T.C

BAHÇEŞEHİR ÜNİVERSİTESİ

INSTITUTE OF SCIENCE

COMPUTER ENGINEERING

A HARDWARE IMPLEMENTATION OF TRUE-MOTION ESTIMATION WITH 3-D RECURSIVE SEARCH BLOCK MATCHING ALGORITHM

Master Thesis

SONER DEDEOĞLU

İSTANBUL, 2008

T.C

BAHÇEŞEHİR ÜNİVERSİTESİ

INSTITUTE OF SCIENCE

COMPUTER ENGINEERING

A HARDWARE IMPLEMENTATION OF TRUE-MOTION ESTIMATION WITH 3-D RECURSIVE SEARCH BLOCK MATCHING ALGORITHM

Master Thesis

SONER DEDEOĞLU

Supervisor: ASST. PROF. DR. HASAN FATİH UĞURDAĞ

İSTANBUL, 2008

T.C

BAHÇEŞEHİR ÜNİVERSİTESİ

INSTITUTE OF SCIENCE

COMPUTER ENGINEERING

Name of the thesis: A Hardware Implementation of True-Motion Estimation with 3-D Recursive Search Block Matching Algorithm
Name/Last Name of the Student: Soner DEDEOĞLU
Date of Thesis Defense: 30 January 2008

The thesis has been approved by the Institute of Science.

Assoc. Prof. Dr. İrini DİMİTRİYADİS
Director

____

I certify that this thesis meets all the requirements as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Adem KARAHOCA
Program Coordinator

____

This is to certify that we have read this thesis and that we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Science.

Examining Committee Members	Signature
Asst. Prof. Dr. Hasan Fatih UĞURDAĞ	____________________
Prof. Dr. Ali GÜNGÖR	____________________
Prof. Dr. Nizamettin AYDIN	____________________
Prof. Dr. Emin ANARIM	____________________
Asst. Prof. Dr. Sezer GÖREN UĞURDAĞ	____________________

ACKNOWLEDGMENTS

This thesis is dedicated to **my family** for their patience and understanding during my master's study and the writing of this thesis. I am also grateful to **Ahmet ÖZUĞURLU** for his moral and spiritual support.

I would like to express my gratitude to **Asst. Prof. Dr. Hasan Fatih UĞURDAĞ** for not only being such a great supervisor but also encouraging and challenging me throughout my academic program.

I wish to thank **Ali SAYINTA, Sinan YALÇIN, and Ümit MALKOÇ** who provided me with great environment at Vestel – Vestek Research and Development Department during my thesis studies.

I also thank **Prof. Dr. Şenay YALÇIN, Prof. Dr. Ali GÜNGÖR, Prof. Dr. Nizamettin AYDIN, Prof. Dr. Emin ANARIM,** and **Asst. Prof. Dr. Sezer GÖREN UĞURDAĞ** for their help on various topics in the areas of digital chip design and digital video processing, for their advice and time.

ABSTRACT

A HARDWARE IMPLEMENTATION OF
TRUE-MOTION ESTIMATION WITH 3-D RECURSIVE
SEARCH BLOCK MATCHING ALGORITHM

DEDEOĞLU, Soner

Computer Engineering

Supervisor: Asst. Prof. Dr. Hasan Fatih UĞURDAĞ

January 2008, 53 pages

Motion estimation, in video processing, is a technique for describing a frame in terms of translated blocks of another reference frame. This technique increases the ratio of video compression by the efficient use of redundancy information between frames. The Block Matching based motion estimation methods, based on dividing frames into blocks and calculating a motion vector for each block, are accepted as motion estimation standards in video encoding systems by international enterprises, such as MPEG, ATSC, DVB and ITU. In this thesis study, a hardware implementation of 3-D Recursive Search Block Matching Algorithm for the motion estimation levels, global and local motion estimations, is presented.

Keywords: Digital Video Processing, Motion Estimation, Very Large Scale Integration

ÖZET

ÜÇ BOYUTLU ÖZYİNELİ ARAMA BLOK UYUMLAMA ALGORİTMASI İLE GERÇEK-HAREKET TAHMİNİNİN DONANIMSAL GERÇEKLEŞTİRMESİ

DEDEOĞLU, Soner

Bilgisayar Mühendisliği

Tez Danışmanı: Yrd. Doç. Dr. Hasan Fatih UĞURDAĞ

Ocak 2008, 53 Sayfa

Hareket tahmini, dijital video işlemede, bir çerçevenin başka bir referans çerçevenin bloklarının çevrilmesi cinsinden tanımlanması tekniğine verilen isimdir. Bu teknik, çerçeveler arası artıklık bilgilerinin daha verimli kullanılması ile video sıkıştırma oranları yükseltilmesini sağlamaktadır. Çerçeveleri bloklara bölerek her blok için bir hareket vektörü hesaplamaya dayanan Blok Uyumlama bazlı hareket tahmini yöntemleri MPEG, ATSC, DVB ve ITU gibi uluslararası kuruluşlar tarafından video kodlama sistemlerinde hareket tahmini standartları olarak kabul edilmiştir. Bu tez çalışmasında hareket tahmini aşamalarından olan global ve lokal hareket tahmini için Üç Boyutlu Özyineli Arama Blok Uyumlama Algoritması donanımsal olarak gerçekleştirilmesi sunulmuştur.

Anahtar Kelimeler: Dijital Video İşleme, Hareket Tahmini, Çok Büyük Ölçekli Tümleşik Devre

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Advanced Television Systems Committee	:	ATSC
Bidirectional Convergence	:	2-D C
Block Motion Compensation	:	BMC
Carry Save Adder	:	CSA
Carry Propagate Adder	:	CPA
Device Under Test	:	DUT
Digital Video Broadcasting	:	DVB
Dual Data Rate	:	DDR
Dynamically Linked Library	:	DLL
Extended Graphics Array	:	XGA
Four Step Search Algorithm	:	FSS
Frame Rate Conversion	:	FRS
Full Search	:	FS
Global Motion Estimation	:	GME
Graphical User Interface	:	GUI
High Definition	:	HD
International Telecommunications Union	:	ITU
Linear Feedback Shift Register	:	LFSR
Liquid Crystal Display	:	LCD
Local Motion Estimation	:	LME
Luminance-Chrominance Color Space	:	YUV
Motion Compensation	:	MC
Motion Estimation	:	ME
Motion Vector	:	MV
Moving Picture Experts Group	:	MPEG
One-At-a-Time Search Algorithm	:	OTS
Orthogonal Search Algorithm	:	OSA
Processing Element	:	PE
Random Access Memory	:	RAM
Recursive Search	:	RS
Red-Green-Blue Color Space	:	RGB

Search Window	:	SW
Sum of Absolute Differences	:	SAD
Three Step Search Algorithm	:	TSS
Two Dimensional Logarithmic Search Algorithm	:	TDL
Wide Extended Graphics Array	:	WXGA

LIST OF SYMBOLS

Candidate set of position $\underline{X}$ on i^{th} frame	:	$CS^{i}(\underline{X},t)$
Candidate vector	:	$\underline{C}(\underline{X},t)$
DDR input data	:	DDR_{in}
DDR output data	:	DDR_{out}
Error function due to candidate vector, $\underline{C}(\underline{X},t)$, on the position $\underline{X}$	:	$\ell(\underline{C},\underline{X},t)$
Global motion vector	:	MV_{global}
Input frame	:	F_{in}
Maximum motion displacement	:	w
Macroblock with upper most left pixel at the location (m,n)	:	$MB_{r}^{i}(m,n)$
Macroblock of position $\underline{X}$	:	$B(\underline{X})$
Output frame	:	F_{out}
Spatial distance	:	r
Spatial recursive vector	:	$\underline{S}_{a}(\underline{X},t)$
Sum of absolute differences due to displacement (u,v)	:	$SAD(u,v)$
Temporal recursive vector	:	$\underline{T}_{a}(\underline{X},t)$
Position on the block grid	:	$\underline{X}$
Prediction vector of position $\underline{X}$ on i^{th} frame	:	$\underline{D}^{i}(\underline{X},t)$
Update value	:	L
Update vector	:	$\underline{U}$
Value of pixel at location (m,n)	:	$p^{i}(m,n)$

1. INTRODUCTION

There have been two significant revolutions in television history. First was in 1954 when the first color TV signals were broadcasted. Nowadays, black-and-white TV signals are unavailable in the airwaves. Second of the revolutions is eventuated by digital TV signals, broadcasted at the end of 1998 first on the air. Analog TV signals have been started to be disappeared from the airwaves as black-and-white TV signals.

Digital TV is not just a provider of quality in video; it also enables many multimedia applications and services to be introduced. While digital video and digital TV technologies are developing rapidly, they triggered the academic researches on the subject, digital video processing. Video processing differs from image processing due to the movements of the objects in video. Understanding how objects move helps us to transmit, store, and manipulate the video in an efficient way. This subject makes the algorithmic development and architectural implementation of motion estimation techniques to be the hottest research topics in multimedia.

This thesis study gives a brief discussion about the well-known motion estimation algorithms and an architectural implementation of true motion estimation with 3-D recursive search block matching algorithm.

In Section 2, the definitions of motion compensation and estimation are given. The proposed well-known motion estimation techniques are briefly listed in algorithmic view and 3-D recursive search block matching algorithm is examined in details at the same section. In Section 3, a hardware implementation for global motion estimation and local motion estimation techniques is proposed. In Section 4, object oriented software to test the architecture is explained at two levels of development: DLL Development and GUI development. In the last section, the Conclusion and future works are given.

2. MOTION ESTIMATION ALGORITHMS

In video compression, motion compensation is a technique for describing a picture in terms of translated copies of portions of a reference picture, often 8x8 or 16x16-pixel blocks. This increases compression ratios by making better use of redundant information between successive frames.

With consumer hardware approaching 1920 pixels per scan line at 24 frames per second for a cinema production a one-pixel-per-frame motion needs more than a minute to cross the screen, many motions are faster. Global motion compensation scrolls the whole screen an integer amount of pixels following a mean motion so that the mentioned methods can work. Block motion compensation divides up the current frame into non-overlapping blocks, and the motion compensation vector tells where those blocks come from in the previous frame, where the source blocks typically overlap.

2.1. GLOBAL MOTION COMPENSATION

In global motion compensation (GMC), the motion model basically reflects camera motions such as dolly (forward, backwards), track (left, right), boom (up, down), pan (left, right), tilt (up, down), and roll (along the view axis). It works best for still scenes without moving objects. There are several advantages of global motion compensation:

- It models precisely the major part of motion usually found in video sequences with just a few parameters. The share in bit-rate of these parameters is negligible.

- It does not partition the frames. This avoids artifacts at partition borders.

- A straight line (in the time direction) of pixels with equal spatial positions in the frame corresponds to a continuously moving point in the real scene. Other MC schemes introduce discontinuities in the time direction.

2.2. BLOCK MOTION COMPENSATION

In block motion compensation (BMC), the frames are partitioned in blocks of pixels. Each block is predicted from a block of equal size in the reference frame. The blocks are not transformed in any way apart from being shifted to the position of the predicted block. This shift is represented by a motion vector.

To exploit the redundancy between neighboring block vectors, it is common to encode only the difference between the current and previous motion vector in the bit-stream. The result of this differencing process is mathematically equivalent to global motion compensation capable of panning. Further down the encoding pipeline, an entropy coder will take advantage of the resulting statistical distribution of the motion vectors around the zero vector to reduce the output size.

It is possible to shift a block by a non-integer number of pixels, which is called sub-pixel precision. The in-between pixels are generated by interpolating the neighboring pixels. Commonly, half-pixel or quarter pixel precision is used. The computational expense of sub-pixel precision is much higher due to the extra processing required for interpolation and on the encoder side, a much greater number of potential source blocks to be evaluated.

The main disadvantage of block motion compensation is that it introduces discontinuities at the block borders (blocking artifacts). These artifacts appear in the form of sharp horizontal and vertical edges which are easily spotted by the human eye and produce ringing effects (large coefficients in high frequency sub-bands) in the Fourier-related transform used for transform coding of the residual frames.

Block motion compensation divides the current frame into non-overlapping blocks, and the motion compensation vector tells where those blocks come from (a common misconception is that the previous frame is divided into non-overlapping blocks, and the motion compensation vectors tell where those blocks move to). The source blocks typically overlap in the source frame. Some video compression algorithms assemble the current frame out of pieces of several different previously-transmitted frames.

2.3. MOTION ESTIMATION

One of the key elements of many video compression schemes is motion estimation (ME). A video sequence consists of a series of frames. To achieve compression, the temporal redundancy between adjacent frames can be exploited. That is, a frame is selected as a reference, and subsequent frames are predicted from the reference using a technique known as motion estimation. The process of video compression using motion estimation is also known as interframe coding.

When using motion estimation, an assumption is made that the objects in the scene have only translational motion. This assumption holds as long as there is no camera pan, zoom, changes in luminance, or rotational motion. However, for scene changes, interframe coding does not work well, because the temporal correlation between frames from different scenes is low. In this case, a second compression technique is used, known as intraframe coding.

In a sequence of frames, the current frame is predicted from a previous frame known as reference frame. The current frame is divided into macroblocks (MB), typically 16x16 pixels in size. This choice of size is a good trade-off between accuracy and computational cost. However, motion estimation techniques may choose different block sizes, and may vary the size of the blocks within a given frame.

Each macroblock is compared to a macroblock in the reference frame using some error measure, and the best matching macroblock is selected. The search is conducted over a predetermined search area, also known as search window (SW). A vector, denoting the displacement of the macroblock in the reference frame with respect to the macroblock in the current frame, is determined. This vector is known as motion vector (MV).

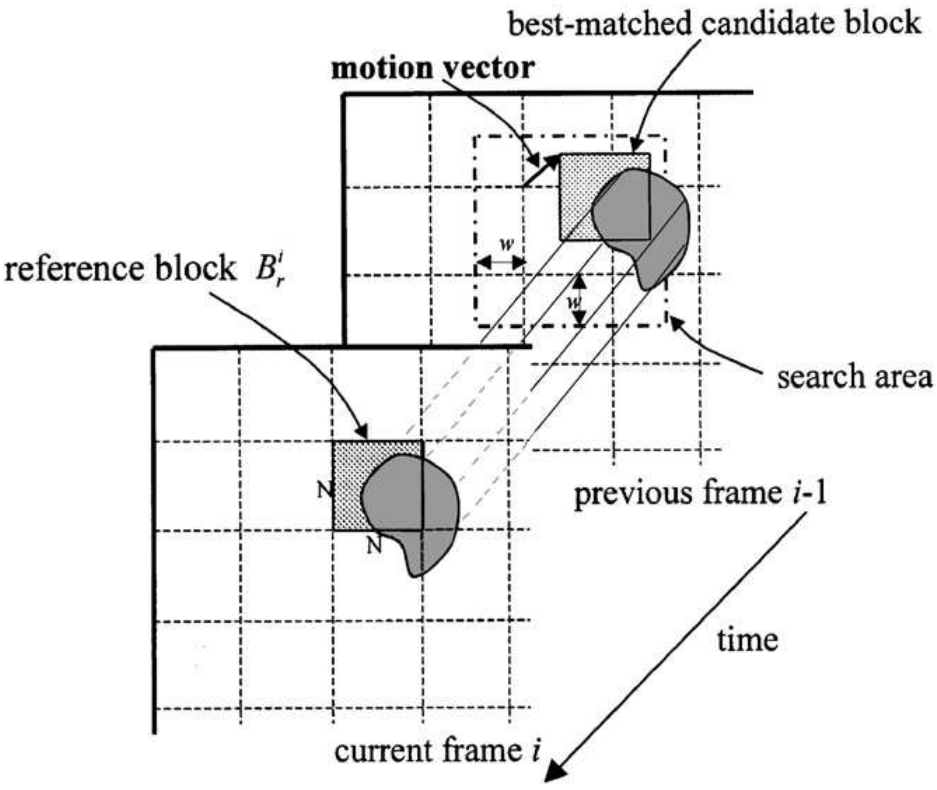


Figure 2.1: Relation between reference frame, current frame and motion vector

When a previous frame is used as a reference, the prediction is referred to as forward prediction. If the reference frame is a future frame, then the prediction is referred to as backwards prediction. Backwards prediction is typically used with forward prediction, and this is referred to as bidirectional prediction.

