UNIVERSITY OF BAHRAIN

College of Information Technology



High Speed On-Chip Motion Estimation Co-Processor for H.265/HEVC Standard

A Thesis Submitted in Partial Fulfillment of the Requirements for the Master Degree in Information Technology

Submitted by

Mohamed Nabil Hammad

20082351

Supervised by

Dr. Yasser Ali Ismail

(Assistant Professor)

University of Bahrain

Kingdom of Bahrain

April, 2016

DECLARATION

I declare that this work is the result of my own investigation and that it has not already been accepted in substance for any degree, nor is it currently submitted for any degree.

Signed: ---------- Date: 6-4-2016

Mohamed Nabil Hammad (Candidate)

The Thesis defense committee considers the Thesis titled:

High Speed On-Chip Motion Estimation Co-Processor for H.265/HEVC Standard

Submitted by Mohamed Nabil Hammad to the College of Information Technology is satisfactory and acceptable for the Master of Science in Information Technology.

Thesis Defense Committee:

Yasser Ismail

Dr. Yasser Ali Ismail
(Supervisor)

Department of Computer Engineering
University of Bahrain

Prof. Abbes Amira
(Member)

Qatar University
Qatar

Dr. Wael Al-Medany
(Member)

Department of Computer Engineering
University of Bahrain

ABSTRACT

H.265/HEVC or High Efficient Video Coding has accomplished up to 50% savings in the transmitted bit-rate compared to the state of the art H.264/AVC video coding standard. As a result, 4K and Ultra High Definition TV (UHD-TV) resolutions can be achieved. Achieving such high bit rate reduction requires removing of both spatial and temporal redundancies. In this thesis we are concentrating on the removal of the temporal redundancies since they are much more effective than the spatial ones. Motion Estimation (ME) process is used as a temporal redundancies removal tool. Since it consumes up to 84% of the total entire encoding time, ME is consider as the most exhaustive processing unit in the H.265/HEVC encoder. There are two main problems when implementing the hardware of the ME unit in the H.265/HEVC standard: the huge number of data required for performing the ME process and the high computational complexity compared to the previous video coding standards. In this thesis, two architectures were implemented to solve the two previous problems. The first architecture solves the issue of the large number of pixels needed from the external memory in the ME process which using some previous work (Ismail, El-Medany, Al-Junaid, & Abdelgawad, 2014). This is achieved by using a smart data flow that efficiently use level A and level B data reuse principle. Data reuse principle is mainly used to reduce the I/O memory bandwidth. Consequently, the overall time required to fetch the data from the external memory is greatly reduced. The implemented ME co-processor in the first architecture can perform the ME process at 9 frame per second using 4K (3840×2160) video resolution at 545 MHz. The second architecture which is using previous software verified ASWS algorithm (Goel, Ismail,

& Bayoumi, 2005) provides reductions in the computations of the ME process. This is achieved by using a prediction and an Adaptive Search Window Size (ASWS) algorithm. The second architecture can perform the ME process at 10 frame per second using 4K (3840×2160) video resolution at 527 MHz with 100% unitization of all hardware components.

DEDICATION

To my sweetheart, Sally

To my son, Yassin

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Dr. Yasser Ali Ismail for his great support, help, and motivation. His guidance helped me in all the time of my research and during the writing of this thesis.

I would like to thank my wife Sally for her love. Her support was endless. I would like also to thank my parents, Nabil Hammad and Abeer Mohamed. I would also dedicate this thesis to my son Yassin.

TABLE OF CONTENTS

Topic	Page Number
APPROVAL	i
ABSTRACT	ii
DEDICATION	iv
ACKNOWLEDGMENTS	v
TABLE OF CONTENTS	vi
LIST OF TABLES	ix
LIST OF FIGURES	x
LIST OF ABBREVIATIONS	xv
Chapter 1 INTRODUCTION	
1. Introduction	2
1.1 Research Problem	3
1.2 Research Goals	6
1.3 Research Area	7
1.4 Contribution	7
1.5 Summary	8
Chapter 2 THEORETICAL FRAMEWORK AND LITERATURE REVIEW	
2.1 Theoretical Framework	10
2.2 Literature Review	12
2.3 Summary	15
Chapter 3 VIDEO CODING CONCEPT	

3.1 Introduction	18
3.2 Video Coding Process	20
3.3 Temporal Model	21
3.4 Video Coding Layer	23
3.4.1 Pictures, Frames, and Fields	23
3.4.2 YC_bC_r Color Space and 4:2:0 Sampling	24
3.5 Motion Estimation and Video Compression Algorithms	25
3.5.1 New Three Step Search (NTSS)	26
3.5.2 Four Step Search (FSS)	27
3.5.3 Diamond Search (DS)	28
3.5.4 Cross Diamond Search (CDS)	29
3.6 Summary	30
Chapter 4 **H.265/HEVC HIGH EFFICIENT VIDEO CODING STANDARD**	
4.1 Introduction	33
4.2 H.264/AVC Video CODEC	35
4.3 Main Motivations of H.265/HEVC Standard	39
4.4 H.265/HEVC CODEC and Features	40
4.5 Summary	44
Chapter 5 **MOTION ESTIMATION CO-PROCESSOR ARCHITECTURE-1**	
5.1 Introduction	46
5.2 Data Reuse Principle	46

5.2.1 Architecture-1: An Overview	49
5.2.2 Architecture Components	51
5.2.3 Data Flow and Process Description	59
5.3 Simulation and Implementation Results	65
5.3.1 VHDL Verification	65
5.3.2 FPGA Implementation	86
5.4 Summary	87
Chapter 6 **MOTION ESTIMATION CO-PROCESSOR ARCHITECTURE-2**	
6.1 Introduction	90
6.1.1 Principle of Prediction Technique	91
6.2 Adaptive Search Window Size Technique (ASWS)	92
6.2.1 Architecture-2: An Overview	94
6.2.2 Architecture Components	95
6.2.3 Challenges and Motivation of Architecture-2	97
6.3 Simulation and Implementation Results	98
6.3.1 VHDL Verification	98
6.3.2 FPGA Implementation	112
6.4 Summary	114
Chapter 7 **CONCLUSION AND RECOMMENDATIONS**	
Conclusion And Recommendations	116
References	119
Abstract (in Arabic)	

LIST OF TABLES

No	Topic	Page Number
1	Table 1.1: Different formats for video transmission (P. & Marikkannan, 2014)	5
2	Table 5.1: Control signals	77
	Table 5.2: Zynq-7000 XC7Z045 FFG900 – 2 key features	90
3	Table 5.3: A summary of implementations for first architecture results using FPGA	92
4	Table 5.4: Calculated frame rate for different formats for video transmission	92
5	Table 6.1: A summary of implementations results for second architecture using FPGA	113
6	Table 6.2: Calculated frame rate for different formats for video transmission	114
7	Table 7.1: Hardware complexity comparison for the two implemented architectures	118

LIST OF FIGURES

No	Topic	Page Number
1	Figure 1.1: Time complexity of the H.265/HEVC encoder (Grellert et al., 2013; Kim et al., 2012)	3
2	Figure 1.2: Full Search Block Matching Motion Estimation (FSBM-ME) algorithm	4
3	Figure 2.1: Adaptive Search Window Size (ASWS) algorithm	11
4	Figure 3.1: Snap-shot of two consecutive frames showing both the spatial and temporal redundancies. (a) Frame n-1. (b) Frame n	19
5	Figure 3.2: Video CODEC block diagram.(Ismail, 2010; Ismail, McNeely, Shaaban, Mahmoud, & Bayoumi, 2012)	21
6	Figure 3.3: Two consecutive frames of the mobile video sequence and their difference. (a) Frame #0. (b) Frame #1. (c) Deference between two consecutive frames	22
7	Figure 3.4: Progressive and interlaced frames and fields (Ismail, 2010)	24
8	Figure 3.5: 4:2:0 sampling patterns (progressive) (Ismail, 2010)	25
9	Figure 3.6: The New Three Step Search algorithm (NTSS). (a) An example of the NTSS algorithm. (b) The flow chart of the NTSS algorithm.(Ismail, 2010; Renxiang et al., 1994)	27
10	Figure 3.7: The Four Step Search Algorithm (Ismail, 2010; Lai-Man & Wing-Chung, 1996)	28
11	Figure 3.8: The Diamond Search algorithm. (a) SDSP and LDSP patterns. (b) An illustrative example for the DS algorithm (Ismail, 2010)	29
12	Figure 3.9: Cross Diamond Pattern. (a) The HDSP. (b) VDSP patterns (Chun-Ho & Lai-Man, 2002)	30
13	Figure 4.1: Existing video coding standards	34
14	Figure 4.2: H.264/AVC Encoder (Wiegand et al., 2003)	39

15	Figure 4.3: Coding Tree Unit (CTU) of H.265/HEVC (Ohm et al., 2012)	41
16	Figure 4.4: Modes and directional orientations for intra-prediction (Ohm et al., 2012)	42
17	Figure 4.5: H.265/HEVC Encoder (Ohm et al., 2012)	43
18	Figure 5.1: Level A and level B data reuse (Ismail et al., 2014)	47
19	Figure 5.2: Level C data reuse (Ismail et al., 2014)	48
20	Figure 5.3: Level D data reuse (Ismail et al., 2014)	48
21	Figure 5.4: Top level architecture of the first implemented ME co-processor	50
22	Figure 5.5: The Processing Element (PE) (Ismail et al., 2014)	51
23	Figure 5.6: Dataflow of the search area data in the PE array (Ismail et al., 2014)	53
24	Figure 5.7: The architecture of the local memory	54
25	Figure 5.8: The search area needed for a 32×32 current block	55
26	Figure 5.9: The sub-memory architecture	55
27	Figure 5.10: 4-2 compressor (Ismail et al., 2014)	56
28	Figure 5.11: Four 4-bit operand adder tree (Ismail et al., 2014)	57
29	Figure 5.12: An illustration of the adder tree used in the implemented co-processor	57
30	Figure 5.13: Control unit	59
31	Figure 5.14: Loading the Current Block Register (CBR) in the PE array	59
32	Figure 5.15: Loading the search area in the left 32 x 32 columns of the local memory	60
33	Figure 5.16: Loading the search area on the right 32×32 columns of the local memory and start loading the most left column into the CBR	61
34	Figure 5.17: Loading the last raw in the right hand side of the local memory and loading the last column of the left hand side of the local memory into the PE array	61
35	Figure 5.18: Loading new row search area into the left	63

	32×32 columns of the local memory and start loading the right 32×32 columns into the PE array	
36	Figure 5.19: Achieving level B data reuse for the right 32×32 columns of the local memory	63
37	Figure 5.20: Loading new row search area into the left 32×32 columns of the local memory and start loading the right 32×32 columns into the PE array	64
38	Figure 5.21: Simulation of PE	66
39	Figure 5.22: Simulation of PE Row	67
40	Figure 5.23: First Simulation of PE Array at 32th Clock Cycle	69
41	Figure 5.24: Simulation of SAD unit	70
42	Figure 5.25: Simulation of Compare Unit	71
43	Figure 5.26: First reference frame with white background and 32×32 black block in position 20H	72
44	Figure 5.27: Simulation of complete ME process, case 1	72
45	Figure 5.28: First reference frame with white background and 32×32 black block in position 3DH	73
46	Figure 5.29: Simulation of complete ME process, case 2	74
47	Figure 5.30: Reference frame (stefan.yuv frame number 0)	75
48	Figure 5.31: Current frame (stefan.yuv frame number 1)	76
49	Figure 5.32: Reference search area	76
50	Figure 5.33: Current block	77
51	Figure 5.34: Simulation results of ME process	77
52	Figure 5.35: Comparing the result of the implemented architecture. (a) Current block. (b) Best match in the search area	78
53	Figure 5.36: Reference frame (stefan.yuv frame number 0)	79
54	Figure 5.37: Current frame (stefan.yuv frame number 1)	79
55	Figure 5.38: Simulation results of ME process	80
56	Figure 5.39: Comparing the result of the implemented architecture. (a) Current block. (b) Best match in the search area	80
57	Figure 5.40: Reference frame (ducks_take_off.yuv frame number 0)	81
58	Figure 5.41: Current frame (ducks_take_off.yuv frame	82

	number 1)	
59	Figure 5.42: Simulation results of ME process	82
60	Figure 5.43: Comparing the result of the implemented architecture. (a) Current block. (b) Best match in the search area	83
61	Figure 5.44: Reference frame (ducks_take_off.yuv frame number 0)	84
62	Figure 5.45: Current frame (ducks_take_off.yuv frame number 1)	84
63	Figure 5.46: Simulation results of ME process	85
64	Figure 5.47: Comparing the result of the implemented architecture. (a) Current block. (b) Best match in the search area	85
65	Figure 5.48: Reference frame (News_ProRes.MOV frame number 0)	86
66	Figure 5.49: Current frame (News_ProRes.MOV frame number 1)	87
67	Figure 5.50: Simulation results of ME process	88
68	Figure 5.51: Comparing the result of the implemented architecture. (a) Current block. (b) Best match in the search area	89
69	Figure 5.52: Device utilization summary using ISE Design Suite 14.5 tool	89
70	Figure 5.53: Device type using ISE Design Suite 14.5 tool	90
71	Figure 6.1: Four current blocks	92
72	Figure 6.2: Top level architecture of the second implemented ME co-processor using prediction and ASWS techniques	95
73	Figure 6.3: AMV Memory used with Prediction unit	96
74	Figure 6.4: Adaptive Search Window Size (ASWS) unit combined with the PU	97
75	Figure 6.5: The parameters which affect the performance of any video coder	97
76	Figure 6.6: Sample frame of size 720×486 for AVM memory	99
77	Figure 6.7: Simulation of AMV memory	99

78	Figure 6.8: ASWS simulation	100
79	Figure 6.9: Reference frame (stefan.yuv frame number 0)	102
80	Figure 6.10: Current frame (stefan.yuv frame number 1)	102
81	Figure 6.11: Reference search area	103
82	Figure 6.12: Current Block	103
83	Figure 6.13: Simulation results of ME process	104
84	Figure 6.14: Comparing the result of the implemented architecture. (a) Current block. (b) Best match in the search area	104
85	Figure 6.15: Reference frame (stefan.yuv frame number 0)	105
86	Figure 6.16: Current frame (stefan.yuv frame number 1)	106
87	Figure 6.17: Simulation results of ME process	106
88	Figure 6.18: Comparing the result of the implemented architecture. (a) Current block. (b) Best match in the search area	107
89	Figure 6.19: Reference frame (ducks_take_off.yuv frame number 0)	108
90	Figure 6.20: Current frame (ducks_take_off.yuv frame number 1)	108
91	Figure 6.21: Simulation results of ME process	109
92	Figure 6.22: Comparing the result of the implemented architecture. (a) Current block. (b) Best match in the search area	109
93	Figure 6.23: Reference frame (ducks_take_off.yuv frame number 0)	110
94	Figure 6.24: Current frame (ducks_take_off.yuv frame number 1)	111
95	Figure 6.25: Simulation results of ME process	111
96	Figure 6.26: Comparing the result of the implemented architecture. (a) Current block. (b) Best match in the search area	112
97	Figure 6.27: Device utilization summary using ISE Design Suite 14.5 tool	113

LIST OF ABBREVIATIONS

AD	Absolute Difference
ADR	Absolute Difference Register
AMV	Actual Motion Vector
ASWS	Adaptive Search Window Size
AVC	Advanced Video Coding
BB-ME	Block Based Motion Estimation
BW	BandWidth
CABAC	Context Adaptive Binary Arithmatic Coding
CC	Computation Complexity
CDS	Cross Diamond Search
CB	Current Block
CBR	Current Block Register
CU	Coding Unit
CODEC	enCOder/DECoder
CTU	Coding Tree Unit
DS	Diamond Search
DSWS	Dynamic Search Window Size
DWS	Dynamic Window Search
DVD	Digital Versatile Disk
FMO	Flexible Macroblock Ordering
FPGA	Field Programmable Gate Array
FSBB-ME	Full Search Block Based Motion Estimation
FSS	Four Step Search
HEVC	High Efficiency Video Coding
HD	High Definition
HDSP	Horizontal Diamond search pattern

ITU	International Telecommunication Union
IEC	International Electrotechnical Commission
ISC	Initial Search Center
ISO	International Organization for Standardization
JPEG	Joint photographic Experts Group
LDSP	Large Diamond Search Pattern
MAD	Mean Absolute Difference
MAF	Motion Activity Field
MC	Motion Compensation
ME	Motion Estimation
MEF	Motion Estimation Partitioning
MV	Motion Vector
MPEG	Moving Picture Experts Group standardized
NTSS	New Three Step Search
PE	Processing Element
PMV	Predicted Motion Vector
PU	Prediction Unit
RB	Reference Block
RBR	Reference Block Register
ROS	Regions of Support
SAD	Sum of Absolute Difference
SAO	Sample Adaptive Offset
SD	Standard Definition
SDTV	Standard-Definition Television
SDSP	Small Diamond search pattern
SIF	Source Input Format
TCC	Total Computation Complexity
TSS	Three Step Search

TU	Transform Unit
UHD	Ultra High Definition
VHDL	Very High Speed Integrated circuit Hardware Description Language
VLSI	Very Large Scale Integration
VCL	Video Coding Layer
VCEG	Video Coding Experts Group
VDSP	Vertical Diamond Search Pattern

Chapter 1

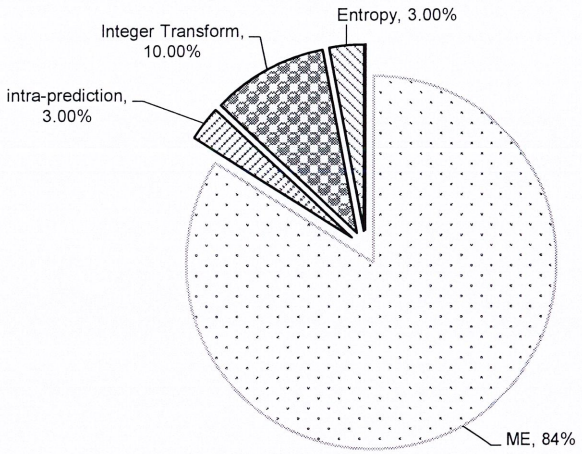
INTRODUCTION

1.1 Research Problem
1.2 Research Goals
1.3 Research Area
1.4 Summary

Chapter 1 INTRODUCTION

H.265/HEVC or High Efficiency Video Coding (HEVC) standard (Ohm, Sullivan, Schwarz, Thiow Keng, & Wiegand, 2012) is a new video coding standard developed jointly by ITU-T VCEG and ISO/IEC Moving Picture Experts Group standardized (MPEG) (ITU-T & ISO/IEC-JTC1, 1994). It was technically finalized in January 2013 and has recently been published by ITU-T and ISO (ITU-T & JVT, 2012). Compared to the previous H.264/AVC (Advanced Video Coding) standard (ITU-T & JVT, 2003), H.265/HEVC has accomplished up to 50% savings in the transmitted bit-rate. Consequently, 4K and Ultra High Definition TV (UHD-TV) resolutions can be achieved (P. & Marikkannan, 2014). Additionally, H.265/HEVC can be used in many applications such as online video streaming, video conferencing, HD-TV broad casting , remote video surveillance, and medical imaging (Ohm et al., 2012).

