LOW POWER MOTION ESTIMATION BASED FRAME RATE UP-CONVERSION HARDWARE DESIGNS

by

TEVFİK ZAFER ÖZCAN

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

Sabancı University

August 2011

LOW POWER MOTION ESTIMATION BASED FRAME RATE UP-CONVERSION HARDWARE DESIGNS

APPROVED BY

Assist. Prof. Dr. İlker HAMZAOĞLU ……..
(Thesis Supervisor)

Assoc. Prof. Dr. Ayhan BOZKURT ……..

Assoc. Prof. Dr. Meriç ÖZCAN ……..

Assist. Prof. Dr. Hakan ERDOĞAN ……..

Dr. Mustafa PARLAK ……..

DATE OF APPROVAL: ……..

LOW POWER MOTION ESTIMATION BASED FRAME RATE UP-CONVERSION HARDWARE DESIGNS

Tevfik Zafer ÖZCAN

EE, MS Thesis, 2011

Thesis Supervisor: Assist. Prof. Dr. İlker HAMZAOĞLU

Keywords: Frame rate up conversion, true motion estimation, adaptive motion estimation, motion compensation, hardware architecture.

Abstract

Recently flat panel high definition television (HDTV) displays with 100 Hz, 120 Hz and 240 Hz picture rates are introduced. However, video materials are captured and broadcast in different temporal resolutions ranging from 24 Hz to 60 Hz. In order to display these video formats correctly on high picture rate displays, new frames should be generated and inserted into the original video sequence to increase its frame rate. Therefore, frame rate up-conversion (FRUC) has become a necessity. Motion compensated FRUC (MC-FRUC) algorithms provide better quality results than non-motion compensated FRUC algorithms. These MC-FRUC algorithms consist of two main stages, motion estimation (ME) and motion compensated interpolation (MCI). In ME, motion vectors (MV) are calculated between successive frames, and in MCI this MV data is used to generate a new frame that is inserted between two successive frames, thus doubling the frame rate. In addition to these two main steps, intermediate steps such as refinement of the MV field by various algorithms like motion vector smoothing and bilateral ME refinement may be used to improve the quality of the interpolated video.

In this thesis, a perfect absolute difference technique for block matching ME hardware is proposed. The proposed technique reduces the power consumption of a full search ME hardware by 2.2% on a XC2VP30-7 FPGA without any PSNR loss. In addition, a global motion estimation (GME) algorithm and its hardware implementation are proposed. The proposed GME algorithm increases PSNR of 3D recursive search ME algorithm by 2.5% and its hardware implementation is capable of processing 341 720p frames per second. An adaptive technique for GME, which reduces the energy consumption of the GME hardware by 14.37% on a XC6VLX75T FPGA with a 0.17% PSNR loss, is also proposed. Furthermore, an early termination technique for the adaptive bilateral motion estimation (ABIME) algorithm is proposed. The proposed technique reduces the energy consumption of the ABIME hardware by 29% with a 0.04% PSNR loss on a XC6VLX75T FPGA. In addition, an efficient weighted coefficient overlapped block motion compensation (WC-OBMC) hardware which reduces the dynamic power consumption of the reference WC-OBMC hardware by 22% is proposed. The proposed hardware is capable of processing 57 720p frames per second on a XC6VLX75T FPGA. Finally, the ABIME hardware is implemented on a Xilinx ML605 FPGA board.

DÜŞÜK GÜÇ TÜKETİMLİ HAREKET TAHMİNİNE DAYALI ÇERÇEVE HIZI ARTIRIMI DONANIM TASARIMLARI

Tevfik Zafer ÖZCAN

EE, Yüksek Lisans Tezi, 2011

Tez Danışmanı: Yard. Doç. Dr. İlker HAMZAOĞLU

Anahtar Kelimeler: Çerçeve hızı artırımı, gerçek hareket tahmini, uyarlanır hareket tahmini, hareket dengeleme, donanım tasarımı.

ÖZET

Video ve ekran teknolojilerindeki ilerlemeler sayesinde, yakın zamanda 100 Hz, 120 Hz ve 240 Hz görüntü hızlarına sahip düz ekran Yüksek Çözünürlüklü Televizyon (YÇT) ekranları piyasaya çıkarıldı. Fakat video görüntüleri 24 Hz'den 60 Hz'e değişen farklı zamansal çözünürlüklerde kaydedilmekte ve yayınlanmaktadır. Bu farklı video çözünürlüklerini yüksek görüntü hızlı ekranlarda doğru bir şekilde görüntülemek için, yeni çerçeveler oluşturulmalı ve görüntü hızını artırabilmek için video dizinine eklenmelidir. Bu nedenle Çerçeve Hızı Artırımı (ÇHA) bir ihtiyaç olmuştur. Hareket Tahminine (HT) dayalı ÇHA algoritmaları, HT dayalı olmayan ÇHA algoritmalarına oranla daha yüksek kaliteli sonuçlar vermektedir. HT dayalı ÇHA algoritmaları iki temel adımdan oluşur, Hareket Tahmini ve Hareket Dengeleme ile Aradeğerleme (HDA). HT'de ardışık çerçeveler arasındaki Hareket Vektörleri (HV) hesaplanır. HDA adımında bu HV bilgisi kullanılarak yeni çerçeve oluşturulur ve orijinal çerçeveler arasına yerleştirilir. Bu iki temel adıma ek olarak oluşturulan çerçevenin kalitesini artıracak ara adımlar da uygulanabilir. Bu ara adımlar genellikle HV alanını iyileştirme, HV Düzleme ve Çift Yönlü HT İyileştirmesi gibi çeşitli algoritmaları içerir.

Bu tezde, blok eşleştirme HT donanımı için hatasız mutlak değer tahmin tekniği önerildi. Önerilen teknik bir Tam Arama HT Donanımı'nın XC2VP30-7 FPGA'sında güç tüketimini PSNR kaybı olmaksızın % 2.2 azalttı. Buna ek olarak bir Global Hareket Tahmini (GHT) algoritması ve donanımı önerildi. GHT algoritması 3 Boyutlu Özyineli Arama (3BÖA) algoritmasının PSNR değerini %2.5 artırırken önerilen donanım saniyede 341 720p çerçeve işleyebilmektedir. Ayrıca GHT için uyarlanır bir teknik önerildi ve bu teknik %0.17 PSNR kaybıyla donanımın XC6VLX75T FPGA'sında enerji tüketimini %14.37 azalttı. Ayrıca Uyarlanır Çift Yönlü Hareket Tahmini (UÇYHT) Donanımı için bir erken sonlandırma tekniği önerildi. Önerilen teknik donanımın XC6VLX75T FPGA'sında enerji tüketimini %0.04 lük bir PSNR kaybıyla % 29 azalttı. Ayrıca Ağırlıklı Katsayılı Çakışmış Blok Hareket Dengeleme (AK-ÇBHD) algoritması için verimli bir donanım önerildi. Önerilen verimli donanım referans donanımının XC6VLX75T FPGA'sında güç tüketimini %22 azalttı ve saniyede 57 720p çerçeve işleyebilmektedir. Son olarak, UÇYHT donanımı Xilinx ML605 FPGA kartında çalıştırıldı.

Acknowledgements

First and foremost I would like to thank my advisor Dr. İlker Hamzaoğlu for his invaluable guidance and support throughout my study. I appreciate very much for his suggestions, detailed reviews, invaluable advices and life lessons. I particularly want to thank him for his confidence and belief in me during my study. He has been a great mentor to me and I feel privileged to be his student.

I am sincerely grateful to my thesis committee members, Dr. Ayhan Bozkurt, Dr. Meriç Özcan, Dr. Hakan Erdoğan and Dr. Mustafa Parlak, for their invaluable feedback.

I would like to thank to all members of System-on-Chip Design and Testing Lab, Yusuf Adıbelli, Serkan Yalıman, Kadir Akın, Aydın Aysu and Onur Can Ulusel who have been greatly supportive during my study. I also would like to thank Defne Kocaoğlu who was always there for me and provided me with endless motivation.

I would also like to express my deepest gratitude for my beloved family, Neşe and Ahmet Özcan, who always believed in me, and always tried their best to make things easier for me.

Finally I would like to acknowledge Sabancı University and TÜBİTAK for supporting me throughout my graduate education.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

3DRS	:	3-D Recursive Search
ABIME	:	Adaptive Bilateral Motion Estimation
AD	:	Absolute Difference
AGME	:	Adaptive Global Motion Estimation
BIME	:	Bilateral Motion Estimation
BM	:	Block Matching
BRAM	:	Block RAM
BSW	:	Bilateral Search Window
CB	:	Current Block
CF	:	Current Frame
ET-ABIME	:	Early Terminated Adaptive Bilateral Motion Estimation
FRUC	:	Frame Rate Up-Conversion
FS	:	Full Search
GME	:	Global Motion Estimation
GMV	:	Global Motion Vector
HD	:	High Definition
LFSR	:	Linear Feedback Shift Register
MB	:	Macro Block
MC-FAVG	:	Motion Compensated Field Averaging
MC-FRUC	:	Motion Compensated Frame Rate Up-Conversion
MCI	:	Motion Compensated Interpolation
ME	:	Motion Estimation
MSE	:	Mean Squared Error
MV	:	Motion Vector
OBMC	:	Overlapped Block Motion Compensation
PE	:	Processing Element
PF	:	Previous Frame
PSNR	:	Peak Signal-to-Noise Ratio

SAD	:	Sum of Absolute Differences
SD	:	Standard Definition
SW	:	Search Window
VCD	:	Value Change Dump
WC-OBMC:		Weighted Coefficient Overlapped Block Motion Compensation

Chapter 1

INTRODUCTION

Recently flat panel high definition television (HDTV) displays with 100 Hz, 120 Hz and 240 Hz picture rates are introduced. However, video materials are captured and broadcast in different temporal resolutions ranging from 24 Hz to 60 Hz. In order to display these video formats correctly on high picture rate displays, new frames should be generated and inserted into the original video sequence to increase its frame rate. Therefore, frame rate up-conversion (FRUC) has become a necessity [1]. An example FRUC scheme in which the frame rate of the input video sequence is multiplied by 4 is shown in Fig. 1.1.

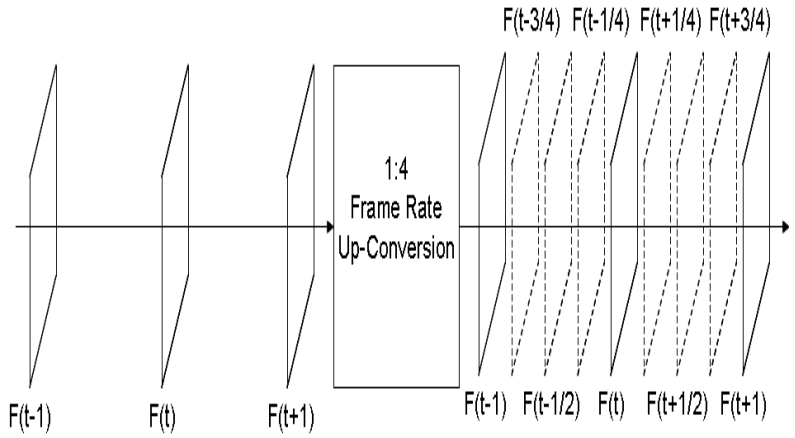
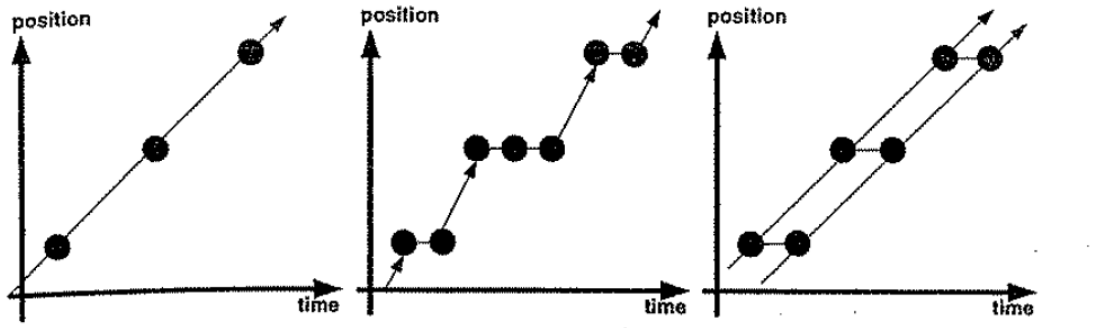


Figure 1.1: An Example FRUC System

The existing FRUC algorithms are mainly classified into two types [2]. First class of algorithms do not take motion of the objects into account, like frame repetition (FR) [3] or linear interpolation (LI) [4]. These algorithms are easy to implement without a significant computational cost. However, at high spatial and temporal resolutions, these algorithms produce visual artifacts [5] like motion judder (if the difference between input and output

frame rate is below 30 Hz) and motion blur (for higher differences). Fig. 1.2 shows the effect of these two situations [1].

In Fig. 1.2(a) the original sequence is shown, where the linear motion of an object is illustrated as a straight line for 3 frames. In Fig. 1.2(b), the case where the motion of the object is recorded by a 24 frames per second (fps) camera and displayed on a 60 Hz display is shown. When picture repetition is applied, some frames will be displayed two times and some will be displayed three times. This is called a 2-3 pull down [6]. In this case the viewer will experience an irregular or jerky motion which is called motion judder. On the other hand, in Fig. 1.2(c), the case where a 50 Hz video is displayed on a 100 Hz display using picture repetition is shown. In this case, the viewer will experience a smooth motion, as the difference between input and output frame rates is higher than 30 Hz. However, the object will be perceived in both positions moving in parallel simultaneously, which will result in a double or blurred object. This is called motion blur.



(a) **(b)** **(c)**

Figure 1.2: Effect of Picture Repetition (a) Original sequence (b) Picture repetition from 24 Hz to 60 Hz (c) Picture repetition from 50 Hz to 100 Hz.

Second class of FRUC algorithms take the motion of objects into account to reduce these artifacts and construct higher quality interpolated frames [2]. These Motion Compensated Frame Rate Up-Conversion (MC-FRUC) algorithms consist of two main stages, Motion Estimation (ME) and Motion Compensated Interpolation (MCI). In ME, a Motion Vector (MV) is calculated between successive frames, and in the MCI step this MV data from the previous step is used to generate a new frame to be inserted between the initial two successive frames, thus doubling the frame rate. This operation can be repeated to further increase the frame rate. In addition to the two main steps, there may be intermediate steps to improve the quality of the interpolated video output. These intermediate steps generally

involve refinement of the MV field by various algorithms like Motion Vector Smoothing and Bilateral ME Refinement.

Among several ME algorithms, Block Matching (BM) is the most preferred method. BM divides the frames of video sequences into *NxN* pixel blocks and tries to find the best matching block according to a cost function from previous frames inside a given search range. The most common cost function is Sum of Absolute Differences (SAD), because of its low computational cost. There are various BM algorithms proposed in the literature. Full Search (FS) algorithm has the best performance as it exhaustively searches every location in the given search range [1]. However, its computational complexity is very high, especially for HD videos. On the other hand, many fast block matching algorithms are available [7-10], which have much less computational complexity while producing acceptable quality results. When motion vectors are generated for FRUC applications, it is important that the vectors represent real motion of the objects which is called the true motion [1]. Although, these algorithms find the best SAD match which is sufficient for video compression, this does not guarantee that those vectors represent the true motion of the objects. Therefore, generally, these algorithms perform poorly when used in frame rate up-conversion applications.

There are several ME algorithms [11-15] which aim to extract the true motion information between the frames of video sequences. These algorithms depend on two assumptions. The objects are larger than blocks so that neighbors of a block should have similar motions, and motions are continuous and spread through a duration of time so that blocks in successive frames of a video sequence should have similar motions. A recursive search algorithm takes advantage of these assumptions, and for the current block evaluates the motion vectors of spatial and temporal neighboring blocks instead of doing an exhaustive or static patterned search. 3-D Recursive Search (3DRS) [11] is one of the best implementations of these assumptions, and produces a smooth and accurate motion vector field suitable for MC-FRUC applications.

The MVs are obtained by ME process which assumes that objects move along the motion trajectory. However, during this process holes and overlapped areas may be produced in the interpolated frames due to no motion trajectory passing through and multiple motion trajectories passing through, respectively [16]. This degrades the quality of generated frames. This problem can be solved by median filtering overlapped pixels [17], and using spatial interpolation methods for holes [18], or prediction methods by analyzing MV fields for covered and uncovered regions [16][19]. However, these methods require complex operations and give unsatisfactory results in cases of non-static backgrounds and camera motions.

Bilateral ME (BIME) algorithms are recently proposed to avoid holes and overlapped areas in interpolated frames more effectively [20]-[23]. BIME algorithms construct a MV field for the interpolated frame and do not produce any overlapped areas or holes during interpolation. BIME algorithms can be used as a refinement step after ME [20].

MVs obtained by ME are used for MCI in ME based FRUC algorithms. BM ME assumes that all the pixels in a block have the same motion since a single MV is found for each block. However, the parts of several objects that move in different directions can be in the same block or MVs obtained by ME may not represent the true motion of the objects because of ME errors [24]. In these cases, block based MCI produces blocking artifacts or block boundary discontinuities which reduce the resulting video quality both in terms of subjective and objective metrics. Overlapped Block Motion Compensation (OBMC) is used to avoid these blocking artifacts and increase the resulting video quality in ME based FRUC [25] and in video compression [26]. OBMC determines the motion of each pixel in a block by considering the MV of the block itself and the MVs of the neighboring blocks. OBMC is performed by increasing the size of the blocks in the interpolated frame during frame interpolation.

1.1 Thesis Contribution

In this thesis, we first propose a perfect absolute difference prediction (P-ADP) method based on the comparison prediction technique proposed in [27]. P-ADP method corrects the incorrect predictions in the same clock cycle. Since the clock period of the ME hardware is long enough to perform two subtractions in one clock cycle, if the prediction is incorrect, the correct subtraction is done by changing the select inputs of the multiplexers in the same clock cycle. The proposed technique reduces the average dynamic power consumption of the 256 processing element (PE) fixed block size ME hardware proposed in [28] by 2.2% without any PSNR loss on a XC2VP30-7 FPGA.