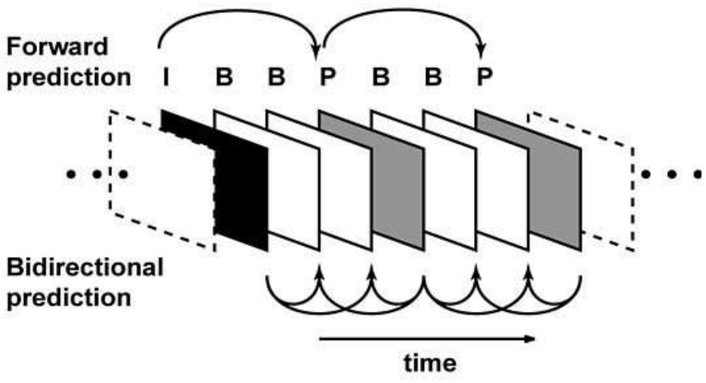


Figure 2.2: Types of frame prediction

2.3.1. Error Function for Block-Matching Algorithms

Block-matching process is performed on the basis of the minimum distortion. In many algorithms SAD, sum of absolute differences, function is adopted as the block distortion measure. Assume $MB_r^i(m,n)$ is the reference block of size NxN pixels whose upper most left pixel is at the location (m,n) of the current frame i, and $MB_r^{i-1}(m+u,n+v)$ is a candidate block within the SW of the previous frame $i-1$ with (u,v) displacement from MB_r^i. Let w be the maximum motion displacement and $p^i(m,n)$ be pixel value at location (m,n) in frame i, then the SAD between MB_r^i and MB_r^{i-1} is defined as

$$SAD(u,v)=\sum_{k=0}^{N-1}\sum_{l=0}^{N-1}\left|p^i(m+k,n+l)-p^{i-1}(m+k+u,n+l+v)\right|, \qquad (2.1)$$

where $-w \le u$, $v \le w$.

The SAD is computed for each candidate block within the SW. A block with the minimum SAD is considered the best-matched block, and the value (u,v) for the best-matched block is called motion vector. That is, motion vector (MV) is given by

$$MV=(u,v)\,|_{\min SAD(u,v)}\,. \qquad (2.2)$$

2.3.2 Full Search (FS) Algorithm

The full search algorithm is the most straightforward brute-force ME algorithm. It matches all possible candidates within the SW. This means that it is at least as accurate (in terms of distortion) as any other block motion estimation algorithm. However, that accuracy comes at the cost of a large number of memory operations and computations. FS is rarely used today, but it remains useful as a benchmark for comparison with other algorithms.

2.3.3. Three Step Search (TSS) Algorithm

This algorithm is introduced by Koga (1981, p. G.5.3.1 - G.5.3.4). It became very popular because of its simplicity, robust and near optimal performance. It searches for the best motion vectors in a coarse to fine search pattern. The algorithm may be described as:

Table 2.1: TSS algorithm

STEP – 1:	An initial step size is picked. Eight blocks at a distance of step size from the center are picked for comparison.
STEP – 2:	The step size is halved. The center is moved to the point with the minimum distortion.
	Steps 1 and 2 are repeated till the step size becomes smaller than 1.

One problem that occurs with the TSS is that it uses a uniformly allocated checking point pattern in the first step, which becomes inefficient for small motion estimation.

2.3.4. Two Dimensional Logarithmic (TDL) Search Algorithm

This algorithm was introduced by Jain and Jain (1981, pp. 1799 – 1808) around the same time that the 3SS was introduced and is closely related to it. Although this algorithm requires more steps than the 3SS, it can be more accurate, especially when the search window is large. The algorithm may be described as:

Table 2.2: TDL algorithm

STEP – 1:	Pick an initial step size. Look at the block at the center and the four blocks at a distance of step size from this on the X and Y axes.

STEP – 2:	If the position of best match is at the center, halve the step size. If however, one of the other four points is the best match, then it becomes the center and step 1 is repeated.
STEP – 3:	When the step size becomes 1, all the nine blocks around the center are chosen for the search and the best among them is picked as the required block.

A lot of variations of this algorithm exist and they differ mainly in the way in which the step size is changed. Some people argue that the step size should be halved at every stage. On the other hand, some people believe that the step size should also be halved if an edge of the SW is reached. However, this idea has been found to fail sometimes.

2.3.5. Four Step Search (FSS) Algorithm

This block matching algorithm was proposed by Po and Ma (1996, pp. 313-317). It is based on the real world image sequence's characteristics of center-biased motion. The algorithm is started with a nine point comparison and then the selection of points for comparison is based on the following algorithm:

Table 2.3: FSS algorithm

STEP – 1:	Start with a step size of 2. Pick nine points around the SW center. Calculate the distortion and find the point with the smallest distortion. If this point is found to be the center of the searching area go to step 4, otherwise go to step 2.
STEP – 2:	Move the center to the point with the smallest distortion. The step size is maintained at 2. The search pattern depends on the position of previous minimum distortion.

a) If the previous minimum point is located at the corner of the previous search area, five points are picked. (Figure 2.3b)

b) If the previous minimum distortion is found at the middle of horizontal or vertical axis of the previous search window, three additional checking points are picked. (Figure 2.3c)

Locate the point the minimum distortion. If this is at the center, go to step 4, otherwise go to step 3.

STEP – 3: The search pattern strategy is the same, however it will finally go to step 4.

STEP – 4: The step size is reduced to 1 and all nine points around the center of the search area examined.

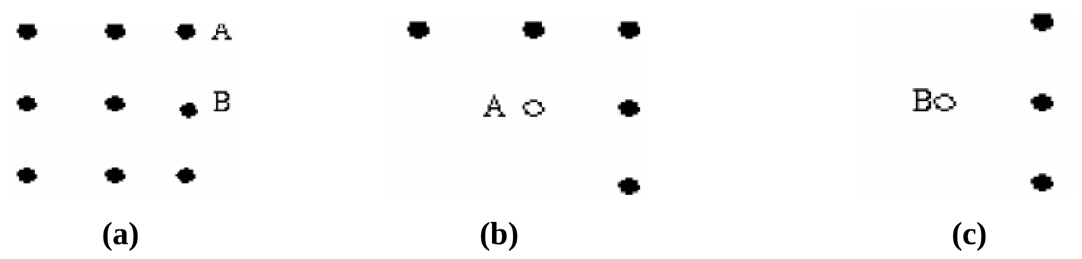


Figure 2.3: Illustration of selection of blocks for different cases is FSS
(a) Initial Configuration. (b) If point A has minimum distortion, pick given five points.
(c) If point B has minimum distortion, pick given three points.

The computational complexity of the FSS is less than that of the TSS, while the performance in terms of quality is better. It is also more robust than the TSS and it maintains its performance for image sequence with complex movements like camera zooming and fast motion. Hence it is a very attractive strategy for motion estimation.

2.3.6. Orthogonal Search Algorithm (OSA)

Puri, Hang and Schilling (1987, pp. 1063 – 1066) introduced the algorithm as a hybrid of the TSS and TDL algorithms. Vertical stage is followed by a horizontal stage to search the optimal block. Steps of algorithm may be listed as follows:

Table 2.4: OSA algorithm

STEP – 1:	Pick a step size (usually half the maximum displacement in the SW). Take two points at a distance of step size in the horizontal direction from the center of the SW and locate the point of minimum distortion. Move the center to this point.
STEP – 2:	Take two points at a distance step size from the center in vertical direction and find the point with the minimum distortion.
STEP – 3:	Halve the step size. If it is greater than or equal to one go to step 1, otherwise halt.

2.4. 3-D RECURSIVE SEARCH BLOCK MATCHING ALGORITHM

Several algorithms, including the algorithms mentioned previous section, have been proposed for frame rate conversion for consumer television applications. There exists a common problem due to complexity of the motion estimator while VLSI implementation of these algorithms; on the other hand, the existing simpler algorithms, such as One-At-a-Time Search (OTS) Algorithm of Srinivasan and Rao (1985, pp. 888 – 896), cause very unnatural artifacts.

De Haan, Biezen, Huijgen and Ojo (1993, pp. 368 – 379) proposed a new recursive block-matching motion estimation algorithm, called 3-D Recursive Search Block-Matching Algorithm. Measured with criteria relevant for the FRC application, this algorithm is shown to have a superior performance over alternative algorithms, with a significantly less complexity.

2.4.1. 1-D Recursive Search

The block-matching algorithms, as the most attractive for VLSI implement, limit number of candidate vectors to be evaluated. This can be realized through recursively optimizing a previously found vector, which can be either spatially or temporally neighboring result.

If spatially and temporally neighboring MVs are believed to predict the displacement reliably, a recursive algorithm should enable true ME, if the amount of updates are around the prediction vector is limited to a minimum. The spatial prediction was excluded for the candidate set:

$$CS^{i}(\underline{X},t)=\left\{\underline{C}\in CS^{\max} \mid \underline{C}=\underline{D}^{i-1}(\underline{X},t)+\underline{U}, \underline{U}=\begin{pmatrix}\pm L\\ 0\end{pmatrix}\vee\begin{pmatrix}0\\ \pm L\end{pmatrix}\right\} \qquad (2.3)$$

where L is the update length, which is measured on the frame grid, $\underline{X}=(X,Y)^{T}$ is the position on the block grid, t is time, and the prediction vector $\underline{D}^{i-1}(\underline{X},t)$ is selected according to:

$$\underline{D}^{i-1}(\underline{X},t)\in\begin{cases}\{\underline{0}\}, & (i=1)\\ \{\underline{C}\in CS^{i-1}(\underline{X},t) \mid \ell(\underline{C},\underline{X},t)\leq \ell(\underline{F},\underline{X},t), \forall \underline{F}\in CS^{i-1}(\underline{X},t)\} & (i>1)\end{cases} \qquad (2.4)$$

and the candidate set is limited to a set $CS^{\max}$:

$$CS^{\max}=\{\underline{C} \mid -N\leq C_{x}\leq +N, -M\leq C_{y}\leq +M\} \qquad (2.5)$$

The resulting estimated MV $\underline{D}(\underline{x},t)$, which is assigned to all pixel positions, $\underline{x}=(x,y)^{T}$, in the block $B(\underline{X})$ of size $X\times Y$ with center $\underline{X}$:

$$B(\underline{X})=\{\underline{x} \mid X_{x}-X/2\leq x\leq X_{x}+X/2\wedge X_{y}-Y/2\leq y\leq X_{y}+Y/2\} \qquad (2.6)$$

equals the candidate vector $\underline{C}(\underline{X},t)$ with the smallest error $\ell(\underline{C},\underline{X},t)$:

$$\forall \underline{x} \in B(\underline{X}) : \underline{D}(\underline{x},t) \in \left\{ \underline{C} \in CS^i(\underline{X},t) \mid \ell(\underline{C},\underline{X},t) \le \ell(\underline{F},\underline{X},t), \forall \underline{F} \in CS^i(\underline{X},t) \right\} \quad (2.7)$$

Errors are calculated as summed absolute differences (SAD):

$$\ell(\underline{C},\underline{X},t) = \sum_{\underline{x} \in B(\underline{X})} \left| F(\underline{x},t) - F(\underline{x} - C, t - n \cdot T) \right| \quad (2.8)$$

where $F(\underline{x},t)$ is the luminance function and T the field period. The block size is fixed to $X = Y = 8$, although experiments indicate little sensitivity of the algorithm to this parameter.

2.4.2. 2-D Recursive Search

It is well known that, the convergence can be improved with predictions calculated from a 2-D area or even a 3-D space. In this section, 2-D prediction strategy is introduced that does not dramatically increase the complexity of the hardware.

The essential difficulty with 1-D recursive algorithm is that it cannot manage the discontinuities at the edges of moving objects. The first impression may be that smoothness constraints exclude good step response in a motion estimator. The dilemma of combining smooth vector fields can be assailed with a good step response.

When the assumption, that the discontinuities in the velocity plane are spaced at a distance that enables convergence of the recursive block matcher in between two discontinuities, the recursive block matcher yields the correct vector value at the first side of the object boundary and begins converging at the opposite side.

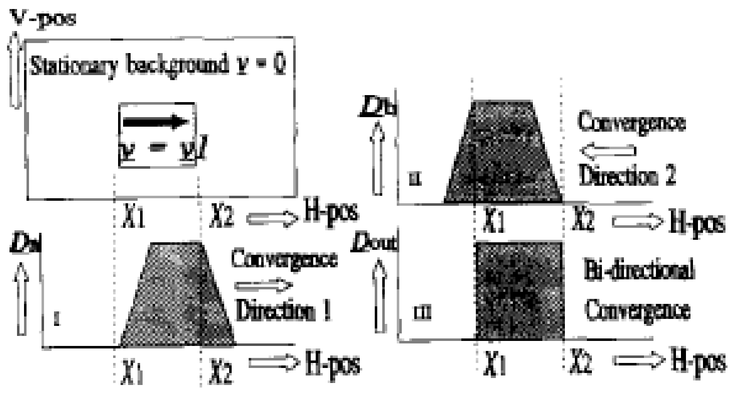


Figure 2.4: The bidirectional convergence (2-D C) principle

It seems attractive to apply two estimator processes at the same time with the opposite convergence directions (Fig. 2.4). SAD of both vectors can be used for selection. 2-D C is formalized as a process that generates a MV:

$$\forall \underline{x} \in B(\underline{X}) : \underline{D}(\underline{x}) = \begin{cases} \underline{D}_a(\underline{X},t), & (\ell(\underline{D}_a,\underline{X},t) \le \ell(\underline{D}_b,\underline{X},t)) \\ \underline{D}_b(\underline{X},t), & (\ell(\underline{D}_a,\underline{X},t) > \ell(\underline{D}_b,\underline{X},t)) \end{cases} \tag{2.9}$$

where

$$\ell(\underline{D}_a,\underline{X},t) = \sum_{\underline{x}\in B(\underline{X})} \left| F(\underline{x},t) - F(\underline{x}-\underline{D}_a,t-T) \right| \tag{2.10}$$

and

$$\ell(\underline{D}_b,\underline{X},t) = \sum_{\underline{x}\in B(\underline{X})} \left| F(\underline{x},t) - F(\underline{x}-\underline{D}_b,t-T) \right| \tag{2.11}$$

while $\underline{D}_a$ and $\underline{D}_b$ are found in a spatial recursive process prediction vectors $\underline{S}_a(\underline{X},t)$:

$$\underline{S}_a(\underline{X},t) = D_a(\underline{X} - \underline{SD}_a,t) \tag{2.12}$$

and $\underline{S}_b(\underline{X},t)$:

$$\underline{S}_b(\underline{X},t) = D_b(\underline{X} - \underline{SD}_b, t) \tag{2.13}$$

where

$$\underline{SD}_a \neq \underline{SD}_b\,. \tag{2.14}$$

The two estimators have unequal spatial recursion vectors $\underline{SD}$. One of these estimators have converged already at the position where the other is yet to do so, that is how 2-D C solves the run-in problem at edges of moving objects, if the two convergence directions are opposite. The attractiveness of a convergence direction varies significantly for hardware (Fig. 2.5). The predictions taken from blocks 1, 2, or 3, are favorable for hardware. Block 4 is less attractive, as it perplexes pipelining of algorithm that the previous result has to be ready before the next can be calculated. Block 5 is not attractive because of reversing the line scan. Blocks 6, 7, and 8 are totally unattractive because of reversing vertical scan direction. Reversing horizontal and vertical scans require extra memories in the hardware.

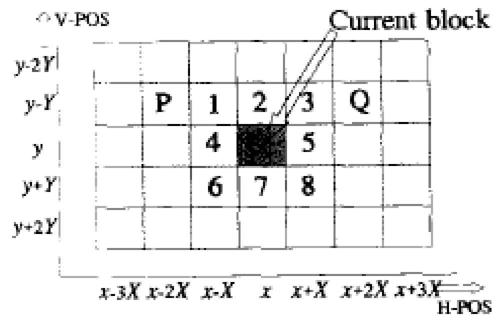


Figure 2.5: Locations around the current block, from which the estimation result could be used as a spatial prediction vector.

When applying only the preferred blocks, the best implementation of 2-D C results with predictions from blocks 1 and 3. By taking predictions from blocks P and Q, it is possible to enlarged the angle between the convergence direction, however, it is observed worse results rather than blocks 1 and 3 for 2-D C.

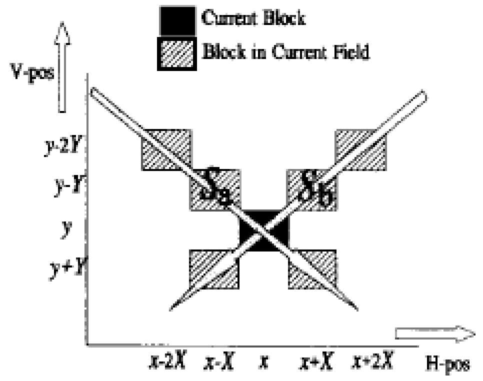


Figure 2.6: Location of the spatial predictions of estimators a and b with respect to the current block.

2.4.3. 3-D Recursive Search

Both estimators, a and b, in algorithm produce four candidate vectors each by updating their spatial predictions $\underline{S}_a(\underline{X},t)$ and $\underline{S}_b(\underline{X},t)$. The spatial predictions were chosen to create two perpendicular diagonal convergence axes:

$$\underline{S}_a(\underline{X},t) = \underline{D}_a\left(\underline{X} - \begin{pmatrix} X \\ Y \end{pmatrix}, t\right) \tag{2.15}$$

and

$$\underline{S}_b(\underline{X},t) = \underline{D}_b\left(\underline{X} - \begin{pmatrix} -X \\ Y \end{pmatrix}, t\right). \tag{2.16}$$

Due to movements in picture, displacements between two consecutive velocity planes are small compared to MB size. The definition of a third and a forth estimators, c and d, is enabled by this assumption.