H.265/HEVC offers many coding tools for achieving high compression gain compared to the H.264/AVC standard. Quarter-sample precision, 7-tap or 8-tap filters, and uni-prediction or bi-prediction or weighted prediction coding techniques are examples of such tools (Ohm et al., 2012). Figure 1.1 shows the encoding time consumed by the H.265/HEVC encoder. It is clear from Figure 1.1 that Motion Estimation (ME) process consumes up to 84% of the total entire encoding time of the H.265/HEVC encoder (Grellert et al., 2013; Kim et al., 2012). This is why we emphasize on reducing the computational complexity of the ME process. Consequently, the overall encoding time of H.265/HEVC encoder can be greatly reduced.



(Figure 1.1)

Time complexity of the H.265/HEVC encoder (Grellert et al., 2013; Kim et al., 2012)

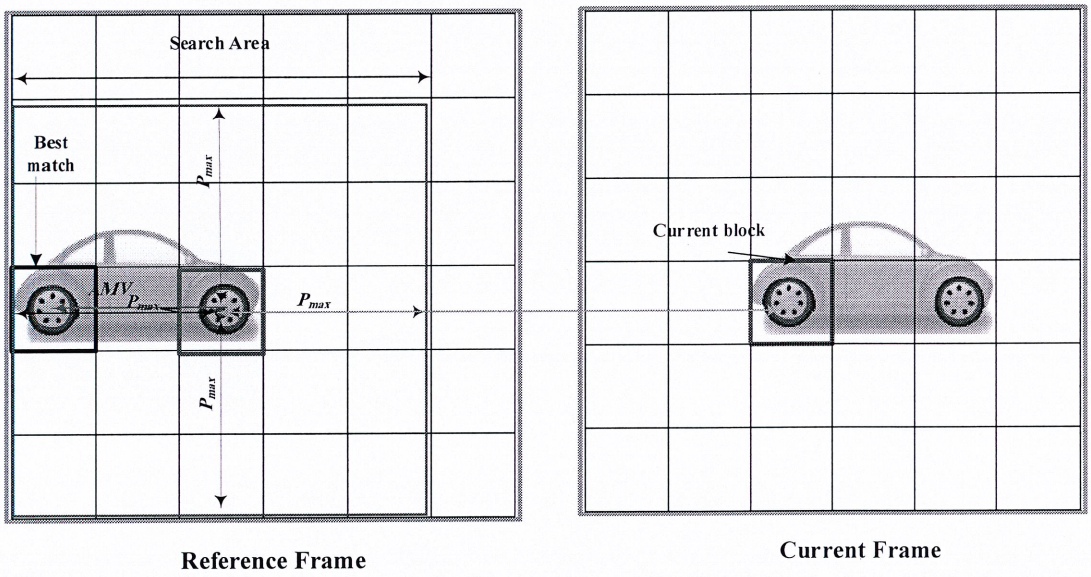
1.1 Research Problem

Motion Estimation (ME) is the process of finding the Actual Motion Vector (AMV) that defines the transformation of a Current Block (CB) image (located in the current frame) from a Reference Block (RB) (located in the reference frame). Full Search Block Based Motion Estimation (FSBB-ME) (Yeong-Kang & Lien-Fei, 2003) is the most popular ME algorithm. During the FSBB-ME algorithm, the current frame is divided into blocks, each of size N×N pixels. Each block searches for its best match RB in the search area located at the reference frame. Figure *1.2* shows how to implement the FSBB-ME algorithm. The best match candidate block is calculated by searching each point in the search area represented by $2P_{max} \times 2P_{max}$.; where $2P_{max}$ is the range of the selected search area. The point located at the smallest cost is selected as the best match candidate block. The cost can be measured using the Sum of Absolute Difference (SAD) metric. The displacement between the

center of the search area and the best match reference block is represented by the Actual Motion Vector (AMV) as seen in Equation 1.1.

$$AMV(k,l)=\text{argmin}\{SAD(k,l;u,v)\} \quad (1.1)$$

Where $-P_{max} \leq u, v \leq P_{max}$, K and l are the coordinates of the Current Block of the search center, u and v are the coordinates of the best match candidate block, and argmin is a Matlab function used to find the minimum value in an input vector.



(Figure 1.2)

Full Search Block Matching Motion Estimation (FSBM-ME) algorithm

Comparing video applications to other multimedia sources such as speech, and text, it consumes much data. Table 1.1 illustrates some different video data formats. For SIF video sequences, 32×32 search area and 16×16 CB size are needed. While for SDTV and HDTV video sequences, 64×64 search area and 32×32 CB size are required (Ismail, El-Medany, Al-Junaid, & Abdelgawad, 2014). We concluded from Table 1.1 two main important notations:

1- Higher number of pixels of the search area is needed from the memory as the resolution of a video has been increased (Jen-Chieh, Tian-Sheuan, & Chein-Wei, 2002). For example, UHDTV broadcasting requires much data to be fetched from the memory than the Video Conferencing which uses SIF video format. Since the memory I/O BandWidth (BW) is limited, the work in this thesis is elaborating architectures for better use of the available memory I/O BW. The elaborated architectures allow the reuse of an existing data inside the ME co-processor. Consequently, no need for fetching large amount of data from the main memory (external memory).
2- The more the resolution of a video sequence is, the more the required computations for performing the FSBB-ME algorithm. In our work, a fast FSBB-ME algorithm and its architecture will be implemented for a great savings in the ME computations.

(Table 1.1)
Different formats for video transmission (P. & Marikkannan, 2014)

Resolution	**Pixels/line**	**Lines/frame**	**Frames/sec**
UHD 8k	7680	4320	30
UHD 4k	3840	2160	30
HDTV broadcast	1920	1080	30
SDTV broadcast (D1)	720	486	30
Video conferencing (SIF)	352	240	30

There are two main problems of the H.265/HEVC standard (Kim et al., 2012): the huge number of data required for performing the ME process and the high computational complexity compared to the previous

standards (Bossen, Bross, Suhring, & Flynn, 2012). The huge data results from two main reasons (Ohm et al., 2012). First, increasing the density of each pixel, as an example, 4K and UHD pictures have 10-bits/pixel. Second, increasing both of the Current Block and the search area sizes, the size of the Current Block and the needed search area are 32×32 pixels and 64×64 pixels, respectively. Consequently, large computations are required to process such huge data.

The problem of the huge number of pixels to be processed in ME process can be solved using a smart internal memory which is important for data reuse (Jen-Chieh et al., 2002). Data reuse principle is very important to reduce the amount of data needed from the main memory. The high computational complexity problem can be solved using a prediction technique combined with an Adaptive Search Window Size (ASWS) technique. Combining such two previous techniques can greatly reduce the number of pixels in the search area needed for the ME process. Consequently the required computations for the ME process will be greatly reduced. As a result the speed of the H.265/HEVC encoder will increase.

1.2 Research Goals

The goal of the work in our thesis can be summarized as follows:

1) Solving the huge number of data required for ME process using the data reuse principle for efficient memory usage and for minimizing the I/O memory BW.
2) Reducing the computational complexity of ME process by implementing ME prediction techniques.

3) Implementing two architectures of ME co-processor to solve the two main problems in H.265\HEVC ME process.

1.3 Research Area

This work is mainly focused on minimizing I/O memory BW, computations, and the hardware complexity of the Block Based Motion Estimation (BB-ME). H.265/HEVC video coding standard is selected as a study case for implementing and elaborating the implemented architectures since it is the recent used standard in a wide range of video application. FSBB-ME is the well-known BB-ME that is used to exhaustively find the best match candidate block in a reference frame.

1.4 Contribution

Our contribution will be implementing two architectures of ME co-processor that are suited for the H.265/HEVC standard as will be explained in the Theoretical Framework section. While implementing the two architectures, optimizing the area, and increasing the ME process speed will be targeted. The first architecture which using some previous work (Ismail, El-Medany, Al-Junaid, & Abdelgawad, 2014) will treat the data reuse problem in H.265/HEVC and it will consider lowering the area of the designed (7z045ffg900-2) FPGA chip. The second architecture which is using previous software verified ASWS algorithm (Goel, Ismail, & Bayoumi, 2005) will add another advantage to the first one by reducing the computations required for performing the ME process using some components already verified in software. Consequently, the speed of the H.265/HEVC encoder will increase. The two architectures will be verified using VHDL tool and will be validated using Matlab tool. Then,

the cost of the proposed co-processors will be estimated using FPGA Xilinx ZYNQ-7 ZC706 using ISE Design Suite 14.5 tool.

1.5 Summary

There are two main problems of the H.265/HEVC standard (Kim et al., 2012): the huge number of data required for performing the ME process and the high computational complexity compared to the previous standards (Bossen et al., 2012). The huge data result from two main reasons (Ohm et al., 2012). First, increasing the density of each pixel in 4K and UHD pictures (10-bits). Second, increasing both of the Current Block and the search area sizes. The size of the Current Block and the needed search area are 32×32 pixels and 64×64 pixels, respectively. Consequently, large computations as well as large data are required. The huge number of data problem can be solved by using a smart internal memory which is important for data reuse (Jen-Chieh et al., 2002). Data reuse principle is very important to reduce the amount of data needed from the main memory. The high computational complexity problem can be solved using prediction techniques combined with Adaptive Search Window Size (ASWS) technique. Combining such two techniques can greatly reduce the number of the required pixels in the search area. Consequently, the required computations for the ME process will be greatly reduced. As a result the speed of the H.265/HEVC encoder will increase.

Chapter 2

THEORETICAL FRAMEWORK AND LITERATURE REVIEW

2.1 Theoretical Framework
2.2 Literature Review
2.3 Summary

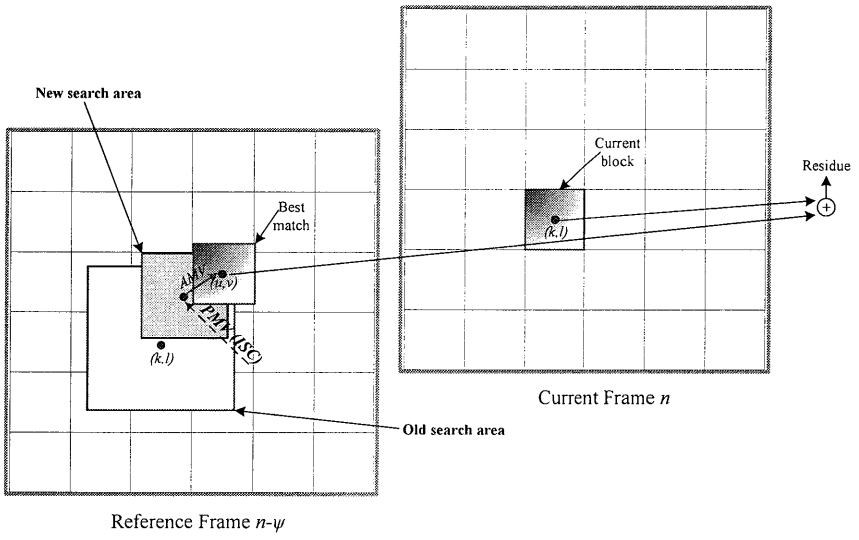
Chapter 2 THEORETICAL FRAMEWORK AND LITERATURE REVIEW

2.1 Theoretical Framework

Motion Estimation consumes up to 84% of the entire computational time of H.265/HEVC encoder (Kim et al., 2012). It is the most exhaustive processing unit in the encoder of the H.265/HEVC standard. We are implementing and elaborating two ME co-processor architectures for efficiently performing the ME process in the H.265/HEVC encoder. The first architecture solves the issue of the large number of pixels needed from the external memory in the ME process. This will be achieved by using a smart internal memory that efficiently use level A and level B data reuse principle. Data reuse principle is mainly used to reduce the I/O memory BW (Jen-Chieh et al., 2002). Consequently, the overall required time for fetching data from the external memory will be greatly reduced. The second architecture benefits from the high correlation between adjacent Current Blocks (CBs) in the current frame to reduce the number of search points in the search area. This can be achieved by expecting the motion activity of the CB that can be used to diminish the search area size. As a result, fewer pixels are fetched from the external memory and fewer computations are required to get the best match candidate block from the search area.

Figure 2.1 shows that there is a possibility of reducing the ME computational complexity by moving the search center point (pixel allocated at point (k, l) in the reference frame of Figure 2.1) to another point near to the best match candidate block position (point (u, v) in

Figure 2.1). As a result, the search area size can be diminished according to the motion activity of the CB. The reduction in the ME complexity can be achieved by using a prediction technique and an Adaptive Search Window Size (ASWS) algorithm. The prediction technique can be used to expect an Initial Search Center (ISC) that is very close to the expected best match candidate block. This can be done using some information from previously encoded blocks in the current frame. Once the ISC is decided, the number of pixels in the search area can be decreased according to the expected motion activity of the CB using the ASWS algorithm.



(Figure 2.1)

Adaptive Search Window Size (ASWS) algorithm

The Total Computational Complexity (TCC) required for obtaining the best match candidate block in the reference frame can be expressed using the following general Equation 2.1:

$$TCC=[(n_1 \times n_2)\ .(Sub+Abs+Add)].M \qquad (22.1)$$

Where, n_1 and n_2 are the number of rows and columns in the Current

Block. Sub, Abs, and Add represent the computational complexity of subtraction, absolute value and addition operations, respectively, that are required for calculating the matching criteria defined by the SAD operation. M is the total number of checked points in the search area in the reference frame.

2.2 Literature Review

There are two main problems of the H.265/HEVC standard (Kim et al., 2012): the huge number of data required for performing the ME process and the high computational complexity compared to the previous standards (Bossen, Bross, Suhring, & Flynn, 2012). And in this study we used FPGA Xilinx (7z045ffg900-2) chip using ISE Design Suite 14.5 tool to estimate the hardware cost. The FSBB-ME algorithm has been successfully implemented in Very Large Scale Integration (VLSI) (Ahmed, Shahid, Martina, Magli, & Masera, 2013; Goel, Ismail, & Bayoumi, 2005; Goel, Ismail, Devulapalli, McNeely, & Bayoumi, 2006; Huong, Klepko, Nam, & Demin, 2011; Pastuszak & Jakubowski, 2013). It is mainly adopted to lower the temporal redundancy between frames. Such algorithm has a huge encoding computational complexity up to 80% and 84% of the total encoding time in case of H.264/AVC and H.265/HEVC standards, respectively. Previous work is mainly mentioned in two sections: Previous data reuse implementations and previous ASWS implementations.

That is why many techniques were developed to reduce the computational complexity of the FSBB-ME algorithm such as Adaptive Search Window Size (ASWS) algorithm. This algorithm is adaptively shrinking the search area based on the predictive motion activity of the Current Block (CB). Other algorithms were also used to minimize the number of search points such as Three Step Search (TSS) (Koga, Iinuma,

& Iijima, 1981), New Three Step Search (NTSS) (Renxiang, Bing, & Liou, 1994), Four Step Search (FSS) (Lai-Man & Wing-Chung, 1996), Diamond Search (DS) (Shan & Kai-Kuang, 1997), and Cross Diamond Search (CDS) (Chun-Ho & Lai-Man, 2002). Although these algorithms are greatly diminish the number of search points in the search area, they have a bad effect on the video visual quality. So that, they can only be used with those applications require less video quality. Additionally, most of them suffer from irregular data flow that may not allow VLSI implementation of such algorithms.

Other algorithms (Nie & Kai-Kuang, 2002; Zhi, Jiajun, Chun, & Xu, 2005) utilizes the smoothness property of the video sequence, so that the searching process can be stopped if the candidate block located at the ISC of the search area is considered as the best match candidate block. A block can be considered as a best match candidate block if its Sum of Absolute Difference (SAD) between the Current Block and the candidate block in the reference frame is less than or equal to a threshold value T_s. If $SAD \leq T_s$, then the candidate block located at the ISC is considered as the best match candidate block, else, continue the exhaustive full search process.

In (Marlow, Ng, & McArdle, 1997), a 2D log-search window size algorithm is based on the distortion (SAD) at the search origin. In this algorithm, an initial step size is selected and then four blocks at a distance s from all four sides are selected. If the position of best match is at the center, halve the step size. Otherwise one of the four points is the best match and will become a new search center, this cycle is repeated till the step size becomes 1. At this stage, the best match is considered by finding the best among the four blocks. In (Feng, Lo, & Karbowiak, 1995), authors used the same algorithm, however, it is categorized into slow,

medium or fast motion activity blocks. Based on this category, the best window size was selected. Same approach was presented in (Kai, 1998) which also contained matching a short-term and long-term assignment parameter that select the step size for each block. In (Alkanhal, Turaga, & Chen, 1999), majority voting scheme with spiral search is presented. In this technique, a decision has to be taken for every block to choose between Tree Step Search (TSS) and Four Step Search (FSS) algorithms depending on the motion activity. If the AMV is large then TSS is selected otherwise FSS is selected. And the spiral search start method from the search center moving outwards. The Mean Absolute Difference (MAD) of previously estimated blocks can determine when to stop the search based on a threshold value.

- **Previous ASWS implementations**

In (Bailo, Bariani, Barbieri, & Raggio, 2004), the search area size depends on the motion activity of the current block. The difference between current block and reference block at same position is calculated. Then, the number of white and black pixels is counted. The more the white pixels are, the bigger the search area will be, as this indicates much motion activity in the current block. Otherwise, if more black pixels are counted, then the search area will be smaller. Another approach is presented in (Minocha & Shanbhag, 1999; Saponara & Fanucci, 2004), where a fixed search window size is selected for all current blocks in the same frame based on the largest Actual Motion Vector (AMV) calculated in the pre-encoded blocks in same current frame. This approach is called window-follower technique. In (Chen et al., 2010) Chen developed a block-pipelined cache. In this algorithm, a real time video sequence was processed (1920x1088@30fps). This algorithm was enhanced by Zhou to

support 3840x2160@60 (Zhou, Zhou, he, & Goto, 2011). Most of the previously implemented algorithms are not suitable for hardware implementation as they lose the regularity of data flow, but they can greatly reduce the computational complexity.

Although most previous algorithms are not suitable for hardware implementation, some state of the art algorithms success to be implemented. In (P. & Marikkannan, 2014), the authors implemented full architecture for H.265/HEVC. In the Motion Estimation unit, temporal and spiral AMVs are used to generate Predicted Motion Vector (PMV) with the help of the Advanced Motion Vector Prediction (AMVP) calculator to improve the throughput of the Motion Estimation. The implemented architecture introduces sharing mechanism to share 1-line buffer between de-blocking filter and intra-prediction to reduce the internal SRAM size. The designed hardware cost is 446.6K gates with 10.21 KB internal SRAM with maximum hardware frequency of 224 MHz to decode a 1080@35fps video.