Then, we propose a Global Motion Estimation (GME) algorithm and its hardware implementation. In order to improve the performance of the 3DRS algorithm [29], a method for including the effect of camera motions which are independent from the movements of the objects is developed. The proposed GME algorithm finds a global motion vector (GMV) for a frame by performing ME using FS algorithm on a set of blocks in this frame, and it uses this GMV as an additional candidate MV for 3DRS ME. The proposed algorithm results in a 2.5%

PSNR increase on average. We also propose an Adaptive GME algorithm (AGME) and its hardware architecture. The adaptive technique checks the validity of the GMV and the presence of global motions by using the statistics of selected MVs during ME with 3DRS and if GME is not necessary it is disabled adaptively. The proposed adaptive technique reduces the energy consumption of the GME hardware by 14.37% with a PSNR loss of 0.17% on average on a XC6VLX75T FPGA. The proposed hardware is capable of processing 341 720p frames per second.

In addition, we propose an early termination technique for the adaptive bilateral motion estimation (ABIME) algorithm proposed in [30]. The proposed early terminated ABIME (ET-ABIME) algorithm exploits the spiral search pattern to adaptively change the size of Bilateral Search Window (BSW) of a MB based on the success of BIME vector refinement process for the neighboring spatial and temporal MBs. The proposed technique reduces the energy consumption of the ABIME hardware by 29% with a PSNR loss of 0.04% on a XC6VLX75T FPGA.

We also propose an efficient WC-OBMC hardware based on the reference hardware presented in [31]. The proposed hardware reduces redundant operations done in the reference hardware by using the prior information about the MVs of the interpolated block and its neighboring blocks and adaptively changes the processing flow based on this data. The proposed hardware also uses a pipelining technique to increase the throughput of the reference hardware. The proposed hardware reduces the dynamic power consumption of the reference hardware by 22% and it is capable of processing 57 720p frames per second.

Finally, the proposed ABIME hardware in [30] is implemented on a ML605 FPGA board which is a state-of-the-art Xilinx board. In the FPGA implementation, we used the ABIME hardware as a slave peripheral and MicroBlaze processor as a master. We also implemented a software running on MicroBlaze processor. Using this software, inputs are transferred to the hardware from a host computer, the outputs of the hardware are sent to the host computer and also displayed on a monitor.

In this thesis, the performances of FRUC algorithms are evaluated as follows. Every even numbered frame is omitted from the sequence and ME is employed between odd frames. Then, MCI step is applied using these MVs to re-synthesize the even numbered frames as shown in Fig. 1.3. After all even numbered frames are generated, the original even numbered frames and interpolated even numbered frames are compared as shown in Fig. 1.4. The comparison is done using Mean Squared Error (MSE) metric by calculating the differences of each pixel at the same locations in the original and interpolated frames and summing the

squares of these values as shown in Equation (1.1). After all MSEs for all even numbered frames are found, the corresponding Peak Signal-to-Noise (PSNR) ratios are found as shown in Equation (1.2). PSNR is a widely used objective evaluation metric for evaluating the quality of video sequences.

(1.1)

where N and M denote the frame height and width respectively, I is the interpolated frame and O is the original frame.

(1.2)

where MAX is the maximum value of a pixel. If pixels are represented by 8 bits, then MAX is 255.

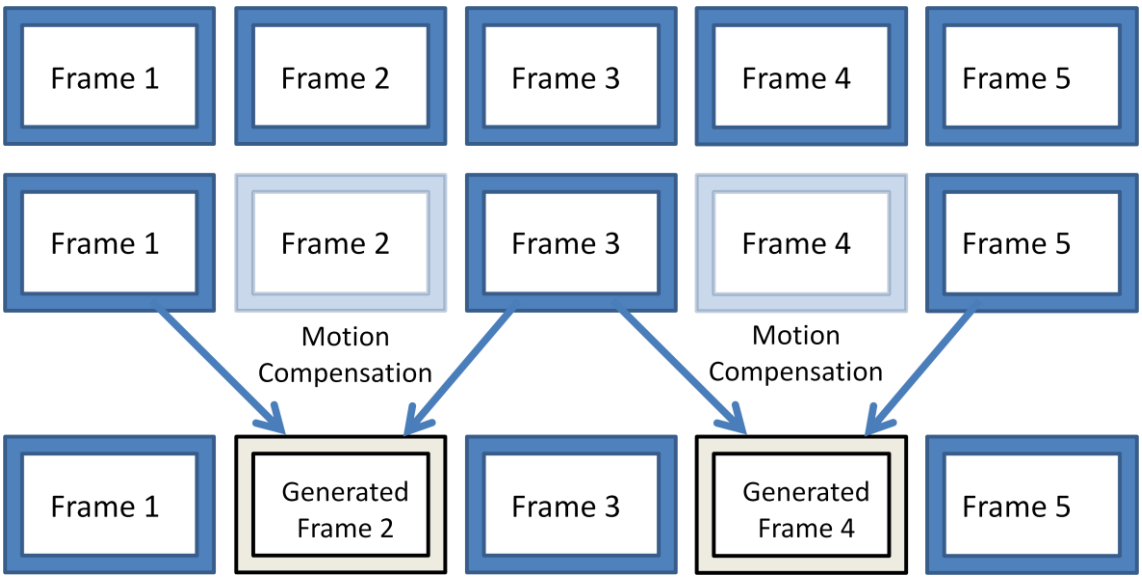


Figure 1.3: Generation of Even Numbered Frames

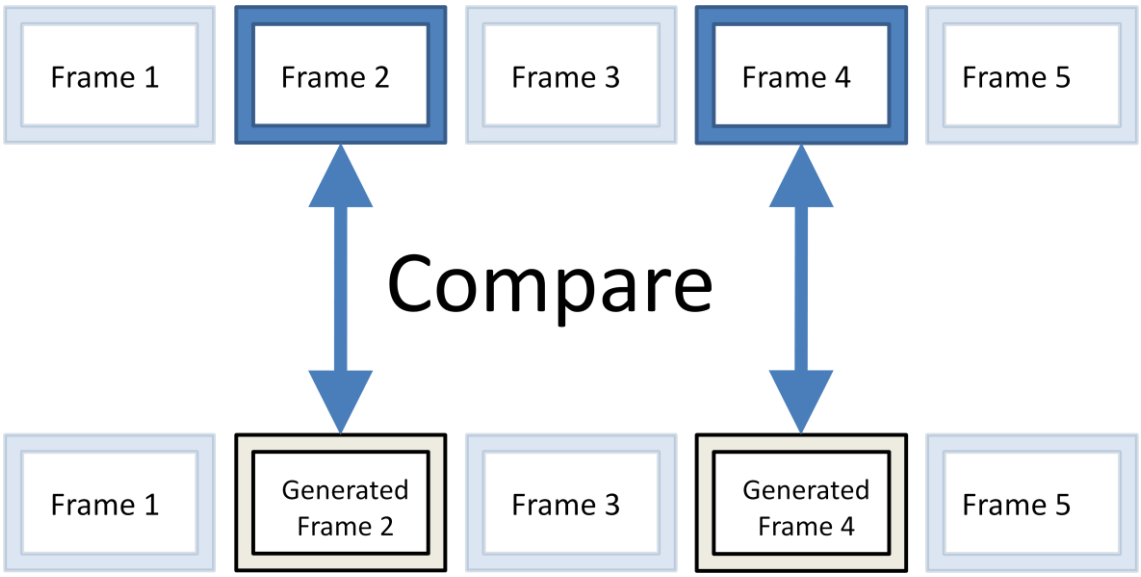


Figure 1.4: Comparison of Even Numbered Frames

The rest of the thesis is organized as follows. In Chapter 2, a perfect absolute difference prediction method for block matching motion estimation hardware is presented. In Chapter 3, a global motion estimation algorithm and its hardware architecture are presented. In addition, an adaptive technique for GME is presented. Chapter 4 explains the early terminated adaptive bilateral motion estimation algorithm and its hardware implementation. In Chapter 5, an efficient weighted coefficient overlapped block motion compensation hardware implementation is presented. In Chapter 6, board implementation of bilateral motion estimation hardware is explained. Finally, Chapter 7 concludes this thesis.

Chapter 2

A PERFECT ABSOLUTE DIFFERENCE PREDICTION METHOD FOR BLOCK MATCHING MOTION ESTIMATION HARDWARE

2.1 Absolute Difference Prediction Technique

The block based ME methods use Block Matching (BM) Algorithms, which divide the frames of video sequences into NxN pixel blocks and try to find the best matching block according to a cost function from previous frames inside a given search range. The most common cost function is Sum of Absolute Differences (SAD) shown in Equation (2.1), because of its low computational complexity. The pixels inside a block are assumed to have the same MV, which is assigned to by BM algorithms.

(2.1)

Full Search (FS) algorithm is based on computing SADs at all possible locations in a given search window. It takes a block in the current frame *n*, whose top left pixel is at position and compares it to every block in the previous frame, *n-1*, inside a pre-defined search area which is also centered at . The motion trajectory connecting the best matching block (with the minimum SAD) in the previous frame with the current block is assigned as the Motion Vector *V* of . This process is illustrated in Fig. 2.1 [1]. The definition of full search is given in Equations (2.2) and (2.3), where *C* denotes the candidate motion vectors pointing to possible search locations inside the search area *SA*, *N* and *M* denotes width and height of *SA* respectively, *V* denotes the selected MV.

(2.2)

(2.3)

FS guarantees finding the minimum SAD value inside a given search range. However, it is not designed to extract the true motion of the objects between frames and it is computationally expensive as it exhaustively evaluates every possible MV candidate.

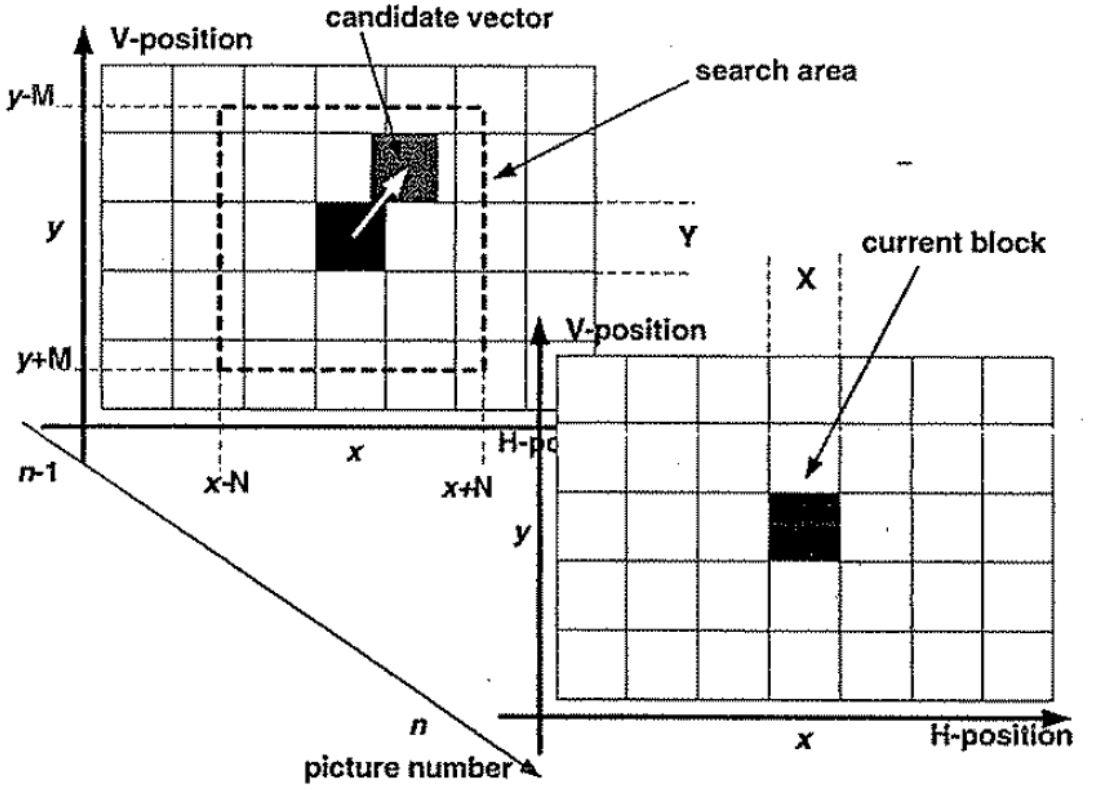


Figure 2.1: Full Search ME

BM ME hardware architectures perform absolute difference (AD) operations for calculating SAD values [28, 32, 33]. The number of AD operations performed by BM ME algorithms is very high. For example, FS algorithm performs 103,809,024 AD operations for finding motion vectors of a CIF (352x288) frame in a [-16, 15] search range. Using larger frame sizes, larger search ranges or multiple reference frames significantly increases the number of AD operations performed.

Therefore a comparison prediction (CP) technique for reducing the power consumption of BM ME hardware by reducing the power consumption of absolute difference operations is proposed in [27]. CP technique replaces the 8-bit comparator in AD hardware with a few gates. CP technique can easily be used in all BM ME hardware.

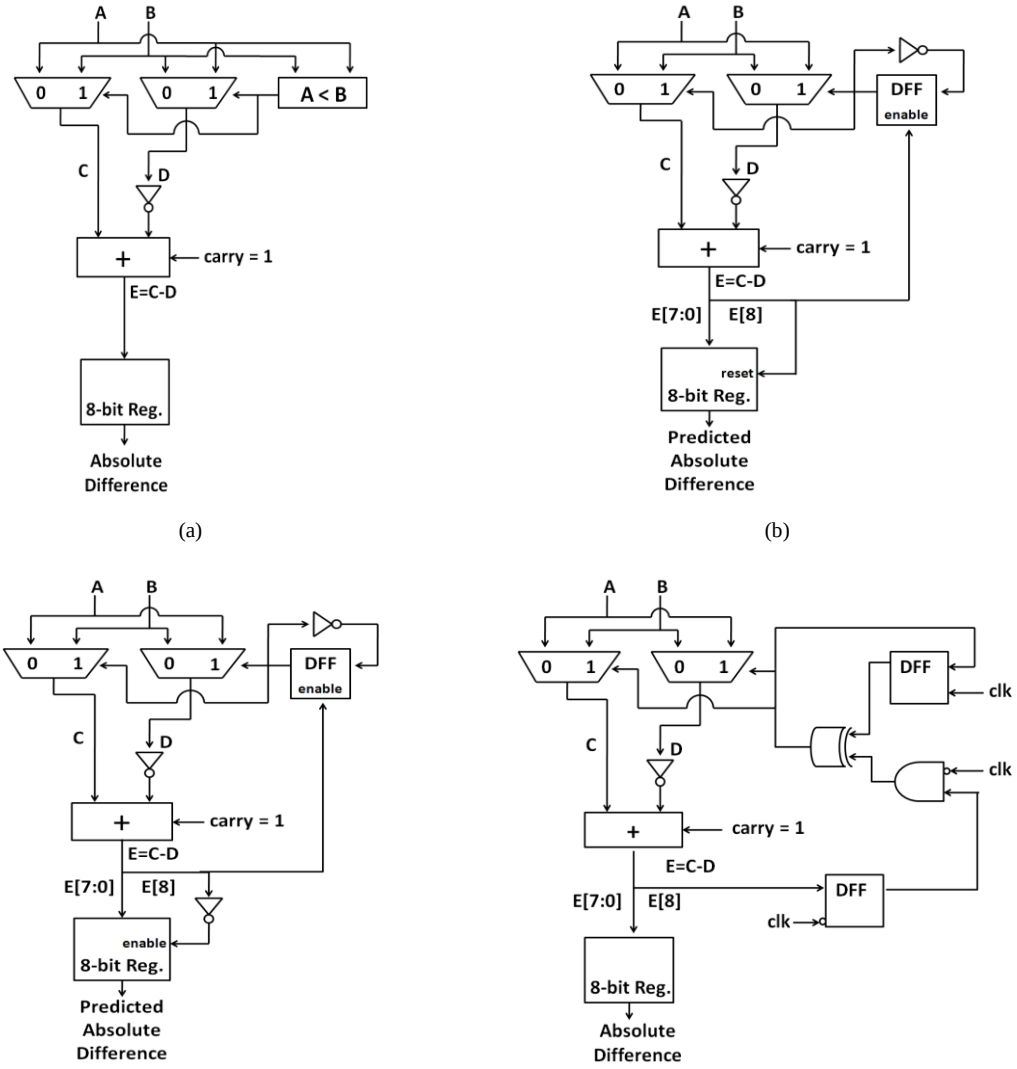


Figure 2.2: (a) AD Hardware, (b) Reset based ADP Hardware (c) Enable based ADP Hardware (d) Perfect ADP Hardware

The proposed technique is applied to the 256 processing element (PE) fixed block size ME hardware proposed in [28]. This ME hardware implements full search algorithm with a zigzag search flow in a [-16, +15] search range. It finds the search location in the search window (SW) that best matches the 16x16 current Macroblock (MB) based on SAD criterion. While the SW is searched for the current MB, each PE stores a current MB pixel and calculates the AD with corresponding pixels in the SW. ADs calculated by the 256 PEs for a search location are added by a pipelined adder tree in order to calculate the SAD value of this search location. After the SAD values for all search locations in the SW are calculated, the search process for the current MB finishes.

Proposed technique reduces the average dynamic power consumption of this ME hardware by 6.1% with a 0.01% Peak Signal-to-Noise Ratio (PSNR) loss and by 9.3% with 0.04% PSNR loss on a XC2VP30-7 FPGA.

The AD hardware used in the 256 PE fixed block size ME hardware is shown in Fig. 2.2 (a). It includes an 8-bit comparator, two 8-bit 2to1 multiplexers, an 8-bit subtractor, and an 8-bit register. AD hardware compares the 8-bit current MB pixel with the 8-bit SW pixel and subtracts the smaller one from the larger one. The result of the comparison is used to select the proper pixels for subtraction so that result of subtraction is always positive.

The proposed CP technique avoids the comparison in the AD hardware by predicting the comparison result using the previous subtraction result. As shown in Fig. 2.2 (b) and (c), the proposed technique stores an initial prediction in a D Flip-Flop (DFF), and updates it after each incorrect prediction. The initial prediction predicts that current MB pixel will be subtracted from the SW pixel. If the sign bit of the subtraction result is 0, the prediction is correct and the DFF is not updated. If the sign bit of the subtraction result is 1, the prediction is incorrect and the prediction in the DFF is reversed. This new prediction is used for predicting the comparison results for the following pixels.