Selection of predictions for estimators, c and d, from position 6 and 8 (Fig. 2.5), respectively, creates additional convergence directions opposite to predictions of a and b; however, the resulting design reduces the convergence speed due to temporal component in prediction delays of c and d.

Instead of choosing additional estimators, c and d, it is suggested to apply vectors from positions opposite to the spatial prediction position as additional candidates in the already defined estimators to save hardware with the calculation of fewer errors. De Haan (1992) keynotes that; working with fewer candidates reduces the risk of inconsistency.

As the algorithm is improved, a fifth candidate in each spatial estimator, a temporal prediction value from previous field accelerates the convergence. These convergence accelerators are taken from a MB shifted diagonally over r MBs and opposite to the MBs from which $\underline{S}_a$ and $\underline{S}_b$ result:

$$\underline{T}_a(\underline{X},t) = \underline{D}\left(\underline{X} + r \cdot \begin{pmatrix} X \\ Y \end{pmatrix}, t - T\right) \qquad (2.17)$$

and

$$\underline{T}_b(\underline{X},t) = \underline{D}\left(\underline{X} + r \cdot \begin{pmatrix} -X \\ Y \end{pmatrix}, t - T\right). \qquad (2.18)$$

By the experimental results, $r = 2$ is the best spatial distance for a MB size of 8x8 pixels.

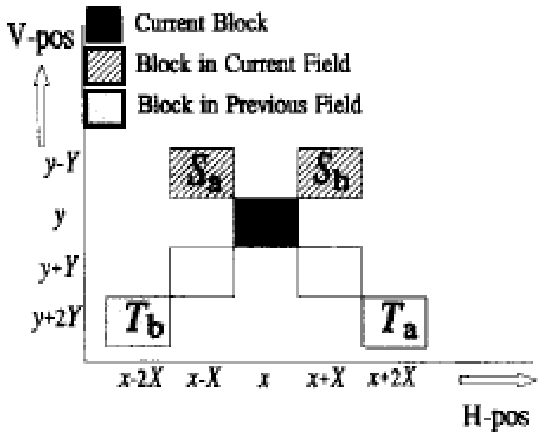


Figure 2.7: The relative positions of the spatial predictors $\underline{S}_a$ and $\underline{S}_b$ and the convergence accelerators $\underline{T}_a$ and $\underline{T}_b$

For the resulting algorithm, $\underline{D}_a(\underline{X},t)$ and $\underline{D}_b(\underline{X},t)$ result from estimators, a and b, calculated in parallel with the candidate set CS_a:

$$\begin{aligned} CS_a(\underline{X},t) &= \left\{ \underline{C} \in CS^{\max} \middle| \underline{C} = \underline{D}_a\left(\underline{X} - \begin{pmatrix} X \\ Y \end{pmatrix}, t \right) + \underline{U}, \right. \\ \underline{U} &= \left. \begin{pmatrix} \pm L \\ 0 \end{pmatrix} \vee \begin{pmatrix} 0 \\ \pm L \end{pmatrix} \right\} \cup \left\{ \underline{D}\left(\underline{X} + 2 \cdot \begin{pmatrix} X \\ Y \end{pmatrix}, t - T \right) \right\} \end{aligned} \tag{2.19}$$

and CS_b:

$$\begin{aligned} CS_b(\underline{X},t) &= \left\{ \underline{C} \in CS^{\max} \middle| \underline{C} = \underline{D}_b\left(\underline{X} - \begin{pmatrix} -X \\ Y \end{pmatrix}, t \right) + \underline{U}, \right. \\ \underline{U} &= \left. \begin{pmatrix} \pm L \\ 0 \end{pmatrix} \vee \begin{pmatrix} 0 \\ \pm L \end{pmatrix} \right\} \cup \left\{ \underline{D}\left(\underline{X} + 2 \cdot \begin{pmatrix} -X \\ Y \end{pmatrix}, t - T \right) \right\} \end{aligned} \tag{2.20}$$

while distortions are assigned to candidate vectors using the SAD function (Eq. 2.8).

2.5. GLOBAL MOTION ESTIMATION TECHNIQUE

Camera effects, i.e., panning, tilting, travelling, and zooming, have very regular character causing very smooth MVs compared to the object motion. Zooming with the camera yields MVs, which linearly change with the spatial position. On the other hand, other camera effects generate a uniform MV, called global motion vector, field for the entire video.

To estimate MV_{global}, a sample set $S(t)$, proposed by De Haan and Biezen (1998, pp. 85 – 92), containing nine MVs, $\underline{D}(\underline{X}, t-1)$ from different positions $\underline{X}$ on the MB grid in a centered window of size $(W-2m)X$ by $(H-2q)Y$ in the picture with the width $W \cdot X$ and the height $H \cdot Y$ from the temporal vector prediction memory according to

$$S(t) = \left\{ \underline{D}(\underline{X}, t-1) \middle| X_X = -\left(\frac{1}{2}W - m\right)X, 0, +\left(\frac{1}{2}W - m\right)X, \right. \\ \left. X_y = -\left(\frac{1}{2}H - q\right)Y, 0, +\left(\frac{1}{2}H - q\right)Y \right\} \quad (2.21)$$

where the values of m and q are noncritical.

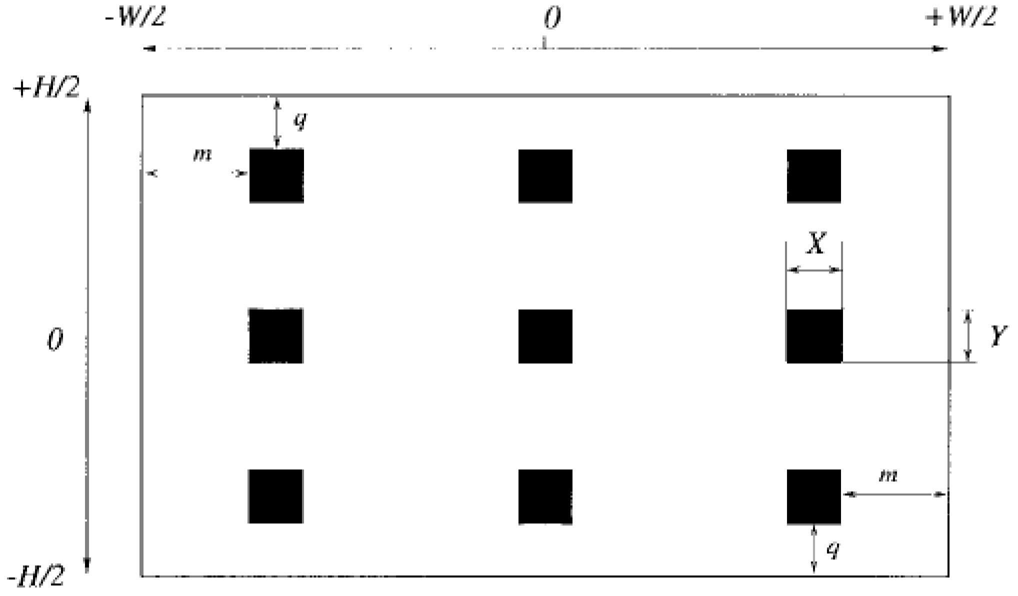


Figure 2.8: Position of the sample SWs to find MV_{global} in the image plane

Global motion estimation to find MV_{global} differs from local motion estimation due to MB sizes. The MB size related to global motion estimation is fixed to $X = Y = 16$.

Another difference between global and local motion estimations is the algorithm to find MVs. MV_{global} is calculated in each SWs by Full Search (FS) Algorithm, on the other hand, local displacement vectors are calculated by 3-D RS. Although it is possible to choose anyone other block matching algorithms instead of FS to reduce the number of computations, with the very limited number of search windows and the aim to find more accurate global displacement vector, FS is performed and $S(t)$ (Eq. 2.21) is filled.

The resultant MV_{global} is derived AS the median vector of each MVs in $S(t)$:

$$MV_{global} = \left(median\left(S_x(0), S_x(1), \ldots, S_x(8)\right), median\left(S_y(0), S_y(1), \ldots, S_y(8)\right)\right) \quad (2.22)$$

and added as an additional candidate vector to candidate set in order to use in local motion estimation.

3. MOTION ESTIMATION HARDWARE

3.1. VIDEO FORMAT

Wide Extended Graphics Array (WXGA) is one of the non-standard resolutions, derived from the XGA, referring to a resolution of 1366x768. WXGA became the most popular one for the LCD and HD televisions in 2006 for wide screen presentation at an aspect ratio of 16:9. Video frames, whose rate to be converted by the motion estimation and compensation in this master thesis work, have WXGA resolution.

A significant point related to the input video format is that it is composed of consecutive repeated frame of each frame (Fig. 3.1).

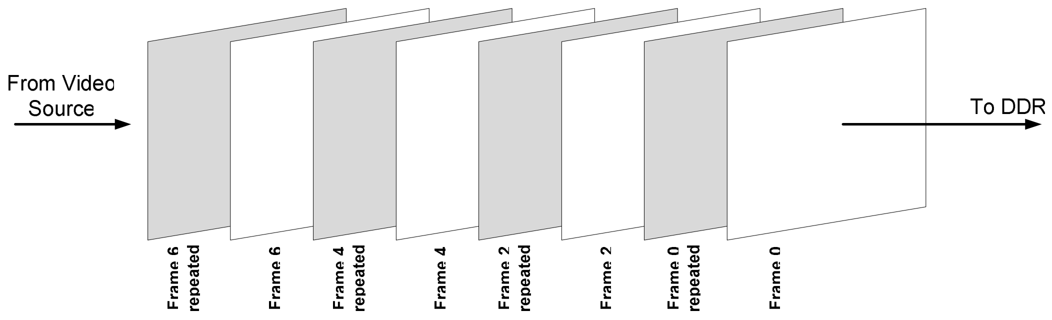


Figure 3.1: Video sequence composed of repeated frames

Because each frame is followed by its duplicated copy, it is not necessary to store all the frames provided by video source into memory. Repeated frames are skipped for memory storage, however, they are not completely omitted. Repeated frames are used while outputting the video frames to the display screen.

Table 3.1: Input frame sequence and storage into DDR

Frame Time	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**…**
F_{in}	F_0	F_0	F_2	F_2	F_4	F_4	F_6	F_6	F_8	F_8	…
DDR_{in}	F_0		F_2		F_4		F_6		F_8		⋯

The objective with ME and MC is to generate new sub-frames by interpolation of MBs with MVs instead of repeated frames and outputting the video frames that have a higher frame rate.

Table 3.2: Timeline representation of DDR access, ME, and generation of output video sequence

Frame Time	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**…**
F_{in}	F_0	F_0	F_2	F_2	F_4	F_4	F_6	F_6	F_8	F_8	…
DDR_{in}	F_0	↓	F_2	↓	F_4	↓	F_6	↓	F_8	↓	…
DDR_{out}		↓	F_0	↓	F_2	↓	F_4	↓	F_6	↓	…
ME		↓	F_1 ↓	↓	F_3 ↓	↓	F_5 ↓	↓	F_7 ↓	↓	…
F_{out}		F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	…

3.2. HIGH-LEVEL ARCHITECTURE OF HARDWARE

Fully implemented motion estimation and compensation hardware consists of five main components: data converters, external memory block, memory interface, motion estimator, and motion compensator.

Color values of each pixel of a video frame are stored in RGB format in video sources, and digital displayers need also RGB pixel values to show the frames, however, motion estimation algorithms are performed on gray-scaled images. A method to obtain gray-scaled image is to convert the color space into YUV color space, which separates the gray-scale (Y - luminance) and color information (U and V) with the equations

$$\begin{aligned} Y &= ((66\times R + 129\times G + 25\times B + 128) >> 8) + 16 \\ U &= ((-38\times R - 74\times G + 112\times B + 128) >> 8) + 128 \, . \\ V &= ((112\times R - 94\times G - 18\times B + 128) >> 8) + 128 \end{aligned} \qquad (3.1)$$

To regenerate RGB data from YUV color space for displaying the frame on display screen a reverse conversion is provided using the equations

$$\begin{aligned}
R &= MIN((9535\times(Y-16)+13074\times(V-128))>>13,255)\\
G &= MIN((9535\times(Y-16)-6660\times(V-128)-3203\times(U-128))>>13,255). \qquad (3.2)\\
B &= MIN((9535\times(Y-16)+16531\times(U-128))>>13,255)
\end{aligned}$$

An RGB2YUV converter hardware block is placed behind the video source; likewise, a YUV2RGB converter is installed in front of the display screen to convert the pixel values to RGB formats.

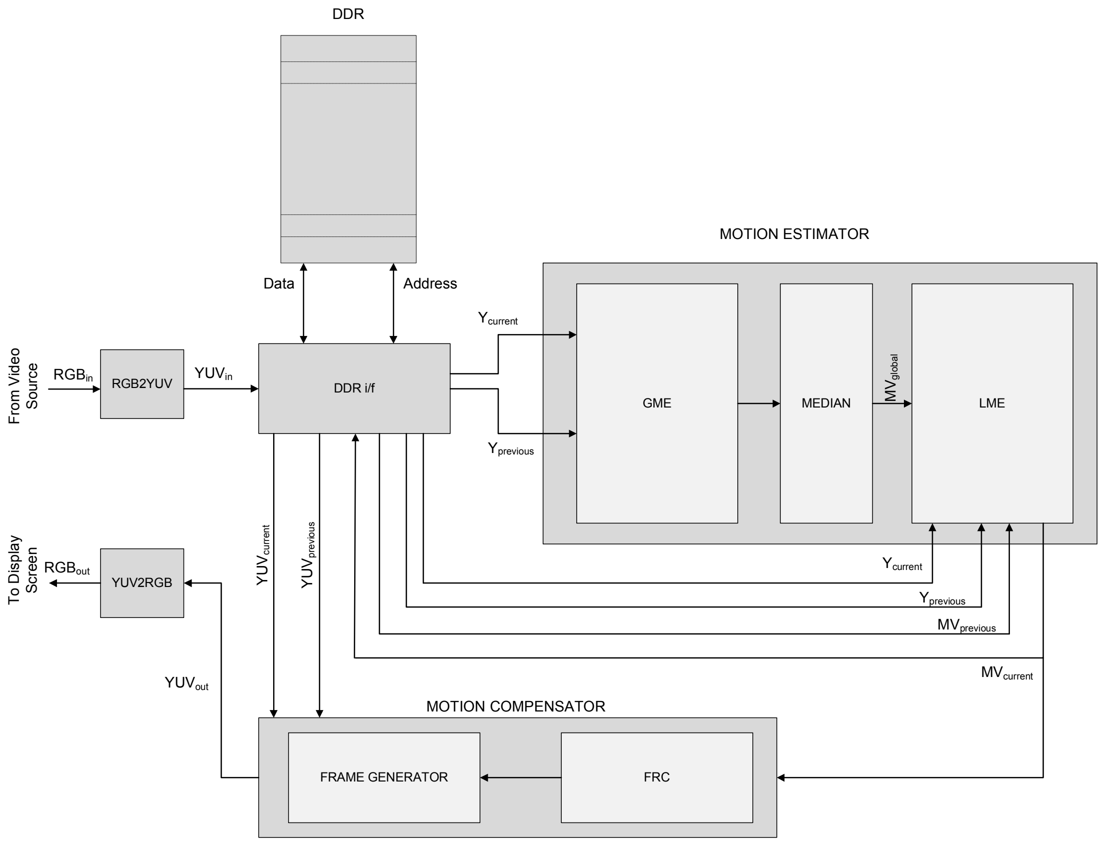


Figure 3.2: High-level block diagram motion estimator/compensator architecture

DDR, as external memory, is used in architecture to store incoming frames and the estimated motion vectors to be used in the following steps of motion estimation.

DDR interface block acts as a global bridge in the system and controls the DDR, Motion Estimator and Motion Compensator blocks. DDR interface is the block where the packing strategy of pixels, presented in following section, is operated.

Motion Estimator is the main hardware component of the whole system whose functionality is presented in details following sections.

Motion Compensator is end-point of the architecture where the estimated vectors to be used for interpolation and generation of interframes to increase the frame rate of the original video sequence.

3.3. PACKING STRATEGY OF PIXELS

In architectures for the block-matching algorithms, memory configuration plays an important role. It enables the exploitation of various techniques such as parallelism and pipelining. The motion-estimation techniques are performed with a great amount data during the computations. This requires a decrement in the number of external memory access and fetching more pixels from DDR at a single cycle.

Pixels from video source are received one by one every pixel clock and converted into YUV color space. Instead of storing 24-bit YUV value of each pixel into each word of external memory, every YUV value is divided into 8-bit Y, which is the only value of pixel used in motion estimation, and 16-bit UV block and for four consecutive pixels 8-bit Y values and 16-bit UV values are buffered in DDR interface. Four pieces of Y values are combined to get a 32-bit word; likewise, two pieces of UV values, selected according to 4:2:2 co-sited sampling, are combined to yield another 32-bit word, and then these words are stored to related address of external memory. This configuration of memory provides the motion estimator to fetch luminance values of four consecutive pixels at a single access to external memory.