- **Data reuse implementations**

In (Sung-Fang et al., 2013), a 8K video H.265/HEVC encoder chip is implemented. The main achievement of this architecture is the internal memory which is achieving up to Level-C data reuse to improve the ME process. The power consumption is 708 mW at 312 MHz for 8K video format. Context Adaptive Binary Arithmetic Coding (CABAC) rate is improved by 50% with 8.90 GB/s external memory BW savings. In (Sugito et al., 2013), super Hi-Vision H.265/HEVC Real-Time encoder has been implemented. The implemented architecture uses Motion Estimation Prediction technique in which it splits the square blocks using Motion Estimation Partitioning (MEP) which is efficient for processing

small motions. 8K video sequence is encoded with 60fps, the architecture provided up to 23% saving in bit-rate with maximum bit-rate of 340 Mbps.

2.3 Summary

ME consumes up to 84% of the entire computational time of H.265/HEVC encoder (Kim et al., 2012). There are two main problems which cause the time consuming issue in the ME process. First, the huge number of the search area pixels needed to be fetched from the main memory. Second, the huge number of computations needed to perform the ME process. Two ME co-processors are implemented to solve the two previous problems. The first ME co-processors will be using a smart internal memory that efficiently use level A and level B data reuse principle. While in the second ME co-processors a huge reduction in the ME complexity can be achieved by using a prediction technique and an Adaptive Search Window Size (ASWS). In the literature review, many algorithms and techniques were explored. Some of them are used to greatly diminish the number of search points in the search area, but they have a bad effect on the video visual quality such as TSS, NTSS, FSS, DS, and CDS. Other algorithms are called Adaptive Search Window Size (ASWS) algorithms. These algorithms are adaptively shrinking the search area based on the predictive motion activity of the Current Block. Many other's hardware implementation of ME co-processor are mentioned in this chapter as well.

Chapter 3

VIDEO CODING CONCEPT

3.1 Introduction

3.2 Video Coding Process

3.3 Temporal Model

3.4 Video Coding Layer

3.4.1 Pictures, Frames, and Fields

3.4.2 YC_bC_r Color Space and 4:2:0 Sampling

3.5 Motion Estimation and Video Compression Algorithms

3.5.1 New Three Step Search (NTSS)

3.5.2 Four Step Search (FSS)

3.5.3 Diamond Search (DS)

3.5.4 Cross Diamond Search (CDS)

3.6 Summary

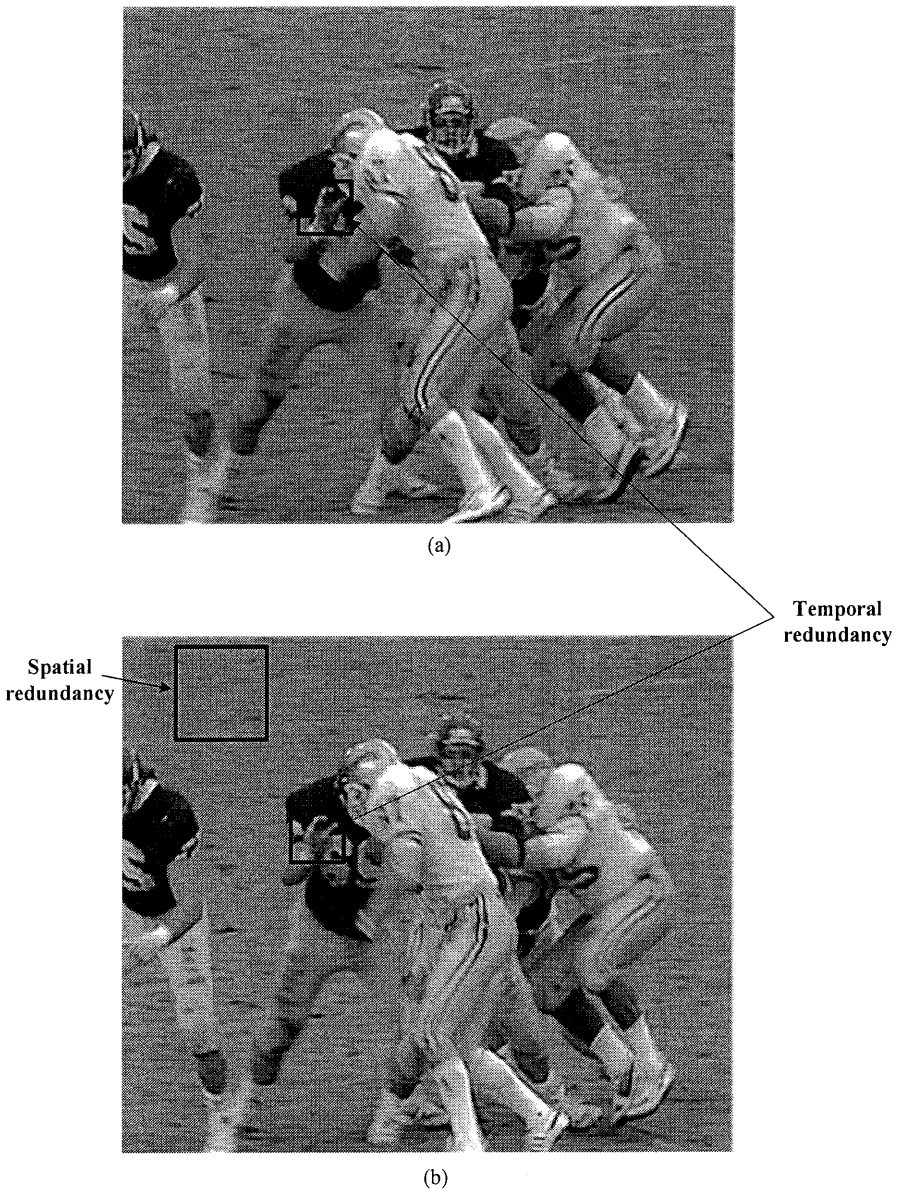
Chapter 3 VIDEO CODING CONCEPT

3.1 Introduction

Video coding compression is a process of compressing digital video data into less number of bits. Transmitting one second of uncompressed TV video quality requires approximately 216 Mbits (Woong, Byeungwoo, & Jechang, 2003). This is why video compression is required to reduce the transmitted channel BW. To achieve video compression, two entities are required: the encoder and the decoder. The encoder converts the video sequence data into a compressed low bit-rate form ready to be transmitted over the transmission channel. The decoder is the second system which does the decompression to return back the compressed data into its original video data. The previous two systems together are called CODEC (enCOder/DECoder).

Video compression is accomplished by eliminating the redundant information from the video data. There are two types of redundancy in video data. First, the spatial redundancy which is eliminated using lossless compression tools in which the reconstructed video is exactly same as the original one. Second, temporal redundancy that can be removed using lossy compression tools which achieves higher video compression. In this type, some parts of video data can be removed without highly impact on the visual quality. Lossy compression tools deals with both temporal and spatial redundancies. In temporal redundancy, there is a high similarity between consecutive frames especially when the frame rate is high. In spatial domain, there is high similarity within the frame between neighboring pixels. Figure 3.1 shows both temporal and spatial redundancies in two consecutive frames.

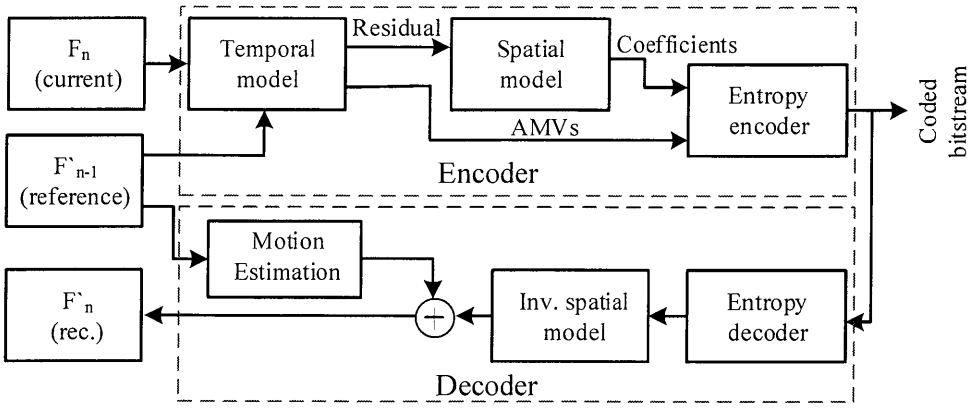
Temporal redundancy can be seen in moving objects between consecutive frames such as moving hands. The spatial redundancy can be seen in the flat background since all pixels values are almost same.



(Figure 3.1)
Snap-shot of two consecutive frames showing both the spatial and temporal redundancies. (a) Frame n-1. (b) Frame n

3.2 Video Coding Process

The encoder transforms the video data into a bit stream. The video encoder consists mainly of three units as shown in Figure *3.2*. These units are the temporal unit, the spatial unit, and the entropy encoder. In the temporal unit, input video is compressed by eliminating temporal redundancies between same video consecutive frames. This process is done by using prediction techniques from one or more of previous frames. The output is a residual frame and Actual Motion Vectors (AMVs). The difference between the current frame and the prediction is the residual. In the spatial unit, the spatial redundancies are removed. In this stage the residual samples are transformed to another domain in which they will be represented by transform coefficients. Insignificant values are removed after quantization process which will keep only small number of significant coefficients to achieve high data compression of the residual frame. The output of temporal and spatial units (AMVs and transformed coefficients) enters the entropy encoder (Wiegand, Sullivan, Bjontegaard, & Luthra, 2003) which remove the statistical redundancy in the video data. In entropy unit, the sorted binary codes are used to represent commonly occurring AMVs and coefficients. The encoded/compressed bit stream can be transmitted and/or stored. The same reference frame/s should be used in both the encoder and the decoder to avoid the drift problem (De Cock, Notebaert, Lambert, & Van de Walle, 2008; Malvar, Hallapuro, Karczewicz, & Kerofsky, 2003). This problem occurs if different reference frame/s is/are used in both encoder and decoder. This problem is avoided by including a feedback loop of the encoder as shown in Figure *3.2* to ensure that the same reference frame/s is used in both encoder and decoder. Where F_n , F'_{n-1} ,and F'_n are current, reference, and recovered Frames.



(Figure 3.2)

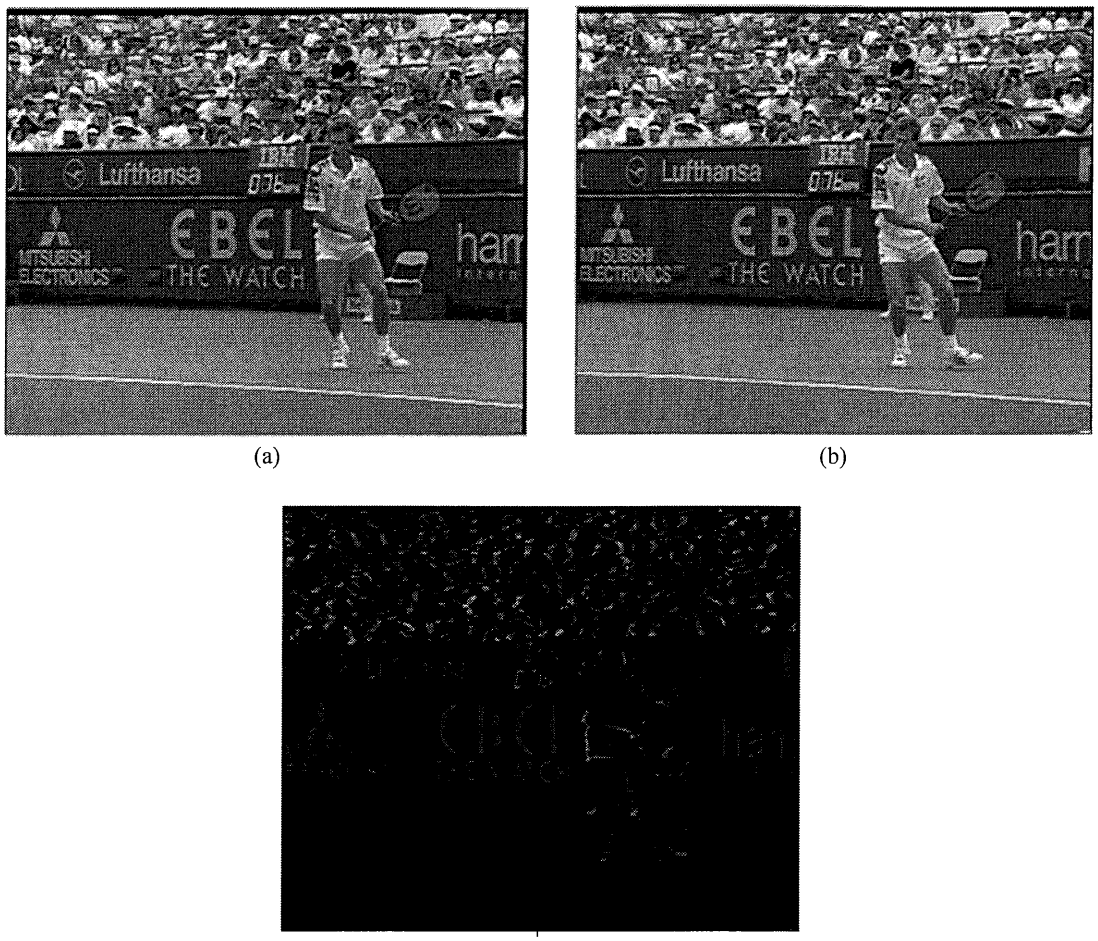
Video CODEC block diagram.(Ismail, 2010; Ismail, McNeely, Shaaban, Mahmoud, & Bayoumi, 2012)

3.3 Temporal Model

The temporal redundancy between the transmitted frames is reduced using the temporal model. The residual frame is formed by subtracting the predicted frame from the current frame. More data compression can be achieved if the predicted frame is more accurate which leads to fewer pixels values in the residual frame.

The predicted frame can be calculated by subtracting the current frame from the previous frame as shown in Figure *3.3*. The mid-gray represents zero residual and light or dark represents positive or negative residual. One drawback of such prediction is that a large amount of energy still remains in the residual frame. Consequently, less compression can be achieved. Compensating for the motion between two or more frames can lower such drawback. Therefore, the compression performance can be enhanced. Smart Motion Estimation technique is required to achieve a high performance.

Motion Estimation (ME) is required to remove temporal redundancy from the video data by finding the best match of the current frame using one or more reference frames. ME process is trying to find the best match to increase the compression performance. In Motion Estimation, the best match frame is subtracted from the current frame to get the residual frame. The lesser the value of the residual frame is, the higher performance of video compression that can be achieved.



(a) (b)

(c)

(Figure 3.3)

Two consecutive frames of the mobile video sequence and their difference.

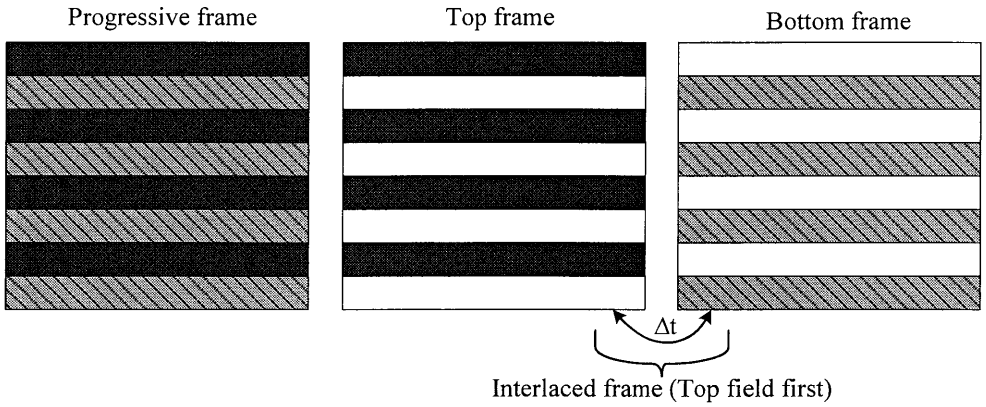
(a) Frame #0. (b) Frame #1. (c) Deference between two consecutive frames

3.4 Video Coding Layer

The video encoder exploits the temporal redundancies from the video data using inter frame prediction. The spatial dependencies are exploited from the prediction residual using a transform coding. H.265/HEVC standard adds a plurality of smaller improvements in each element in the VCL to greatly improve the coding efficiency. In the following subsections, we will explain the improvements that are added by the H.265/HEVC standard that are not used by the previous standards.

3.4.1 Pictures, Frames, and Fields

A coded picture in (ITU-T & JVT, 2003) can be represented by a frame or a single field. The video frames contain two interleaved fields, a top and a bottom field. Defining the number of rows in a frame as H, the top field contains even numbered rows 0, 2, ..., H/2-1 and the bottom field contains the odd numbered rows starting from the second line of the frame. If these two fields are captured at the same time, the frame is called a progressive frame, and otherwise, the frame is called an interlaced frame as seen in Figure 3.4.

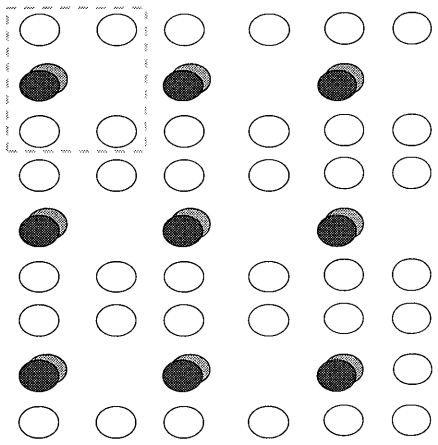


(Figure 3.4)

Progressive and interlaced frames and fields (Ismail, 2010)

3.4.2 YC_bC_r Color Space and 4:2:0 Sampling

The human eye is much more discriminating to the changes in brightness than color. Given that Y is the luma component (representing the luminance or the brightness) and C_b and Cr are the chroma components (representing the chrominance or the color), H.264/AVC and H.265/HEVC standards takes advantage of the previous observation by using YC_bC_r color space together with reducing the sampling resolution of C_b and C_r chroma information. H.264/AVC and H.265/HEVC use a sampling structure (Sullivan, Ohm, Woo-Jin, & Wiegand, 2012; Wiegand et al., 2003), as seen in Figure 3.5, in which the chroma component has one fourth of the number of samples than the luma component. In other words, chroma component has the half number of samples in both the horizontal and the vertical dimensions. This is called 4:2:0 sampling with 8 bits and 10 bits precision per sample in case of using H.264/AVC and H.265/HEVC standards, respectively.