When the comparison prediction is incorrect, the result of the subtraction operation is different from the absolute difference of the two input pixels. This causes PSNR loss. Since the pixels in the SW usually have high spatial correlation, CP technique has very high prediction accuracy. Therefore, it causes very small PSNR loss. The accuracy of the comparison prediction is determined on 5 video sequences each with 80 frames. The results show that the proposed CP technique correctly predicts the results of 90.1% of the comparisons performed by the PEs.

When there is an incorrect prediction, the larger pixel value is subtracted from the smaller one and the subtraction result is negative. Using this negative value for SAD calculation will result in an incorrect SAD. In order to reduce the impact of this negative value on the SAD and therefore reduce the impact of using incorrect predictions on the PSNR obtained by ME, four different methods; reset based AD prediction (R-ADP), enable based AD prediction (E-ADP), R-ADP used with a checkerboard pattern (CR-ADP) and E-ADP used with a checkerboard pattern (CE-ADP) are proposed in [27].

As shown in Fig. 2.2 (b), R-ADP method uses the sign bit of the subtraction result as a reset signal for the 8-bit register used for storing the absolute difference result. When an incorrect prediction is done, this 8-bit register is set to 0. Therefore, instead of a negative value, 0 is used for SAD calculation. Since the SW pixels have high spatial correlation, in

case of consecutive incorrect comparison predictions, it is likely that the current MB pixel value is close to the SW pixel values. Therefore, predicting absolute difference as 0 will have a small impact on SAD.

As shown in Fig. 2.2 (c), E-ADP method uses the inverse of the sign bit of subtraction result as an enable signal for the 8-bit register used for storing absolute difference result. When an incorrect prediction is done, this 8-bit register is disabled. Therefore, instead of a negative value, previous absolute difference is used for SAD calculation. In case of consecutive incorrect comparison predictions, predicting absolute differences as 0 may cause the SAD to be smaller than it should be and this SAD value may incorrectly be selected as the minimum SAD. E-ADP method avoids this by using the previous absolute difference in case of incorrect comparison prediction. In addition, since E-ADP method keeps previous AD value in the 8-bit register, it does not consume dynamic power for setting the 8-bit register to 0.

CR-ADP method applies the R-ADP method to 128 of the 256 PEs in the PE array. CE-ADP method applies the E-ADP method to 128 of the 256 PEs in the PE array. In both CR-ADP and CE-ADP methods, the 128 PEs are determined by a checkerboard pattern. The

		Comparison Prediction Accuracy	Quantization Parameter	Absolute Difference		Proposed Methods							
						R-ADP		CR-ADP		E-ADP		CE-ADP	
				Bit Rate	PSNR	Bit Rate	PSNR	Bit Rate	PSNR	Bit Rate	PSNR	Bit Rate	PSNR
Video Sequences	Foreman (288x352)	90.9%	25	1753	38.916	2088	38.893	1800	38.909	1814	38.902	2070	38.910
			30	804	35.588	1016	35.466	837	35.556	1023	35.474	833	35.563
			35	398	32.252	534	31.920	417	32.147	525	31.968	411	32.197
	Mobile (288x352)	86.5%	25	9440	38.226	9382	38.216	9331	38.221	9487	38.216	9338	38.225
			30	6647	33.146	6692	33.130	6602	33.140	6557	33.139	6566	33.141
			35	4444	28.074	4496	28.036	4376	28.069	4481	28.048	4383	28.071
	Mother & Daughter (288x352)	91.3%	25	2149	39.972	2217	39.942	2170	39.966	2163	39.948	2150	39.967
			30	1198	36.496	1248	36.437	1200	36.480	1203	36.459	1214	36.462
			35	658	32.966	663	32.829	660	32.861	699	32.834	662	32.901
	Akiyo (288x352)	93.3%	25	2219	41.193	2258	41.166	2211	41.183	2329	41.180	2257	41.186
			30	1447	37.336	1495	37.254	1415	37.299	1444	37.279	1422	37.300
			35	955	35.183	903	35.006	955	35.128	955	35.100	898	35.174
	Paris (288x352)	89.6%	25	6102	38.703	6288	38.678	6243	38.690	6170	38.689	6121	38.695
			30	4126	34.155	4268	34.129	4136	34.145	4221	34.139	4178	34.145
			35	2716	29.395	2817	29.357	2719	29.388	2775	29.363	2718	29.390
Average PSNR Loss			25	–	0%	–	0.06%	–	0.02%	–	0.04%	–	0.01%
			30		0%		0.17%		0.06%		0.13%		0.06%
			35		0%		0.44%		0.17%		0.34%		0.08%

Table 2.1: Average PSNR and Bit Rate for Several Video Sequences

proposed R-ADP, E-ADP, CR-ADP, CE-ADP methods are integrated to H.264 JM reference encoder software version 14.2. The average PSNR (dB) and bit rate (Kbps) obtained by these methods for several video sequences for 16x16 fixed block size full search motion estimation in a [-16, 15] search range with zigzag search flow are given in Table 2.1.

2.2 Perfect Absolute Difference Prediction Method

In this thesis, we propose perfect ADP (P-ADP) technique. As shown in Fig. 2.2 (d), P-ADP method corrects the incorrect predictions in the same clock cycle. Since the clock period of the ME hardware is long enough to perform two subtractions in one clock cycle, if the prediction is incorrect, the correct subtraction is done by changing the select inputs of the multiplexers in the same clock cycle. The predicted absolute difference is calculated before the negative edge of the clock. In order to detect the incorrect prediction, the sign bit of the prediction is stored in a negative edge triggered DFF. When the clock is low, if the output of this DFF is 1, the select inputs of the multiplexers are corrected by an XOR operation with the current incorrect prediction, and the correct prediction is stored in the DFF. Therefore, if the prediction is correct, the absolute difference is calculated with one subtraction operation. If the prediction is incorrect, the absolute difference is calculated with two subtraction operations. Since 90.1% of the absolute differences are calculated with one subtraction operation, P-ADP method reduces the average dynamic power consumption of the ME hardware with no PSNR loss.

The proposed P-ADP method is implemented in Verilog HDL and this hardware implementation is integrated to the 256 PE ME hardware. The resulting Verilog RTL codes are synthesized to a XC2VP30-7 FPGA using Precision RTL 2005b and mapped to the same FPGA using ISE 8.2i. The ME hardware implementations are verified with post place & route simulations using Modelsim 6.1c.

The power consumptions of the ME hardware are estimated using Xilinx XPower tool. In order to estimate the dynamic power consumption of a ME hardware, timing simulation of the placed & routed netlist of that ME hardware is done at 50 MHz for a full frame of the Foreman video sequence using Mentor Graphics ModelSim 6.1c and the signal activities are stored in a Value Change Dump (VCD) file. This VCD file is used for estimating the dynamic power consumption of that ME hardware.

		AD	Proposed Methods									
			R-ADP		CR-ADP		E-ADP		CE-ADP		P-ADP	
		value	value	% red.	value	% red.	value	% red.	value	% red.	value	% red.
Area	Slices	9353	8628	8	8934	4	8542	9	9039	3	8885	5
	LUTs	16145	14353	11	15210	6	14217	12	15261	5	14621	9
	DFFs	7377	7633	-3	7505	-2	7633	-3	7505	-2	7893	-6
Average Dynamic Power (mW)		775.90	713.39	8.1	729.57	6.0	704.09	9.3	728.77	6.1	758.36	2.2

Table 2.2: Comparison of Motion Estimation Hardware Architectures

The area and power consumptions of ME hardware with standard AD and ME hardware with proposed methods are given in Table 2.2. The ME hardware with R-ADP, E-ADP, CR-ADP, CE-ADP and P-ADP methods use 8%, 9%, 4%, 3% and 5% less slices than the ME hardware with standard AD. The ME hardware with R-ADP, E-ADP, CR-ADP, CE-ADP and P-ADP methods have 8.1%, 9.3%, 6.0%, 6.1% and 2.2% less dynamic power consumption than the ME hardware with standard AD.

R-ADP and E-ADP methods reduce the dynamic power consumption and area of the ME hardware more than the CR-ADP and CE-ADP methods. However, R-ADP and E-ADP methods have a PSNR loss of 0.06% and 0.04% respectively, whereas CR-ADP and CE-ADP methods have a PSNR loss of 0.02% and 0.01% respectively. The P-ADP method reduces the dynamic power consumption less than the other methods, but it obtains the same PSNR as the full search algorithm. Therefore, one of these five methods can be used for ME depending on performance and power consumption requirement of the video compression or video enhancement application.

Chapter 3

GLOBAL MOTION ESTIMATION ALGORITHM AND HARDWARE

3.1 Global Motion Estimation Algorithm

The physical three-dimensional motion projected onto two-dimensional space is referred to as true motion. The ability to track true motion by observing changes in luminance intensity is critical to many video applications such as FRUC [24]. Different from the other motion estimation algorithms like FS, a true motion estimation algorithm should also take other measures into account like spatio-temporal consistency of the MV field around objects. This is based on two assumptions. Objects are larger than blocks so that MV field around a block should be smooth and objects have inertia, i.e. object motions are spread through time to several frames. Therefore, motions of the objects can also be tracked by analyzing previous frames.

There are several true motion estimation algorithms in the literature [11-15] that check the spatio-temporal consistency around blocks to obtain the true motion of the object containing that block. Three Dimensional Recursive Search (3DRS) [11] is one of the best implementations of these two assumptions. Instead of evaluating all possible candidate locations in a search window, 3-D recursive search algorithm uses spatial and temporal predictions to select only a few candidate vectors from the 3-D neighborhood (spatial and temporal neighbors) of the current block, thus reducing computational complexity of ME which is the most computationally expensive part of MC-FRUC and also resulting in a smooth and accurate true MV field.

There are two problems with the first assumption in 3DRS. First, because of the processing order of the blocks (starting from top-left block and ending with the bottom-right block), not all of the spatial neighboring blocks of the current block (CB) are available, e.g. the blocks to the right of the CB and the blocks that are below the CB. This problem is solved

with the second assumption. Since the motion of the object continues over several frames, instead of the motion vectors of the spatial neighboring blocks that are not yet calculated the motion vectors of the corresponding temporal neighboring blocks are used.

Second, all vectors are zero or undefined at initialization. Therefore, the motion vector of the object cannot be found in any of the neighboring blocks in the first frame. This problem is solved by adding random update vectors from a pre-defined set of noise vectors, filling the MV field with not accurate but possible motion data. In [29], it is proposed to use the candidate set shown in Equation (3.1) and illustrated in Fig 3.1. Squares marked as *S* are vectors taken from spatial neighbors and square marked as *T* is the vector taken from the previous frame. CB denotes the current block.

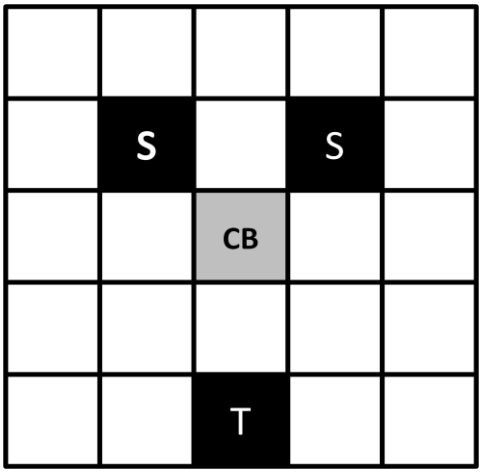


Figure 3.1: Candidate Search Locations Set for 3DRS

(3.1)

where the update vectors and are randomly selected from the following update set:

(3.2)

In order to improve the performance of the 3DRS algorithm [29], a method for including the effect of camera motions which are independent from the movements of the objects is developed in this thesis. For this purpose a Global Motion Vector (GMV) is added to the candidate set of 3DRS algorithm. FS algorithm is used to find the GMV because of its suitability for hardware implementation. The proposed Global Motion Estimation (GME) algorithm performs ME using FS algorithm on a set of blocks in a frame. As shown for a 1280x720 size frame in Fig 3.2, these blocks are selected from the blocks that are close to the edges since they are more likely to belong to the background. After ME is performed on these selected blocks, average MVs are calculated for each edge. Then, GMV is calculated by finding the median of these 4 MVs and averaging the median two values. One GMV is found for each frame and used as an additional candidate in 3DRS ME algorithm for the following frame in time.

Considering that the camera movements might be continuous for several frames and there might not be any camera movements between the frames, performing GME between every frame can result in redundant operations. In order to avoid these redundant operations, a technique for detecting the necessity of the GME is also proposed. After GME is performed on the first frame in time, if the GMV is selected more than 1/8 of the blocks in the frame during ME with 3DRS, it is assumed that the GMV is still valid. Therefore, GME is not performed again and the current GMV is used in the next frame. In addition, the {0, 0} vector

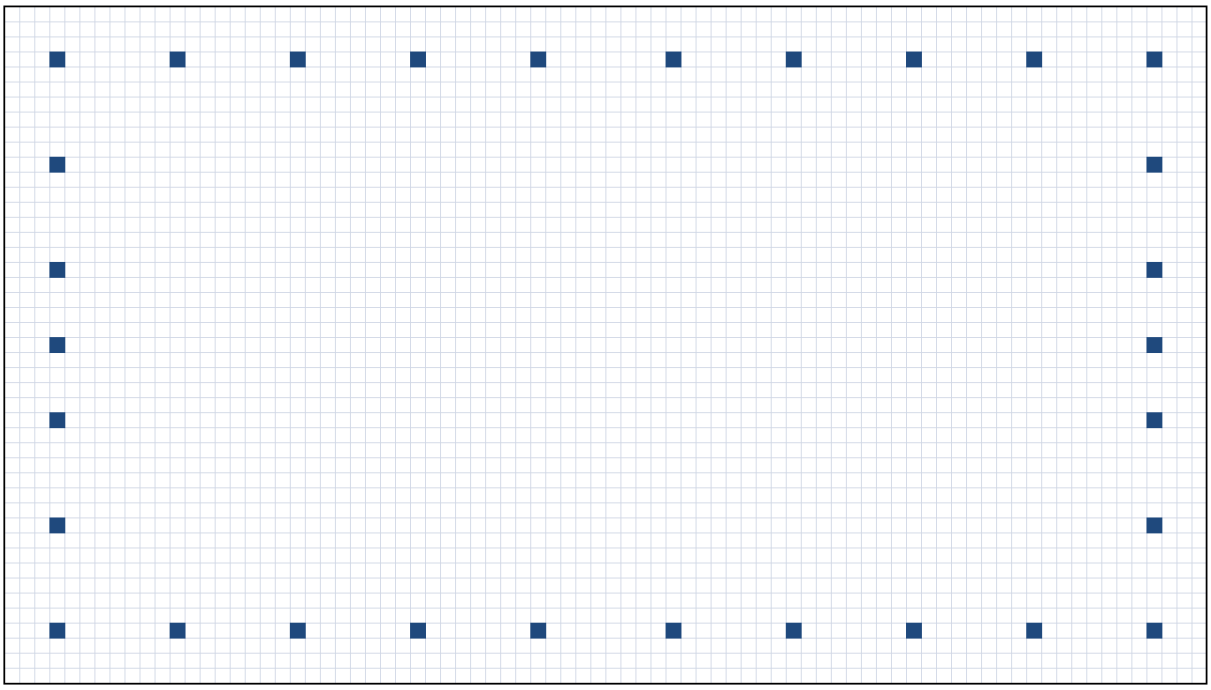


Figure 3.2: Blocks Used for GME (1280x720 Resolution)

Video	Resolution	FR	LI	3DRS	GME		AGME	
		PSNR (dB)	PSNR (dB)	PSNR (dB)	PSNR (dB)	PSNR Diff. (%)	PSNR (dB)	PSNR Diff. (%)
Mobile	352 x 288	20.6415	25.2320	24.5158	26.1554	6.69%	26.1770	6.78%
Foreman	352 x 288	26.2470	29.8586	30.5020	31.3653	2.83%	31.4403	3.08%
Spiderman	720 x 576	21.6702	23.6874	23.9321	23.8181	-0.48%	23.9591	0.11%
Irobot	720 x 576	21.6651	23.4856	24.3188	24.4581	0.57%	24.3753	0.23%
Gladiator	720 x 576	19.9773	22.0586	23.4640	23.1621	-1.29%	23.1284	-1.43%
ParkJoy	1280 x 720	18.2158	20.1119	22.5827	24.1018	6.73%	23.9542	6.07%
SthmlPan	1280 x 720	21.9768	23.9610	33.1125	33.9925	2.66%	34.0646	2.88%
NewMobCal	1280 x 720	26.1470	29.7561	31.8369	32.8922	3.31%	32.9302	3.43%
ParkJoy	1920x1080	18.2866	20.1520	23.3245	24.5479	5.25%	24.0730	3.21%
CrowdRun	1920x1080	21.5343	24.2368	26.3229	26.4787	0.59%	26.4757	0.58%
DucksTakeOff	1920x1080	26.3352	29.4377	29.5996	29.8033	0.69%	29.7990	0.67%
	Average					2.50%		2.33%

Table 3.1: PSNR Comparison of the Proposed Techniques

Video	Resolution	3DRS	GME	AGME	
		Number of SAD Calculations	Number of SAD Calculations	Number of SAD Calculations	Reduction (%)
Mobile	352 x 288	57024	12118272	586560	-95.16%
Foreman	352 x 288	57024	12118272	1078080	-91.10%
Spiderman	720 x 576	233280	13557504	929472	-93.14%
Irobot	720 x 576	233280	13557504	659136	-95.14%
Gladiator	720 x 576	233280	13557504	929472	-93.14%
ParkJoy	1280 x 720	518400	20360192	14110464	-30.70%
SthmlPan	1280 x 720	518400	20360192	8892160	-56.33%
NewMobCal	1280 x 720	518400	20360192	14110464	-30.70%
ParkJoy	1920x1080	1157760	21212672	15577472	-26.57%
CrowdRun	1920x1080	1157760	21212672	2732416	-87.12%
DucksTakeOff	1920x1080	1157760	21212672	2732416	-87.12%
	Average				-71.47%