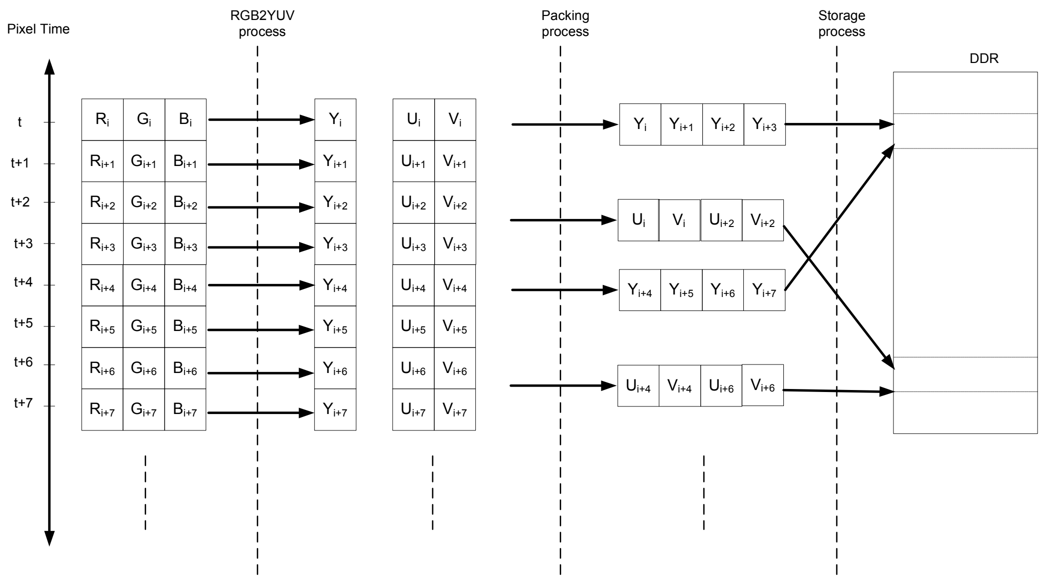


Figure 3.3: Packing strategy of pixels and DDR storage

3.4. GLOBAL MOTION ESTIMATOR

Global motion estimator is the component to detect the global movements in the background image of frame as a result of camera effects. It is based on FS block-matching strategy on fixed reference locations of each frame and extracting a global MV after scanning the reference SWs.

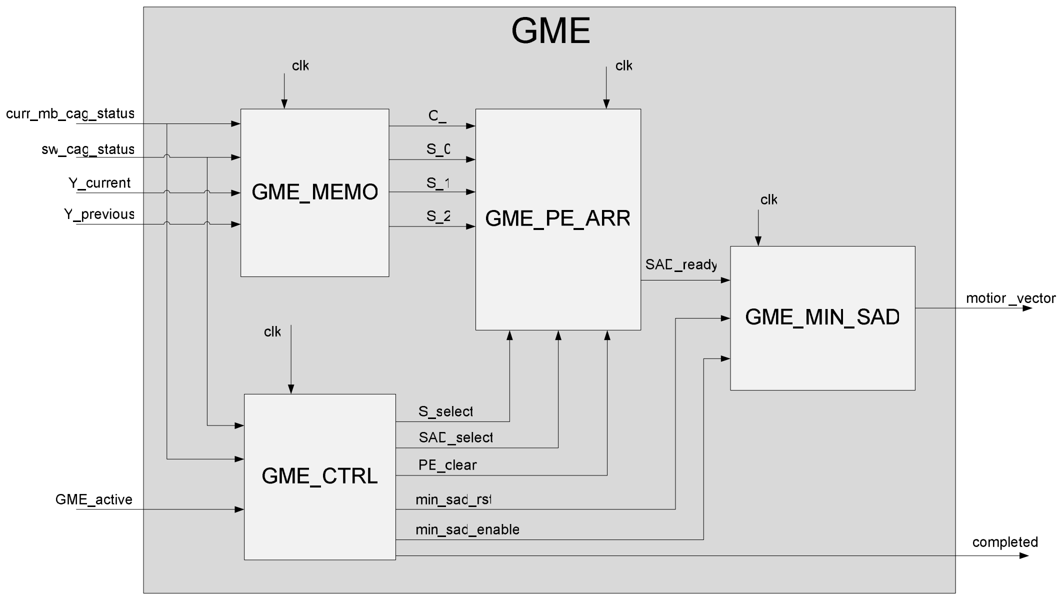


Figure 3.4: Global motion estimator block diagram

3.4.1. GME Memory Structure

FS block-matching algorithm is performed between current frame with the MB of 16x16 in size and previous frame with the SW of 48x36 in size, calculated with the search range of ±16 in horizontal and ±10 in vertical.

To reduce the number of access to external memory, MB and SW are totally fetched to internal memories, i.e. Block RAMs of FPGA, before the FS is started. The structure of the DDR words, internal block-RAMs is set to 32-bit in width. Because each word consists of four luminance values, the numbers of addresses of SW block RAM and MB block RAM are set to $\frac{(48\times36)}{4} = 432$ and $\frac{(16\times16)}{4} = 64$, respectively.

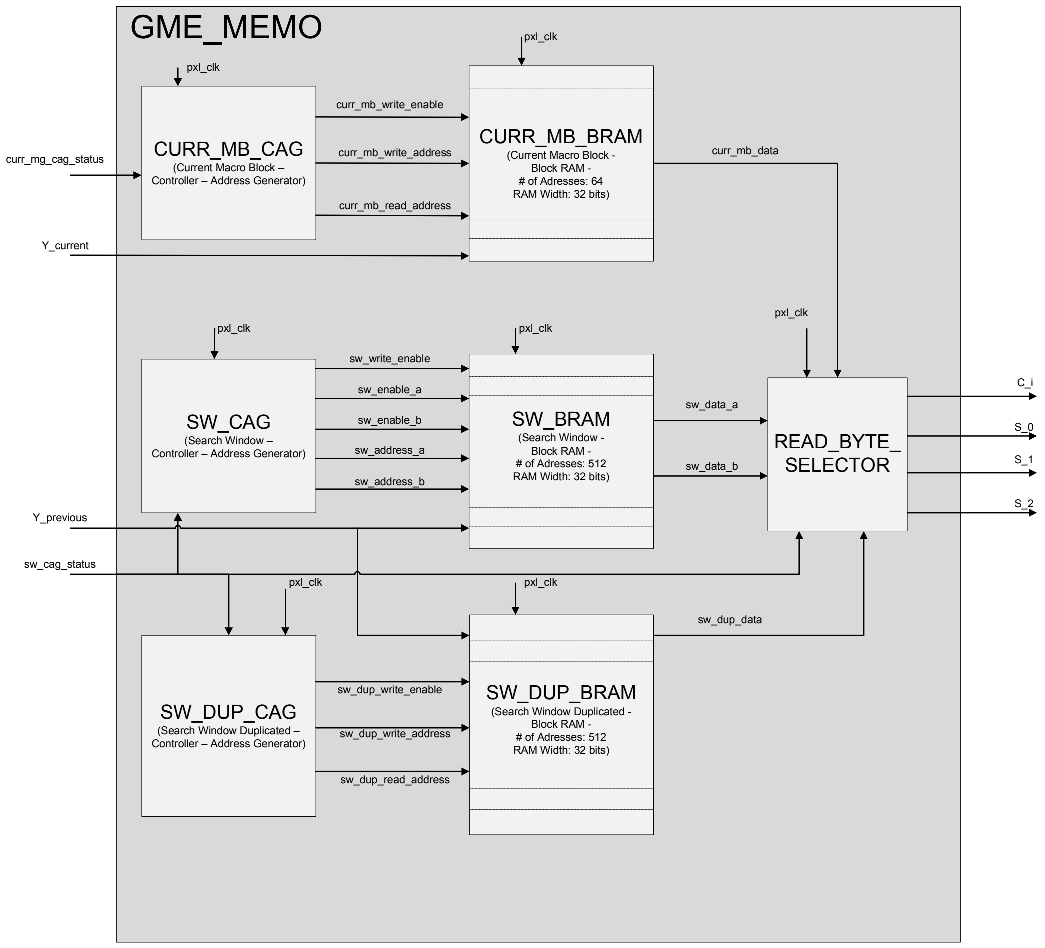

GME_MEMO
pxl_clk
CURR_MB_CAG
(Current Macro Block – Controller – Address Generator)
curr_mb_write_enable
curr_mb_write_address
curr_mb_read_address
CURR_MB_BRAM
(Current Macro Block - Block RAM - # of Adresses: 64 RAM Width: 32 bits)
curr_mb_data
curr_mg_cag_status
Y_current
SW_CAG
(Search Window – Controller – Address Generator)
sw_write_enable
sw_enable_a
sw_enable_b
sw_address_a
sw_address_b
SW_BRAM
(Search Window - Block RAM - # of Adresses: 512 RAM Width: 32 bits)
sw_data_a
sw_data_b
READ_BYTE_ SELECTOR
C_i
S_0
S_1
S_2
Y_previous
sw_cag_status
SW_DUP_CAG
(Search Window Duplicated – Controller – Address Generator)
sw_dup_write_enable
sw_dup_write_address
sw_dup_read_address
SW_DUP_BRAM
(Search Window Duplicated - Block RAM - # of Adresses: 512 RAM Width: 32 bits)
sw_dup_data

Figure 3.5: Memory structure of global motion estimator

Three luminance values of SW are required to provide the regularity of the data flow to processing elements in FS algorithm; however, a single block-RAM is eligible to provide two pixel data, S_0 and S_1, over its one read and one read/write ports. So an additional block-RAM, labeled as SW_DUP_BRAM in Fig. 3.5, is installed on global motion estimation structure to transmit the third necessary data, S_2, to processing elements. The contents of additional memory block and the original SW memory block are identical.

READ_BYTE_SELECTOR is a multiplexing structure to select the essential bytes for PEs from the 32-bit outputs of block-RAMs, CURR_MB_BRAM, SW_BRAM, and SW_DUP_BRAM. It decides the luminance to be selected by a simple 2-bit counter inside.

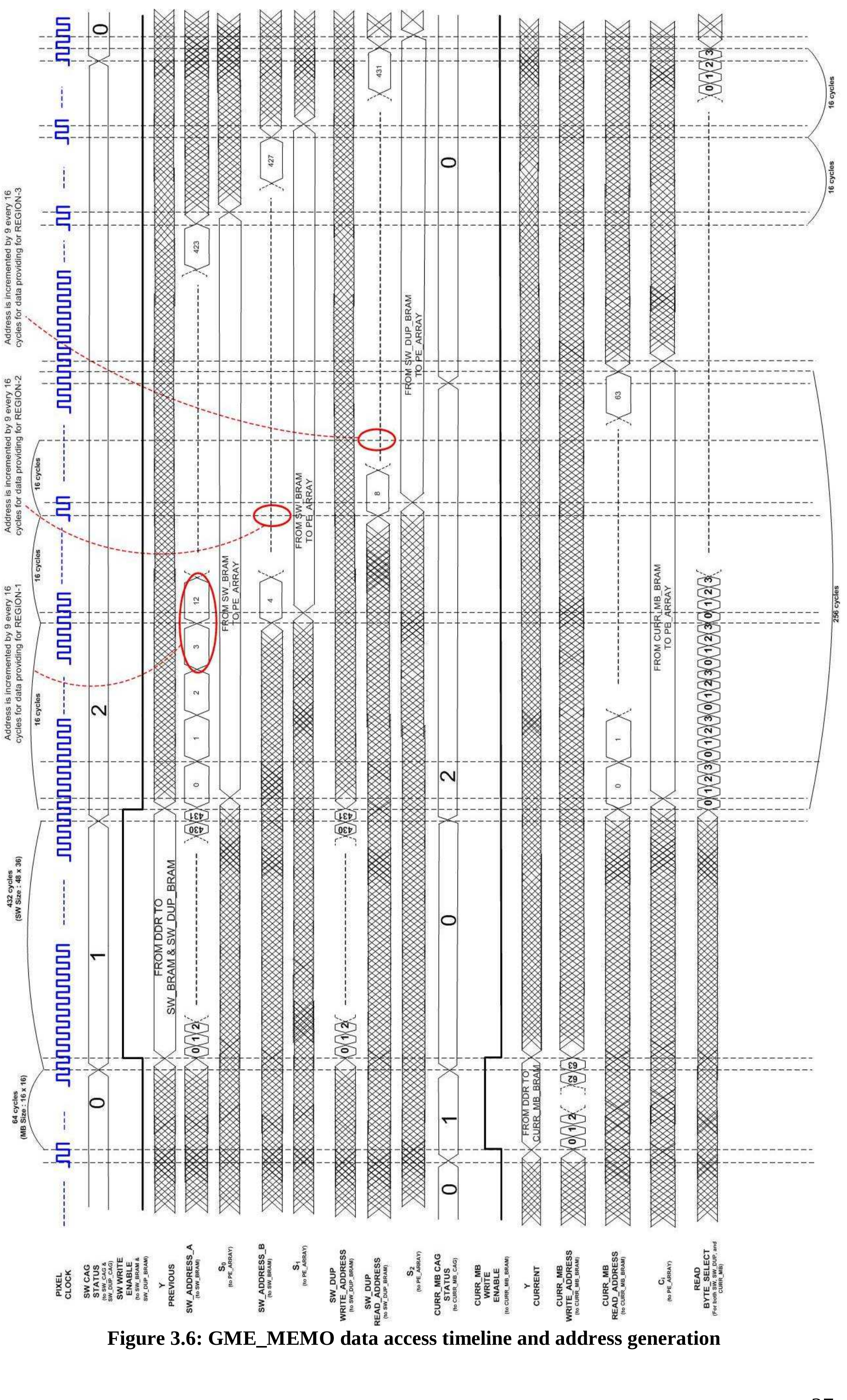
Address generators of block-RAMs are controlled by the status inputs (Table 3.3 and Table 3.4), fed from DDR interface.

Table 3.3: Address generator states

State Number	Meaning
0	IDLE
1	WRITE TO BLOCK-RAM
2	READ FROM BLOCK-RAM

Table 3.4: Address generation algorithm

Previous State	Current State	To Do
0	0	Do nothing
0	1	Enable writing over block-RAM. Reset write address.
0	2	Enable reading from block-RAM. Reset read address.
1	0	Disable writing.
1	1	Increase write address by appropriate value.
1	2	Disable writing. Enable reading from block-RAM. Reset read address.
2	0	Disable reading.
2	1	Unreachable state transition.
2	2	Increase/Decrease read address by appropriate value.



PIXEL CLOCK
SW CAG STATUS (to SW_CAG & SW_DUP_CAG)
SW WRITE ENABLE (to SW_BRAM & SW_DUP_BRAM)
Y PREVIOUS
SW_ADDRESS_A (to SW_BRAM)
S0 (to PE_ARRAY)
SW_ADDRESS_B (to SW_BRAM)
S1 (to PE_ARRAY)
SW_DUP WRITE_ADDRESS (to SW_DUP_BRAM)
SW_DUP READ_ADDRESS (to SW_DUP_BRAM)
S2 (to PE_ARRAY)
CURR_MB CAG STATUS (to CURR_MB_CAG)
CURR_MB WRITE ENABLE (to CURR_MB_BRAM)
Y CURRENT
CURR_MB WRITE_ADDRESS (to CURR_MB_BRAM)
CURR_MB READ_ADDRESS (to CURR_MB_BRAM)
Ci (to PE_ARRAY)
READ BYTE_SELECT (For both SW_DUP, and CURR_MB)
64 cycles (MB Size : 16 x 16)
432 cycles (SW Size : 48 x 36)
16 cycles
256 cycles
Address is incremented by 9 every 16 cycles for data providing for REGION-1
Address is incremented by 9 every 16 cycles for data providing for REGION-2
Address is incremented by 9 every 16 cycles for data providing for REGION-3
FROM DDR TO CURR_MB_BRAM
FROM DDR TO SW_BRAM & SW_DUP_BRAM
FROM SW_BRAM TO PE_ARRAY
FROM SW_DUP_BRAM TO PE_ARRAY
FROM CURR_MB_BRAM TO PE_ARRAY

Figure 3.6: GME_MEMO data access timeline and address generation

3.4.2. GME Processing Element Array

Due to the search range of ± 16 locations in horizontal, there exists 33 search points for each line of SW. This enables a set of parallel 33 PEs in processing element array structure of global motion estimator. Each PE is assigned to calculate the SAD of the corresponding search location. After a completed calculation PE is assigned for a new SAD calculation of the search location in next line with the same column.