(Figure 3.5)

4:2:0 sampling patterns (progressive) (Ismail, 2010)

3.5 Motion Estimation and Video Compression Algorithms

Motion Estimation (ME) technique is used to find Actual Motion Vectors (AMVs) between two frames in order to remove temporal redundancy. Motion Estimation process consumes up to 84% of the entire encoding time of the H.265/HEVC encoder (Grellert et al., 2013; Kim et al., 2012). Block Based Motion Estimation (BB-ME) algorithm is the most popular algorithm for ME process and is adopted in the latest video coding standards because of its good compression performance achievements. BB-ME is processed on the luminance components of a video frame. Macro blocks size of N×N pixels of the current frame to be encoded are called Current Blocks (CBs). In ME process, we are trying to find the best match of a CB in the reference frame within a search area. The search area is centered at same position of the CB but in the reference frame. The transmission bit rate is reduced by sending only the difference between the Current Block and the best match candidate block. FSBB-ME technique achieves a very high video quality and a high compression ratio, but its computational complexity is extremely high. Many algorithms have been implemented in order to reduce the computation complexity problem of FSBB-ME algorithm by reducing the number of search points needed to find the best match block in the search area. Three Step Search (TSS) (Koga et al., 1981), New Three Step Search (NTSS) (Renxiang et al., 1994), Four Step Search (FSS) (Lai-Man & Wing-Chung, 1996), Diamond Search (DS) (Shan & Kai-Kuang, 1997), and Cross Diamond Search (CDS) (Chun-Ho & Lai-Man, 2002) are examples for such algorithms. The previous mentioned methods can reduce the computational complexity; however, they can lead to considerable video quality degradation. They also are not suitable for

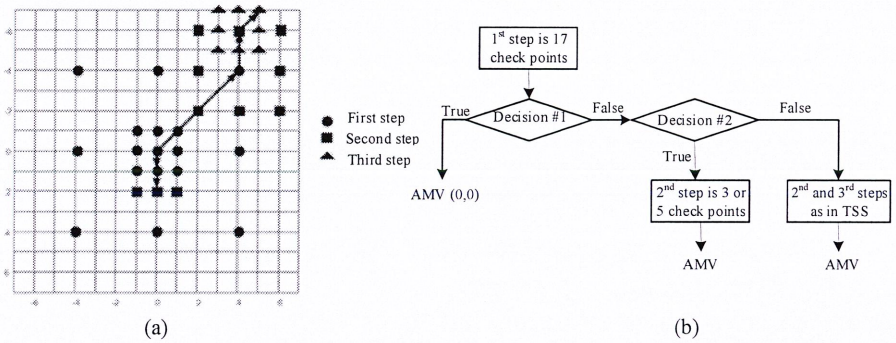
hardware implementation since they lake of regular data flow (Ismail et al., 2012).

3.5.1 New Three Step Search (NTSS)

The main problem with the Three Step Search (TSS) algorithm (Koga et al., 1981) is that large number of searching points pattern used to start the process. TSS will not be efficient in case of having small motion activity and most probably this can lead to a local minima problem (Renxiang et al., 1994). NTSS algorithm is used to eliminate the previously mentioned problem and can be summarized as follows using Figure *3.6*:

1) NTSS has eight more points than TSS, neighboring the center point of the block to be checked as shown in Figure *3.6* (a), so total of 16 points are picked for compression.
2) If the minimum SAD is located at the search center of the first step, then stop the search and select the search center block as the best match candidate block. Else, if the minimum SAD is found in one of the other eight neighbors of the search center in the first step, then the search will be performed only on the eight neighboring points in the next step, this is called second step stop.

If the minimum SAD is found in the outer eight points, then TSS search will be performed. Figure *3.6* shows block diagram of NTSS algorithm and related example.



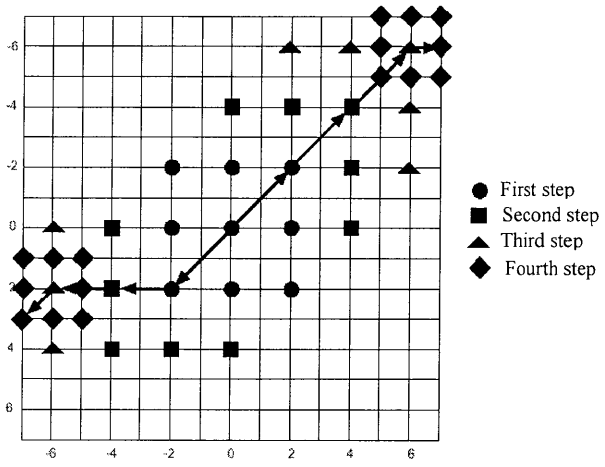
(Figure 3.6)

The New Three Step Search algorithm (NTSS). (a) An example of the NTSS algorithm. (b) The flow chart of the NTSS algorithm.(Ismail, 2010; Renxiang et al., 1994)

3.5.2 Four Step Search (FSS)

This algorithm is implemented to minimize the local minima problem by using a smaller initial step size compared to the TSS algorithm (Koga et al., 1981). FSS algorithm is summarized as follows:

1) A minimum SAD is found from nine checking points pattern on a 5x5 window located at the center of 15x15 search area as shown in Figure 3.7. If the minimum is found at the center of the search area, then go to step 4; otherwise go to step 2.
2) The search window is maintained in 5x5 checking points. However, the search patterns depend on the location of the minimum SAD point, as shown in Figure 3.7 it is located, then go to step 4; otherwise go to the next step.
3) The Search pattern is same as step 2.
4) The search window is reduced to 3x3 checking points and the direction of the overall MV is considered as the minimum SAD point among the nine checking points.

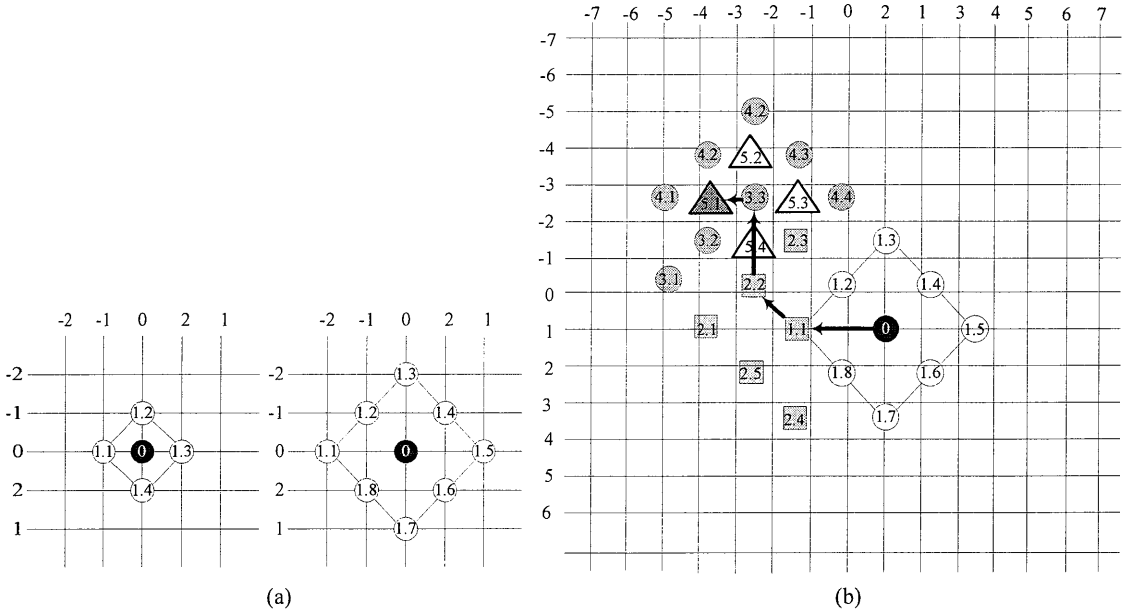


(Figure 3.7)
The Four Step Search Algorithm (Ismail, 2010; Lai-Man & Wing-Chung, 1996)

3.5.3 Diamond Search (DS)

The DS algorithm uses nine candidate search points; this pattern is called Large Diamond Search Pattern (LDSP). Eight points of the nine are surrounding the search center to form a diamond search. The second pattern consists of five checking points, forming a smaller diamond shape, is called Small Diamond search pattern (SDSP) as shown in Figure 3.8 (a).

In the DS algorithm, LDSP is repeatedly used to find the minimum SAD at the search point. Then, the search pattern is transformed from the LDSP to SDSP as going to the final search stage. The AMV is found in the minimum SAD position of the best match candidate block as shown in Figure 3.8 (b).



(Figure 3.8)

The Diamond Search algorithm. (a) SDSP and LDSP patterns. (b) An illustrative example for the DS algorithm (Ismail, 2010)

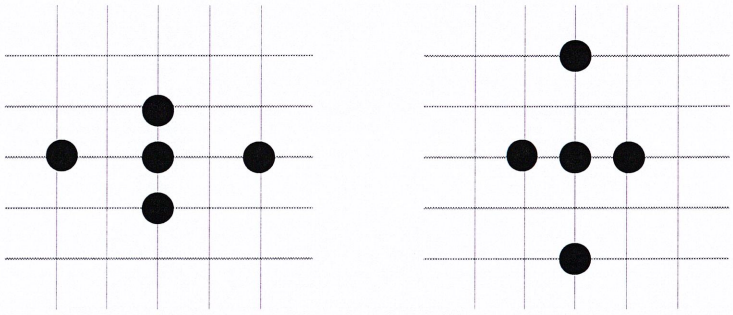
3.5.4 Cross Diamond Search (CDS)

In the CDS algorithm, if the point with the minimum SAD is found in the center point, then stop search. Otherwise, continue with the cross patterns as seen in Figure 3.9. The CDS algorithm is summarized as follows:

1) The cross pattern can be determined by finding the proportion relation between the Horizontal Diamond Search Pattern (HDSP) and the Vertical Diamond Search Pattern (VDSP) motion sizes in the last reference frame. According to the position of the current block, minimum SAD is calculated for the center point and four other points surrounding it. Best match block is considered to be the minimum SAD among all five

previous points. If the minimum SAD is found at the center of the search area, then go to step 3; otherwise go to step 2.

2) Select the diamond pattern that is found in step 1. Choose the minimum SAD point from step 1 as the center point, then the search starts to find the best match. If the minimum SAD is found in the center, then go to step 3. Otherwise, this step will be repeated until the best match point is allocated at the center.
3) Select the minimum SAD value as the best match point, and then the Actual Motion Vector is calculated. CDS algorithm determines the motion direction and consequently starts the search process based on that.



(a) HDSP (B) VDSP

(Figure 3.9)

Cross Diamond Pattern. (a) The HDSP. (b) VDSP patterns (Chun-Ho & Lai-Man, 2002)

3.6 Summary

This chapter has briefly discussed the video coding process and a comprehensive survey of the existing Motion Estimation techniques. It has been shown that Motion Estimation algorithm forms the core operation of the video CODEC system. It is the most exhaustive time consuming part in the whole video encoding system. It is clear that the

most popular ME algorithm, i.e., FSBB-ME, is computationally intensive. Some fast ME techniques are also briefly described in this chapter. Although such fast ME algorithms greatly reduce the computations and speed up the video encoding process, however, most of them suffer from irregularity of their data flow. Consequently, they are hard to be implemented in VLSI and offer low video quality. This is why FSBB-ME algorithm is still considered the most usable algorithm due to its regularity in data flow and high resolution of video quality. The next chapter presents a comprehensive survey of the H.265/HEVC standard that is used in a wide range of video applications nowadays which will be discussed in the next chapter.

Chapter 4

H.265/HEVC HIGH EFFICIENT VIDEO CODING STANDARD

4.1 Introduction
4.2 H.264/AVC Video CODEC
4.3 Main Motivations of H.265/HEVC Standard
4.4 H.265/HEVC CODEC and Features
4.5 Summary

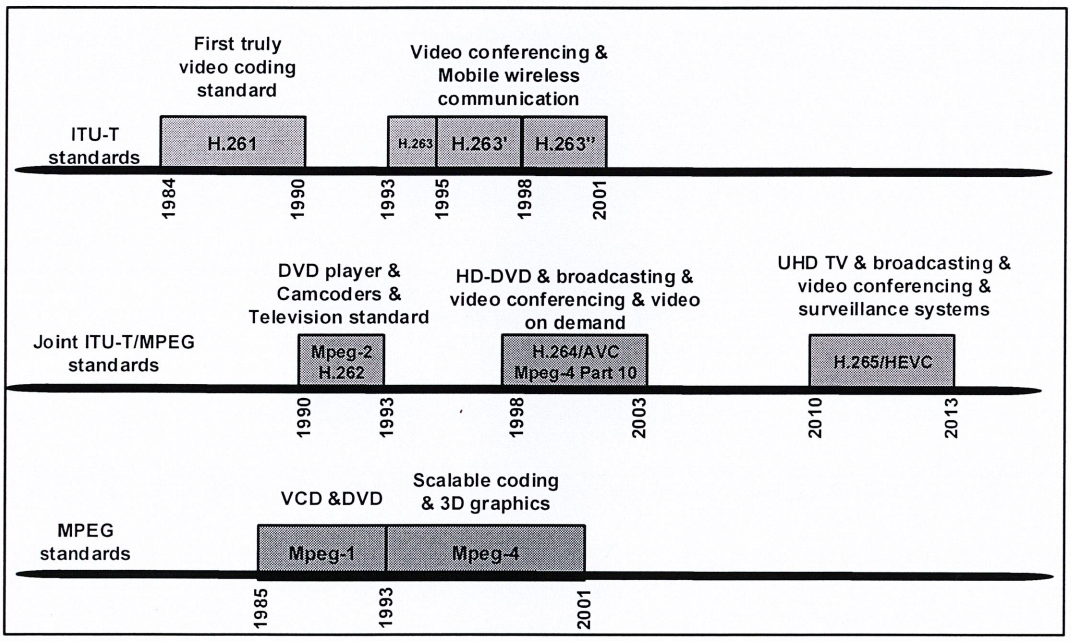
Chapter 4 H.265/HEVC HIGH EFFICIENT VIDEO CODING STANDARD

4.1 Introduction

Referring to Figure *4.1*, in 1984 International Telecommunication Union (ITU) started the development of H.261 standard. And in 1990 (ITU-T & JVT, 2003), H.261 was completely developed. H.261 is using a hybrid video coding scheme that is the base of many video coding techniques nowadays. MPEG-2 (known as H.262) was the most popular standard created by ISO/IEC and ITU and was published in 1993 (ITU-T & JVT, 2003). H.262 video standard has been used in many applications such as digital-TV, Digital Versatile Disc (DVD) (ITU-T & JVT, 2003). In 1995, H.263 was mainly developed by the ITU ("Video coding for low bit rate communication," 2000) for video conferencing at low bit rates in mobile wireless communication applications. MPEG-4 standardization began in 1993 ("Video coding for low bit rate communication," 2000). It supports new applications such as scalable coding and 3D graphics. MPEG-4 was completely developed along with H.263 standard in 2001. The simple profile of MPEG-4 is very close to H.263. It added support to the SD video and additional tools to further increase the compression efficiency.

H.264/AVC Advanced Video Coding Standard was developed to reduce the transmission bit-rate by approximately 50% compared to all other previous standards (ITU-T & ISO/IEC-JTC1, 1994). H.264/AVC has accomplished a high BW savings when it comes to video streaming over the internet, especially for High Definition (HD) videos. However, the Motion Estimation in H.264/AVC is still considered as a complex

process, because it consumes more than 80% of the total encoding time (Woong et al., 2003). H.264/AVC achieved up to 50% bit-rate savings by enhancing the motion-prediction capabilities, using small block-size, exact-match transform, adaptive in-loop deblocking filter, and enhance entropy coding methods. H.264/AVC can be used in many applications such as multimedia streaming over the internet or wireless network, HDTV broadcasting, Blu-Ray discs, video conferencing, remote video surveillance, and medical image processing field (Ohm et al., 2012).



(Figure 4.1)

Existing video coding standards (Timhillone Software Co.,Ltd., 2015)

High resolution videos are growing in popularity. They are creating bigger needs for high coding efficiency as the current network infrastructure is challenging the traffic caused by high video resolution applications. H.265/HEVC or High Efficiency Video Coding (HEVC) standard has been originally developed and finalized by ITU-T VCEG

and ISO/IEC MPEG (ITU-T & JVT, 2012). The H.265/HEVC standard achieves two main goals. First, encoding the high resolution video such as 4K and Ultra High Definition (UHD) using lower bit-rate compared to all other previous standards. Second, utilizing parallel processing operations in order to avoid the delay results from the added tools that are used to lower the bit-rate. The main raised issue in H.265/HEVC standard is that the computational complexity has been dramatically increased compared to previous standards. H.265/HEVC encoder is four-time more complex than older standards (Bossen et al., 2012).

One of the most exhaustive processing-unit in the H.265/HEVC video CODEC is the Motion Estimation (ME) unit which consumes up to 84% of the total encoding time (Grellert et al., 2013; Kim et al., 2012) . FSBB-ME is the most popular ME algorithm that is utilized to accordingly find the best match candidate block in the reference frame. FSBB-ME goes through all points in a specified search area, so that it costs a massive computational time. Additionally, computational time significantly increases in Motion Estimation process when considering variable block sizes and more than one reference frame. Although FSBB-ME gives the best quality compared to all other fast ME algorithms, it consumes large computations as well as large data to be processed. This is why working in algorithms and architectures that facilitate the computations and reduce the amount of data needed from the main memory to be processed is a hot area nowadays.

4.2 H.264/AVC Video CODEC

The main goal of the H.264/AVC standard is to provide a video quality at significantly lower bit-rate compared to the previous standards. H.264/AVC has many applications such as: DVD storage, low and high

video resolution, and video conferencing. H.264/AVC highlights some features discussed below (Wiegand et al., 2003):

1) **Variable block size Motion Estimation with small block sizes**: H.264/AVC provides more flexibility in the selection of Motion Estimation block sizes than other previous standards. It also uses a minimum luma block size as small as 4×4.
2) **Quarter-pixel-accurate Motion Estimation:** H.264/AVC standard improves the encoding accuracy by adding quarter-pixel-accuracy to the Motion Estimation process. Most previous techniques only use half-pixel-accuracy.
3) **Multiple reference picture Motion Estimation:** Most of the previous standards use one previous encoded frame to predict the values of the incoming (current) frame. H.264/AVC standard greatly improves the encoding efficiency by enabling the encoder select to use multiple previously encoded frames in the ME process.
4) **Weighted prediction:** H.264/AVC allows the predicted video data to be weighted by amounts specified by the encoder. This can improve the coding efficiency for scenes containing fades.
5) **Small block size transform:** All previous video coding standards use a transform block size of 8×8. H.264/AVC allows the encoder to represent video data in more locally adaptive fashion by using a transform block size of 4×4. This greatly increases the accuracy of the transformation process.
6) **Hierarchical block transform:** Although the 4×4 transform is useful in improving the coding efficiency, there are some cases that can use longer basis function for the transform coding without affecting the quality of the coding efficiency. These

cases happen when video data contains sufficient correlation. H.264/AVC standard enables this in two ways. The first way is to use the hierarchical transform to extend the used block size to 8×8 array in case of low frequency chroma information. The second way is to allow the encoder to enable the extension of the 16×16 block size in case of using luma transform for low frequency information in intra coding.