Table 3.2: Number of SAD Calculations Comparison

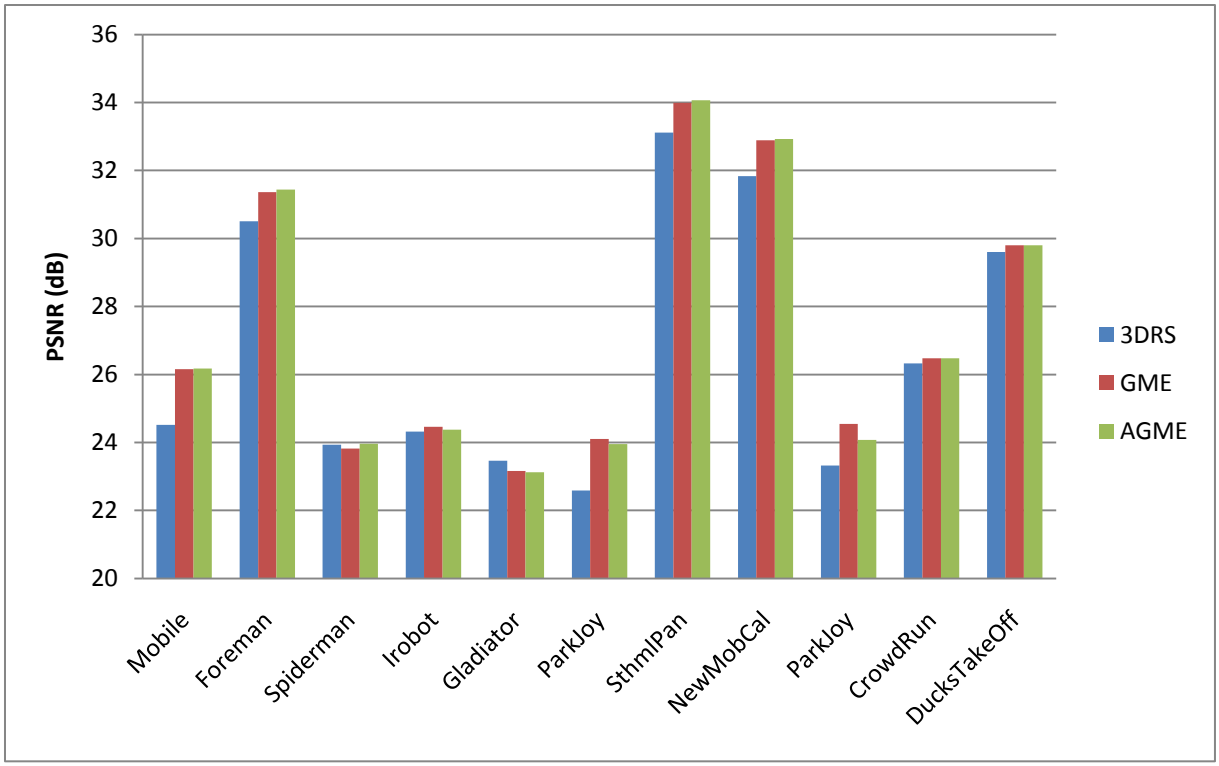


Figure 3.3: PSNR Comparison of the Proposed Techniques

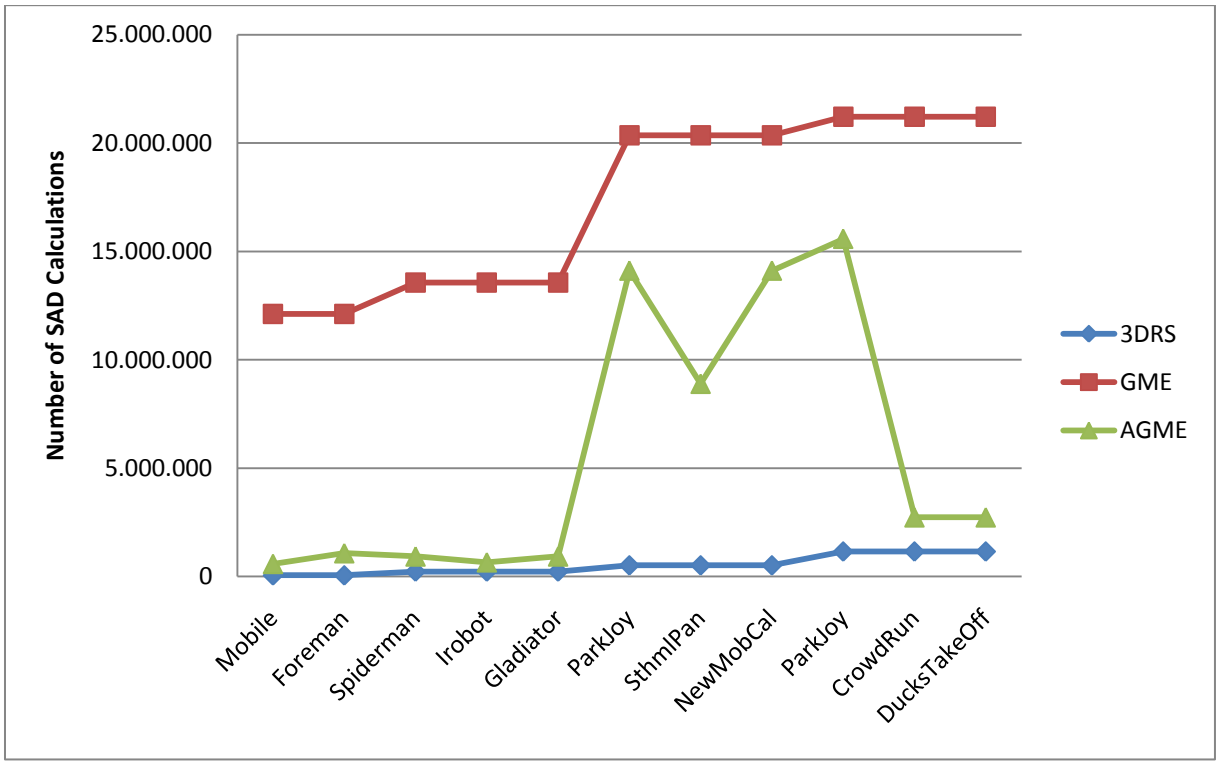


Figure 3.4: Number of SAD Calculations Comparison

is added to the candidate set of 3DRS and if this vector is selected more than 1/8 of the blocks in the frame during ME with 3DRS, it is assumed that there is no global motion present.

Therefore, GME is not performed. Using these two techniques, an adaptive GME (AGME) algorithm is developed.

The performance and computational cost comparison of the proposed GME and AGME algorithms with 3DRS algorithm is shown in Tables 3.1 and 3.2, and Figures 3.3 and 3.4. In addition, Table 3.1 shows that motion compensated frame rate up conversion techniques give better performance than frame repetition (FR) and linear interpolation (LI) techniques. The proposed GME algorithm increases PSNR of 3DRS algorithm by 2.50% on average. AGME algorithm reduces the computational load of GME algorithm by 71.47% at the expense of an average PSNR loss of 0.17%.

3.2 Global Motion Estimation Hardware

For performance and power consumption comparison, efficient hardware architectures for implementing 3DRS, GME and AGME algorithms are proposed. The 3DRS ME hardware

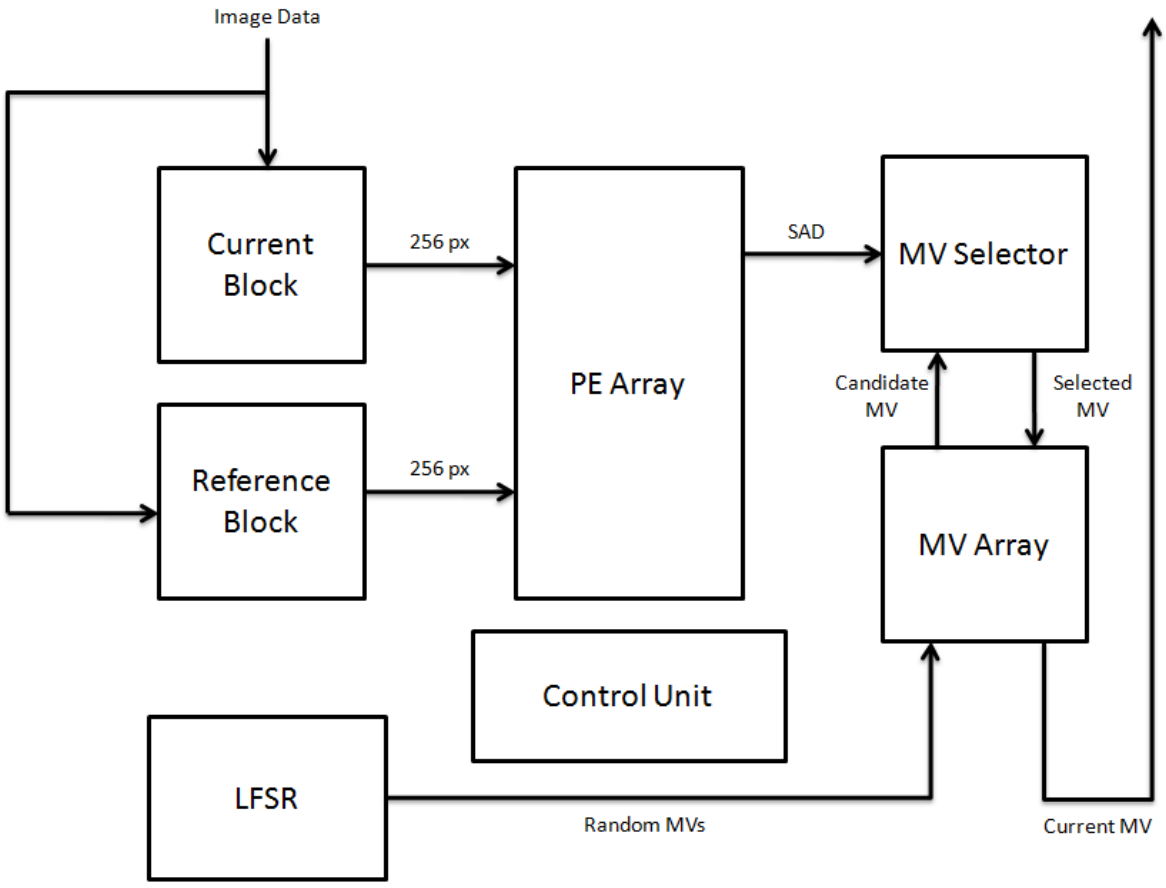


Figure 3.5: 3DRS Motion Estimation Hardware

is shown in Fig 3.5. It consists of control unit, MV array, MV selector, 256 PE array, LFSR module and the data arrays for reference and current frames. When the hardware starts processing the first frame, since the MV array is empty, it is filled by the MVs generated using the LFSR. After the MV array is initialized with random MVs, the processing of the following frame starts. Control unit sends the position of the current block to the MV array. Using this information MV array reads the MVs that will be used during ME, updates the ones which will be updated and forms the candidate MVs. At the same time, the current block is written to current block data array. Then, the reference block for the current candidate MV is written to the reference block data array. When both the current and the reference blocks are ready, SAD calculation is performed using the 256 PE array and the result is sent to the MV selector. For each candidate MV, new data for the reference block is written to the reference block data array. After SAD value of each candidate MV is found, MV selector selects the MV with the smallest SAD and stores it in the MV array. This process is repeated for all the blocks.

The GME hardware is shown in Fig 3.6. It consists of the 3DRS ME hardware, a global motion estimation module and a BRAM array to store the search window used during FS. GME module checks the position of the current block. If this block is one of the sampling blocks shown in Fig 3.2, it generates a signal indicating that GME is enabled and sends this signal to the relevant parts of the hardware. This enable signal indicates that for that block, FS will be performed on a [-32, +32] search range. In that case, the necessary search window is written to the BRAM array. Then, for each search location starting from the upper left corner to the lower right corner, the reference block array is filled in a zigzag pattern. While the FS ME is performed for the current block, at the same time, the 3DRS ME for the current block is performed. The MV found by FS ME is sent to the GME module, and it is added to the register of the edge the current block belongs to. For the blocks that are not used for GME, only 3DRS ME is performed. After the ME is completed for all the blocks in a frame, GMV is calculated using the information stored in the GME module and used in the ME of the following frame.

In the AGME hardware, while ME is done using 3DRS, the number of times GMV is selected and the number of times {0, 0} vector is selected are stored. Using this information, GME module decides if GME will be performed or not, and generates the GME enable signal. If GME is not enabled, the hardware only performs ME with 3DRS and the GMV is not updated.

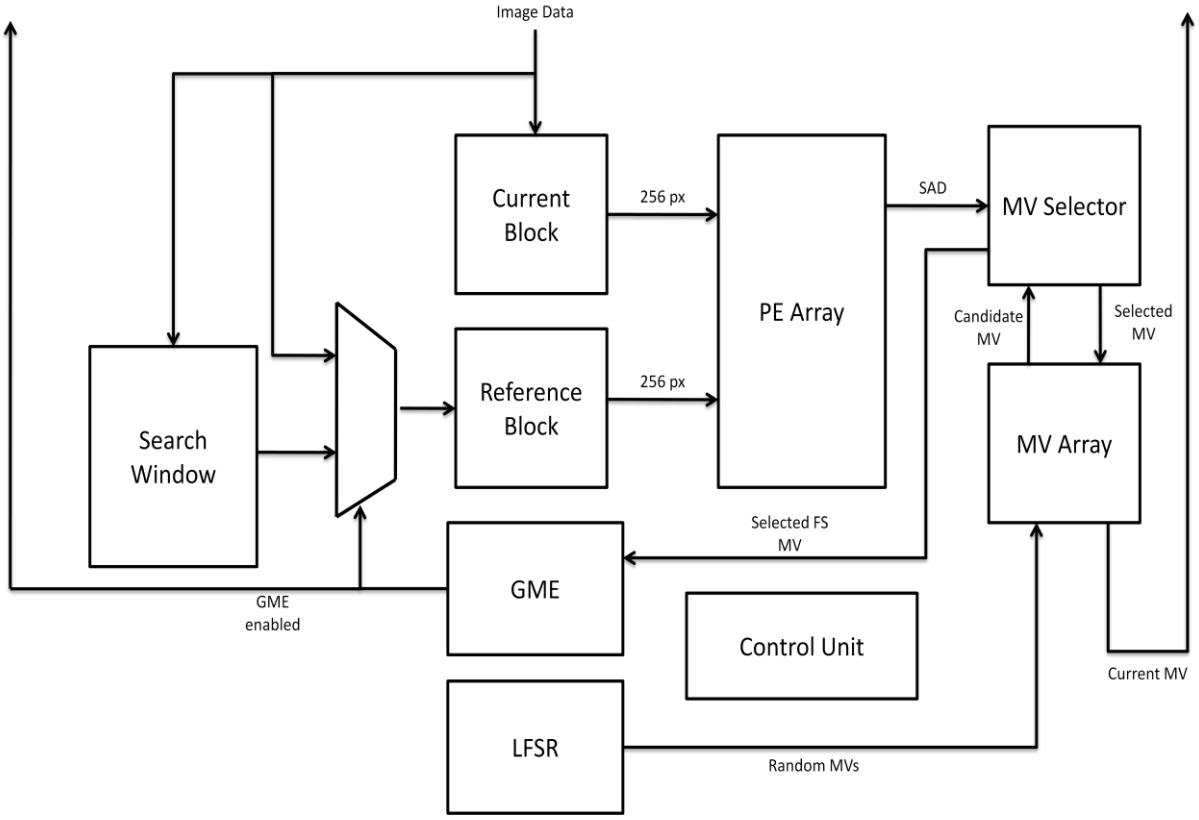


Figure 3.6: GME Hardware

The proposed 3DRS, GME and AGME hardware architectures are implemented in Verilog HDL. The resulting Verilog RTL codes are synthesized to a Xilinx XC6LVX75T Virtex-6 FPGA using Synopsys Synplify Pro and mapped to the same FPGA using Xilinx ISE 11.4. The ME hardware implementations are verified with post place & route simulations using Modelsim 6.1c. The performance and the area usage of these hardware are shown in Table 3.3. The power consumptions of these ME hardware are estimated using Xilinx XPower tool and shown in Table 3.4. In order to estimate the dynamic power consumption of a ME hardware, timing simulation of the placed & routed netlist of that ME hardware is done at 50 MHz for the even frames of first 11 frames of the NewMobCal video sequence using Mentor Graphics ModelSim 6.1c, and the signal activities are stored in a Value Change Dump (VCD) file. This VCD file is used for estimating the dynamic power consumption of that ME hardware. The results show that the AGME hardware has 14.37% less energy consumption than the GME hardware.

	3DRS Hardware	GME Hardware	AGME Hardware
LUTs	7999	9324	9286
Slices	2160	2504	2546
BRAMs	4	24	24
Frequency (MHz)	280	210	210

Table 3.3: Area and Performance Comparison of the Proposed Techniques

	3DRS Hardware	GME Hardware	AGME Hardware	Reduction
Average Power Consumption per Frame (mW)	107.495	158.830	144.762	
Execution Time per Frame (msec)	7.073	12.442	11.661	
Average Energy Consumption per Frame (mJ)	0.760	1.976	1.692	14.37%

Table 3.4: Power and Energy Consumptions of the Proposed Techniques

Chapter 4

EARLY TERMINATED ADAPTIVE BILATERAL MOTION ESTIMATION ALGORITHM AND HARDWARE

4.1 Bilateral Motion Estimation Algorithm and Hardware

One of the potential problems with BM algorithms for FRUC is the possible hole and overlapped areas in the interpolated frames. Since a new frame is generated by interpolation between previous frame (PF) and current frame (CF) based on motion vectors (MV) and these vectors are obtained by ME which assumes that objects move along the motion trajectory, holes and overlapped areas may be produced in the interpolated frames due to no motion trajectory passing through and multiple motion trajectories passing through, respectively [16]. This degrades the quality of generated frames as shown in Fig 4.1. This problem can be solved by median filtering overlapped pixels [17], using spatial interpolation methods for holes [18], or prediction methods by analyzing MV fields for covered and uncovered regions [16][19]. However, these methods have high computational complexity and give

Figure 4.1: (a) Hole and Overlapping Regions (b) Frame Generated by Bilateral ME

unsatisfactory results, especially in cases of non-static backgrounds and camera motions. To overcome this problem more efficiently, Bilateral Motion Estimation (Bi-ME) methods are proposed [20]-[23], which construct a MV field from the viewpoint of the to-be-interpolated frame, and therefore do not produce any overlapped areas or holes during interpolation.