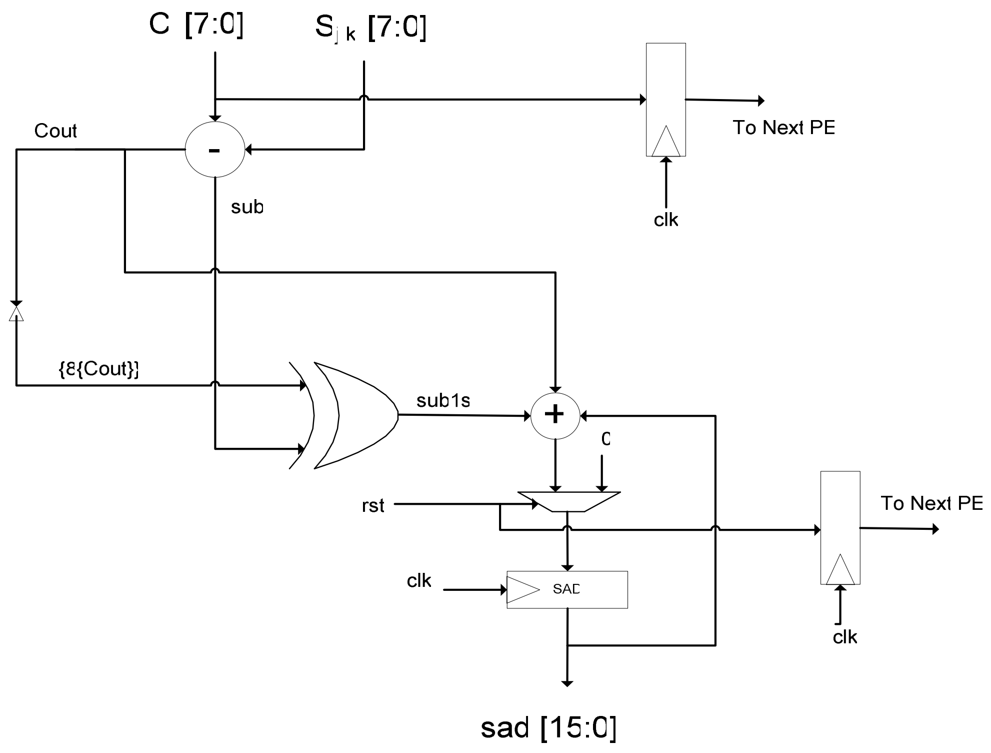


Figure 3.7: Structure of GME processing element

PE array data of SW is provided by the GME memory structure over 3 luminance ports S_0, S_1 and S_2, however, each PE uses only 1 or 2 of this luminance values due to the region of the corresponding its search location. A SW consists of 3 search regions. Columns 0-15, 17-32, 33-48 are defined as region-0, region-1, and region-2, respectively. The data providing of these regions are shared to the luminance ports S_0, S_1, and S_2; on the other hand, the luminance values of current MB are provided over single port, labeled as C_i, in serial and shifted from a PE to the following PE.

Table 3.5: Data flow to processing elements over input luminance ports

INPUT PORTS					PROCESSING ELEMENTS													
C_i	S_0	S_1	S_2		PE0	PE1	PE2	PE3	PE4	...	PE14	PE15	PE16	PE17	PE18	...	PE31	PE32
C_0	$S_{0,0}$	X	X		$C_0 - S_{0,0}$	IDLE	IDLE	IDLE	IDLE	...	IDLE	IDLE	IDLE	IDLE	IDLE	...	IDLE	IDLE
C_1	$S_{0,1}$	X	X		$C_1 - S_{0,1}$	$C_0 - S_{0,1}$	IDLE	IDLE	IDLE	...	IDLE	IDLE	IDLE	IDLE	IDLE	...	IDLE	IDLE
C_2	$S_{0,2}$	X	X		$C_2 - S_{0,2}$	$C_1 - S_{0,2}$	$C_0 - S_{0,2}$	IDLE	IDLE	...	IDLE	IDLE	IDLE	IDLE	IDLE	...	IDLE	IDLE
C_3	$S_{0,3}$	X	X		$C_3 - S_{0,3}$	$C_2 - S_{0,3}$	$C_1 - S_{0,3}$	$C_0 - S_{0,3}$	IDLE	...	IDLE	IDLE	IDLE	IDLE	IDLE	...	IDLE	IDLE
C_4	$S_{0,4}$	X	X		$C_4 - S_{0,4}$	$C_3 - S_{0,4}$	$C_2 - S_{0,4}$	$C_1 - S_{0,4}$	$C_0 - S_{0,4}$	...	IDLE	IDLE	IDLE	IDLE	IDLE	...	IDLE	IDLE
C_5	$S_{0,5}$	X	X		...	...	...	...	...	...	IDLE	IDLE	IDLE	IDLE	IDLE	...	IDLE	IDLE
C_6	$S_{0,6}$	X	X							...	IDLE	IDLE	IDLE	IDLE	IDLE	...	IDLE	IDLE
C_7	$S_{0,7}$	X	X							...	IDLE	IDLE	IDLE	IDLE	IDLE	...	IDLE	IDLE
C_8	$S_{0,8}$	X	X							...	IDLE	IDLE	IDLE	IDLE	IDLE	...	IDLE	IDLE
C_9	$S_{0,9}$	X	X							...	IDLE	IDLE	IDLE	IDLE	IDLE	...	IDLE	IDLE
C_{10}	$S_{0,10}$	X	X							...	IDLE	IDLE	IDLE	IDLE	IDLE	...	IDLE	IDLE
C_{11}	$S_{0,11}$	X	X							...	IDLE	IDLE	IDLE	IDLE	IDLE	...	IDLE	IDLE
C_{12}	$S_{0,12}$	X	X							...	IDLE	IDLE	IDLE	IDLE	IDLE	...	IDLE	IDLE
C_{13}	$S_{0,13}$	X	X							...	IDLE	IDLE	IDLE	IDLE	IDLE	...	IDLE	IDLE
C_{14}	$S_{0,14}$	X	X		...	...	...			...	$C_0 - S_{0,14}$	IDLE	IDLE	IDLE	IDLE	...	IDLE	IDLE
C_{15}	$S_{0,15}$	X	X		$C_{15} - S_{0,15}$	$C_{14} - S_{0,15}$	$C_{13} - S_{0,15}$	$C_{12} - S_{0,15}$	$C_{11} - S_{0,15}$	...	$C_1 - S_{0,15}$	$C_0 - S_{0,15}$	IDLE	IDLE	IDLE	...	IDLE	IDLE
C_{16}	$S_{1,0}$	$S_{0,16}$	X		$C_{16} - S_{1,0}$	$C_{15} - S_{0,16}$	$C_{14} - S_{0,16}$	$C_{13} - S_{0,16}$	$C_{12} - S_{0,16}$	...	$C_2 - S_{0,16}$	$C_1 - S_{0,16}$	$C_0 - S_{0,16}$	IDLE	IDLE	...	IDLE	IDLE
C_{17}	$S_{1,1}$	$S_{0,17}$	X		$C_{17} - S_{1,1}$	$C_{16} - S_{1,1}$	$C_{15} - S_{0,17}$	$C_{14} - S_{0,17}$	$C_{13} - S_{0,17}$	...	$C_3 - S_{0,17}$	$C_2 - S_{0,17}$	$C_1 - S_{0,17}$	$C_0 - S_{0,17}$	IDLE	...	IDLE	IDLE
C_{18}	$S_{1,2}$	$S_{0,18}$	X		$C_{18} - S_{1,2}$	$C_{17} - S_{1,2}$	$C_{16} - S_{1,2}$	$C_{15} - S_{0,18}$	$C_{14} - S_{0,18}$	...						...	IDLE	IDLE
C_{19}	$S_{1,3}$	$S_{0,19}$	X					$C_{16} - S_{1,3}$	$C_{15} - S_{0,19}$	...						...	IDLE	IDLE
C_{20}	$S_{1,4}$	$S_{0,20}$	X						$C_{16} - S_{1,4}$	...						...	IDLE	IDLE
C_{21}	$S_{1,5}$	$S_{0,21}$	X							...						...	IDLE	IDLE
C_{22}	$S_{1,6}$	$S_{0,22}$	X							...						...	IDLE	IDLE
C_{23}	$S_{1,7}$	$S_{0,23}$	X							...						...	IDLE	IDLE
C_{24}	$S_{1,8}$	$S_{0,24}$	X							...						...	IDLE	IDLE
C_{25}	$S_{1,9}$	$S_{0,25}$	X							...						...	IDLE	IDLE
C_{26}	$S_{1,10}$	$S_{0,26}$	X							...						...	IDLE	IDLE
C_{27}	$S_{1,11}$	$S_{0,27}$	X							...						...	IDLE	IDLE
C_{28}	$S_{1,12}$	$S_{0,28}$	X							...						...	IDLE	IDLE
C_{29}	$S_{1,13}$	$S_{0,29}$	X							...						...	IDLE	IDLE
C_{30}	$S_{1,14}$	$S_{0,30}$	X							...						...	IDLE	IDLE
C_{31}	$S_{1,15}$	$S_{0,31}$	X		$C_{31} - S_{1,15}$	$C_{30} - S_{1,15}$	$C_{29} - S_{1,15}$	$C_{28} - S_{1,15}$	$C_{27} - S_{1,15}$	...						...	$C_0 - S_{0,31}$	IDLE
C_{32}	$S_{2,0}$	$S_{1,16}$	$S_{0,32}$		$C_{32} - S_{2,0}$	$C_{31} - S_{1,16}$	$C_{30} - S_{1,16}$	$C_{29} - S_{1,16}$	$C_{28} - S_{1,16}$	...						...		$C_0 - S_{0,32}$
C_{33}	$S_{2,1}$	$S_{1,17}$	$S_{0,33}$			$C_{32} - S_{2,1}$	$C_{31} - S_{1,17}$	$C_{30} - S_{1,17}$	$C_{29} - S_{1,17}$	...						...		
C_{34}	$S_{2,2}$	$S_{1,18}$	$S_{0,34}$				$C_{32} - S_{2,2}$	$C_{31} - S_{1,18}$	$C_{30} - S_{1,18}$	...						...		
C_{35}	$S_{2,3}$	$S_{1,19}$	$S_{0,35}$					$C_{32} - S_{2,3}$	$C_{31} - S_{1,19}$	...						...		
C_{36}	$S_{2,4}$	$S_{1,20}$	$S_{0,36}$						$C_{32} - S_{2,4}$	...						...		
C_{37}	$S_{2,5}$	$S_{1,21}$	$S_{0,37}$							...						...		
C_{38}	$S_{2,6}$	$S_{1,22}$	$S_{0,38}$							...						...		
C_{39}	$S_{2,7}$	$S_{1,23}$	$S_{0,39}$							...						...		
C_{40}	$S_{2,8}$	$S_{1,24}$	$S_{0,40}$							...						...		
C_{41}	$S_{2,9}$	$S_{1,25}$	$S_{0,41}$							...						...		
C_{42}	$S_{2,10}$	$S_{1,26}$	$S_{0,42}$							...						...		
C_{43}	$S_{2,11}$	$S_{1,27}$	$S_{0,43}$							...						...		
C_{44}	$S_{2,12}$	$S_{1,28}$	$S_{0,44}$							...						...		
C_{45}	$S_{2,13}$	$S_{1,29}$	$S_{0,45}$							...						...		
C_{46}	$S_{2,14}$	$S_{1,30}$	$S_{0,46}$							...						...		
C_{47}	$S_{2,15}$	$S_{1,31}$	$S_{0,47}$							...						...		
.	.	.	.		.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.		.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.		.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.		.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.		.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.		.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.		.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.		.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.		.	.	.	.	.	.	.	.	.	.	.	.	.	.

As it is given in Table 3.5, all processing elements do not use every input port, and ports corresponded to PEs are changing cycle by cycle. This requires an adaptive multiplexing structure for switching between input ports. This structure is built by simple 2×1 multiplexers in front of processing elements and the select inputs of these multiplexers are fed by S_Select port from the GME controller.

Table 3.6: PEs vs. corresponding luminance ports

PE index	Corresponding Luminance Ports
0	S_0
1-15	S_0 and S_1
16	S_1
17-31	S_1 and S_2
32	S_2

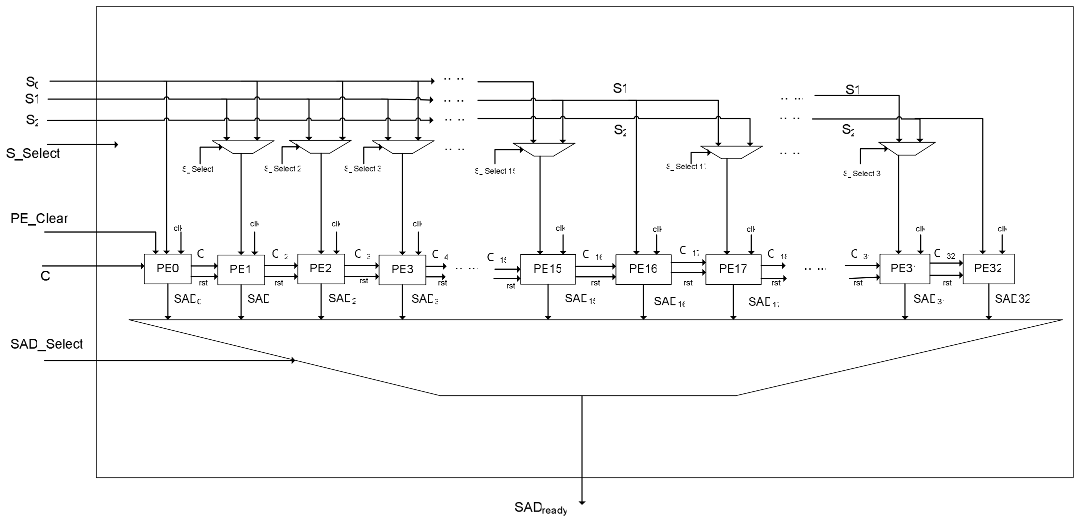


Figure 3.8: GME PE array structure

Since the MB size is fixed to 16×16, an SAD calculation time equals to 256 cycles for a single search location. Total execution time of PE array for whole SW can be calculated by the formula:

$$T_{PE\_array} = 256 \times n + t = 256 \times 21 + 32 = 5408 \qquad (3.3)$$

where n is the number of vertical search locations in a SW column, and t is the delay of pipelining due to the number of PEs.

3.4.3. GME Minimum SAD Comparator

A motion vector in a SW is decided by the location of minimum distortion (SAD). PE array calculates all the SAD values and passes to the minimum SAD comparator component of the GME structure. This component finds the minimum distortion with comparison between incoming SAD value and the SAD value stored in currentMin register. If the comparison results as true, the motion vector is updated by the values of counters, triggered by enable port.

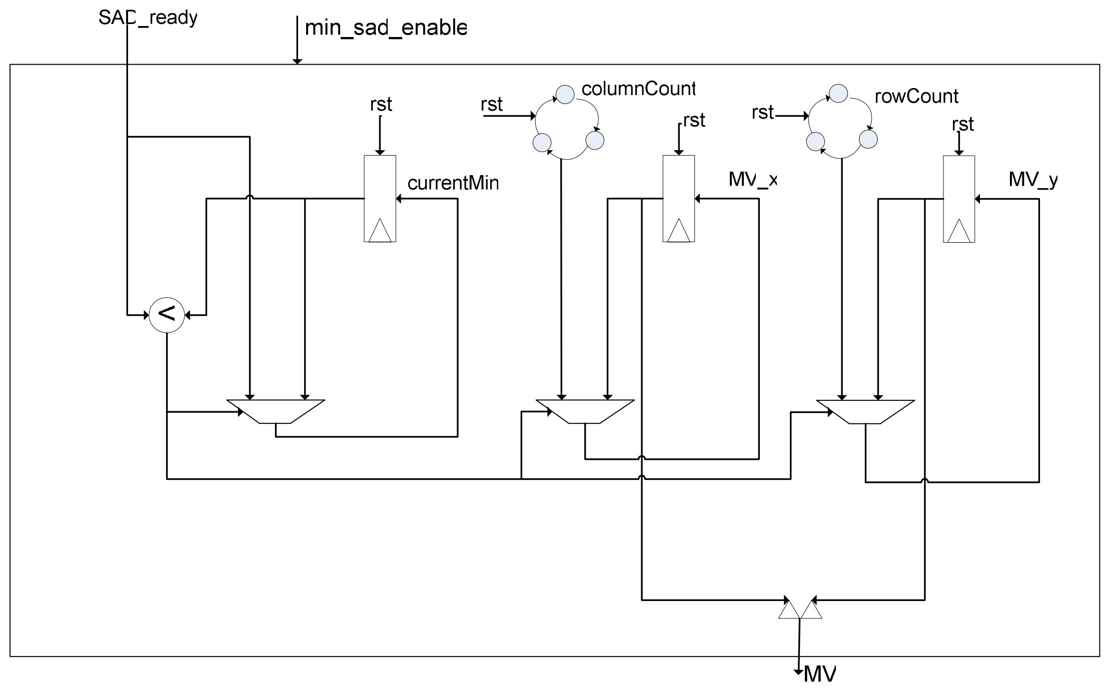


Figure 3.9: Structure of GME minimum SAD comparator

3.5. MEDIAN VECTOR GENERATION

Nine different reference points are set to find the global motion vector defining the camera movements. Each reference point generates its own motion vector. MV_{global} is determined by the median vector of these nine different motion vectors. (Eq. 2.22)

There exists several algorithms to find the median vector; however, due to the clock frequency of input video and the size of chip, it is not feasible to implement a hardware block to find the median vector in a single cycle. In this study, median vector generator is implemented by a serial bubble sorter, which takes:

$$(n-1)\times(n-2) = (9-1)\times(8-1) = 56 \qquad (3.4)$$

cycles in $O(n^2)$ complexity where n is the number of motion vectors to be sorted. Middle element of both x and y components array generates the median vector, said to be MV_{global}.