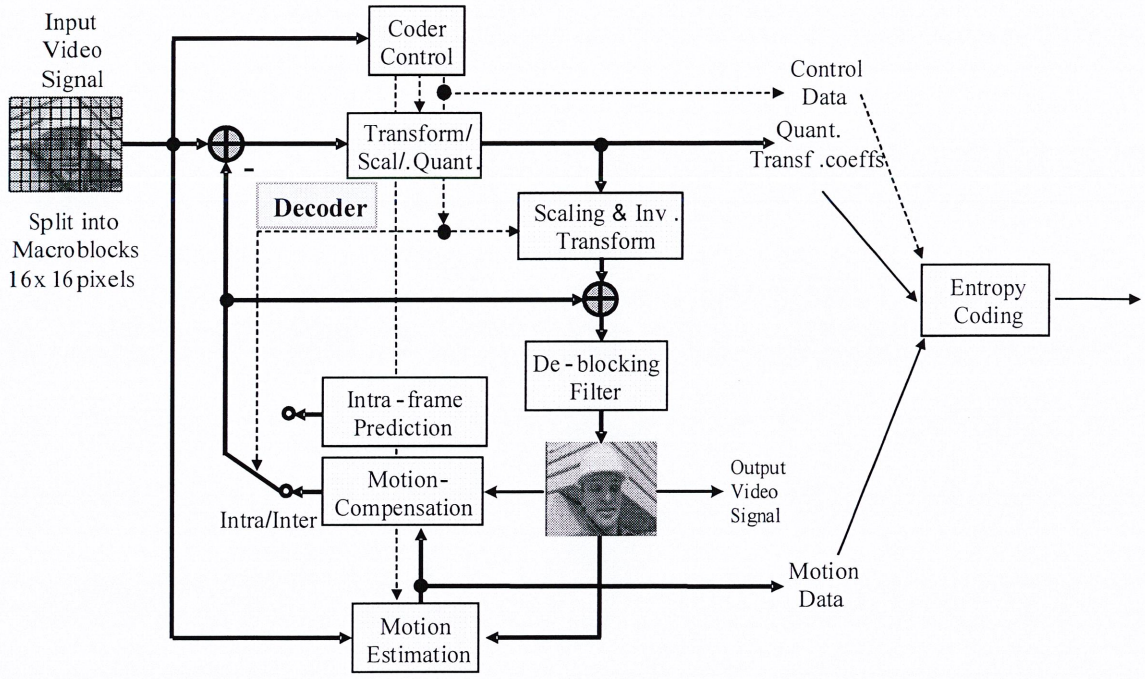
7) **Flexible Macroblock Ordering (FMO):** H.264/AVC standard offers a new feature of partitioning the video frame into independently decoded slices. Flexible macroblock ordering, if used effectively, can enhance the robustness to data losses by benefits from the spatial correlation between blocks in each slice.
8) **Redundant pictures:** H.264/AVC allows the encoder to send redundant representations of regions of pictures in case of losing the primary representations during data transmission.

All luma and chroma pixels of a macroblock are temporally or spatially predicted (Wiegand et al., 2003) and the resulting predicting residual is encoded using a transform coding. The transformed coefficients are then quantized and encoded using entropy coding method. Figure *4.2* shows the block diagram of the H.264/AVC CODEC. H.264/AVC added some new features such as intra-frame prediction and De-blocking filter (Wiegand et al., 2003). The encoder in Figure *4.2* includes two dataflow paths, a forward path (left to right) and a reconstruction path (right to left). The operation of the encoding process using one macro block of size 16×16 can be summarized as follows:

- **Encoder (Forward Path):** The input video frame is divided into macroblocks. Each macroblock is encoded using inter or intra

mode to estimate a predicted macroblock from the reference frame/s. The predicted macroblock is subtracted from the current macroblock to produce the residual macroblock. The residual macroblock is then transformed using block transform and quantized to give a set of quantized transform coefficients which are recorded and entropy encoded. The entropy encoded coefficients and the side information (prediction mode, quantizer parameter, actual motion vectors) are required in the decoder to reconstruct a representation of the original macroblock.

- **Encoder (Reconstruction Path):** As mentioned before, it is important for both the encoder and the decoder to use the same reference frame in order to avoid the accumulation of error in the reconstruction process which called drift problem (De Cock et al., 2008; Malvar et al., 2003). The problem results if non identical reference frame/s is used in both the encoder and the decoder. To avoid this problem, a decoder part is included in the feedback loop of the encoder as shown in Figure *4.2* to be sure that the encoding process uses the same distorted reference frame/s as the one used in the decoder part. A filter is applied to reduce the effects of blocking distortion.



(Figure 4.2)

H.264/AVC Encoder (Wiegand et al., 2003)

4.3 Main Motivations of H.265/HEVC Standard

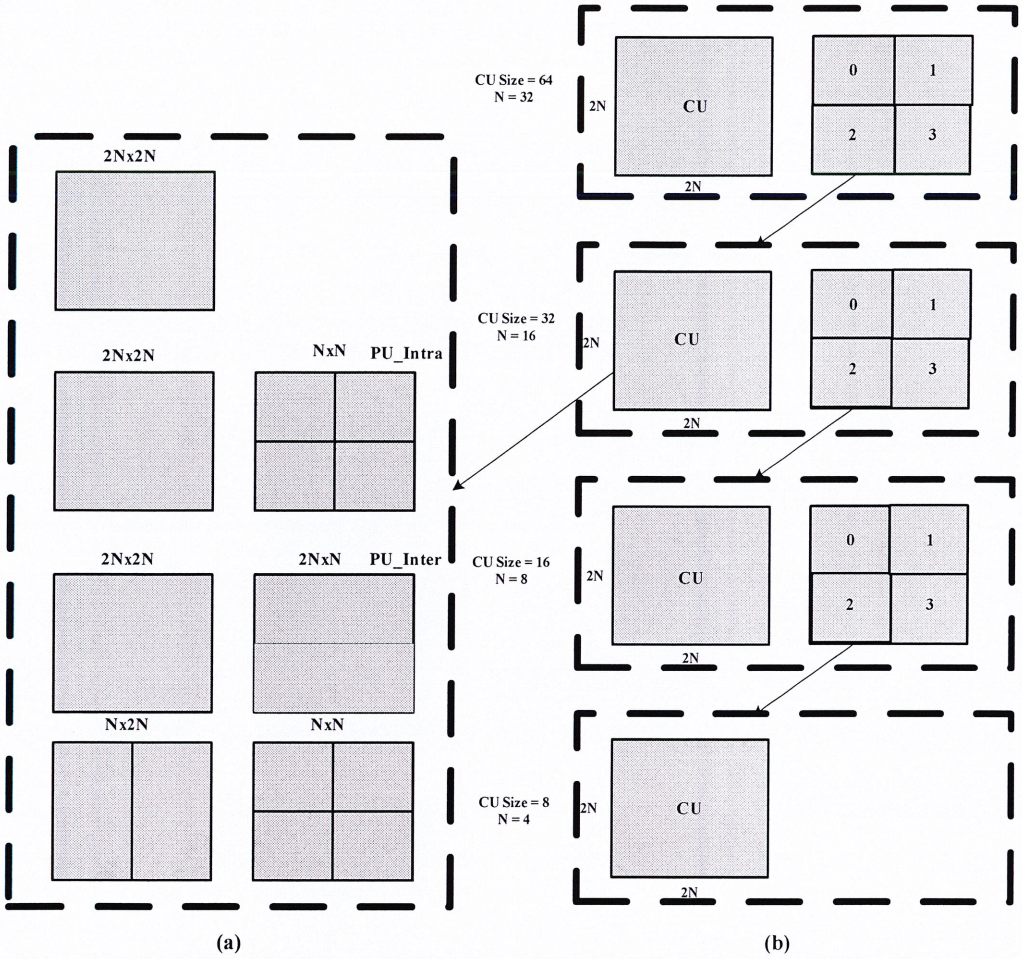
H.265/HEVC standard can encode high resolution videos with lower bit-rate compared to all previous standards. Also, it utilizes parallel processing operations in order to avoid delays that result from the new added components. The density of each pixel in 4K and UHD videos is now 10-bits (compared to 8-bits in H.264/AVC standard). On the other hand, the computational complexity of the H.265/HEVC encoder is increased compared to the H.264/AVC standard. Additionally, the computational time complexity of the Motion Estimation process increases dramatically as many new features are added in the H.265/HEVC standard compared to previous video coding standards. Most of the efforts have been done to speed up the Motion Estimation process which is consuming most of the encoding time. H.265/HEVC has

many applications such as wireless video surveillance system, broadcast of High Definition (HD) TV, Blu-ray Discs, Security applications, Camcoders, Mobile video, Digital cinema, and Video conferencing.

4.4 H.265/HEVC CODEC and Features

H.265/HEVC standard highlights some features that are discussed below (Ohm et al., 2012):

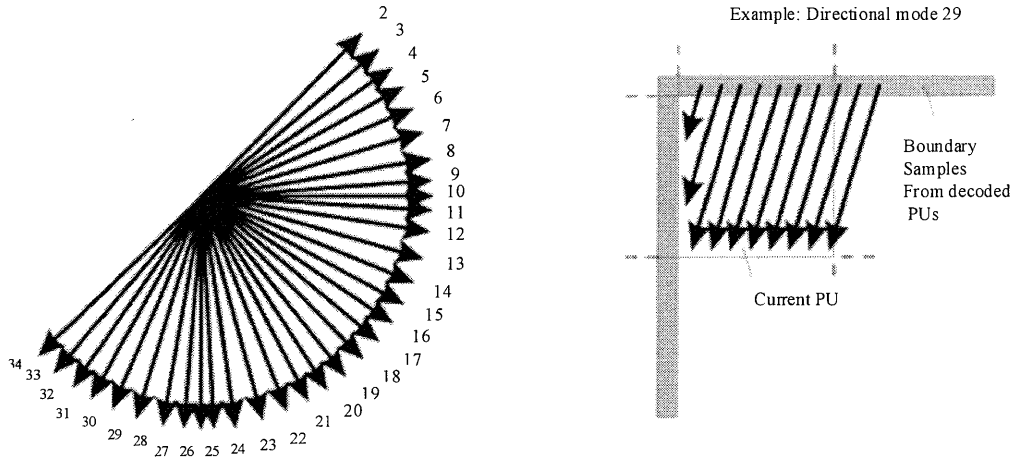
1) **Block-based coding:** H.265/HEVC standard increases the block size up to (64×64) compared to (32×32) in H.264/AVC standard. H.265/HEVC also uses Coding Tree Unit (CTU) structure which consists of three block concepts: Coding Unit (CU), Prediction Unit (PU), and Transform Unit (TU). CU luma block size is N×N, where N can be chosen as N=16 or 32 or 64 samples. H.265/HEVC supports variable PU sizes from 64×64 down to 4×4 samples. TU is defined for square block sizes; i.e., 4×4, 8×8, 16×16, and 32×32 block sizes. Figure 4.3 shows the coding tree unit used in the H.265/HEVC standard.



(Figure 4.3)

Coding Tree Unit (CTU) of H.265/HEVC (Ohm et al., 2012)

2) **Intra-Prediction in H.265/HEVC:** The Intra-prediction technique is currently used in H.265/HEVC and supports two directional methods: the arbitrary direction and angular intra-prediction. The Intra-prediction in H.265/HEVC defines 33 different directional orientations with block sizes from 4×4 up to 32×32. Figure 4.4 shows different directional modes. The main advantage of intra-prediction in H265/HEVC is that it sends low-compression frame in-between inter-predicted frames to enhance the resolution of the encoded frames.



(Figure 4.4)

Modes and directional orientations for intra-prediction (Ohm et al., 2012)

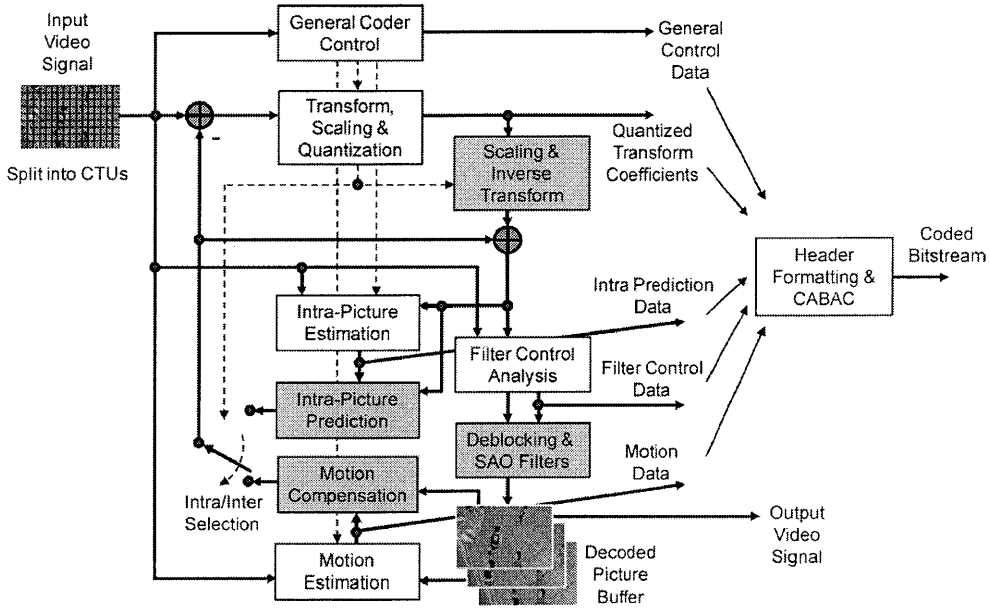
3) **Inter-prediction in H.265/HEVC:** The inter-prediction in H.265/HEVC uses reference frames stored in a memory to perform the ME process. Any search area should be selected using a frame index. And best match candidate block within such search area is represented using Actual Motion Vectors (AMVs). Both of the encoder and the decoder are synchronized by generating the same inter-prediction signals which are transmitted as side information. H.265/HEVC supports more Prediction Block (PB) partition shapes for inter-prediction CBs. The HEVC inter picture prediction consists of RBs and MVs for each inter coded prediction unit and performing quarter-sample accurate motion compensated prediction with a 2D separable interpolation filter. The quadtree segmentation of HEVC coding units allows also non-square inter coded prediction units to be processed. In the intra coding case, inter coded blocks are also transform coded and a set of various

post-filtering options can be applied to enhance the reconstruction further.

4) **Entropy coding in H.265/HEVC**: Context Adaptive Binary Arithmatic Coding (CABAC) is specified as the only entropy coding methods in H.265/HEVC (compared to two such methods in H.264/AVC: variable length coding and CABAC). H.265/HEVC encoder has many improvements in CABAC in order to speed up the parallel processing, increase the compression performance, and reduce memory requirements. The coding performance can be better parallelized and has higher throughput per processing cycle in software of hardware implementation than CABAC, although being very similar to it. It must be noted that the compression performance of this design can be significantly higher than the lower-complexity VLC.

5) **Motion Estimation in H.265/HEVC:** Interpolation of fractional-sample in H.265/HEVC is using 7-tap or 8-tap filters (compared to six-tap in H.264/AVC), and quarter-sample precision is used for AMVs. Multiple reference frames can be used in Motion Estimation exactly same as in H.264/AVC. Motion Estimation (ME) is the process of finding the Actual Motion Vector (AMV) that defines the transformation of a Current Block (CB) image (located in the current frame) from a Reference Block (RB) (located in the reference frame). FSBB-ME (Yeong-Kang & Lien-Fei, 2003) is the most popular ME algorithm. During the FSBB-ME algorithm, the current frame is divided into blocks, each of size N×N pixels. Each block searches for its best match RB in the search area located at the reference frame.

Figure *4.5* shows the block diagram of H.265/HEVC encoder considering the addition of some new features. Intra-prediction estimation, filter control analysis, and SAO filters are examples of such features. The huge compression ratio (up to 50% compared to previous H.264/AVC standard) is due to previous listed features. Figure *4.5* shows the same two data flow paths of H.2654/AVC are used in the latest standard H.265/HEVC (i.e. forward path (Encoder), and reconstruction path (Decoder)).



(Figure 4.5)

H.265/HEVC Encoder (Ohm et al., 2012)

4.5 Summary

H.265/HEVC is the latest standard that achieves approximately 50% bit-rate savings with equivalent video quality compared to the H.264/AVC previous standard. There are many new features of H.265/HEVC standard such as Quarter-sample precision, 7-tap or 8-tap filters, and uni-prediction or bi-prediction or weighted prediction coding technique. H.265/HEVC improves the coding efficiency (i.e., bit-rate and

video visual quality) by adding some new features such as block-based coding which increases the block size up to 64×64, also using Coding Tree Unit (CTU) structure in ME process, and process intra-prediction with 33 different directional orientations. Adding these features to the encoder greatly increases the computational complexity of the encoding process. Consequently, the speed of the encoding process will slow down. The most exhaustive processing-unit in the H.265/HEVC video CODEC is the Motion Estimation (ME) unit which consumes up to 84% of the total encoding time (Grellert et al., 2013; Kim et al., 2012). In this thesis, several techniques are implemented in order to enhance both the computational complexity and the encoding speed.

Chapter 5

MOTION ESTIMATION CO-PROCESSOR ARCHITECTURE-1

5.1 Introduction

5.2 Data Reuse Principle

5.2.1 Architecture-1: An Overview

5.2.2 Architecture Components

5.2.3 Data Flow and Process Description

5.3 Simulation and Implementation Results

5.3.1 VHDL Verification

5.3.2 FPGA Implementation

5.4 Summary

Chapter 5 MOTION ESTIMATION CO-PROCESSOR ARCHITECTURE-1

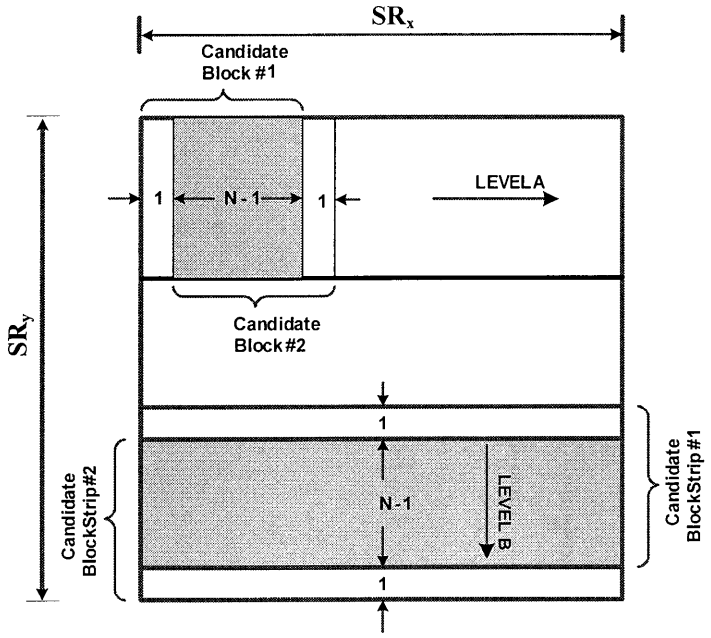
5.1 Introduction

In the following sections, data reuse principle is discussed in details. Additionally, the whole architecture of the implemented ME co-processor will be discussed and elaborated in details. It is worth mentioning that using the data reuse principle in our ME co-processor can help in reducing the I/O memory BW.