In other ME algorithms, an NxN size block from CF, CB, is kept stationary and a match for this CB is searched inside a search window in PF. In Bi-ME, an imaginary frame is assumed to exist which will be the intermediate frame after it is interpolated, and ME is performed from the viewpoint of this frame. Therefore, the block inside the to-be-interpolated frame is kept stationary and a match for this block is tried to be found both in CF and PF at symmetric locations to each other. The trajectory connecting two symmetric blocks in CF and PF always passes through the stationary block inside the to-be-interpolated frame. When the best match is found, the trajectory between two symmetric blocks is assigned as the MV to the block that will be interpolated. The Bi-ME process is shown in Fig. 4.2.

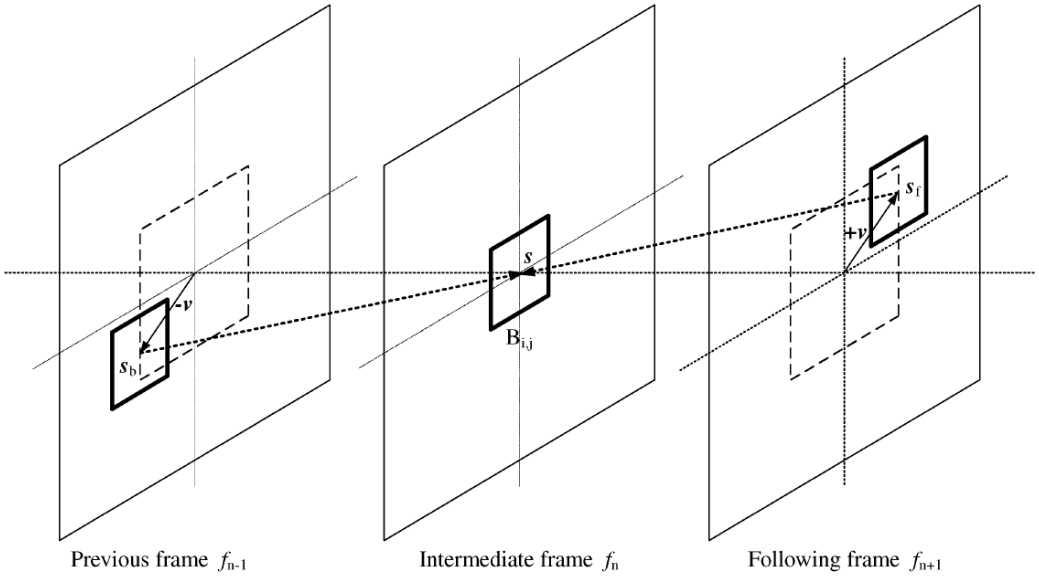


Figure 4.2: Bilateral Motion Estimation

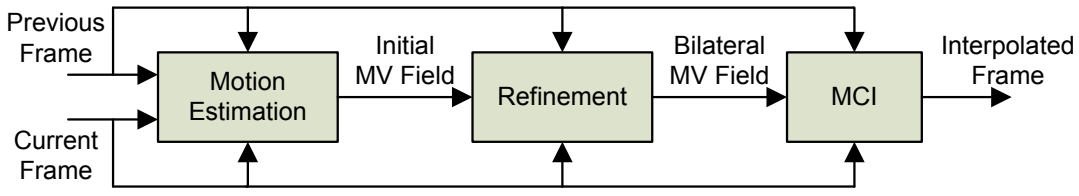


Figure 4.3: Bilateral ME as a Refinement Step

Bi-ME, when used exclusively as the ME step, does not yield acceptable results for MC-FRUC applications due to its lack of true motion estimation capability. It is proposed in [27] that Bi-ME can be used as a refinement step to a ME algorithm as shown in Figure 4.3.

An adaptive bilateral motion estimation (ABIME) algorithm and its hardware implementation are proposed in [30]. The proposed ABIME algorithm refines the motion vector field between successive frames by employing a spiral search pattern and by adaptively assigning weights to candidate search locations. The ABIME algorithm searches the best SAD match in the Bilateral Search Window (BSW) starting from the center and evaluates the candidate search locations by assigning weights. It conserves the true motion property of the motion vector field by favoring the candidate search locations near the center where the initial MV points to.

The proposed ABIME algorithm refines MVs found by a true ME algorithm to improve the FRUC quality. The search process is performed by calculating the SAD between the 16x16 current MB and 16x16 reference MB at each candidate search location in their respective BSWs. After the SAD value for a search location is calculated, current MB and reference MB moved symmetrically in the current frame (CF) BSW and the reference frame (RF) BSW.

The block diagram of the proposed ABIME hardware is shown in Fig. 4.4. The hardware is composed of 16 BRAMs, 2 Vertical Rotators, 2 Horizontal Splitters, 2 Horizontal Shifters, PE Array, Control Unit, Adder Tree and Comparator & MV Updater. The hardware finds refined MV of a 16x16 MB using the proposed adaptive Bi-ME algorithm in a [-4, 4] pixel search range. Its latency is 9 clock cycles; 1 cycle for Control Unit, 1 cycle for synchronous read from memory, 1 cycle for Vertical Rotator, 4 cycles for Adder Tree and 2 cycles for Comparator & MV Updater. The Control Unit generates the required address and control signals to compute the weighted sum of bilateral absolute difference (WSBAD) values of candidate search locations in BSWs in RF and CF. After the WSBAD value of a search location is calculated, Comparator & MV Updater compares this WSBAD value with the minimum WSBAD value in order to determine the search location that produces minimum WSBAD value and the corresponding refined MV.

In conventional BM ME algorithms, current macroblock (MB) pixels do not change during the search process of a MB, only the reference MB pixels change for each search location. However, in Bi-ME algorithms, both current MB pixels and reference MB pixels change during the search process. This increases control overhead and memory accesses. The proposed all connected 256 processing element (PE) systolic array, ladder type memory

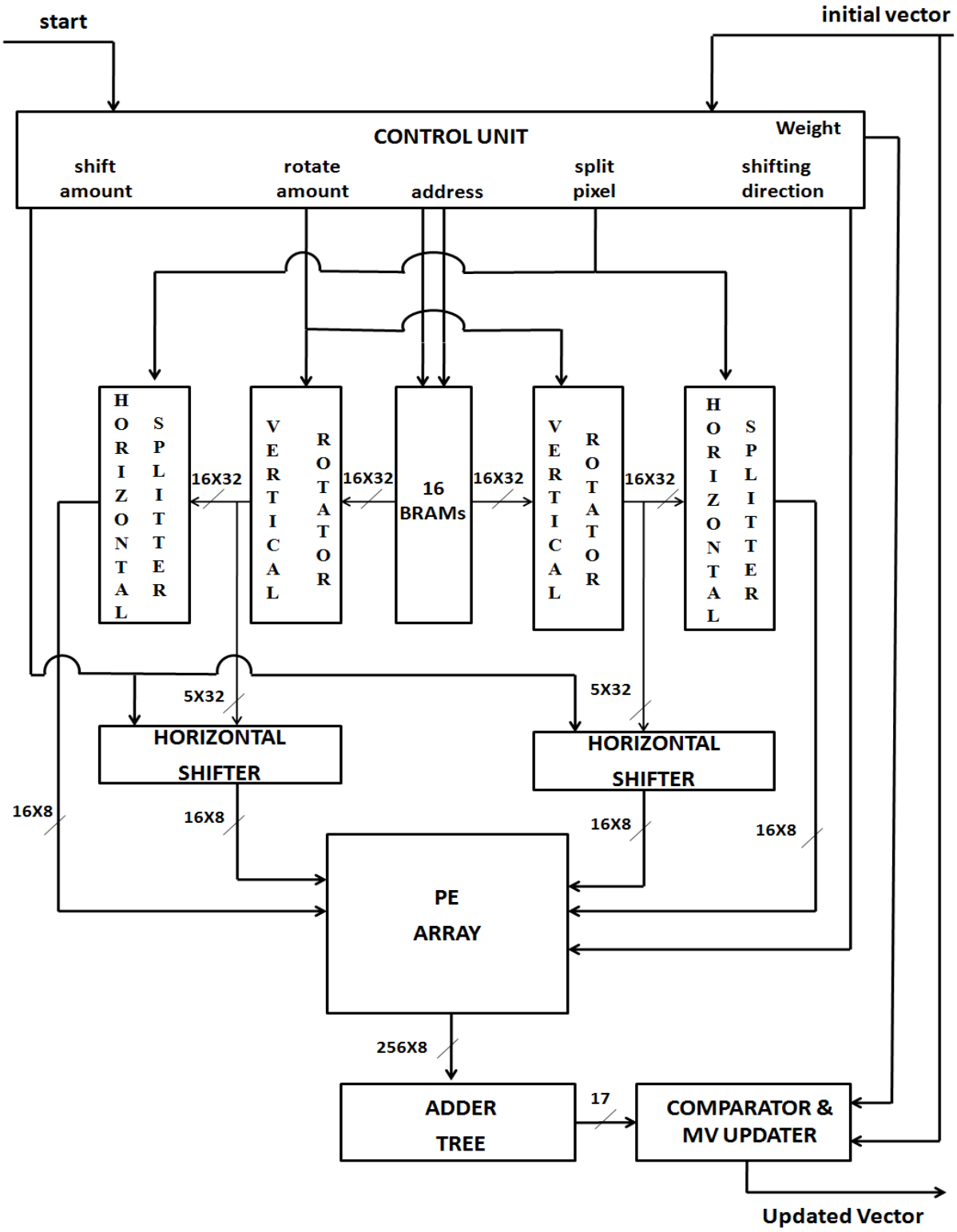


Figure 4.4: ABIME Hardware

organization, symmetric data placement in memory and data alignment techniques reduce the amount of memory accesses by enabling high amounts of data reuse.

4.2 Early Terminated Adaptive Bilateral Motion Estimation Algorithm and Hardware

In this thesis, we propose an early termination technique for reducing the computational complexity of the ABIME algorithm. The proposed early terminated ABIME (ET-ABIME) algorithm exploits the spiral search pattern to adaptively change the size of BSW of a MB based on the success of BIME vector refinement process for the neighboring spatial and temporal MBs. Our experiments showed that the success of BIME vector refinement for a MB is correlated with the success of BIME vector refinement for the neighboring MBs. Therefore, in order to determine the size of BSW of the current MB, the proposed technique checks the number of its spatial and temporal neighboring MBs whose MVs are updated by the ABIME algorithm. If the MVs of at least 5 out of 9 spatial and temporal neighboring MBs of the current MB are updated by the ABIME algorithm, it uses a [-4, +4] BSW for the current MB. Otherwise, it uses a [-2, +2] BSW for the current MB. As shown in Table 4.1, our experiments also showed that 83.27% of the MVs updated at the 3rd and 4th levels of the spiral search pattern are also updated at the lower levels. Therefore, when the proposed technique decides to use a [-4, +4] BSW for the current MB, ABIME is performed on the 3rd and 4th levels of the BSW only if the initial MV is updated in any of the lower levels.

Video	Resolution	Number of MVs Updated at 3rd or 4th Levels	Number of These MVs also Updated at Lower Levels	%
Football	352 x 240	435	339	77.93%
Mobile	352 x 288	6	5	83.33%
Foreman	352 x 288	52	39	75.00%
Spiderman	720 x 576	4383	4310	98.33%
Irobot	720 x 576	2220	2036	91.71%
Gladiator	720 x 576	3960	3835	96.84%
ParkJoy	1280 x 720	4027	3814	94.71%
SthmlPan	1280 x 720	470	298	63.40%
NewMobCal	1280 x 720	4	3	75.00%
DucksTakeOff	1280 x 720	216	165	76.39%
	Average			83.27%

Table 4.1: Number of Updated Vectors at Level 3 and 4 and Number of These Vectors Also Updated at Lower Levels

	Resolution	3DRS	3DRS+ BIME [20]	3DRS+ EBIME [22]	3DRS+ ABIME [30]	3DRS+ ET-ABIME [proposed]
Bi-ME Search Range	-	-	-4,+4	-4,+4	-4,+4	Adaptive
Football	352x240	20.56	21.79	**22.03**	21.67	21.58
Mobile	352x288	27.74	26.29	26.43	**28.07**	**28.07**
Foreman	352x288	31.85	33.42	**33.65**	33.19	33.13
Spiderman	720x576	23.98	**24.59**	24.25	24.41	24.32
Irobot	720x576	24.33	24.55	24.43	**24.92**	24.77
Gladiator	720x576	22.90	**24.77**	23.10	24.06	24.25
ParkJoy	1280x720	24.10	24.61	**24.81**	24.76	24.71
SthlmPan	1280x720	34.00	34.13	34.06	**34.73**	34.61
NewMobCal	1280x720	33.70	33.85	33.79	**34.76**	**34.76**

Table 4.2: PSNR Comparison of Proposed Techniques

	Resolution	EBIME [22]	BIME[20] ABIME	ET-ABIME	Diff. (%) ET-ABIME vs. ABIME
Football	352x240	4,949,343	1,309,770	554,778	-57.64%
Mobile	352x288	5,973,345	1,571,724	592,452	-62.31%
Foreman	352x288	5,973,345	1,571,724	540,932	-65.58%
Spiderman	720x576	25,080,111	6,429,780	4,295,900	-33.19%
Irobot	720x576	25,080,111	6,429,780	2,625,756	-59.16%
Gladiator	720x576	25,080,111	6,429,780	3,640,252	-43.38%
ParkJoy	1280x720	56,165,319	14,288,400	6,814,696	-52.31%
SthlmPan	1280x720	56,165,319	14,288,400	4,601,856	-67.79%
NewMobCal	1280x720	56,165,319	14,288,400	4,908,624	-65.65%
Average					-56%

Table 4.3: SAD Calculation Comparison of Proposed Techniques

PSNR and computational load comparison of ABIME and ET-ABIME techniques and previously proposed bilateral motion estimation algorithms are shown in Tables 4.2 and 4.3. The comparison of ABIME and ET-ABIME techniques is also shown in Figures 4.5 and 4.6.

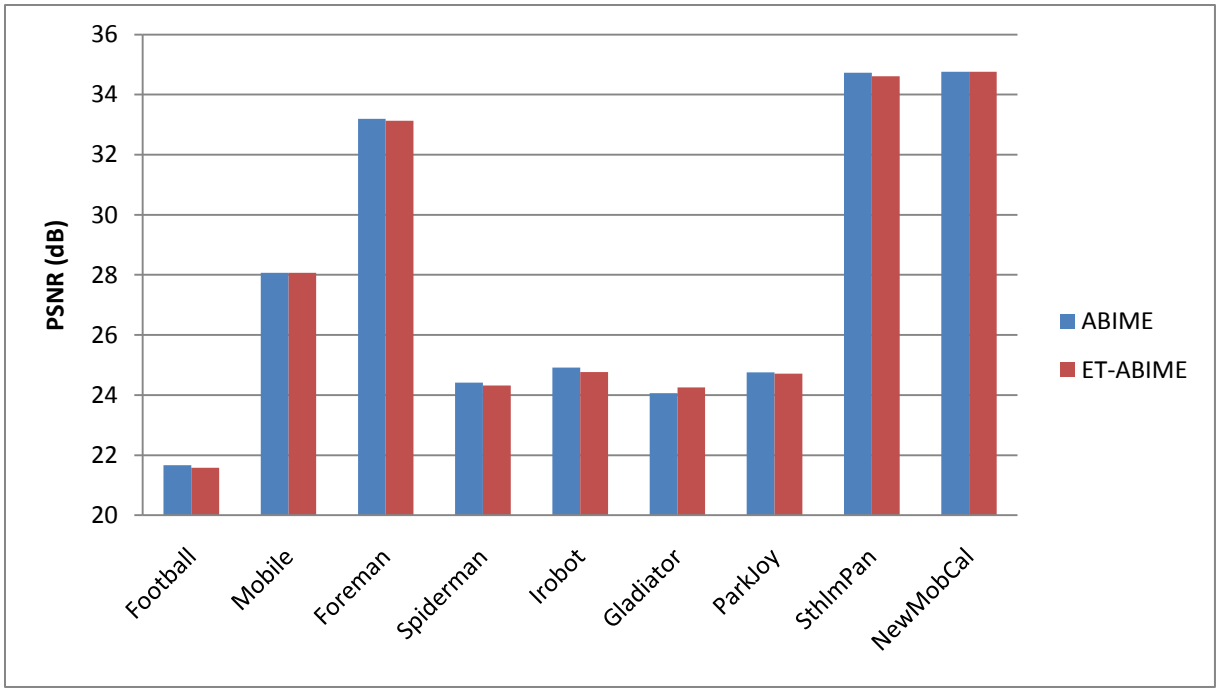


Figure 4.5: PSNR Comparison of Proposed Techniques

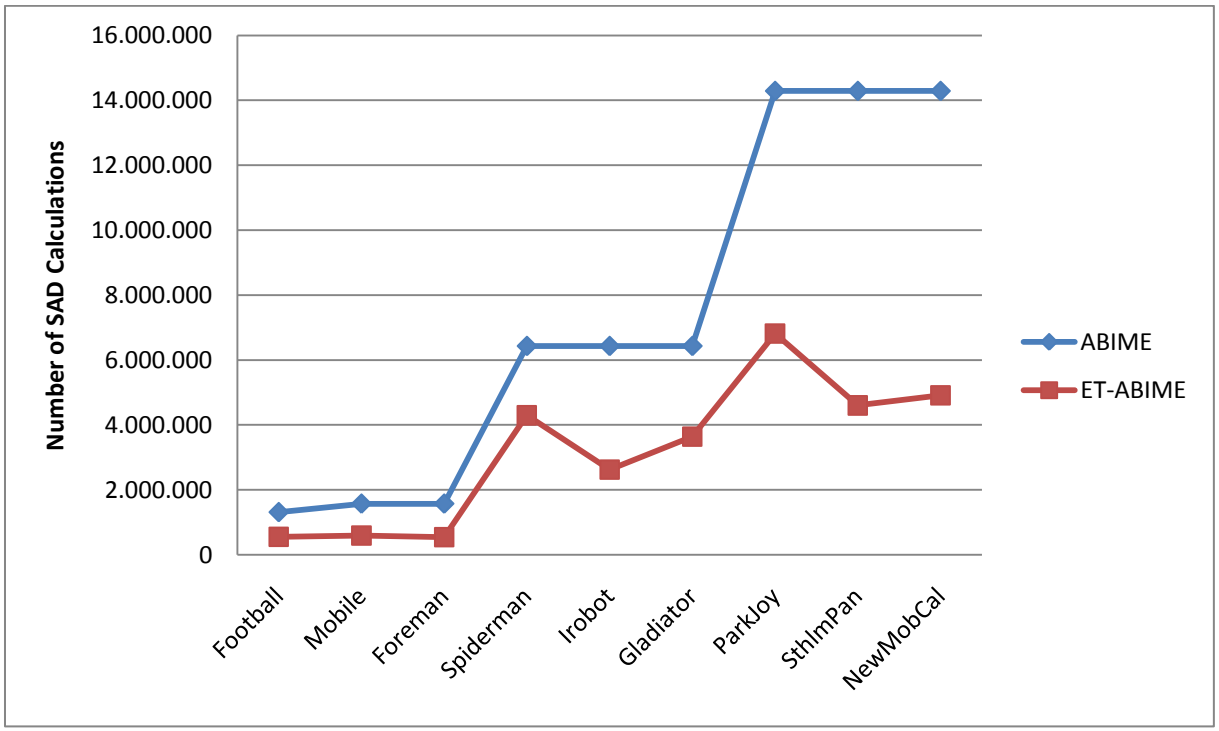


Figure 4.6: SAD Calculation Comparison of Proposed Techniques

The results show that the proposed ET-ABIME algorithm produces better PSNR results for all video sequences than basic 3DRS algorithm. For 4 of 9 video sequences either ABIME or ET-ABIME provides better quality than both BIME and EBIME algorithms. For the 2 of the 12 video sequences BIME provides the best quality but our proposed ET-ABIME algorithm

provides on the average 0.24 dB better quality than Bi-ME. For the 3 video sequences EBIME provides the best quality. However, as shown in Table 4.3, EBIME is more than 8 times computationally intensive than ET-ABIME algorithm. On the average, ET-ABIME produces 0.03% less PSNR than ABIME. But, it performs 56.33% less SAD calculations than ABIME.