3.6. LOCAL MOTION ESTIMATOR

By the local motion estimator, it is targeted to find the motion vectors for moving objects. The hardware architecture is based on 3-D RS block-matching algorithm which is explained in Sec. 2.4.

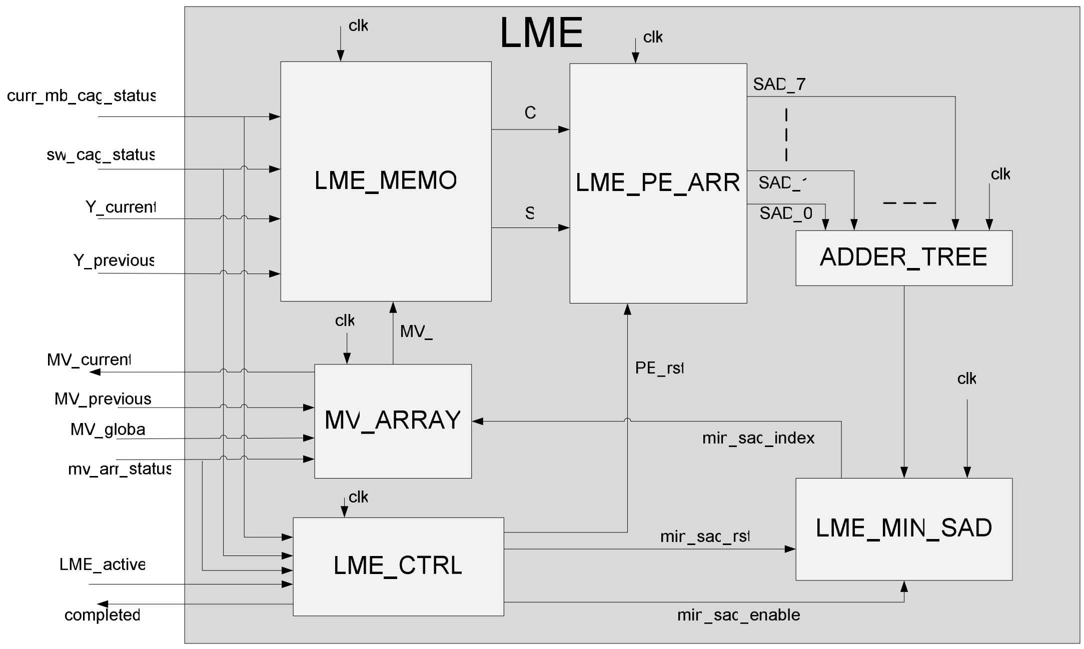


Figure 3.10: Local motion estimator block diagram

3.6.1. Motion Vector Array

3-D RS algorithm is based on the motion vectors calculated during the motion estimation between previous 2 frames.

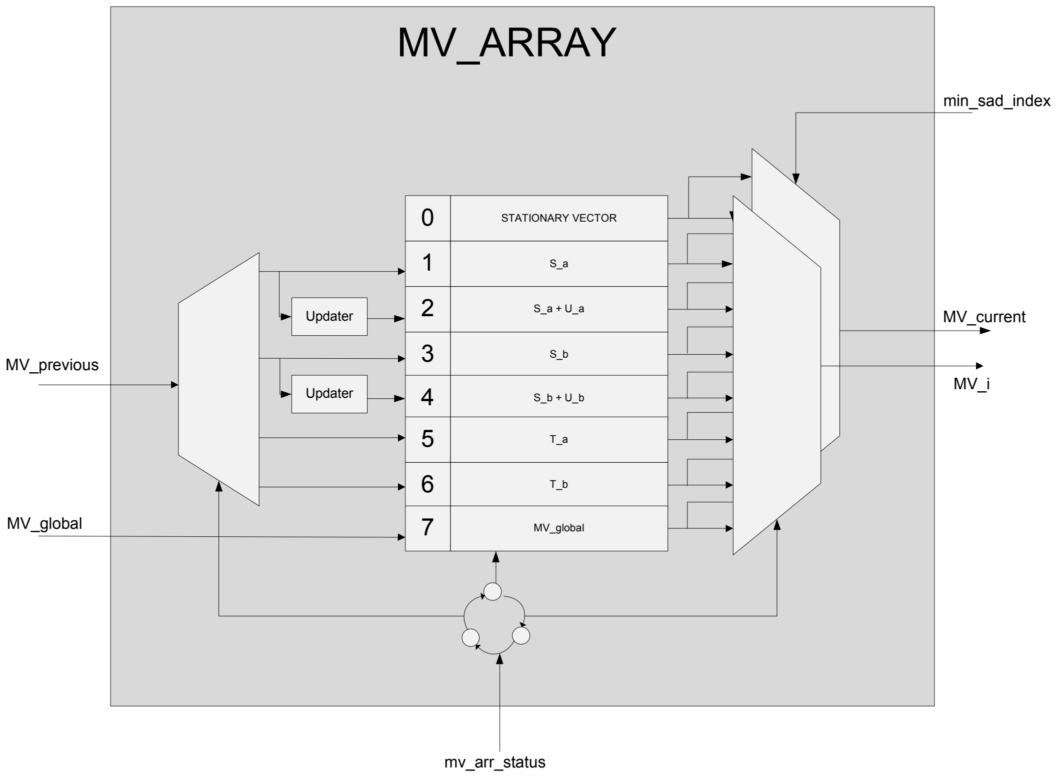


Figure 3.11: Structure of motion vector array

Local motion vectors are computed by 8 different motion vectors, four of those are directly related to the motion vectors from previous estimation (S_a, S_b, T_a, and T_b). These four vectors are fetched from DDR and stored into the register block of the MV array structure. Two vectors are generated by the updaters. Remaining two vectors are the stationary vector, showing the same search location of MB on SW, and MV_{global}, calculated by global motion estimator and the median vector generator.

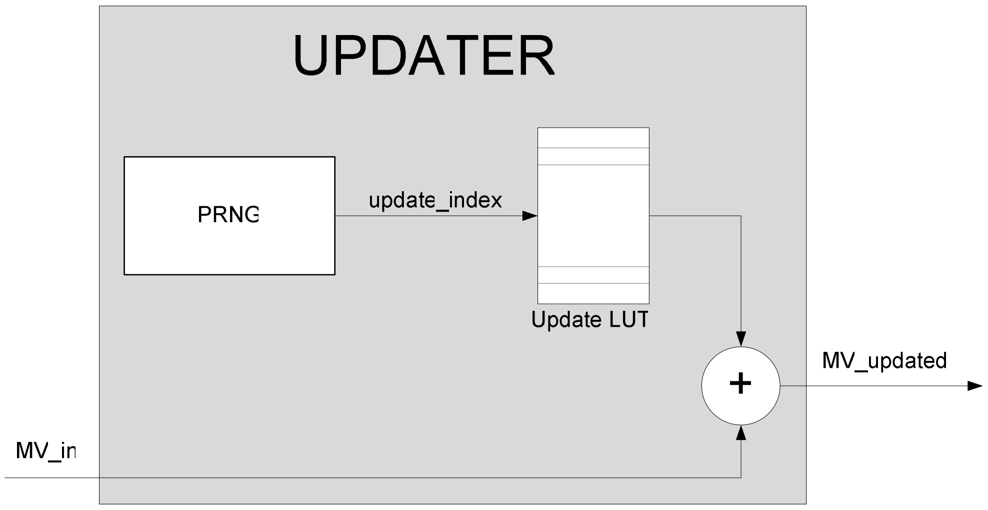


Figure 3.12: Structure of updater

Updater blocks inside the MV array generate two new motion vectors to be searched by adding update vectors from an update set:

$$U=\left\{\begin{pmatrix}1\\0\end{pmatrix},\begin{pmatrix}0\\1\end{pmatrix},\begin{pmatrix}-1\\0\end{pmatrix},\begin{pmatrix}0\\-1\end{pmatrix},\begin{pmatrix}2\\0\end{pmatrix},\begin{pmatrix}0\\2\end{pmatrix},\begin{pmatrix}-2\\0\end{pmatrix},\begin{pmatrix}0\\-2\end{pmatrix},\right.$$
$$\left.\begin{pmatrix}3\\0\end{pmatrix},\begin{pmatrix}0\\3\end{pmatrix},\begin{pmatrix}-3\\0\end{pmatrix},\begin{pmatrix}0\\-3\end{pmatrix},\begin{pmatrix}4\\0\end{pmatrix},\begin{pmatrix}0\\4\end{pmatrix},\begin{pmatrix}-4\\0\end{pmatrix},\begin{pmatrix}0\\-4\end{pmatrix}\right\} \tag{3.5}$$

over spatial vectors, S_a and S_b. The update vectors are listed in a LUT which is fed by a randomly generated update index. The randomization of this index is provided by a pseudo-random number generator, which is designed on the basics of Galois LFSR in this thesis study.

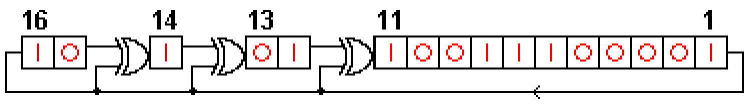


Figure 3.13: Galois LFSR

3.6.2. LME Memory Structure

Like FS algorithm in GME, 3-D RS is performed between MBs from current frame and the SWs from previous frame; however, the sizes of these blocks differ from GME. MB is set to be 8×8 in size that reduces the size of SW to 40×28 due to the search range of ±16 in horizontal and ±10 in vertical. MB and SW are fetched to internal memories as same as the GME to reduce the number of access to external memory. The configuration of words to write into block-RAMs is also identical to configuration in GME. The only difference related to the block-RAMs is in numbers of addresses of SW block-RAM and MB block-RAM that are $\frac{(40\times 28)}{4}=280$ and $\frac{(8\times 8)}{4}=16$, respectively, due to the block sizes.

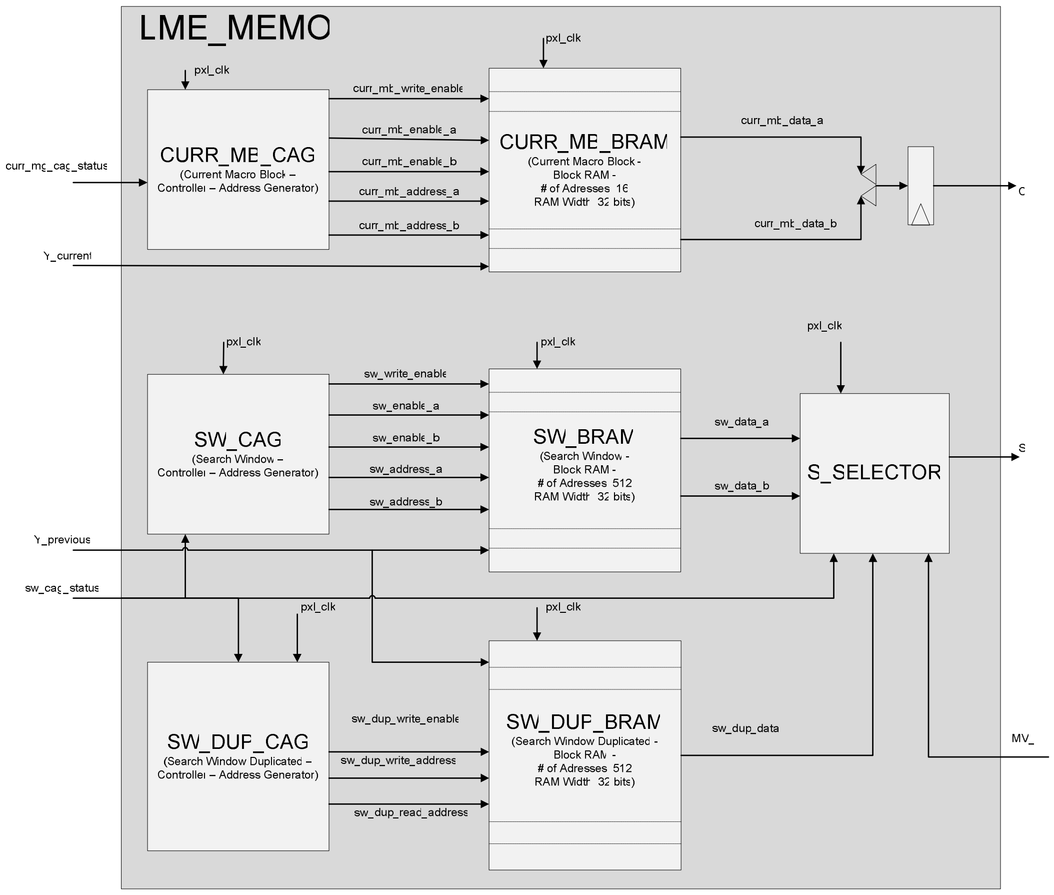


Figure 3.14: Memory structure for local motion estimator

Another difference between the GME and LME memory structures is the width of the output ports. In GME, there exist four output ports, C_i, S_0, S_1, and S_2, of 8 bits in width to run the FS data flow. This enables the calculation of SAD for 33 different search locations. In LME, the strategy of minimum distortion calculation is completely different, where eight blocks in SW, pointed with eight independent motion vectors, are correlated with MB of current frame. This means that the pixels search blocks are not listed consecutively in SW block-RAMs. The situation of the block-RAM configuration prevents the calculation of eight different distortions in parallel with a small number of block-RAMs in structure. Because the number of block-RAMs in FPGAs is very limited, it is necessary to design a structure reducing the block-RAM demand for data providing to processing elements.

To reduce the number of block-RAMs, the parallelism strategy is converted from Parallel-Serial (minimum distortion calculation of different search locations in parallel by feeding PEs with corresponding search pixels of different search locations in serial) to Serial-Parallel (minimum distortion calculation of different search location in serial by feeding PEs with corresponding search pixels of same search location in parallel). The structure can be implemented by two output ports, C and S, each of which is 64 bits in width.

Due to the value of motion vector, that decides the macroblock from SW to be correlated with current MB, eight luminance values of previous MB might be distributed to 2 or 3 words in block-RAM related to search window; on the other hand, the luminance values of current MB are placed in every two words of its own block-RAM. A block-RAM is able to output two values with its one read and one read/write port. This enables that the current MB values can be provided by a single block-RAM; otherwise, for search window, a second block-RAM, with an identical content with original SW block-RAM, is required to provide the data because of the possibility of distribution of necessary values in 3 words due to the MVs.

After fetching these three words from block-RAMs, a multiplexing structure has to be installed behind the block-RAMs to select the correct eight luminance values out of twelve values, fetched from two block-RAMs, due to the MV.

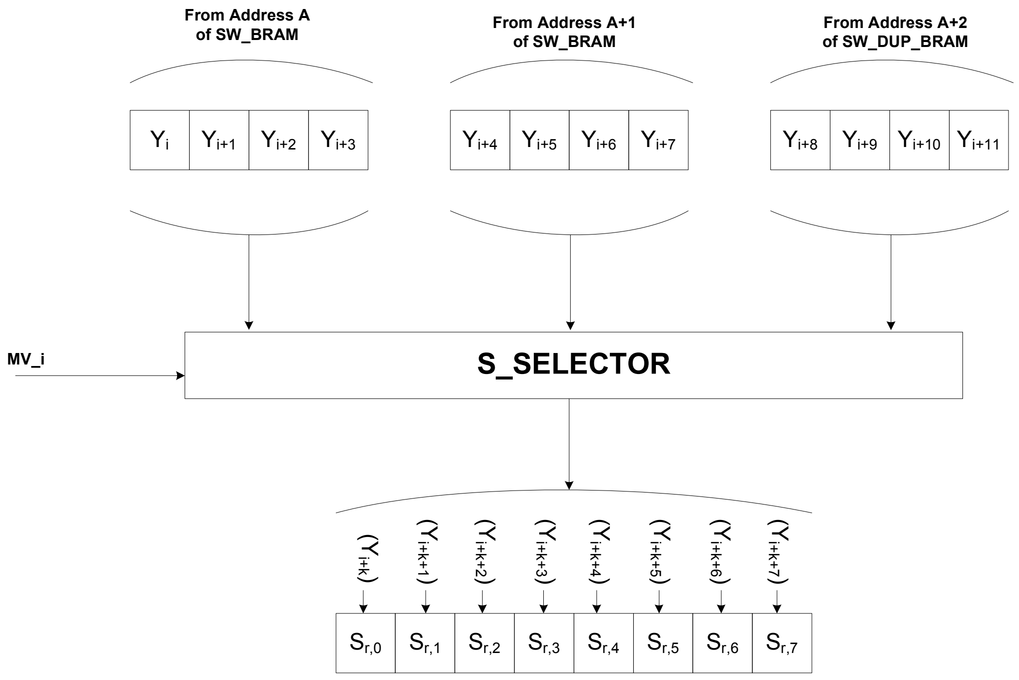

From Address A of SW_BRAM
From Address A+1 of SW_BRAM
From Address A+2 of SW_DUP_BRAM
Y_i Y_{i+1} Y_{i+2} Y_{i+3}
Y_{i+4} Y_{i+5} Y_{i+6} Y_{i+7}
Y_{i+8} Y_{i+9} Y_{i+10} Y_{i+11}
MV_i
S_SELECTOR
(Y_{i+k}) (Y_{i+k+1}) (Y_{i+k+2}) (Y_{i+k+3}) (Y_{i+k+4}) (Y_{i+k+5}) (Y_{i+k+6}) (Y_{i+k+7})
$S_{r,0}$ $S_{r,1}$ $S_{r,2}$ $S_{r,3}$ $S_{r,4}$ $S_{r,5}$ $S_{r,6}$ $S_{r,7}$

Figure 3.15: Selection of correct luminance value

S_SELECTOR component multiplexes these luminance values for correct data providing. It analyzes the motion vector and generates a *k* value pointing the corresponding head luminance and forms the 64-bit S value composed of 8 luminance values.