5.2 Data Reuse Principle

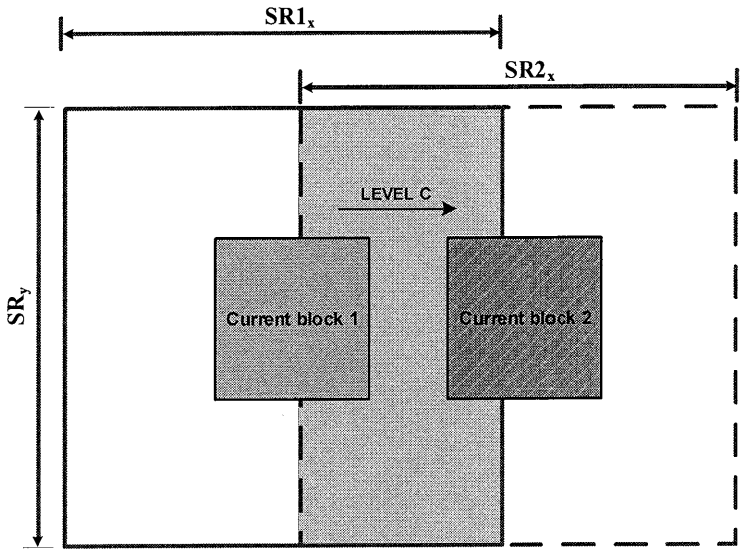
High quality video transmission requires high memory BW due to the using of bigger frames and search area sizes (Jen-Chieh et al., 2002). Data reuse principle has been implemented successfully to reduce the memory BW required for the search area while performing the ME process (Jen-Chieh et al., 2002). There are four levels of data reuse that may provide maximum BW efficiency. Figure *5.1* shows the first level defined as Level A data reuse. This level can be represented by pixels of any two following blocks that are overlapped in the search area. Figure *5.1* shows that candidate block #2 data is overlapping with candidate block #1 data, so that the overlapped pixels will be accessed only one time from the memory. The second level is called Level B data reuse. It can be represented by huge overlapped pixels in two vertically adjacent strips. Figure *5.1* shows that there are N-1 candidate block strip can be fetched from the memory only one time. The third level is called Level C data reuse. Figure *5.2* shows that the data reuse is described horizontally

within two different search areas belonging to two consecutive current blocks. The last level of data reuse is called Level D data reuse. Figure 5.3 describes the data reuse in level D as data is frequently reused vertically through already loaded two search areas belonging to two different current blocks (Jen-Chieh et al., 2002). It is worth mentioning that the first two data reuse levels (Level A and level B) are implemented in this thesis due to their hardware implementation simplicity and efficient decrease in the I/O memory BW.



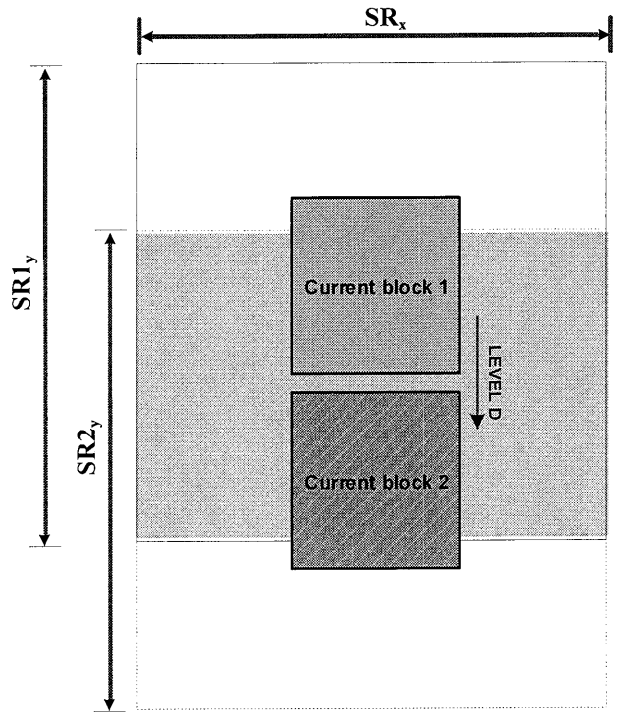
(Figure 5.1)

Level A and level B data reuse (Ismail et al., 2014)



(Figure 5.2)

Level C data reuse (Ismail et al., 2014)



(Figure 5.3)

Level D data reuse (Ismail et al., 2014)

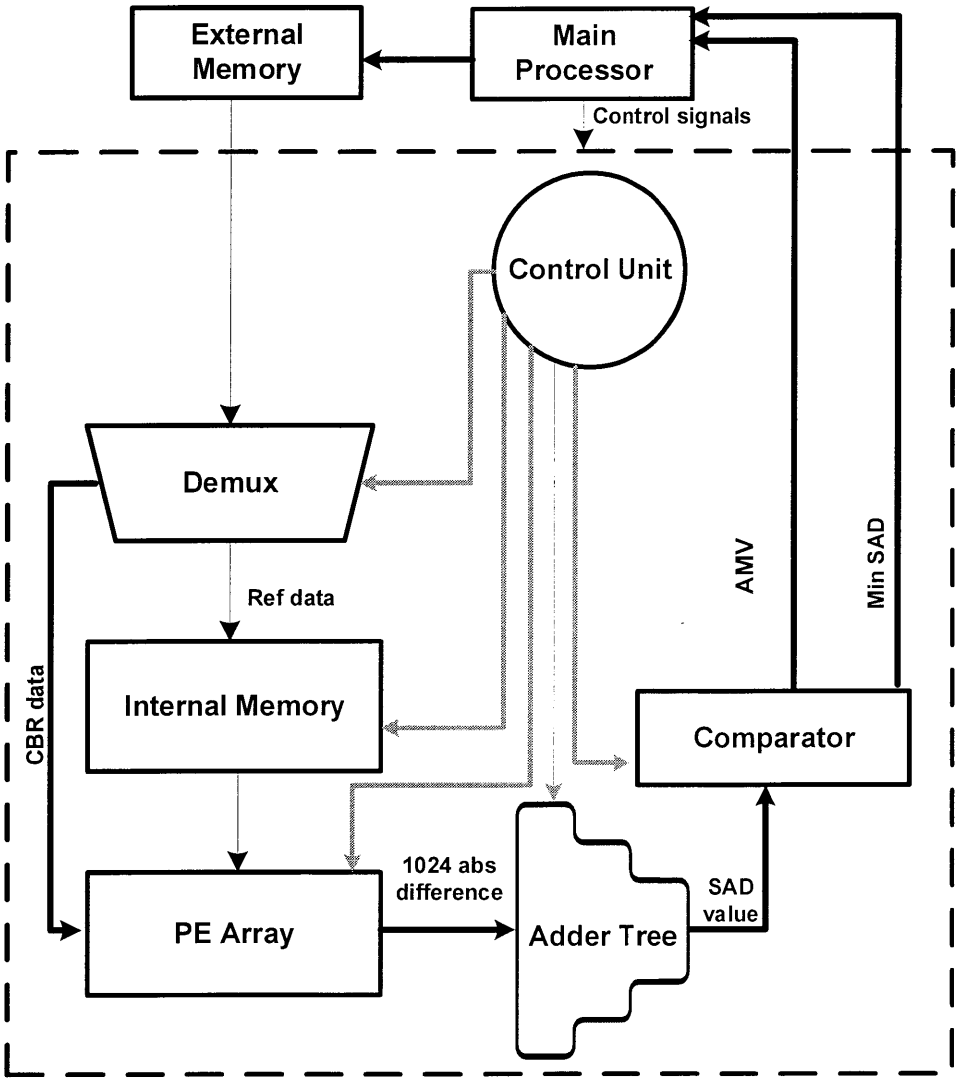
5.2.1 Architecture-1: An Overview

We have implemented in this chapter an efficient ME co-processor architecture suited for the H.265/HEVC video CODEC that uses a smart data reuse level A and level B. The main purpose of the implemented architecture in this chapter is to decrease the I/O memory BW. Consequently less data can be fetched from the main memory. This will speed up the ME process by decreasing the memory access and waiting time for getting the search area data required for the ME process. The implemented architecture uses a CB of size 32×32 pixels which is recommended by H.265 Standard (ITU-T & JVT, 2012). The size of the search area (fetched from the main memory) is 64×64 pixels which is compatible with UHD 4K video sequences (Zhi et al., 2005). Figure *5.4* shows the top level of the implemented architecture of the ME co-processor.

The Processing Element (PE) array consists of a 32×32 PEs. They compute the (AD) values between the Current Block and a candidate block from the search area. To reduce the memory BW required for the search area while performing the ME process, a small local memory is used for the purpose of data reuse. The local memory stores the pixels of the search area using the data reuse principle. The Sum of the 1024 (SAD) is calculated in parallel using the adder tree. The final SAD value is sent to a compare unit to estimate the current minimum distortion (SAD_{min}) between the CB and all candidates in the search area and its corresponding Actual Motion Vector (AMV). The controller generates all the necessary control signals required for all the modules in our ME co-processor.

The main processor stores the current and the reference frames in an external memory. The ME co-processor stores both of the current and the

reference blocks as well as the search area. The ME process starts by fetching the CB pixels and the search area from the external memory to the local registers of PE array and the local memory, consequently. The CB and the search area are loaded using smart mechanism for maximizing the coding performance. The implemented ME co-processor will accomplish the following requirements: (1) data reuse to reduce the memory BW, and (2) smooth and regular data flow.



(Figure 5.4)

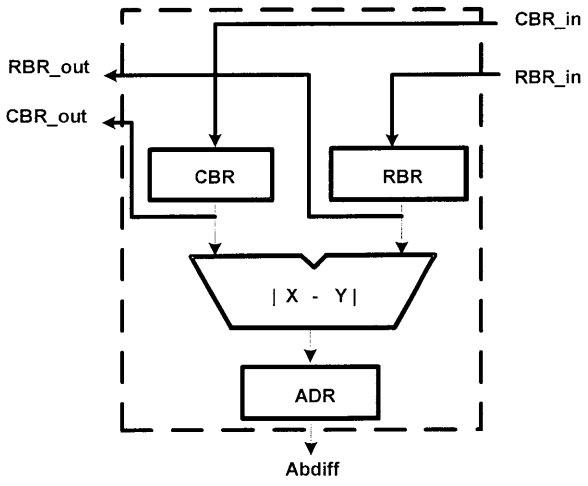
Top level architecture of the first implemented ME co-processor

5.2.2 Architecture Components

The main units of the implemented ME co-processor can be summarized as follows:

A. Processing Element (PE):

The PE contains the Absolute Difference (AD) unit, a Reference Block Register (RBR), and a Current Block Register (CBR) as shown in Figure 5.5. The main purpose of the PE design is to keep the CB data fixed and changing the candidate block coming from the search area using an intelligent data flow. Then calculate the AD between the CB and the candidate block. The CB data is loaded into the CBR register using the CBR_in terminal. The RB data is loaded to the RBR register using the RBR_in terminal. The Abs is calculated and stored in an Absolute Difference Register (ADR), once the data is available in CBR and RBR registers.



(Figure 5.5)

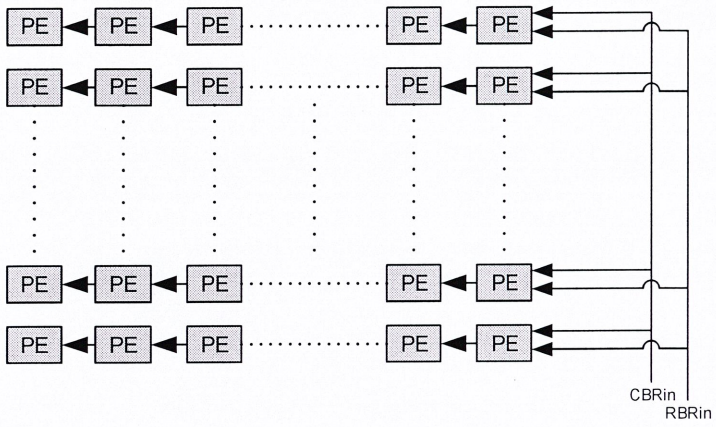
The Processing Element (PE) (Ismail et al., 2014)

B. Processing Element (PE) array:

The Processing Element (PE) array is responsible for calculating the Absolute Difference values between the current and the reference blocks. The PE array consists of 32×32 PEs, each PE is responsible for computing the AD between a pixel from the CB and a pixel from the Reference Block located in the search area. In the implemented ME co-processor, the CB remains the same whereas the RB data moves horizontally from right to left. The Pre-load time is defined as the required time to load all CBRs and RBRs in the PE array with the CB and the candidate block, respectively, to begin the ME operations. Referring to Figure *5.4* and Figure *5.6*, during the first 32 cycles, the de-multiplexer output will feed the PE array with the CB pixels. During the pre-load time, the CB data will be pushed horizontally from right to the left direction into the PE array in forms of 32 pixels column by column. In the next 32 clock cycles, the Reference Block (candidate block) is pushed into the PE array exactly like the CB, but in this time the de-multiplexer output will be connected to terminal output #1. Figure *5.6* shows the data flow of both the CB and the RB into the PE array. Connecting each PE only to its immediate neighbor therefore shorter interconnects are obtained. This adds many advantages such as: smaller delays, less area, low cross-talk noise, and less power consumption. In the implemented ME co-processor, each register data in the search area is accessed only once. A small internal memory is used to perform data reuse principle. Data reuse is achieved within a row of RB, by pushing new 32 pixels column data every clock cycle using RBR in terminal in Figure *5.6*. Before data is

written into the RBR registers, the old data is shifted to the left sideways. With this method the data in the PE array is reused.

Using a [-32, 31] search window, each new 32 pixel row of reference blocks takes 64 clock cycles to finish.



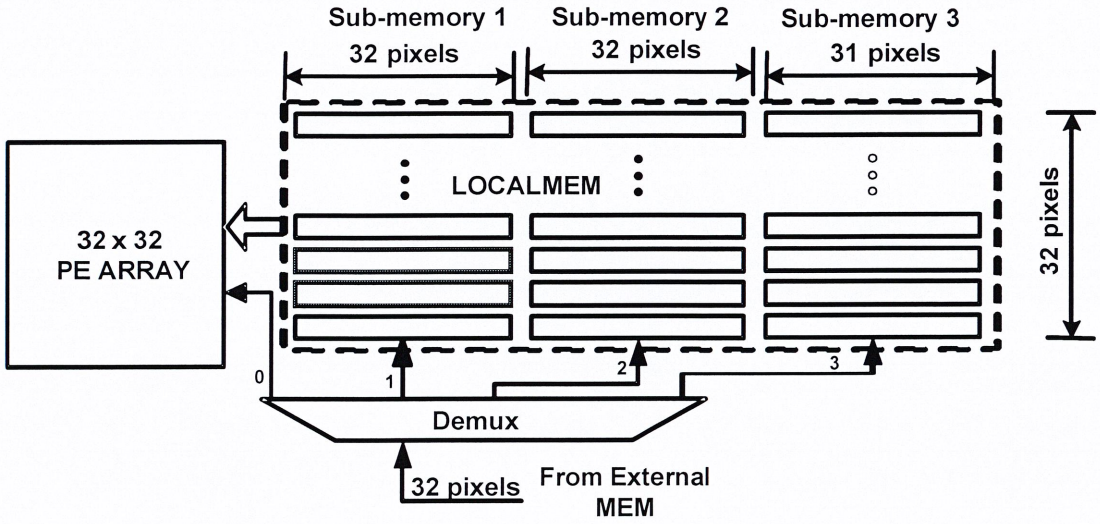
(Figure 5.6)

Dataflow of the search area data in the PE array (Ismail et al., 2014)

C. Local memory:

The data reuse is achieved by using the local memory unit. It is used to store the search area data considering the reduction of the I/O memory BW. Additionally, only one external memory access is required for each search area. Figure *5.7* shows the used architecture for the local memory connected to the 32×32 PE array. The local memory consists of: (1) De-multiplexer (DEMUX) and the sub-memory units. Each sub memory consists of 32×32 pixels. Each pixel is represented by 10 bits as this is defined in H.265 standard (ITU-T & JVT, 2012). Consequently, each sub memory consists of 32×32 10 bits registers as seen in Figure *5.7*.

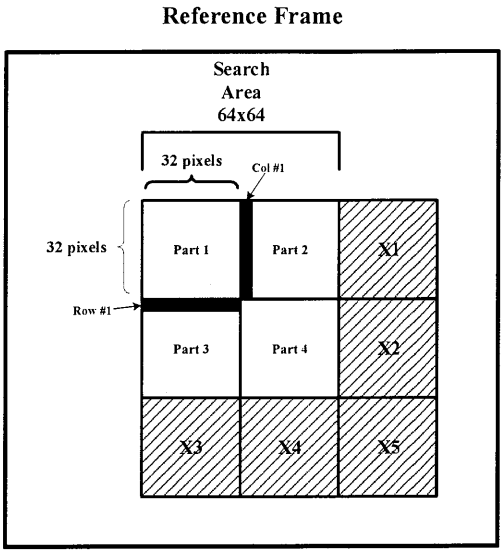
Output terminal 0 of the DEMUX is used to pass the CB pixels values from the external memory to the PE array. Output terminals 1, 2, and 3 of the DEMUX are used to pass the search area to sub memories 1, 2, and 3, respectively. A 320-bits (32 pixels wide) data bus is used to move the data from the external memory. A 32 pixels row of data is pushed from the bottom row registers using the de-multiplexer while existing data is shifted to top coming from external memory. Data is loaded to the PE array as 32 pixels columns. The addressing needed to read/write from/into the local memory is very simple compared to other architectures in (Goel et al., 2005; Jen-Chieh et al., 2002).



(Figure 5.7)

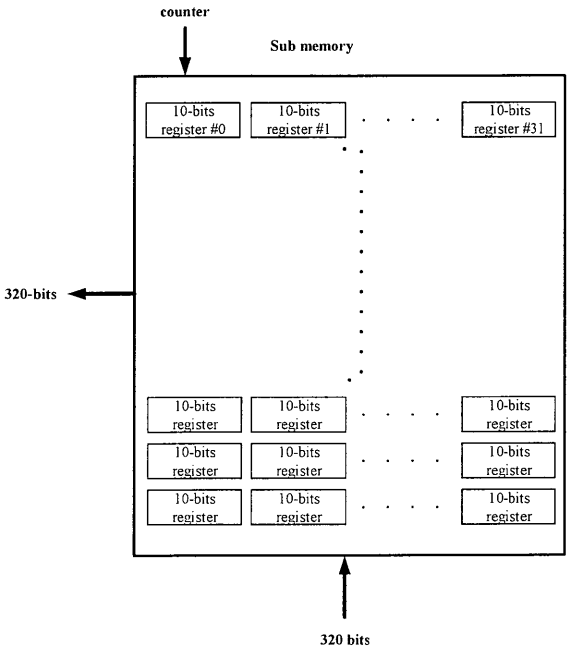
The architecture of the local memory

Figure *5.8* and Figure *5.9* show the 64×64 search area used for the ME process in the H.265/HEVC standard. The data flow can be summarized as follows.