We also integrated the proposed early termination technique to ABIME hardware. This is done by adding one BRAM to the ABIME hardware to store the information whether the MVs of the blocks are updated by ABIME algorithm or not, and by modifying the control unit shown in Fig. 4.4 to determine the size of the BSW based on this information. The ET-ABIME hardware consumes less energy than the ABIME hardware by both adaptively reducing the size of the BSW and using the early termination technique. In addition, when a [-2, +2] BSW is used, only the data in the BRAMs required for a [-2, +2] BSW are updated. Therefore, the switching activity in the BRAMs are reduced which further reduces the energy consumption.

The proposed ET-ABIME hardware architecture is implemented in Verilog HDL. The Verilog RTL codes are synthesized to a Xilinx XC6VLX75T FPGA using Synopsys Synplify Pro and mapped to the same FPGA using Xilinx ISE 11.4. The hardware implementations are verified with post place & route simulations using Mentor Graphics Modelsim 6.1c.

As shown in Table 4.4, the performances and areas of the ABIME and ET-ABIME hardware implementations are almost the same. The ABIME hardware consumes 4320 slices (14067 LUTs and 6869 DFFs), which is 37% of all slices of a XC6VLX75T FPGA. PE array consumes 2376 slices (8192 LUTs), one Vertical Rotator consumes 316 slices (1024 LUTs), one Horizontal Splitter consumes 70 slices (128 LUTs), one Horizontal Shifter consumes 159 slices (243 LUTs), Adder Tree consumes 883 slices (1927 LUTs) and the remaining slices are used for Comparator & MV Updater, Control Unit and multiplexers before address ports of the BRAMs. In addition, 9216 bits on-chip memory is used for storing CF BSW and RF BSW. These 9216 bits are stored in 16 BRAMs.

The power consumptions of the ABIME and ET-ABIME hardware are estimated using Xilinx XPower tool. In order to estimate the dynamic power consumption of a BIME hardware, timing simulation of the placed & routed netlist of that BIME hardware is done at 50 MHz for the even numbered frames of the first 11 frames of the SthlmPan video sequence using Mentor Graphics ModelSim 6.1c and the signal activities are stored in a Value Change Dump (VCD) file. This VCD file is used for estimating the dynamic power consumption of

that BIME hardware. As shown in Table 4.5, the proposed early termination technique reduced the energy consumption of the ABIME hardware by 29.37%.

	ABIME Hardware	ET-ABIME Hardware
LUTs	14067	13919
Slices	4320	4441
BRAMs	16	17
Frequency (MHz)	240	220

Table 4.4: Area and Performance Comparison of Proposed Techniques

	ABIME Hardware	ET-ABIME Hardware
Average Power Consumption per Frame (mW)	192.056	165.706
Execution Time per Frame (msec)	20.452	16.741
Average Energy Consumption per Frame (mJ)	3.928	2.774
Reduction in Energy Consumption (%)		29.37%

Table 4.5: Power and Energy Consumptions of Proposed Techniques

Chapter 5

AN EFFICIENT WEIGHTED COEFFICIENT OVERLAPPED BLOCK MOTION COMPENSATION HARDWARE

5.1 Weighted Coefficient Overlapped Block Motion Compensation Algorithm and Hardware

Motion Compensated Field Averaging (MC-FAVG) [1] is the most basic MCI method. MC-FAVG algorithm combines two adjacent frames linearly. Each block in the PF is shifted towards the CF according to the value of its MV, and similarly each block in the CF is shifted towards PF along its motion trajectory. The algorithm is shown in Equation (5.1)

–

(5.1)

where denotes the intensity value of the pixel at location in frame *n*, α denotes the up-conversion ratio (0.5 for doubling the frame rate), and is the MV associated with that pixel.

The block based ME uses the assumption that all the pixels in a block have the same motion as there exists a single motion vector for each block. However, different parts of objects that move in different directions can be in the same block or MV field generated by the ME step may not represent the correct motion of the objects due to ME errors. In these cases, conventional block based interpolation may produce blocking artifacts or block boundary discontinuities that reduce the quality of the video both in subjective and objective metrics.

Overlapped Block Motion Compensation [25] is developed in order to avoid these blocking artifacts and increase the quality of the resulting frame in MC-FRUC. It is also used in video compression standards such as H.263 [26]. The main idea of OBMC is based on

determining the motion of each pixel in a block by considering the motion vector of the block itself and the motion vectors of its neighboring blocks.

A simple OBMC technique is implemented in [20]. It employs OBMC during the interpolation stage by enlarging every *NxN* block in the to-be-interpolated frame to *(N+2w) x (N+2w)* block which form overlapped areas of width *w* in every block as shown in Fig 5.1. The purpose of this operation is having a smooth transition between adjacent blocks. The pixels at the corners of an *NxN* block are located in the overlapped area of the 4 neighboring blocks. The intensities of these pixels are calculated by averaging the intensity values generated by the motion vectors of each respective block. The intensities of the pixels that are located at the side boundaries of the interpolated block are calculated by averaging the intensity values generated by the motion vectors of the interpolated block and the adjacent block. The remaining interpolation is done by only using the motion vector of the to-be-interpolated block.

For example, in Fig 5.1, OBMC is not applied to the pixels in *R1* region as these pixels belong to a single block. The pixels that are located in *R2* regions should be interpolated by taking motion vectors of both adjacent blocks into account, as these pixels belong to both blocks. The pixels in *R3* region are in the overlapped area of 4 neighboring blocks, therefore the interpolations of these pixels are performed by using 4 different motion vectors.

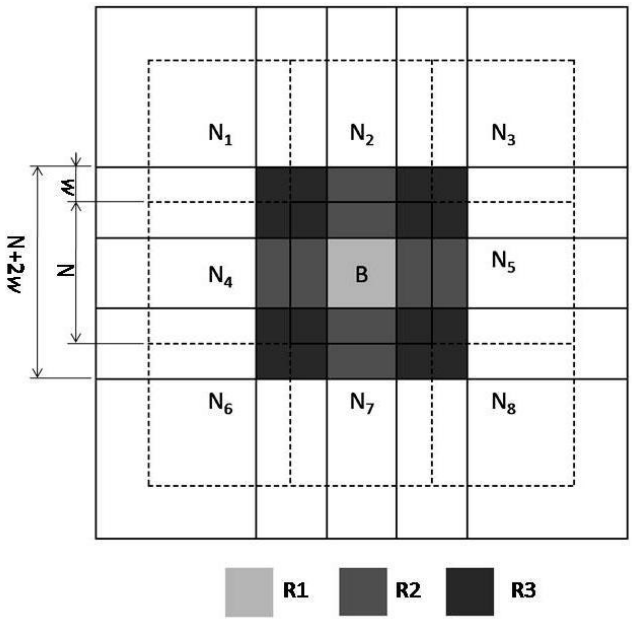


Figure 5.1: Overlapping Regions in OBMC

The interpolation of the block *B* is defined as in Equations (5.2), (5.3) and (5.4) where the neighboring blocks are *Ni= 1, 2… 8,* refers to the motion vector of the block *B* at position and denote the motion compensated field averaging for pixel at using motion vector *V* of block *B*.

1. For *R1*:

(5.2)

2. For *R2*:

–

where $N_i \in \{N_2, N_4, N_5, N_7\}$. (5.3)

3. For *R3*:

–

(5.4)

where S_k is the sum of the MC-FAVG results for the neighboring blocks overlapped with *B* in *R3* and defined as:

(5.5)

A sinusoidal OBMC algorithm is proposed in [20]. In sinusoidal OBMC algorithm, *NxN* blocks are enlarged to *2N x 2N* blocks as shown in Fig. 5.2. In the figure, the dashed lines show the enlarged versions of the blocks. Therefore, each pixel in an interpolated block is located in the overlapped area of the interpolated block and its 3 neighboring blocks. For example, the neighboring blocks N_1, N_2 and N_4 are used for interpolation of the first quarter of block B. The other 3 quarters of block B are interpolated similarly using the appropriate neighboring blocks.

In [20], the coefficients of the pixels are determined by a sinusoidal function. We used the following sinusoidal equation

(5.6)

where *c* denotes the coefficient of a pixel in the interpolated block and *x* is determined by the distance of the position of this pixel to the center of the interpolated block. As shown in Fig. 5.2, the coefficient of the pixel that is closest to the center of the interpolated block is given the largest weight. The coefficients of the pixels in the interpolated block are then reduced according to the sinusoidal function towards the boundaries of the interpolated block. In the figure, the coefficients of the pixels in the interpolated block are represented by a dashed line which is between 1 and 0.5, while the coefficients of the pixels in the enlarged neighboring block are represented by a straight line which is between 0 and 0.5.

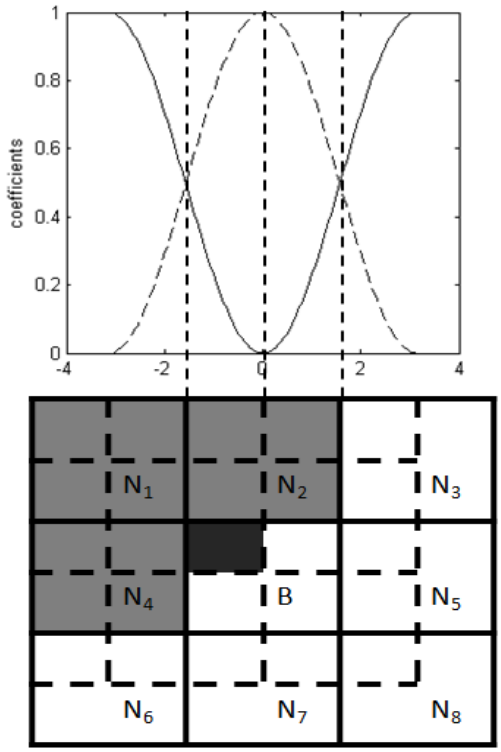


Figure 5.2: Block Overlapping for the First Quarter of Block B

Since there are 3 overlapping neighboring blocks for each quarter of the interpolated block, the coefficients are determined individually between the interpolated block and one of its neighboring blocks. For the vertical and horizontal neighbors, the coefficients are determined by the distances in x and y directions, and for the diagonal neighbor, the coefficients are calculated using the data obtained from horizontal and vertical neighbors.

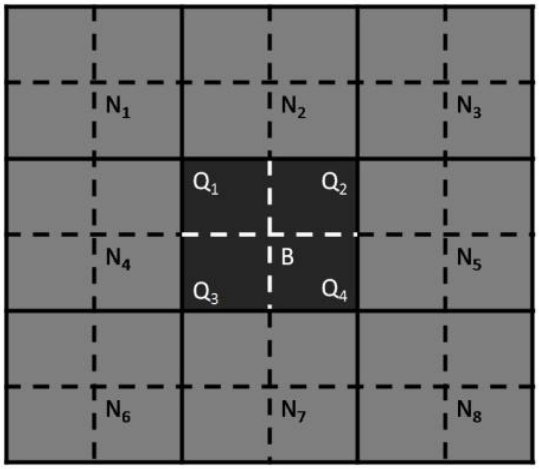


Figure 5.3: Overlapping Blocks for WC-OBMC

In [31], a novel OBMC algorithm, WC-OBMC, which has both lower computational complexity and higher PSNR results is proposed. In WC-OBMC algorithm, same as sinusoidal OBMC algorithm, *NxN* blocks are enlarged to *2N x 2N* blocks. Therefore, an interpolated block has 4 quarter areas, and each pixel in a quarter is located in the overlapped area of the interpolated block and its vertical, horizontal and diagonal neighboring blocks. After the MCFA results for the interpolated pixel are computed for each motion vector, the interpolated pixel is calculated by weighted averaging of these values.

Unlike sinusoidal OBMC algorithm, in WC-OBMC algorithm, the coefficients of the pixels in the interpolated block and in the enlarged neighboring blocks are constant for each block. Because of this, WC-OBMC has lower computational complexity and is easier to implement than sinusoidal OBMC. For each quarter, the weight of the interpolated block is determined as *5/8* and the weights of the neighboring blocks are determined as *1/8* in order to implement the division with shift operations.

The computations required for each pixel in block *B* in Fig. 5.3 are shown in (5.7), (5.8), (5.9) and (5.10).

For $\vec{p} \in Q1$:

– – (5.7)

For $\vec{p} \in Q2$:

– – (5.8)

For $\vec{p} \in Q3$:

– – (5.9)

For $\vec{p} \in Q4$:

– – (5.10)

The basic OBMC [20], sinusoidal OBMC [21], and WC-OBMC algorithms are implemented in C in [31]. For the basic OBMC algorithm *NxN* blocks are enlarged to *(N + N/2) x (N + N/2)* blocks, and the adaptive technique for determining the coefficients used in sinusoidal OBMC is not implemented. Table 5.1 shows the simulation results for three 1280x720 HD video sequences. The results show that basic OBMC algorithm performs better than MCFA with no OBMC algorithm. Sinusoidal OBMC algorithm performs better than basic OBMC algorithm. However, it has high computational complexity. WC-OBMC algorithm, in addition to its low computational complexity, produces better PSNR results than both the basic OBMC and sinusoidal OBMC algorithms.

	PSNR (dB)		
	NewMobCal	ParkJoy	SthlmPan
MCFA	33.69	24.19	34.15
Basic OBMC	33.80	24.45	34.23
Sinusoidal OBMC	33.80	24.48	34.24
WC-OBMC	33.87	24.60	34.29

Table 5.1: Simulation Results for OBMC Algorithms

A hardware architecture for WC-OBMC algorithm is also proposed in [31]. The block diagram of this WC-OBMC hardware is shown in Fig. 5.4. This hardware is composed of 3 BRAMs which store the motion vectors, the current frame and the reference frame, Accumulator, Frame Address Generator, Frame Border Detector, Motion Vector Address Generator, MV Splitter and Control Unit.

The current and reference frames are stored in off-chip SRAM. Since off-chip SRAM access has higher access time and consumes more power than BRAM access, 2 BRAMs are used to store 80x80 current frame pixels and 80x80 reference frame pixels for the interpolated block. The horizontal and vertical components of MVs for each block are assumed to be previously found by a ME algorithm and stored in a BRAM. The interpolated frame is

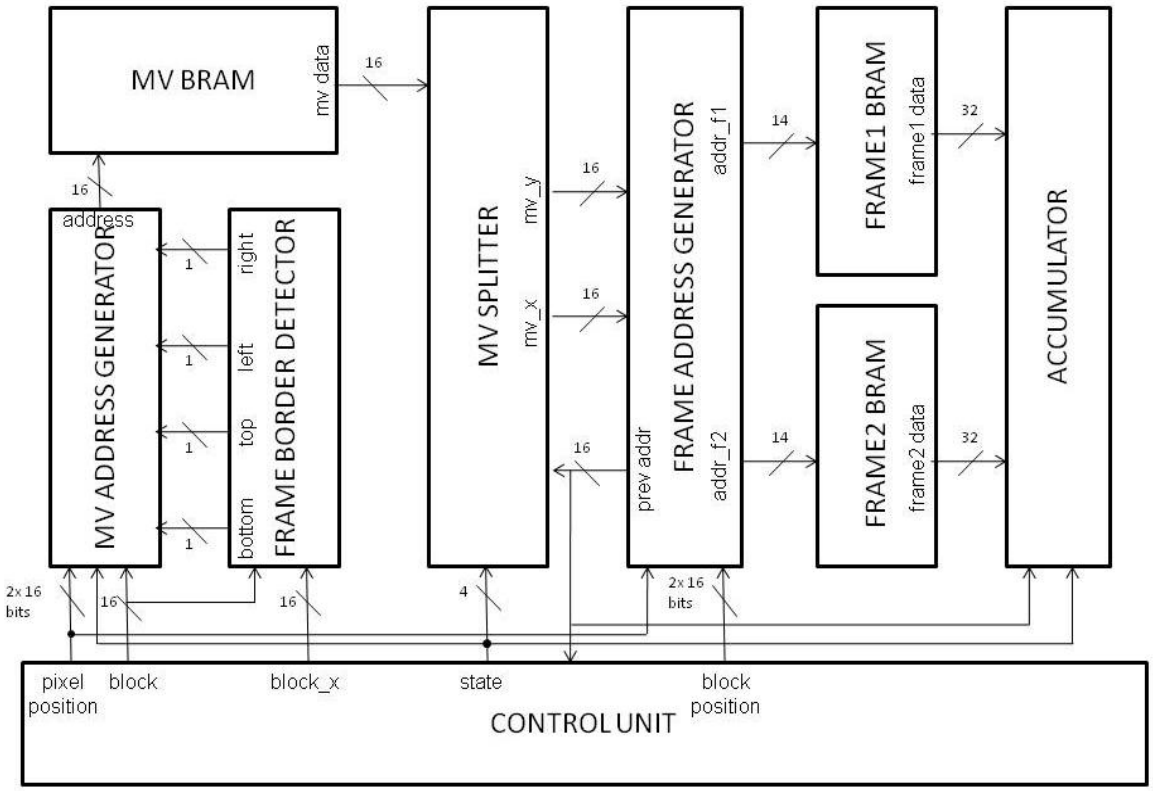


Figure 5.4: Top-level Block Diagram of WC-OBMC Hardware

generated by applying WC-OBMC algorithm to each pixel in all the blocks, and stored in off-chip SRAM. For each pixel, the corresponding motion vectors are accessed, and these motion vectors are used to determine the location of the pixels that will be used from the current and reference frames.