Table 3.7: Value of pointer k due to motion vector input

(X coordinate of MV_i) mod 4	Value of k
0	0
1	1
2	2
3	3

3.6.3. LME Processing Element Array

In every pixel clock during the local motion estimation, memory structure feeds eight luminance values for both current MB and search MB. Since the block size of each macroblock is 8×8 and 8 luminance values are fed every single cycle by memory, there could be installed 8 PEs to accumulate the SADs of each column of macroblocks in eight cycles.

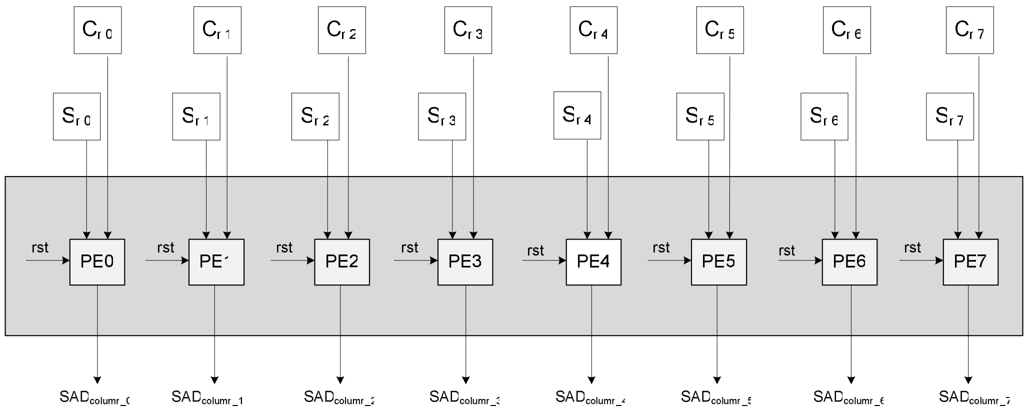


Figure 3.16: LME PE array structure

PEs, used in LME, are similar to the PEs in GME; however; they are not exactly identical. Because global motion estimation is performed by the FS algorithm where the SAD calculation strategy is Parallel-Serial, explained in Sec. 3.6.2, luminance values of current MB are transferred from one PE to the following one to start the correlation between the block of search location and the current MB. The structures of PEs are also different in reset input. While every PE in GME is reset right after the previous PE in sequence of the array, the reset signal of a PE is also shifted to the following PE.

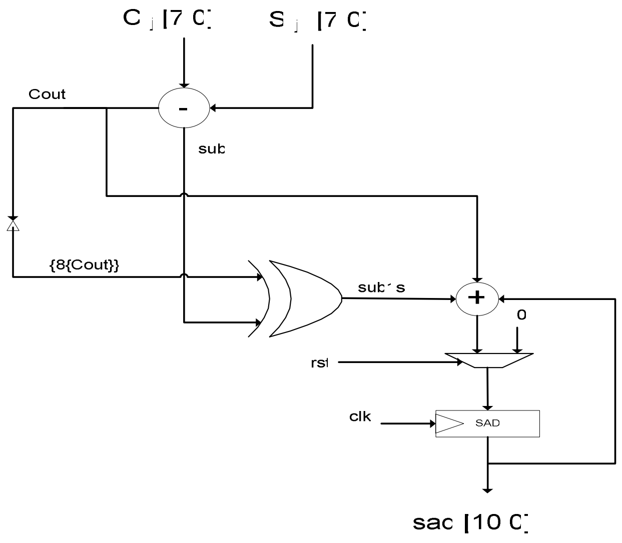


Figure 3.17: Structure of LME processing element

The last difference between the PEs of GME and LME is the width of the output SAD port. The width of this port varies the number of accumulation for resultant

SAD value for each PE. In GME, a PE is responsible for a whole MB correlation where the block size is 16×16; then it takes 256 cycles to finish the process. The worst distortion value would be $255 \times 256 = 65280$, where 255 is the value of the worst distortion (between white and black pixel) and 256 is the number of pixels in whole MB, could be represented in 16 bits. On the other hand, in LME, a PE is responsible only one column of an MB, where the block size is set to 8×8; it would take only 8 cycles to finish the SAD process. In the worst case of LME, the maximum distortion would be $255 \times 8 = 2040$, where 255 has the same meaning with GME and 8 is the number of the pixels in an MB column, could be represented in narrower bit width, 11.

3.6.4. Adder Tree

The PE array unit comprises 8 PEs, with each PE computing the SAD for one column of the block. After every 8 pixel clock cycles, the SADs of all the 8 columns are summed up using an 8-input high-speed parallel adder to produce the SAD for the entire block.

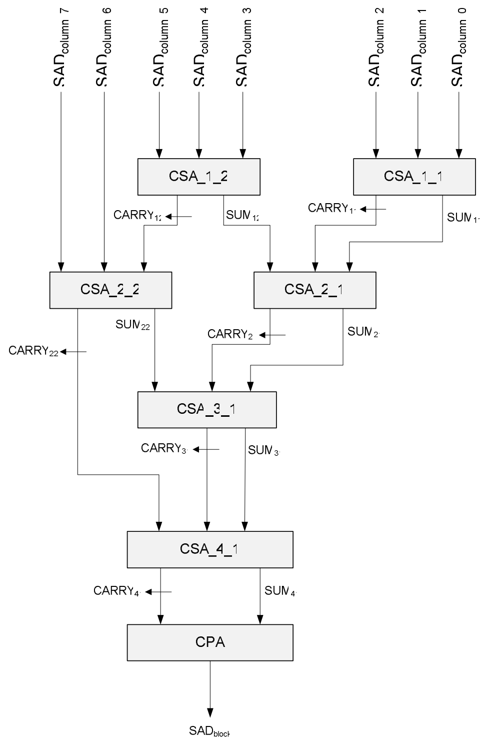


Figure 3.18: Computation of SAD for an LME block

The parallel adder is made up of 6 CSAs and one CPA in the final stage. Each CSA takes three 14-bit inputs and produces a 14-bit sum and a 14-bit carry at the output. Since the PE outputs are only of 11-bit length, the maximum length of the sum of 8 such inputs can be 14 bits, and hence an adder with a final output of length 14 bits is sufficient. In this 8-input parallel adder, 2 CSAs are present in first stage to accept 6 inputs and produce 2 sum outputs and 2 carry outputs. The remaining two inputs are simply carried to the next stage. The 2 carry outputs are left-shifted by 1 bit. In a similar manner, 2 CSAs are used in the second stage, and so on. The final CSA gives one sum output and one carry output. In the final stage, a CPA adds up the sum and the left-shifted carry to produce the final sum of 14-bit length. This becomes the SAD for the entire block.

3.6.5. LME Minimum SAD Comparator

Like GME minimum SAD comparator, the comparator in LME finds the minimum distortion which decides the motion vector output; however, the counter in LME structure does not directly count the motion vector value, but the index of the motion vector stored in the register array of MV array unit of local motion estimator.

When the enable port is high, the component checks the incoming SAD value as if it is smaller than the minimum SAD value inside the currentMin register. If the comparison gives true as the result, index value pointing the MV array is updated by the value of index counter.

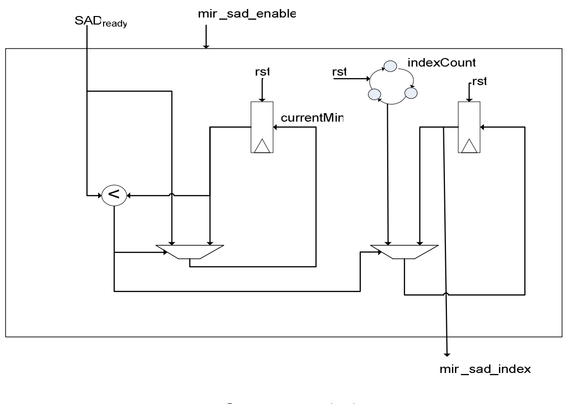


Figure 3.19: Structure of LME minimum SAD comparator

4. VERIFICATION STRATEGY AND TOOLS

Hardware verification is the process of verifying the functional characteristics of models at any level of abstraction. Simulators, such as Modelsim, Veriwell, Icarus, etc., are used to simulate the hardware models. To verify if the RTL code meets the functional requirements of the specification, it has to be observed if all RTL blocks are functionally correct. To achieve this, a testbench is needed to be written, which generates clock, reset and the required stimulus.

The waveform output from the simulator is used to see if the device under test is functionally correct. As the design becomes more complicated, self checking testbench is preferred, where the testbench applies the test vector, then compares the output of DUT with the expected value.

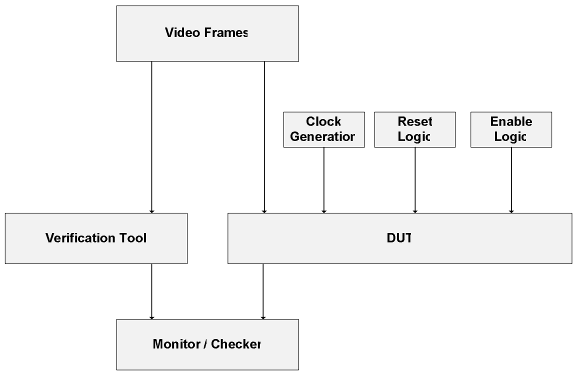


Figure 4.1: Testbench environment

In this thesis study, Modelsim, as the simulator software, is only used for applying test vectors and generating the DUT outputs; however, the expected values are not generated by the testbench. Remaining steps of are provided by the verification tool, developed in C# language. This tool is designed at two levels: DLL development, and GUI development.

4.1. MOTION DLL CLASSES

A reusable dynamic linked library is developed for the verification strategy of the study, composed of many classes with the inherited structure between each other.

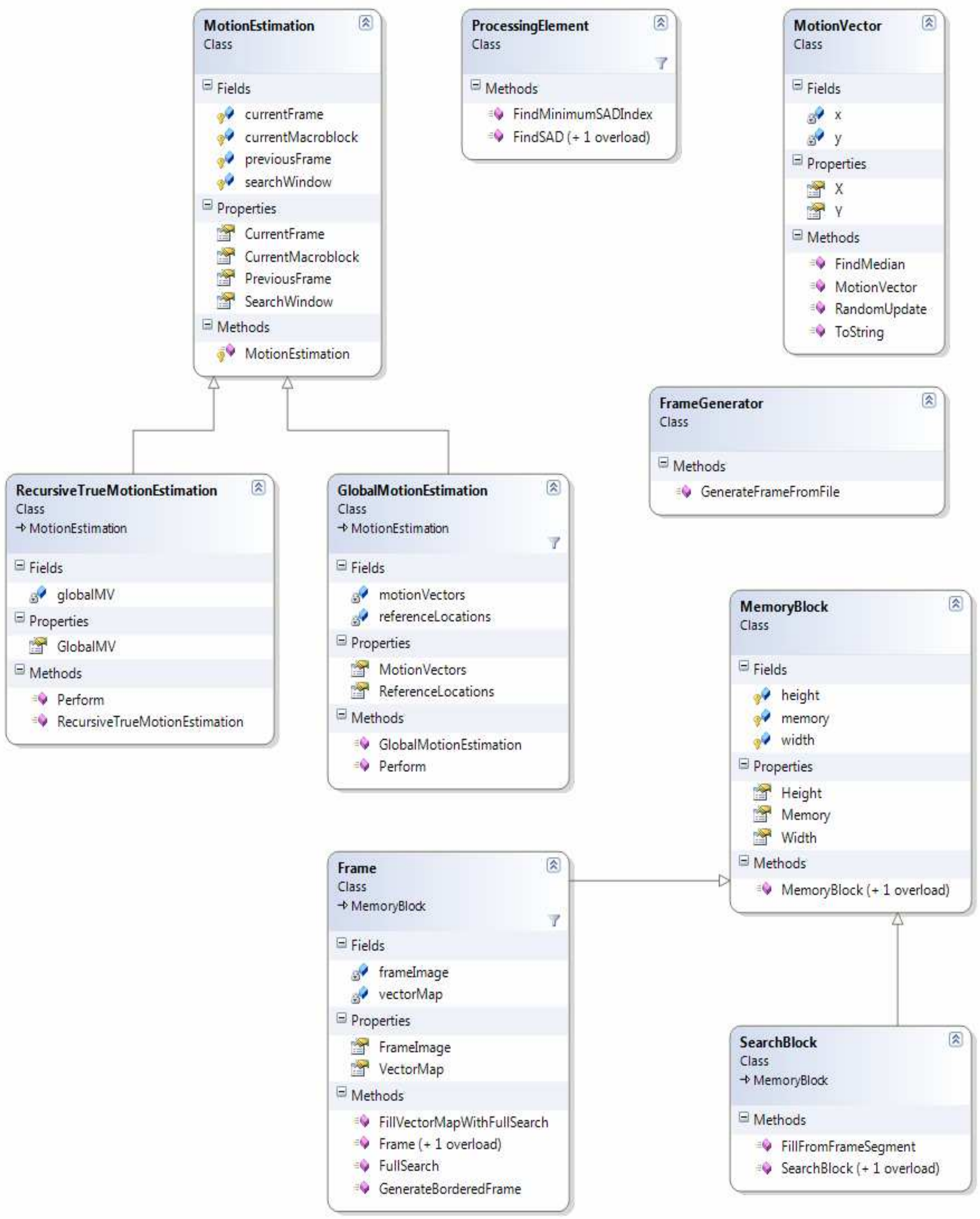


Figure 4.2: Class Diagram of Motion DLL

4.1.1. MemoryBlock

MemoryBlock represents a two-dimensional integer array with the given height and width, and their values. Sub-classes, Frame and SearchBlock, are inherited from this class, since they are two-dimensional array like structures.

4.1.2. SearchBlock

Search block is a kind of memory block, that is windowed by a given size and is filled by coordinate values of x and y on a specified frame. For the motion estimation algorithms all blocks, i.e. current MB and SW, are defined as the instantiations of this class. The size of blocks are changing due to the algorithm used for motion estimation.

4.1.3. Frame

Frame is another memory block, which can be also defined as the pixel matrix of an image. The pixel values of the frame are stored by only its gray values, calculated by the conversion formula between RGB and YUV color spaces, while only the gray-scale image is needed for motion estimation.

Because the purpose of the software development is to test the 3-D RS algorithm, and since 3-D RS needs the motion vectors of previous estimation between two previous images, a function called FillVectorMapWithFullSearch is implemented to generate the initial motion vectors for motion estimation. FS algorithm is performed by a static function inside the class and motion vector map of the first frame of video sequence is filled by the result of this static function.

One of the problems with the motion estimation algorithms is to find the motion vector of macroblocks at border, where some pixels of corresponding SW do not exists. The software handles this problem by another static function of the class, GenerateBorderedFrame, which adds additional black pixels for the missing parts of SW, needed.

4.1.4. FrameGenerator

FrameGenerator is a single-static-functioned class that generates a frame from a given bitmap file by converting the RGB data into YUV and saves only the Y values inside the frame memory array.

4.1.5. MotionVector

Motion vector of macroblocks are modeled by this class. Motion vector is composed of two coordinates that refer displacement in both vertical and horizontal.

Motion vectors due to spatial domain in 3-D RS algorithm are updated by random numbers to generate two additional candidate vectors for algorithm. This process is provided by the function RandomUpdate.

Global motion estimation algorithm generates 9 motion vectors due to 9 different reference locations. MV_{global} is calculated by the median operation over these 9 motion vectors. The median vector, labeled as MV_{global}, is computed by a static function of class over an input parameter with type of motion vector array.