(Figure 5.8)

The search area needed for a 32×32 current block



(Figure 5.9)

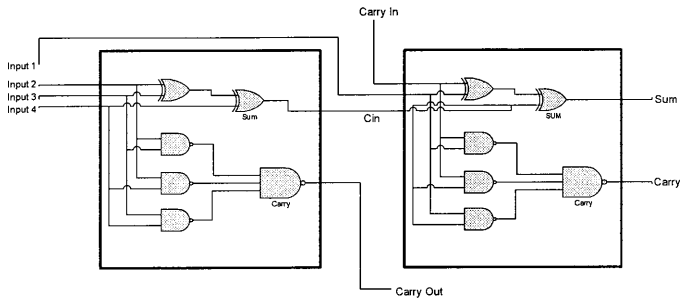
The sub-memory architecture

The pre-load time is 96 clock cycles. First 32 clock cycles are needed to fill the PE array with the CB pixels values via terminal 0 of the

DEMUX as seen in Figure *5.7*. During the second 32 clock cycles, sub-memory 1 is filled via terminal 1 of the DEMUX. During the last 32 clock cycles of the pre-load time, all values inside sub-memory 1 will be installed inside the PE array. And at same time, sub-memory 2 will be filled as seen in Figure *5.7*. After that, sub-memory 3 is filled while loading the PE array with sub-memory 2 data column by column. Using the previous operations schedule, Level A data reuse is achieved.

D. Adder Tree:

The 1024 AD values coming from the PE array are added together to get the SAD value using an adder tree. Adder tree can perform additions in a parallel fashion to speed up the addition process. The main unit in the adder tree is the 4-2 compressor shown in Figure *5.10*. This unit is used to add 4-bits from different numbers A, B, C, and D at a time. The output of the compressor circuits are added in a final adder stage as seen in the following example.



(Figure 5.10)

4-2 compressor (Ismail et al., 2014)

Assume that we have four 4-bit numbers:

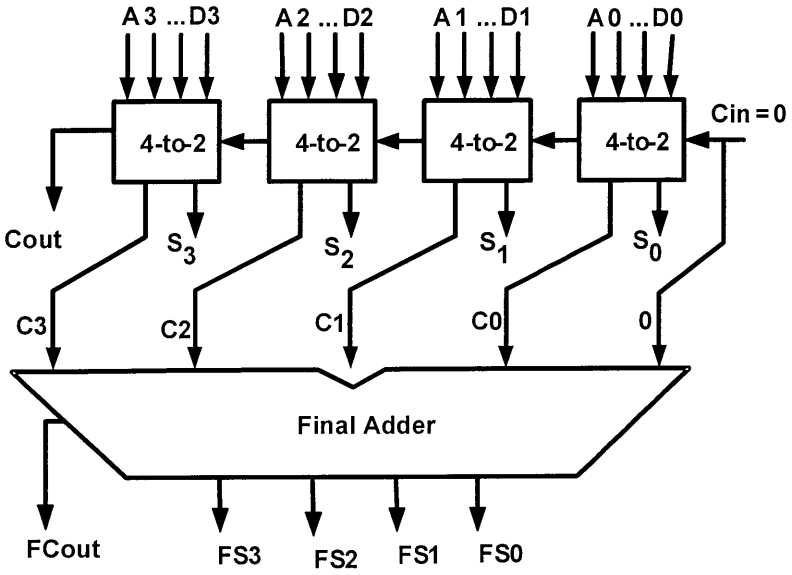
First: A3 A2 A1 A0

Second: B3 B2 B1 B0

Third: C3 C2 C1 C0

Fourth: D3 D2 D1 D0

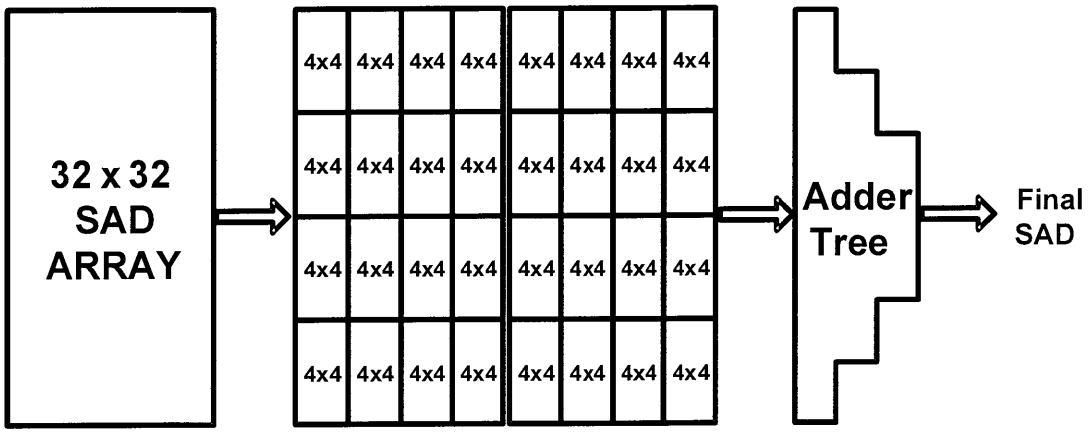
These four 4-bit numbers will enter to 4-2 compressors as seen in Figure 5.11. The Cout of first stage will be Cin for the next stage. The final summation will be calculated in final stage where all summation and carry outs will be added together using 5-bit adder.



(Figure 5.11)

Four 4-bit operand adder tree (Ismail et al., 2014)

The whole 32×32 AD values result from the PE array are divided into 32 4×4 array shown in Figure *5.12*. Each group uses the adder tree to add all of its AD values. Multiple parallel operations are performed at same time to get the final SAD value. 4×4 sub-array is selected to reduce the complexity of summation operation and achieve high speed operation. The more the size of the sub-array is, the slower the adder tree will be.



(Figure 5.12)

An illustration of the adder tree used in the implemented co-processor

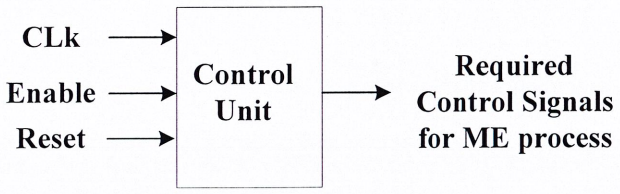
E. Control unit:

The control unit is responsible for producing all the required control signals of the designed ME co-processor. The control unit is considered to be the main and the most complex unit in the whole architecture. It controls all components in the implemented architecture. The control unit consists mainly of an up counter and control signals controller. The controller unit has only three inputs, which are enable, reset, and system clock as seen in Figure 5.13. As the name complies, the up counter is used to count the number of the clock cycles needed for a complete ME process. In the implemented architecture the search area is 64×64 which requires 13-bit counter to finish the whole ME process with 6116 clock cycles. The counter can be started using enable, reset, and system clock. The counter value should be reset every new ME process. This counter is used also to determine the position of the candidate block inside the search area. The control signals are generated based on the value of up counter. These control signals are used to control and initialize all components

of whole ME architecture. The control unit produces many control signals during the ME as shown in Table 5.1:

(Table 5.1)
Control signals

Control Signal Name	Purpose
selector	Demux selector to choose the required sub-memory
AVM_mem_en	To enable/disable AMV memory
mem1_en	To enable/disable Sub memory 1
mem1_en	To enable/disable Sub memory 2
mem1_en	To enable/disable Sub memory 3
comp_en	To enable/disable comparator unit
pos	To count for the AMV value



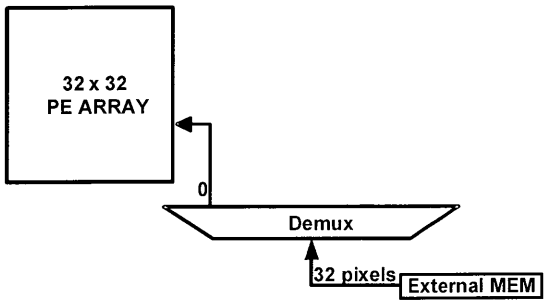
(Figure 5.13)

Control unit

5.2.3 Data Flow and Process Description

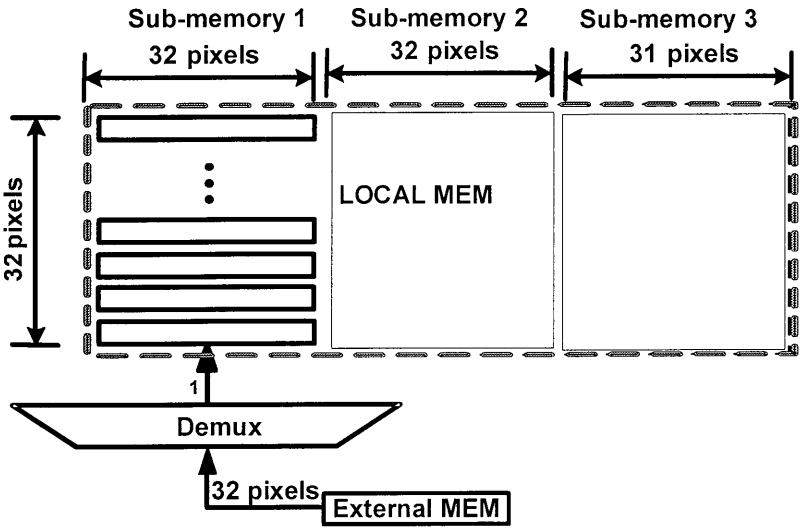
In this section, data flow and process description of the whole ME co-process will be discussed using one CB and one search area. The ME process starts by receiving a start control signal from the main processor. During the first 32 clock cycles of the pre-load time, the DEMUX output will be connected to the PE array via terminal #0 as seen in Figure *5.14*. The external memory starts loading the PE array with the 32×32 pixels values of the CB, column by column starting from the right column PEs.

Every clock cycle, 32 pixels values of the CB enter the right PEs column and shift from right to left direction as seen in Figure *5.14*. In the second 32 clock cycles, sub-memory 1 of the local memory will be filled by 32×32 pixels of the search area (part 1), as seen in Figure *5.8*, from the external memory and via terminal #1 of the DEMUX. Filling sub-memory 1 will be row by row from bottom to top direction as seen in Figure 5.15.



(Figure 5.14)

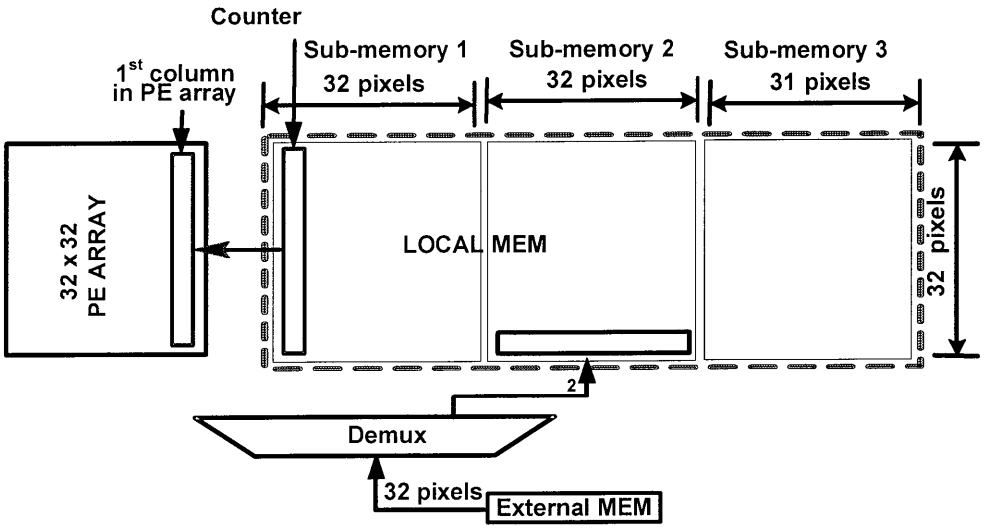
Loading the Current Block Register (CBR) in the PE array



(Figure 5.15)

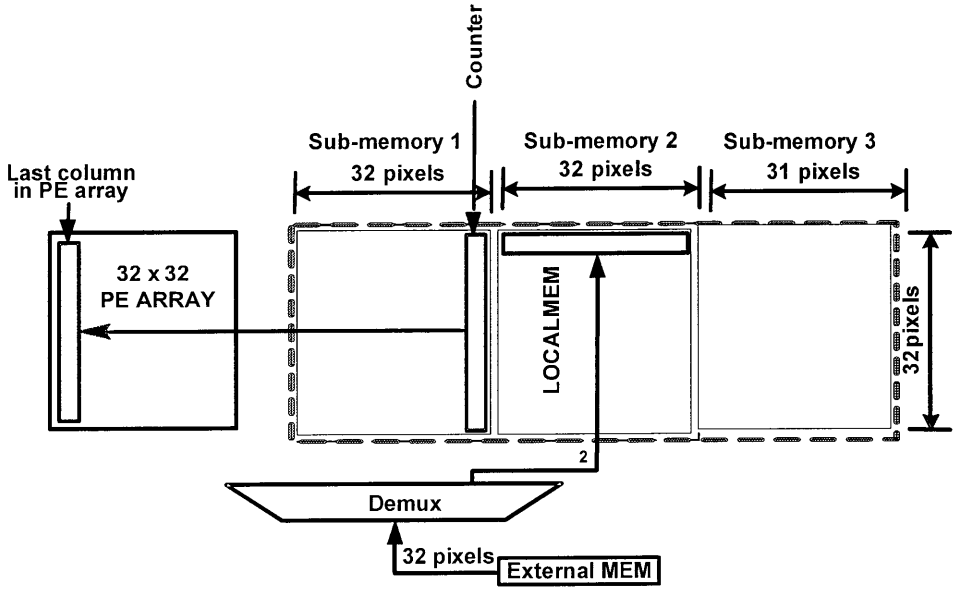
Loading the search area in the left 32 x 32 columns of the local memory

In clock cycle number 65, the PE array starts reading data part 1 from sub-memory 1 column by column and sub-memory 2 will start reading 32×32 search area part 2 via terminal #2 of the DEMUX as seen in Figure 5.16 and Figure *5.8*. An internal counter will be used to read column by column of sub-memory 1 into the PE array. The search area will be pushed inside the PE array column by column from the right to the left direction. By the end of clock cycle number 96, all sub-memory 1 contents are filled into the PE array and sub-memory 2 is filled by Part #2 of the search area as seen in Figure *5.8* and Figure 5.17. Two additional clock cycles are needed for getting the SAD value and one for the compare unit to decide the minimum SAD value. The PE array will start filling the left column of part 2 search area in Figure *5.8* in clock cycle 97. It is worth mentioning that every time new column enters the PE array, there will be a shift for all old candidate block values from right to left direction. Additionally, sub-memory 3 will continue filling part X1 of the search area in Figure *5.8* while the PE array loading previous candidate blocks of the first strip. This will achieve level A data reuse. It takes 159 clock cycles to fill the whole pixels of the first strip search area (part 1, part 2, and part X1 in Figure *5.8* in the PE array.



(Figure 5.16)

Loading the search area on the right 32×32 columns of the local memory and start loading the most left column into the CBR



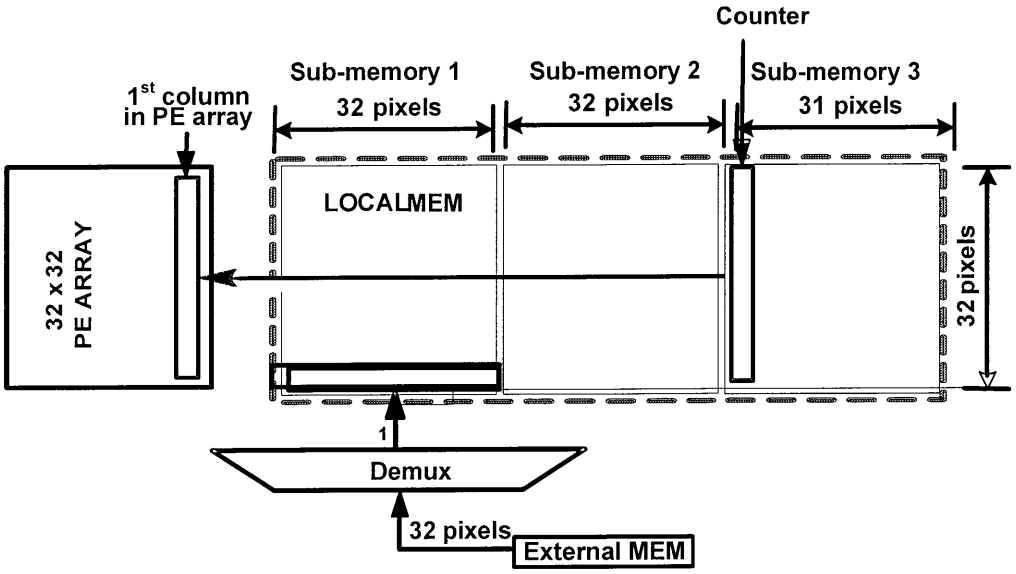
(Figure 5.17)

Loading the last raw in the right hand side of the local memory and loading the last column of the left hand side of the local memory into the PE array

To achieve level B data reuse, the top row of part 3 and part 4 of the search area, in Figure *5.8*, will be pushed to the bottom of sub-memory 1 and sub-memory 2 respectively, while filling sub-memory 3 into the PE array as seen in Figure 5.18 and Figure 5.19.