The motion vector address generator generates the addresses for the MVs of the interpolated block and its neighboring blocks. It decides which MVs will be accessed by using the block and pixel position provided by the control unit. Since some of the neighboring blocks do not exist at the borders of a frame, the inputs from the frame border detector are used to check if the neighbors of the interpolated block exist or not. If a neighboring block does not exist, the MV of the interpolated block is used instead of the MV of the neighboring block. The motion vector address generator module sends the address of the required MV to the BRAM containing the MV.

The MV read from the BRAM is split into its x and y components and these components are expanded to 16 bits by the motion vector splitter. The MV is stored in a buffer in the motion vector splitter, because it may be required for two cycles. The MV components are sent to the frame address generator and they are combined with the current

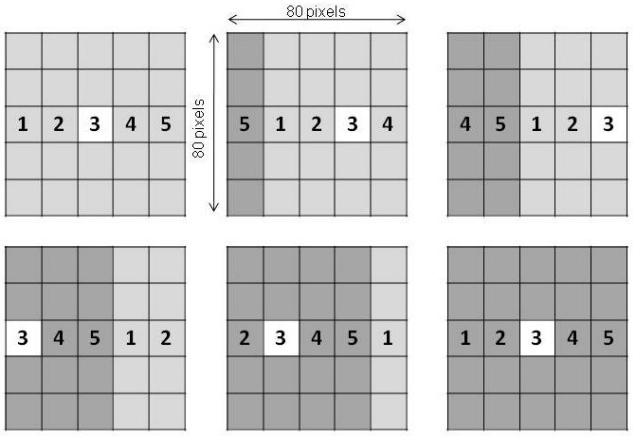


Figure 5.5: Data re-use

block and pixel position for calculating the addresses of the required pixels in both current and reference frames. The calculated addresses are sent to the frame1 and frame2 BRAMs.

The BRAMs for current and reference frames store 80x80 pixels each. Since [-64, +64] pixels search window is generally enough to obtain high quality results for FRUC and only half of the values of motion vectors are used for interpolation, 80x80 pixel BRAMs store all the pixels needed for a specific block. Since the off-chip SRAM is accessed by a 32 bit data bus and each pixel is 8 bits, in each cycle 4 pixels are read from SRAM and written to BRAM. Storing 80x80 pixels in the BRAMs allows large amount of data re-use. A BRAM consists of 5x5 blocks, 16x16 pixels each. The block that will be interpolated, shown as the white block in Fig. 5.5, is located at the center of the 80x80 pixels. After the interpolation of a block is finished, for the next block, only 5x1 blocks will be read from the off-chip SRAM and written to BRAMs in place of the blocks that are not needed. The 5x4 blocks (80x64 pixels) in the BRAMs will be re-used for the next block. The replaced blocks are shown as the dark grey blocks in Fig. 5.5. As an exception, all the blocks (80x80 pixels) will be replaced for the first blocks in each row of the interpolated frame.

Fig. 5.5 also shows the data layout in the 80x80 pixels BRAM and the block replacements during the interpolation of the consecutive blocks. The numbers in Fig. 5.5 represent the order of the columns in the original frames. As a result of this data re-use scheme, orders of the columns in the BRAMs are not always the same. Therefore, frame address generator maps the positions of the required pixels to the corresponding addresses of the BRAMs.

Since the required pixel positions in the BRAMs are determined with respect to position of the interpolated block in the BRAM, the interpolated block position is provided to the

frame address generator. The frame address generator calculates the horizontal and vertical positions of the required pixels using the position of the interpolated block in the BRAM and the corresponding MV components. Since the interpolated block is not always located at the center of the BRAM, in some cases the calculated horizontal position might point to a location where no data is available.

Therefore, the calculated position is checked to determine whether an out of bounds condition occurred or not. If an out of bounds occurs, the horizontal position is adjusted so that it maps to the physical address of the required pixel. For example, in the third step in Fig. 5.5, if the required pixel is in the column 4, the calculated position will point to the right of the interpolated block where there is no data available, while the required pixel is in the first column of the BRAM. Therefore, the horizontal position is adjusted in order to map it to the first column of the BRAM and the address of the required pixel is calculated using this adjusted horizontal position.

As it can be seen in Fig. 5.5, in the final step, the layout of the blocks in the BRAM returns to the initial state where the interpolated block is located at the center of the BRAM. This rotating data order allows using the same flow for each row of blocks in a frame.

Since 4 consecutive pixels are accessed in one cycle from a BRAM, data hit and data miss occur. Every address location contains 4 pixels. When the required 4 pixels are located in the same address location, data hit occurs as shown in Fig. 5.6. On the other hand, when the motion vector value does not allow accessing all required 4 pixels from a single address location, data miss occurs as shown in Fig. 5.7. In this case, one more clock cycle is required to access the next address line. A control signal indicating whether a data miss occurred or not

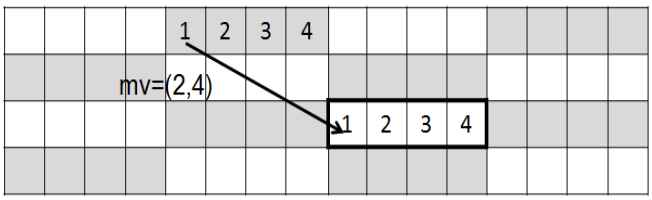


Figure 5.6: Data Hit

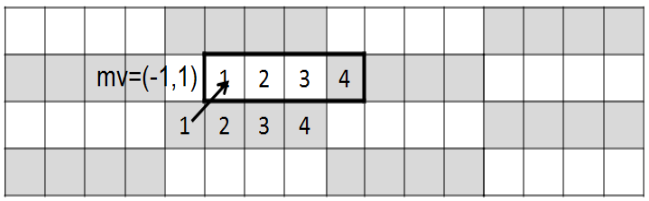


Figure 5.7: Data Miss

is generated by frame address generator, and sent to the relevant modules. Depending on this signal, the control unit changes the state machine dynamically in order to complete the data access and motion vector splitter decides whether to use the previously accessed MV or currently accessed MV. When data miss occurs, the addresses are calculated using the previous MV stored in motion vector splitter and sent to the BRAMs.

After the pixels are read from BRAMs, the accumulator splits each input from the BRAMs into 4 pixels. After the pixels are separated, the coefficients for multiplications are determined using the state information. The accumulator also checks for data miss using the frame address information, and if data miss occurs, it selects the requested input pixels and performs multiplication and accumulation only for these selected pixels. After all the required pixels are read and processed by the accumulator, the resulting pixel values are put into 32 bit words containing 4 8-bit pixels and sent to the output of the OBMC hardware. The OBMC hardware generates 4 pixels in at most 11 and at least 7 clock cycles depending on whether data hit or miss occurred.

The WC-OBMC hardware architecture is implemented in Verilog HDL. The Verilog RTL codes are synthesized and mapped to an 8 million gate FPGA implemented in 0.15 µm CMOS technology. The WC-OBMC hardware consumes 922 slices (1692 LUTs), which corresponds to 6% of the slices of the same FPGA. In addition, 12 BRAMs are used to store motion vectors, current and reference frame data.

Since 4 pixel output is generated between 7 and 11 clock cycles, the hardware requires 1,612,800 clock cycles in best case and 2,534,400 clock cycles in worst case for a 1280x720 HD frame. The hardware is simulated to generate first 5 odd frames of the ParkJoy sequence and the simulation results show that the hardware generates a 1280x720 HD frame in 2,266,525 clock cycles on average without considering memory loadings. Since FPGA implementation works at 61 MHz, it is capable of processing 26 1280x720 HD frames per second.

5.2 An Efficient Weighted Coefficient Overlapped Block Motion Compensation Hardware

In this thesis, we propose an efficient hardware architecture for the WC-OBMC algorithm by modifying the reference hardware architecture proposed in [31]. The proposed hardware avoids the redundant memory accesses and arithmetic operations done in the

reference hardware by using the prior information about the MVs of the interpolated block and its neighboring blocks. In the WC-OBMC algorithm, the positions of the required pixels for interpolation are determined using the MV of the interpolated block and the MVs of three neighboring blocks for each quarter of the interpolated block. The reference hardware uses all of these four MVs in a pre-defined order in the state machine with their corresponding coefficients. If any of these four MVs point to the same pixel position, the same pixel is accessed again and used for accumulation with the coefficient of its own MV. In the proposed hardware, this pre-defined processing order is replaced with an adaptive scheme. The MVs used for each quarter of the interpolated block are compared. If there are same MVs among the MVs used for a certain quarter, the pixel pointed by those same MVs is only accessed once and the coefficient of that pixel is adjusted in the accumulation stage. For example, if the MVs of the interpolated block and two of its neighboring blocks are identical, the coefficient corresponding to this MV is set to 7/8 while the other coefficient remains as 1/8, and only two memory accesses are done to the BRAMs which store the frames instead of four memory accesses. This avoids redundant memory accesses and arithmetic operations.

The block diagram of the proposed WC-OBMC hardware is shown in Fig. 5.8. The Motion Vector Address Generator and Motion Vector Splitter modules are removed from the reference hardware, and Motion Vector Buffer and Coefficient Decider modules are added. The motion vector buffer generates addresses for the MVs of the interpolated block and its neighboring blocks at the beginning of interpolation and stores the MVs in a buffer for further use. The motion vector buffer uses inputs from frame border detector which checks if the neighbors of the interpolated block exist or not. If a neighboring block does not exist, the MV of the interpolated block is used instead of the MV of the neighboring block. After all the required MVs are stored in the buffer, it compares the MVs of each quarter and sends the result of the comparisons to coefficient decider.

The coefficient decider determines how many distinct MVs exist for each quarter of the interpolated block and sends this information to the control unit. It generates two output vectors, the index of the MV in the motion vector buffer and the corresponding coefficient for each distinct MV, and sends them to the motion vector buffer. The control unit adaptively sets the number of states required for the interpolation of each quarter of the interpolated block. The motion vector buffer extends the x and y components of the distinct MVs to 16 bits and sends them to the frame address generator.

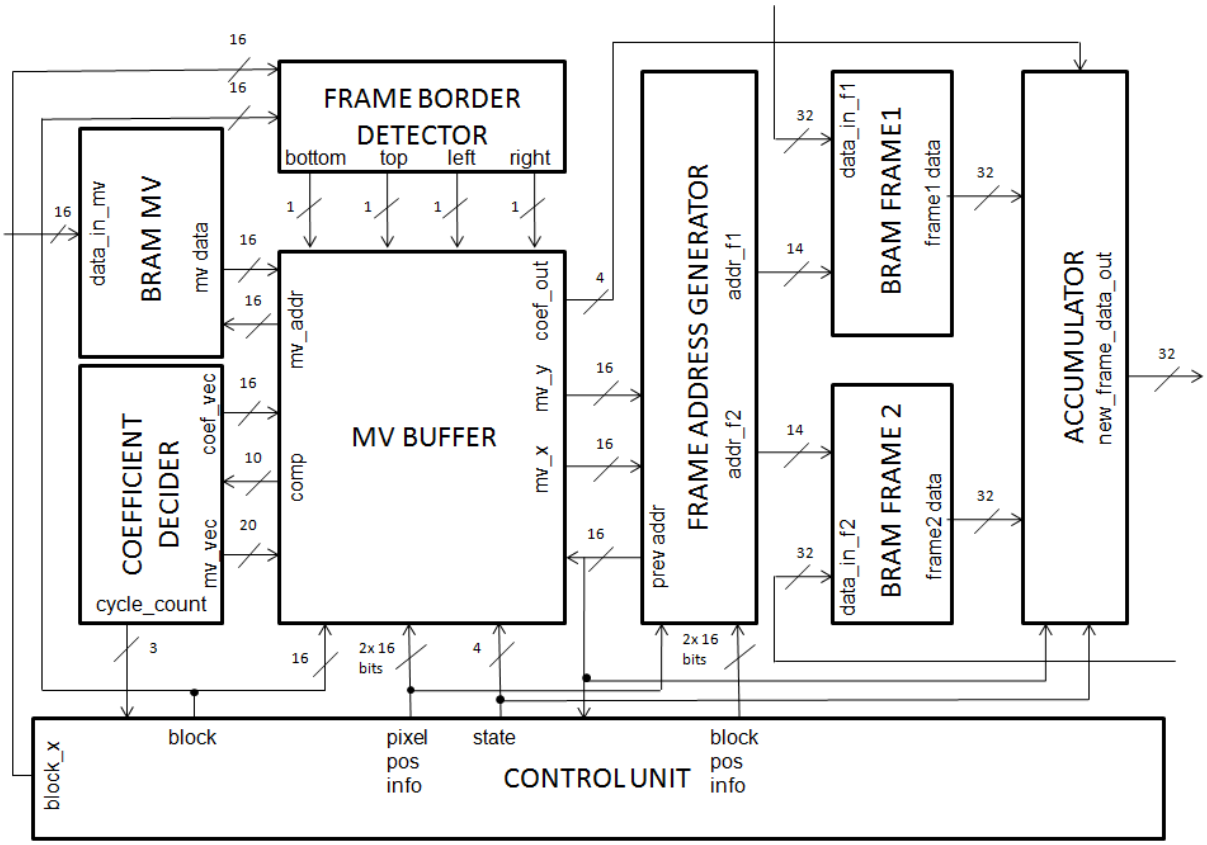


Figure 5.8: Top-level Block Diagram of Proposed WC-OBMC Hardware

The frame address generator calculates the addresses using the MV, current block and pixel position information. The calculated addresses are sent to the frame1 and frame2 BRAMs. A pipeline stage is implemented in frame address generator. This increases the clock frequency with an overhead of one clock cycle for each output generated for 4 pixels. Since the same data layout with the reference hardware is used, there is data re-use and data hit or miss occur. Therefore, the frame address generator sends a control signal to motion vector buffer and control unit to indicate whether a data miss occurred or not. When data miss occurs, the motion vector buffer provides the same MV for another clock cycle and the control unit adjusts the state machine to complete the data access.

After the pixels are read from BRAMs, the accumulator splits each input from the BRAMs into 4 pixels. The motion vector buffer provides the corresponding coefficients for the pixels to the accumulator. The accumulator checks whether data miss occurred or not using the frame address information, and if data miss occurred, it selects the requested input pixels and performs multiplication and accumulation only for these selected pixels. After all the required pixels are processed by the accumulator, the resulting pixel values are put into 32 bit words containing 4 8-bit pixels and sent to the output of the OBMC hardware.

The proposed hardware architecture is implemented in Verilog HDL. The Verilog RTL codes are synthesized and mapped to an 8 million gate FPGA implemented in 0.15 µm CMOS technology. The architecture occupies 998 slices (1875 LUTs), which corresponds to 7% of the same FPGA. In addition, 12 BRAMs are used to store motion vectors, current and reference frame data. The hardware is simulated to generate first 5 odd frames of the ParkJoy sequence and the simulation results show that the hardware generates a 1280x720 HD frame in 1,984,594 clock cycles on average without considering memory loadings. Since FPGA implementation works at 113 MHz, it is capable of processing 57 1280x720 HD frames per second.

The power consumptions of both WC-OBMC hardware implementations on the same FPGA are estimated at 50 MHz using a gate level power estimation tool. In order to estimate the power consumption of a WC-OBMC hardware, timing simulation of its placed and routed netlist is done. ParkJoy frame is used as input for timing simulations and the signal activities are stored in VCD files. These VCD files are used for estimating the power consumptions of WC-OBMC hardware implementations using this power estimation tool.

The performance, area and power consumption comparison of the WC-OBMC hardware implementations are shown in Table 5.2. The proposed hardware is faster than the reference hardware, but its area is larger than the reference hardware. The proposed hardware has 22% less power consumption than the reference hardware.

	Area	Maximum Clock Frequency (MHz)	Execution Time (clock cycles per frame)	Performance (1280x720 fps)	Power Consumption (mW)
Reference Hardware	922 (Slices) 1692 (LUTs)	61	2,266,525	26	108.22
Efficient Hardware	998 (Slices) 1875 (LUTs)	113	1,984,594	57	84.09

Table 5.2: Comparison of Hardware Architectures

Chapter 6

ADAPTIVE BILATERAL MOTION ESTIMATION HARDWARE IMPLEMENTATION ON AN FPGA BOARD

6.1 XILINX ML605 FPGA Board

In this thesis, the ABIME hardware proposed in [30] is implemented on a ML605 FPGA board which is a state-of-the-art Xilinx board. As seen in Fig 6.1, the board is composed of a Virtex 6 XC6VLX240T FPGA, 512 MB DDR RAM and 32 MB Flash memory and interfaces such as UART and DVI. The board also has MicroBlaze support as shown in Fig 6.2.