4.1.6. MotionEstimation

The class is designed as the base class for motion estimation algorithms. Since every motion estimation algorithm requires a previous, a current frame, a SW and a current MB with fixed sizes, these attributes are packed in the MotionEstimation class.

4.1.7. GlobalMotionEstimation

The class is the implementation of global motion estimation with the strategy explained in Sec. 2.5. After an object instantiation of class, reference locations are calculated by Eq. 2.2.

Perform function runs FS algorithm for each reference location and stores the motion vectors in an array, called MotionVectors. Finally it generates the global motion vector by calling the static function, FindMedian, and of MotionVector class passing the array the function as input parameter.

4.1.8. RecursiveTrueMotionEstimation

Implementation of 3-D RS algorithm is coded in this class. The constructor of the class takes MV_{global} as an additional input parameter, computed by the execution of global motion estimation algorithm.

Perform function of the class runs the algorithm for each macroblock in current frame and fills the vector map of current frame by the vectors found by the algorithm.

4.2. GUI FOR TEST SOFTWARE

GUI is the second level for the test software development. The application runs with the given scenario:

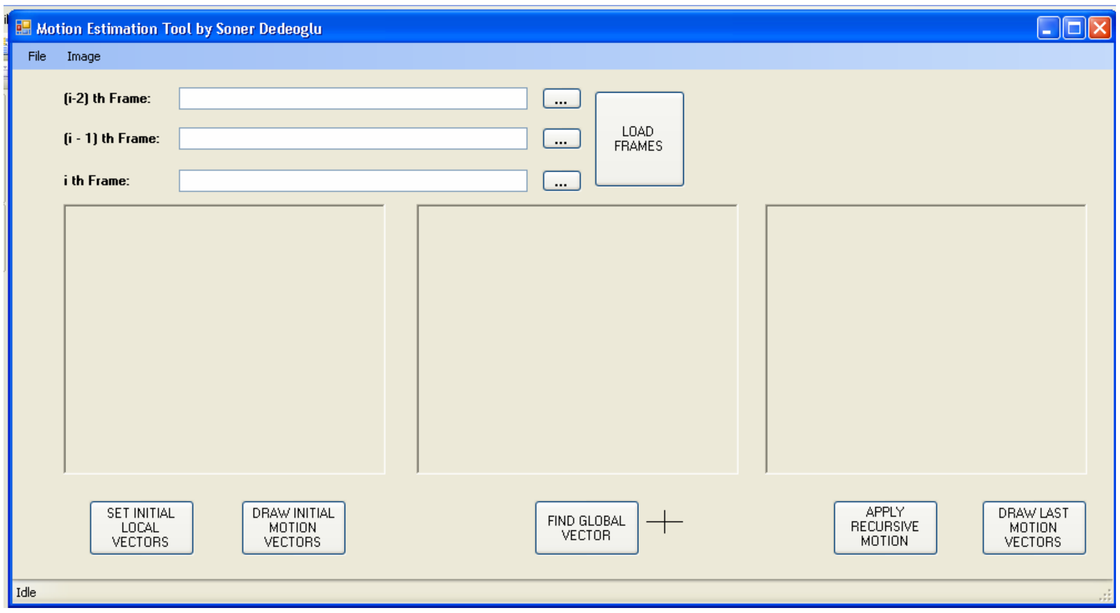
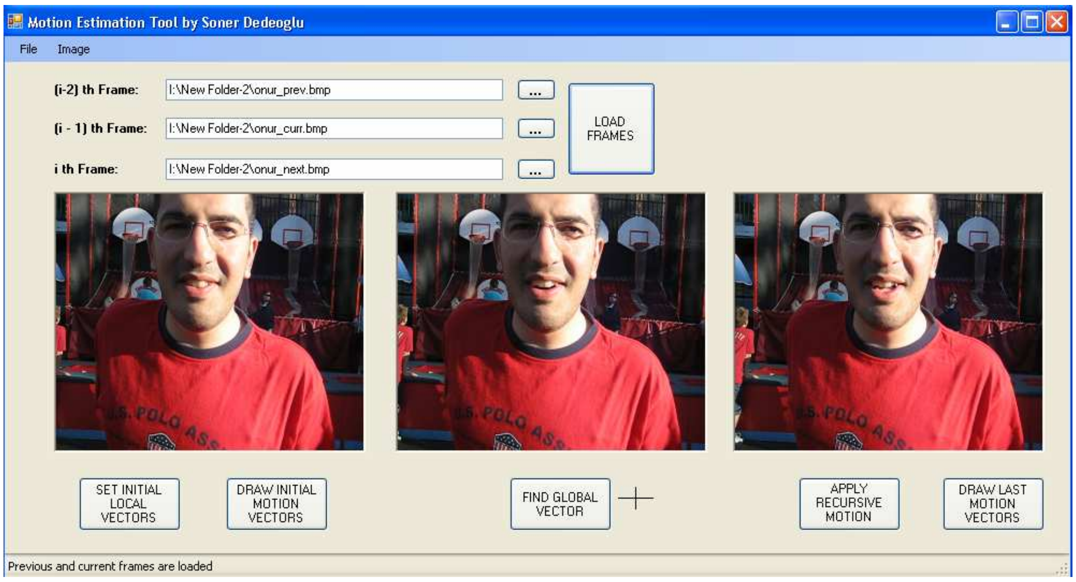


Figure 4.3: Main user form of motion estimation test software

Three consecutive video frames are opened, where the first two frames, labeled as $(i-2)^{th}$ and $(i-1)^{th}$ are used to find the initial local vectors. Initial vector calculation step is performed by FS algorithm. The initial vectors would be used in next step, 3-

D RS, as previous motion vectors. Global motion estimation and 3-D RS algorithms are applied between last two frames, loaded as $(i-1)^{th}$ and i^{th}.


Motion Estimation Tool by Soner Dedeoglu
File
Image
...
LOAD FRAMES
SET INITIAL LOCAL VECTORS
DRAW INITIAL MOTION VECTORS
FIND GLOBAL VECTOR
APPLY RECURSIVE MOTION
DRAW LAST MOTION VECTORS

(i-2) th Frame: I:\New Folder-2\onur_prev.bmp

(i - 1) th Frame: I:\New Folder-2\onur_curr.bmp

i th Frame: I:\New Folder-2\onur_next.bmp

Previous and current frames are loaded

Figure 4.4: "Baskirt - Amusement Park" sequence is loaded

After computation of local motion vectors, the motion vectors are drawn on a local vectors form. Each vector line begins from the left upper corner of each macroblock and ends at the location pointed by the MV.

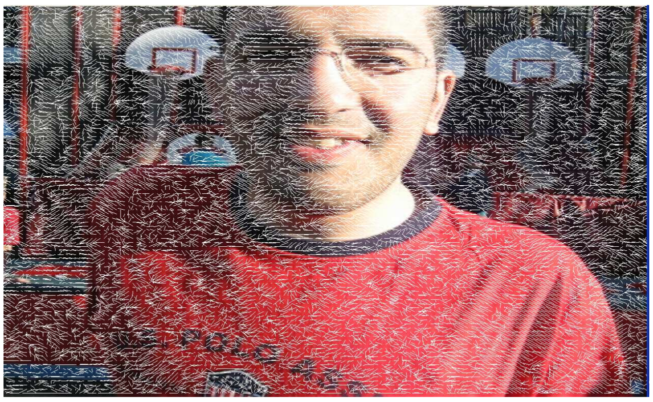


Figure 4.5: Initial motion vectors calculated by FS algorithm

MV_{global} is calculated by the algorithm explained in Sec. 2.5 right after the calculation of initial vectors. The global vector is drawn on a coordinate plane by a red line.

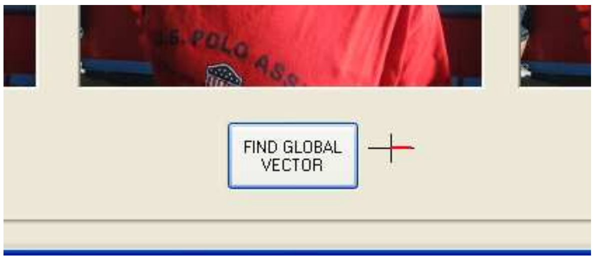


Figure 4.6: Global motion vector on coordinate plane

After the processes, finding initial vectors and global vector, are finished, software is ready to estimate the local vectors between $(i-1)^{th}$ and i^{th} frames by 3-D RS algorithm.

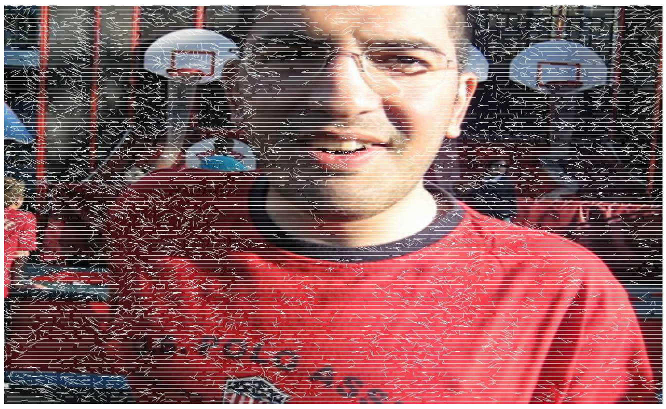


Figure 4.7: Local motion vectors calculated by 3-D RS algorithm

“Baskirt – Amusement Park” sequence is a test video with WXGA resolution; likewise, the software, as a result of the modularity in DLL, is also capable of motion estimation over videos with any resolutions.

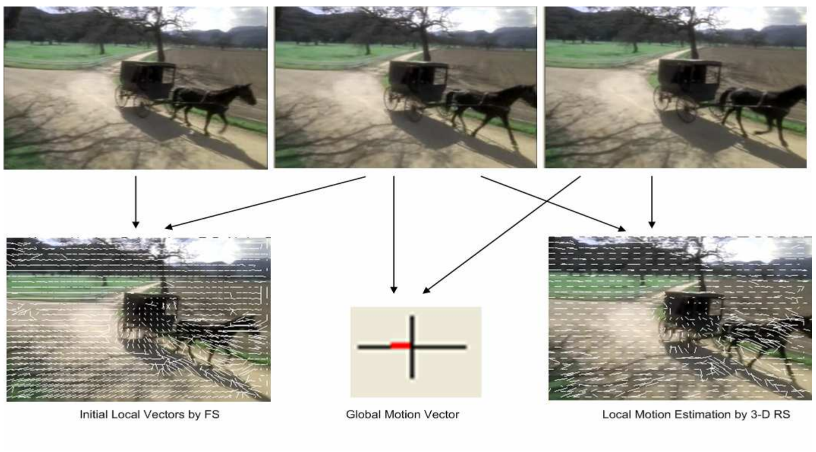

Initial Local Vectors by FS
Global Motion Vector
Local Motion Estimation by 3-D RS

Figure 4.8: Motion estimation over “Phaeton” sequence with 352×240 resolution

5. CONCLUSION AND FUTURE WORKS

During this thesis study, an architectural design is given for global motion estimation using FS algorithm and local motion estimation using 3-D RS algorithm. Global motion estimation level of the project is fully implemented by Verilog and verified by the object oriented software developed.

The coded hardware design is synthesized for Xilinx Spartan 3E FPGA with the version xc3s1600e-4fg484 by Xilinx ISE 8.1. Design summary for synthesized GME and median hardware blocks are given by the table below:

Table 5.1: Design summary for GME and median hardware blocks

Hardware Block	Number of Slices	Number of Slice Flip-Flops	Number of 4-input LUTs	Number of BRAMs	Clock Frequency (MHz)
GME_MEMO	112	111	208	3	101.513
GME_PE_ARR	718	816	1099	0	122.714
GME_CTRL	71	52	109	0	145.433
MEDIAN	611	155	1113	0	82.902

By looking at the synthesis results of the GME and median blocks, it is feasible to implement the design for the given Xilinx chip using about 11% of total slices in it. The current design is also feasible for a video sequence having 80 MHz in pixel clock frequency.

As a future work, it is decided to implement the given architectural design of LME in Verilog and get the design summary for the same FPGA. It will be examined as if the whole architecture is feasible for given version of Xilinx Spartan 3E FPGA.

In architectural view, it is possible to implement a simple MEDIAN block structure, what significantly decreases the maximum clock frequency of the chip and occupies significantly more space in terms of number of slices for a single median operation.

Another possible architectural study is available about reusing the pre-fetched luminance values of a SW for the next ME of next MB in LME. Because the significant amount of values in BRAMs is also needed for next MB of the same line, development of a structure to hold the needed data on BRAM and only fetch the new columns from the DDR will extremely decrease the number of DDR accesses and number of cycles for LME in total.

In this thesis study, an object-oriented verification tool for motion estimation techniques is developed. By the modularity in DLL development stage, as a future, this tool can be extended to realize other proposed motion estimation algorithms using classes.

REFERENCES

Books

Jack, K., 2007. *Video demystified: a handbook for the digital engineer*, 5th edn., USA: Newnes Press.

Periodical Publications

Jain, J. R. & Jain, A. K., 1981. Displacement measurement and its application in interframe image coding, *IEEE Transactions on Communications*, **29** (12), pp 1799 – 1808.

Po, L. M. & Ma, W. C., 1996. A novel four-step search algorithm for fast block motion estimation, *IEEE Transactions on Circuits and Systems for Video Technology*, **6**, pp 313 – 317.

Srinivasan, R. & Rao, K. R., 1985. Predictive coding based on efficient motion estimation, *IEEE Transactions on Commun.,* **33** (8), pp 888 – 896.

De Haan, G., Biezen P. W. A. C., Huijgen H. & Ojo O. A., 1993. True-motion estimation with 3-D recursive search block matching, *IEEE Transactions on Circuits and Systems for Video Technology,* **3** (5), pp 368 – 379.

De Haan, G. & Biezen P. W. A. C., 1998. An efficient true-motion estimator using candidate vectors from a parametric motion model, *IEEE Transactions on Circuits and Systems for Video Technology,* **8** (1), pp 85 – 91.

Fanucci, L., Saponara, S. & Bertini, L., 2001. A parametric VLSI architecture for video motion estimation, *The VLSI Journal of Integration*, **31**, pp 79 – 100.

Swamy, P. N., Chakrabarti, I. & Ghosh, D., 2002. Architecture for motion estimation using the one-dimensional hierarchical search block-matching algorithm, *IEE Proc.-Comput. Digit. Tech.,* **149** (5), pp 229 – 239.

Yang, S., Wolf, W. & Vijaykrishnan, N., 2005. Power and performance analysis of motion estimation based on hardware and software realizations, *IEEE Transactions on Computers***,** **54** (6), pp 714 – 726.

Other Publications

Toga, T., 1981. Motion-compensated inter-frame coding for video conferencing, *NTC '81*, New Orleans, LA, November 1981, p G.5.3.1 – G.5.3.4.

Puri, A., Hang, H. M. & Schilling, D., 1987. An efficient block-matching algorithm for motion-compensated coding, *IEEE International Conference on Acoustics, Speech, and Signal Processing'87*, Dallas, TX, 6-9 April 1987, pp 1063 – 1066.

Chen, Y. K., (1998). True motion estimation – theory, application, and implementation. *PhD Thesis.* Princeton, NJ: Princeton University Electrical Engineering Department.

Turaga, D., & Alkanhal, M., 1998. Search algorithms for block-matching in motion estimation [online], Carnegie Mellon University, http://www.ece.cmu.edu/~ee899/project/deepak_mid.htm [citied 2 October 2006]

VITAE

Name Surname	: Soner DEDEOĞLU
Address	: Bahçeşehir Üniversitesi Mühendislik Fakültesi Çırağan Cd. Osmanpaşa Mektebi Sk. No: 4 – 6 34349 Beşiktaş / İstanbul / Türkiye
Birth Place / Year	: İstanbul - 1983
Languages	: Turkish (native) - English
Elementary School	: Oğuzkaan College – 1994
High School	: Beşiktaş Atatürk Anatolian High School - 2001
BSc	: Bahçeşehir University - 2005
MSc	: Bahçeşehir University - 2008
Name of Institute	: Institute of Science
Name of Program	: Computer Engineering
Publications	: Ugurdag H.F., Sahin Y., Baskirt O., **Dedeoglu S**., Ugurdag S.G. & Kocak Y. S., 2006. Population-based FPGA solution to mastermind game, *1st NASA/ESA Conference on Adaptive Hardware and Systems,* Istanbul, Turkey, 15-18 June 2006.
	Dedeoglu S. & Kocak Y. S., (2005), Digital net-list simulator with JAVA, *BSc Graduation Project*. Istanbul, Turkey: Bahçeşehir University Computer Engineering Department.
Work Experience	: Bahçeşehir University Computer Engineering Department Teaching and Research Assistant (August 2005 – Today)
	Bahçeşehir University Computer Engineering Department Student Assistant (October 2002 – January 2005)
	Kent State University School of Technology Research Intern (September 2002)