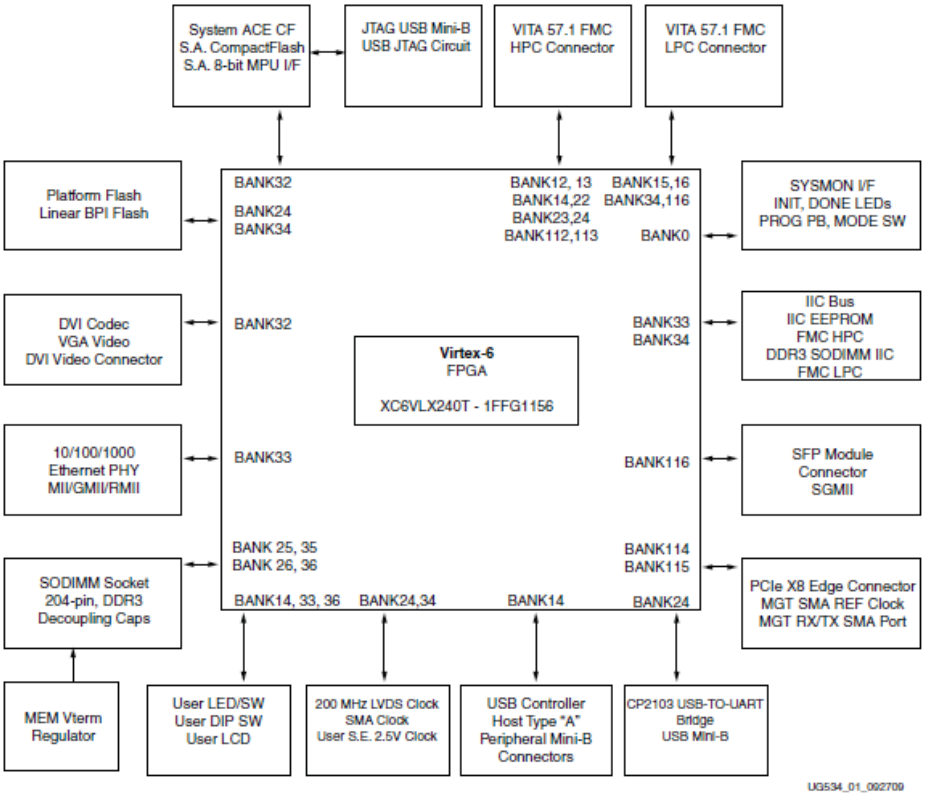


Figure 6.1: Block Diagram of Xilinx ML605 FPGA Board

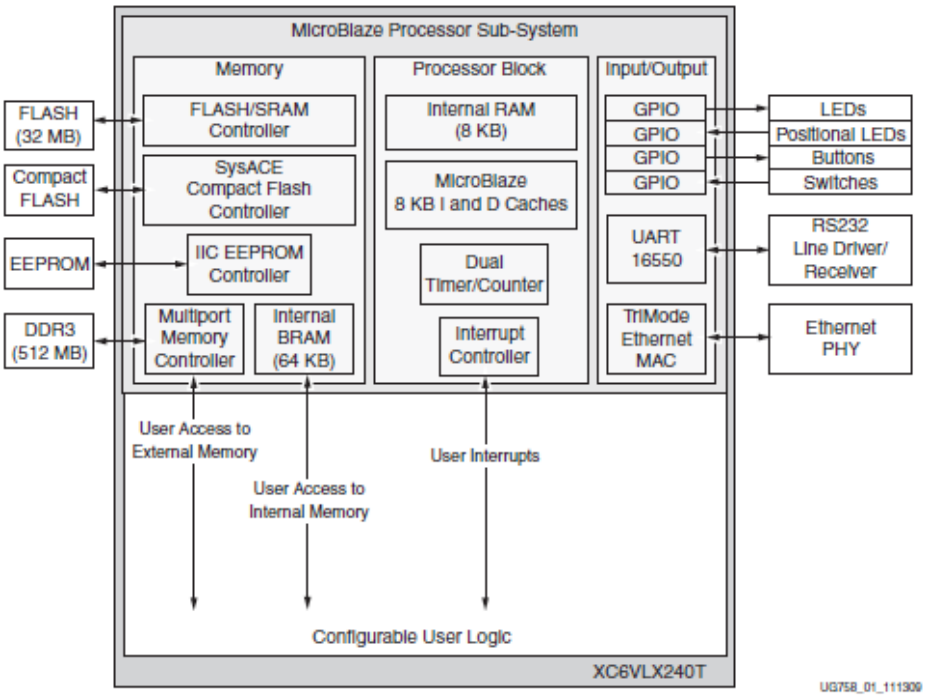


Figure 6.2: Block Diagram of MicroBlaze Processor Sub-System

6.2 Adaptive Bilateral Motion Estimation Hardware Implementation

Since the board was not previously used for any hardware implementation, we first implemented a basic image processing application, contrast enhancement hardware, in order to observe the process of the hardware implementation on the board. Using the experience gained from this preliminary work, a method for implementing the ABIME hardware is proposed.

A software running on MicroBlaze processor is developed to transfer the inputs of the ABIME hardware from a host computer in an appropriate order and to gather the outputs of the hardware for sending them back to the host computer and displaying the resulting frame on a monitor. The ABIME hardware is added as a peripheral to a bus where the MicroBlaze processor is the master. For this purpose the ABIME hardware is modified to be a slave peripheral for this data bus and 4 software accessible registers are added to the hardware. 2 of these registers are used by the software running on MicroBlaze for writing the inputs to the hardware and the other 2 are used for gathering the outputs and the status information from the hardware.

The software gets 2 input frames and the MVs found by 3DRS ME algorithm from the host computer using the UART interface and writes them to a DDR RAM. Then, it loads the

BRAMs of the ABIME hardware with the pixels in two BSWs. After the ABIME hardware generates the done signal, the software reads the MV updated by the ABIME hardware and writes it to the DDR RAM. This process is repeated for all the MBs. After all the MBs are processed, the software runs an interpolation algorithm with the updated MVs and writes the interpolated frame to the DDR RAM. Finally, the interpolated frame is displayed on a monitor using the DVI interface of the FPGA board as shown in Fig. 6.3.

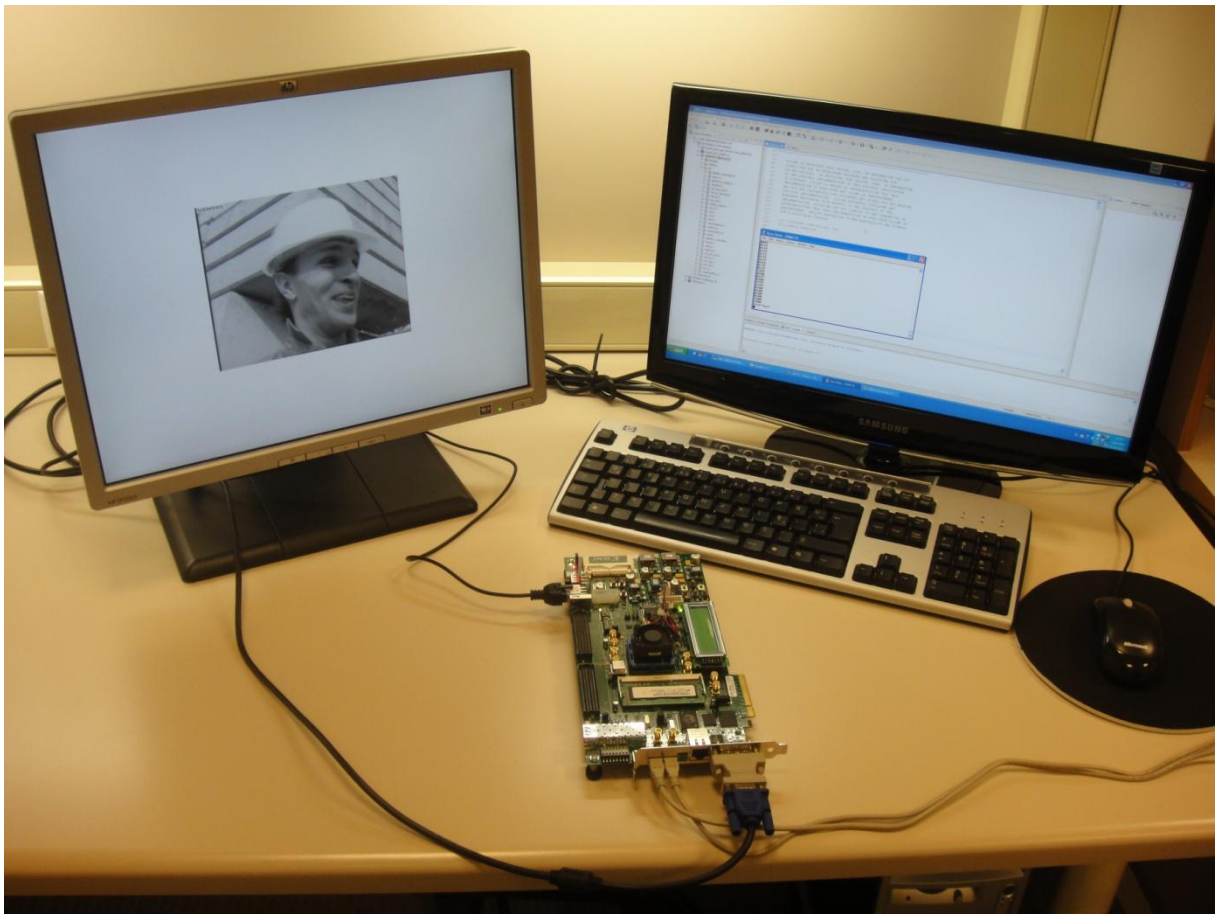


Figure 6.3: Board Implementation of ABIME Hardware

Chapter 7

CONCLUSION AND FUTURE WORK

In this thesis, we first proposed a perfect absolute difference prediction (P-ADP) method based on the comparison prediction technique proposed in [27]. P-ADP method corrects the incorrect predictions in the same clock cycle. Since the clock period of the ME hardware is long enough to perform two subtractions in one clock cycle, if the prediction is incorrect, the correct subtraction is done by changing the select inputs of the multiplexers in the same clock cycle. The proposed technique reduces the average dynamic power consumption of the 256 processing element (PE) fixed block size ME hardware proposed in [28] by 2.2% without any PSNR loss on a XC2VP30-7 FPGA.

Then, we proposed a Global Motion Estimation (GME) algorithm and its hardware implementation. In order to improve the performance of the 3DRS algorithm [29], a method for including the effect of camera motions which are independent from the movements of the objects is developed. The proposed GME algorithm finds a global motion vector (GMV) for a frame by performing ME using FS algorithm on a set of blocks in this frame, and it uses this GMV as an additional candidate MV for 3DRS ME. The proposed algorithm results in a 2.5% PSNR increase on average. We also proposed an Adaptive GME algorithm (AGME) and its hardware architecture. The adaptive technique checks the validity of the GMV and the presence of global motions by using the statistics of selected MVs during ME with 3DRS and if GME is not necessary it is disabled adaptively. The proposed adaptive technique reduces the energy consumption of the GME hardware by 14.37% with a PSNR loss of 0.17% on average on a XC6VLX75T FPGA. The proposed hardware is capable of processing 341 720p frames per second.

In addition, we proposed an early termination technique for the adaptive bilateral motion estimation (ABIME) implementation proposed in [30]. The proposed early terminated ABIME (ET-ABIME) algorithm exploits the spiral search pattern to adaptively change the

size of Bilateral Search Window (BSW) of a MB based on the success of BIME vector refinement process for the neighboring spatial and temporal MBs. The proposed technique reduces the energy consumption of the ABIME hardware by 29% with a PSNR loss of 0.04% on a XC6VLX75T FPGA.

We also proposed an efficient WC-OBMC hardware based on the reference hardware implemented in [31]. The proposed hardware reduces redundant operations done in the reference hardware by using the prior information about the MVs of the interpolated block and its neighboring blocks and adaptively changes the processing flow based on this data. The proposed hardware also uses a pipelining technique to increase the throughput of the reference hardware. The proposed hardware reduces the dynamic power consumption of the reference hardware by 22% and it is capable of processing 57 720p frames per second.

Finally, we implemented the ABIME hardware proposed in [30] on a ML605 FPGA board which is a state-of-the-art Xilinx board. In the FPGA implementation, we used the ABIME hardware as a slave peripheral and MicroBlaze processor as a master. We also implemented a software running on MicroBlaze processor. Using this software, inputs are transferred to the hardware from a host computer, the outputs of the hardware are sent to the host computer and also displayed on a monitor.

As future work, a complete FRUC hardware can be developed by integrating the proposed hardware architectures, and it can be implemented on the ML605 FPGA board.

BIBLIOGRAPHY

[1] G. de Haan, *Video Processing for Multimedia Systems*. Univ. Press Eindhoven, ISBN 90-9014015-8, 2001.

[2] O. A. Ojo and G. de Haan, "Robust Motion-Compensated Video Upconversion," *IEEE Trans. Consum. Electron.*, vol. 43, no. 4, pp. 1045-1056, Nov. 1997.

[3] M. Tekalp, *Digital video processing.* Englewood Cliffs, NJ: Prentice Hall, 1995.

[4] N. Netravali and J. D. Robbins, "Motion-adaptive interpolation of television frames," *Proc. Picture Coding Symp.*, p. 115, June 1981.

[5] K. A. Bugwadia, E. D. Petajan, and N. N. Puri, "Progressive-Scan Rate Up-Conversion of 24/30 Source Materials for HDTV," *IEEE Trans. Consum. Electron.*, vol. 42, no.3, pp. 312-321, Aug. 1996.

[6] K. A. Bugwadia, E. D. Petajan, N. N. Puri, "Progressive-Scan Rate Up-Conversion of 24/30 Hz Source Materials for HDTV," *IEEE Trans. Consum. Electron.*, vol. 42, no. 3, pp. 312-321, Aug. 1996.

[7] T. Koga, K.Iinuma, and T. Ishiguro, "Motion compensated interframe coding for video conferencing," *Proc. NTC*, pp. G5.3.1-G5.3.5, Dec. 1981.

[8] A. Puri, H. M. Hang, and D. L. Schilling, "An efficient block matching algorithm for motion compensated coding," *Proc. IEEE ICASSP*, pp. 1063-1066, Apr. 1987.

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

[10] L. M. Po and W. C. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no.3, pp. 313–317, June 1996.

[11] 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 Trans. Circuits Syst. Video Technol*., vol. 3, no. 5, pp. 368-379, Oct. 1993.

[12] F. Dufaux and F. Moscheni, "Motion Estimation Techniques for Digital TV: A Review and a New Contribution," *Proceedings of the IEEE*, vol. 83, no. 6, pp. 858–876, 1995.

[13] M. T. Orchard, "Predictive Motion-Field Segmentation for Image Sequence Coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 3, no. 1, pp. 54–70, 1993.

[14] V. Seferidis and M. Ghanbari, "Generalized Block-Matching Motion Estimation Using Quad-Tree Structured Spatial Decomposition," *IEEE Proc.-Vis. Image Signal Process*, vol. 141, no. 6, pp. 446–452, 1994.

[15] Y-K. Chen and S.Y. Kung, "Rate optimization by true motion estimation," *Proc. of IEEE Workshop on Multimedia Signal Processing*, pp. 187-194, June 1997.

[16] B.-W. Jeon, G.-I. Lee, S.-H. Lee, and R.-H. Park, "Coarse-to-fine frame interpolation for frame rate up-conversion using pyramid structure," *IEEE Trans. Consum. Electron.*, vol. 49, no.3, pp. 499-508, Aug. 2003.

[17] T.Y. Kuo and C.-C.J. Kuo, "Motion-compensated interpolation for low-bit-rate video quality enhancement," *SPIE Visual Communications and Image Processing*, vol. 3460, pp. 277-288, July 1998.

[18] A. Kaup and T. Aach, "Efficient prediction of uncovered background in interframe coding using spatial extrapolation," *ICASSP*, vol. 5, pp. 501- 504, 1994.

[19] R. J. Schutten and G. D. Haan, "Real-time 2–3 pull-down elimination applying motion estimation/compensation in a programmable device," *IEEE Trans. Consum.Electron.*, vol. 44, no. 3, pp. 501–504, Aug. 1998.

[20] B-T. Choi, S-H. Lee, and S-J. Ko, "New frame rate up-conversion using bi-directional motion estimation," *IEEE Trans. Consum. Electron.*, vol.46, no.3, pp. 603-609, Aug. 2000.

[21] B-D. Choi, J-W. Han, C-S. Kim, and S-J. Ko, "Motion-compensated frame interpolation using bilateral motion estimation and adaptive overlapped block motion compensation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 4, pp. 407-416, Apr. 2007.

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

[23] 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 Trans. Consum. Electron.*, vol. 54, no.4, pp. 1830-1838, Nov. 2008.

[24] S.-C. Tai, Y.-R. Chen, Z.-B. Huang, C.-C. Wang, "A Multi-Pass True Motion Estimation Scheme With Motion Vector Propagation for Frame Rate Up-Conversion Applications", IEEE/OSA J. Disp. Technol., vol. 4, no. 2, pp. 188-197, June 2008.

[25] M. Orchard and G. Sullivan, "Overlapped block motion compensation: An estimation-theoretic approach", *IEEE Trans. Image Processing*, vol. 3, pp. 693–699, May 1994.

[26] ITU-T, Draft ITU-T Recommendation H.263, "Video Coding for Low Bit Rate Communication," 1997.

[27] A. Akın, O. C. Ulusel, T. Z. Ozcan, G. Sayılar, I. Hamzaoglu, "A Novel Power Reduction Technique for Block Matching Motion Estimation Hardware", *Int. Conference on Field Programmable Logic and Applications*, Sep. 2011.

[28] C. Kalaycioglu, O. C. Ulusel, and I. Hamzaoglu, "Low Power Techniques for Motion Estimation Hardware", *Int. Conference on Field Programmable Logic and Applications*, Aug. 2009.

[29] G. de Haan, "Progress in motion estimation for consumer video format conversion," *IEEE Trans. Consum. Electron.*, vol. 46, no. 3, pp. 449-459, Aug. 2000.

[30] A. Akin, M. Cetin, B. Erbagci, O. Karakaya, I. Hamzaoglu, "An Adaptive Bilateral Motion Estimation Algorithm and its Hardware Architecture", *IEEE/IFIP International Conference on VLSI and System-on-Chip*, Sep. 2010.

[31] T. Z. Özcan, Ç. Çakır, M. Çetin, I. Hamzaoglu, "An Overlapped Block Motion Compensation Hardware for Frame Rate Conversion", *Euromicro Conference on Digital System Design*, Sep. 2011.

[32] W. M. Chao, C. W. Hsu, Y. C. Chang, and L. G. Chen, "A Novel Motion Estimator Supporting Diamond Search and Fast Full Search," *IEEE ISCAS*, May 2002.

[33] 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 Trans. on Consumer Electronics*, vol. 55, no. 3, Aug. 2009.