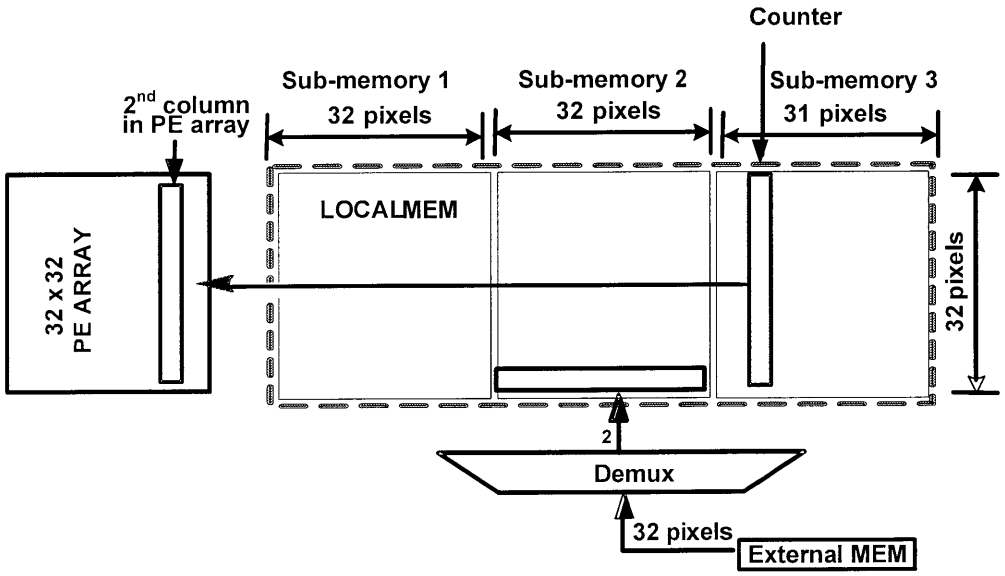
At clock cycle number 97 the PE array gives 1024 Absolute Differences

(ADs) to be summed using the adder tree. At clock cycle # 98, the first left column of sub-memory 2 will be loaded into the right column of the PE array as seen in Figure 5.20. At same clock cycle, sub-memory 3 starts filling part X1 as seen in Figure *5.8*. The SAD value will be calculated in clock cycle 98, the adder tree will calculate the SAD value and transfer it to the compare unit and first column of sub-memory 2 will be pushed to the PE array to achieve level A data reuse, see Figure 5.20. I in clock cycle number 136, the compare unit is done be calculating (SAD_{min}). Each sub-memory needs 32 clock cycles to be filled. Level B data reuse is achieved by pushing new row to sub memories 1, 2, and 3 respectively. In clock cycle number 133 also 32 pixels will fill the bottom row of sub-memory 1. In this case all sub-memory 1 values will be shifted upward by one row to achieve level B data reuse. After filling the contents of sub-memory 3 in the PE array, new candidate value of group 1 starts to enter the PE array. Operations will be repeated until all pixels in the search area are loaded into the PE array. In this design, sub-memory 1 and sub-memory 2 takes 32 clock cycles to read column by column. Sub-memory 3 only takes 31 clock cycles to fill 31 columns into the PE array.



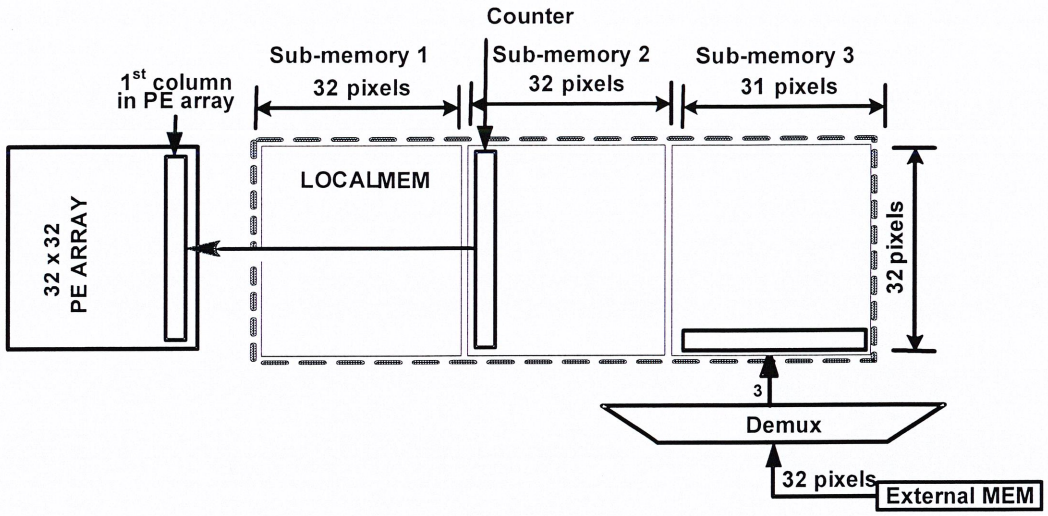
(Figure 5.18)

Loading new row search area into the left 32×32 columns of the local memory and start loading the right 32×32 columns into the PE array



(Figure 5.19)

Achieving level B data reuse for the right 32×32 columns of the local memory



(Figure 5.20)

Loading new row search area into the left 32×32 columns of the local memory and start loading the right 32×32 columns into the PE array

To achieve level B data reuse, data is pushed to sub-memories row by row while level A data reuse is achieved by reading columns of each sub-memory. 32 clock cycles are needed to fill sub-memory 2 while reading sub-memory 1. Additionally, 32 clock cycles to read from sub-memory 2 while writing sub-memory 3 row by row. Finally, 31 clock cycles are needed to read from sub-memory 3, while writing the bottom row of sub-memory 1 and sub-memory 2 (level B data reuse) as seen in Figure 5.18 and Figure 5.19. The previous operations take 95 clock cycles. Those 95 clock cycles are repeated for 64 slice rows of the search area. Adding 32 clock cycles for loading the Current Block into the PE array, 32 clock cycles for loading the first candidate block inside the PE array, two clock cycles for getting the first SAD, two clock cycles for the compare unit, and one clock cycle for resetting all registers at the beginning of the ME process. The total setup clock cycles are 36. The whole ME process in the architecture is taking (64 row search area) × (95

clock cycles for reading one slice using sub-memory 1, 2, and 3) + (36 setup clock cycles) which are 6116 clock cycles in total.

5.3 Simulation and Implementation Results

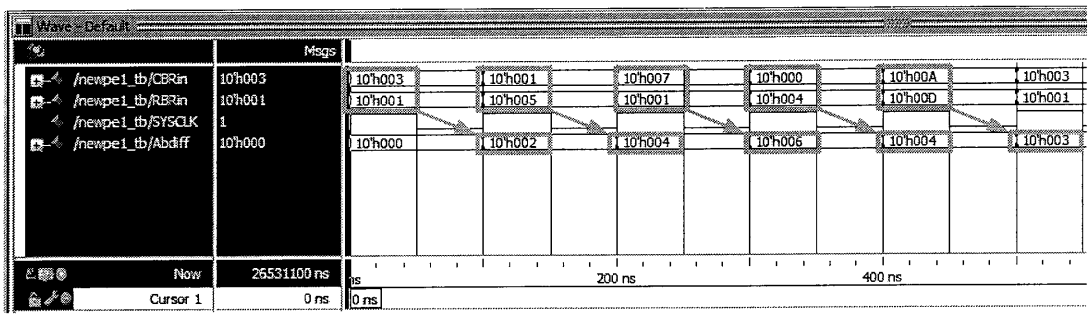
Frames are divided into blocks of size 32×32 pixels. The size of the used search area is 64×64 pixels. VHDL (Modelism-version10.4a) is used as a time verification tool for ME architecture-1. It is worth mentioning that the clock cycle period used in the Modelsim is 100 ns. Real frames are used as test benches with (288×352 , and 720×486 pixels). The hardware complexity of the implemented co-processor is calculated using Field Programmable Gate Array (FPGA) tool. The simulation and implementation results are summarized in the following subsections.

5.3.1 VHDL Verification

In this subsection we evaluated the validity and the time verification of all components of the implemented ME co-processor using VHDL tool.

A. Process Element (PE)

The AD unit calculates the difference between the two input pixels from the CB and the RB. The output of this unit goes as an input to the (ADR). Figure 5.21 shows the output of the PE for different inputs. The result is the AD between these two input values. The input values are (|3-1|=2), (|1-5|=4), (|7-1|=6), (|0-4|=4), and (|10-13|=3). It is worth mentioning that each PE takes one clock cycle to get an output.



(Figure 5.21)

Simulation of PE

B. PE Row

In this subsection, we are verifying the operation of one row of the PE array. 32 clock cycles are needed to enter 32 pixels data in both the current and the reference registers of each PE. The output will be calculated in clock cycle number 33 as shown in Figure 5.22. The following matrices show the value of CBR and RBR in one row of the PE array.

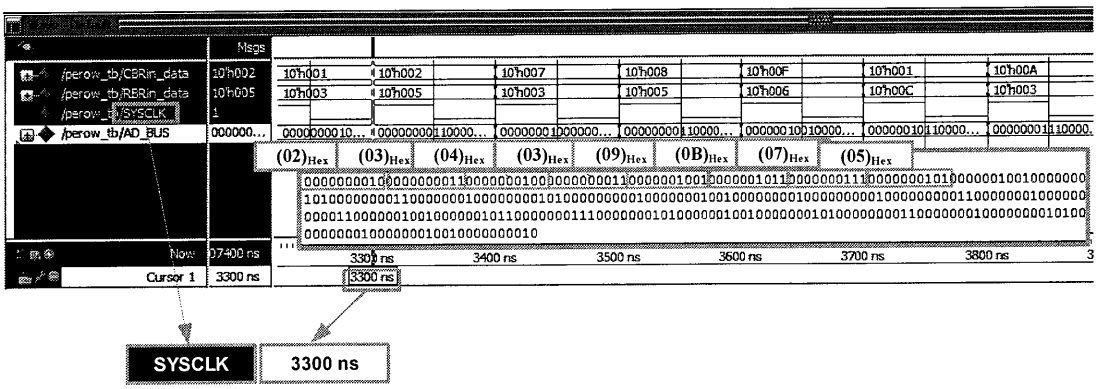
CBR
=[01 02 07 08 0F 01 0A 09 0B 08 01 09 0E 01 0A 04 01 02 07 08 0F 01 0A 09 0B 08 01 09 0E 01 0A 04]

RBR =
[03 05 03 05 06 0C 03 04 02 03 04 05 04 03 01 02 03 05 03 05 06 0C 03 04 02 03 04 05 04 03 01 02]

PE Row Output =
[02 03 04 03 09 0B 07 05 09 05 03 04 0A 02 09 02 02 03 04 03 09 0B 07 05 09 05 03 04 0A 02 09 02]

The below calculation is performed to get the output shown in Figure 5.22.

(|1-3|=2), (|2-5|=3), (|7-3|=4), (|8-5|=3), (|15-6|=9), (|1-12|=B), (|10-3|=7), (|9-4|=5), (|10-2|=9), (|8-3|=5), (|1-4|=3), (|9-5|=4), (|14-4|=A), (|1-3|=2), (|10-1|=9), and (|4-2|=2).



(Figure 5.22)

Simulation of PE Row

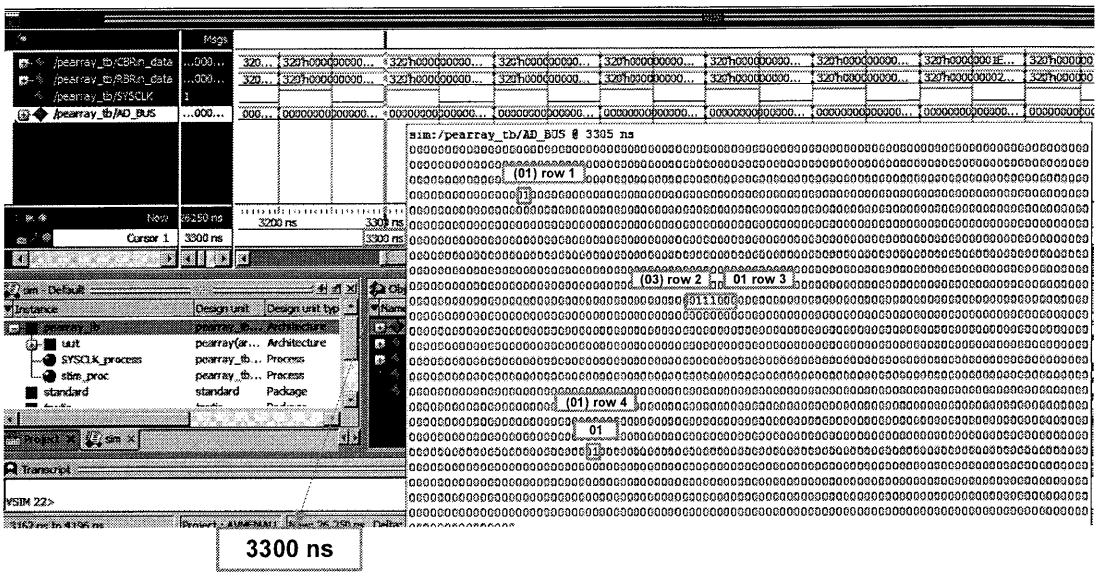
C. PE Array

Figure 5.23 shows the output of the PE array using the matrices below, the input data is only for one PE for each row to simplify the simulation results. The following matrices are 32×32 CB and 32×32 RB, the output matrix is calculated to show the absolute difference between the CB and RB matrices. 32 clock cycles are needed to enter 32 rows of data form right to left, and the output will be calculated in clock cycle number 33 as shown in Figure 5.23.

$$CB=\begin{bmatrix}00\ 00\ 00\ldots\ldots\ 00\ 02\\ 00\ 00\ 00\ldots\ldots\ 00\ 05\\ 00\ 00\ 00\ldots\ldots\ 00\ 04\\ 00\ 00\ 00\ldots\ldots\ 00\ 06\\ 00\ 00\ 00\ldots\ldots\ 00\ 0F\\ 00\ 00\ 00\ldots\ldots\ 00\ 02\\ 00\ 00\ 00\ldots\ldots\ 00\ 0F\\ 00\ 00\ 00\ldots\ldots\ 00\ 00\\ \vdots\\ 00\ 00\ 00\ldots\ldots\ 00\ 00\\ 00\ 00\ 00\ldots\ldots\ 00\ 03\end{bmatrix}$$

$$RB=\begin{bmatrix}00\ 00\ 00\ldots\ldots\ 00\ 01\\ 00\ 00\ 00\ldots\ldots\ 00\ 02\\ 00\ 00\ 00\ldots\ldots\ 00\ 03\\ 00\ 00\ 00\ldots\ldots\ 00\ 05\\ 00\ 00\ 00\ldots\ldots\ 00\ 03\\ 00\ 00\ 00\ldots\ldots\ 00\ 0A\\ 00\ 00\ 00\ldots\ldots\ 00\ 01\\ 00\ 00\ 00\ldots\ldots\ 00\ 00\\ \vdots\\ 00\ 00\ 00\ldots\ldots\ 00\ 00\\ 00\ 00\ 00\ldots\ldots\ 00\ 07\end{bmatrix}$$

$$\text{PE Array Output}=\begin{bmatrix}00\ 00\ 00\ldots\ldots\ 00\ 01\\ 00\ 00\ 00\ldots\ldots\ 00\ 03\\ 00\ 00\ 00\ldots\ldots\ 00\ 01\\ 00\ 00\ 00\ldots\ldots\ 00\ 01\\ 00\ 00\ 00\ldots\ldots\ 00\ 0C\\ 00\ 00\ 00\ldots\ldots\ 00\ 08\\ 00\ 00\ 00\ldots\ldots\ 00\ 0E\\ 00\ 00\ 00\ldots\ldots\ 00\ 00\\ \vdots\\ 00\ 00\ 00\ldots\ldots\ 00\ 00\\ 00\ 00\ 00\ldots\ldots\ 00\ 04\end{bmatrix}$$

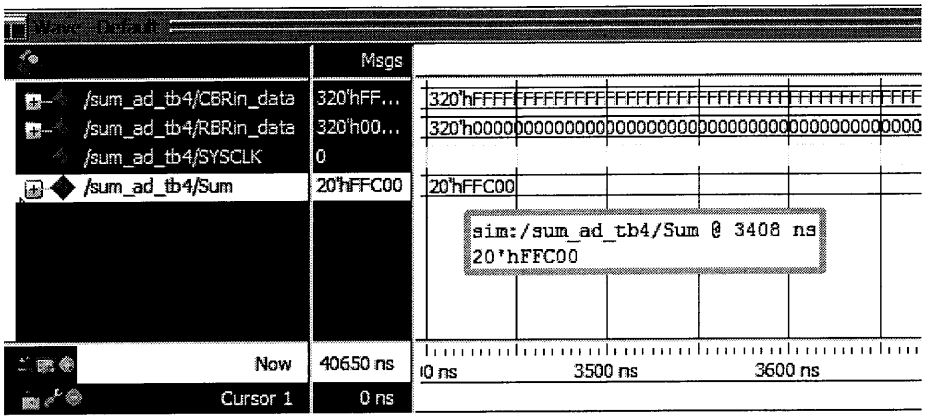


(Figure 5.23)

First Simulation of PE Array at 32 Clock Cycle

D. SAD Unit

Figure 5.24 shows the simulation results of the SAD unit. The output (Absolute Difference Values) of the PE Array will be sent to the SAD unit to calculate the SAD value. In this case, we select the values of each pixel in the CB matrix to be $(1023)_{10}$. The pixels of the RB matrix are selected to be 0's. The SAD unit will calculate the SAD between CB and RB matrices. In the first 32 clock cycles, PE array will be filled after that 2 more clock cycles are needed to get the final SAD value which is "FFC00" as shown in Figure 5.24, so the total clock cycles number for SAD is 34.

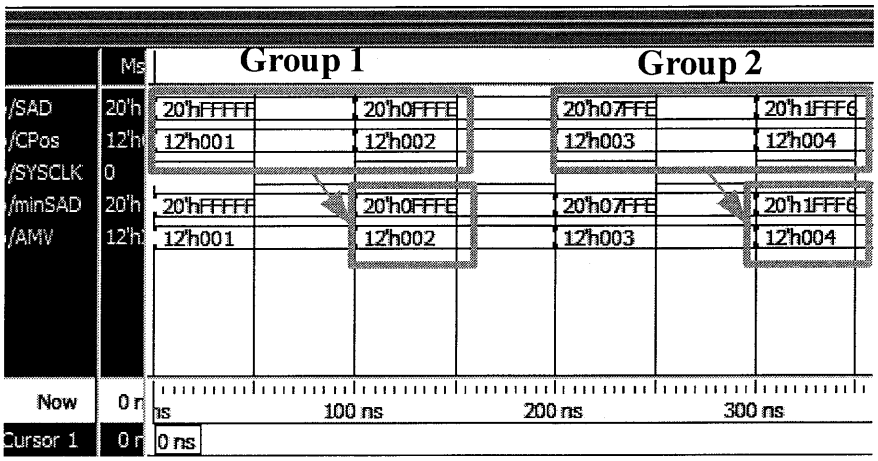


(Figure 5.24)

Simulation of SAD unit

E. Compare Unit

The compare unit is used to compare two SAD values and decide the minimum SAD value. The output of the compare unit should be the minimum SAD and its corresponding position in the search area. As an example, assume two groups of two SAD values group 1 {"FFFFF", "0FFFE"}, and group 2 {"07FFE", "1FFF6"} with the corresponding positions 1, 2, 3, and 4 respectively. As shown in Figure 5.25, the minimum value according to the input values of SAD in the first group is "0FFFE" at position number 2. And the minimum value according to the input values of SAD in the second group is "1FFF6" at position number 4. The minimum SAD and its position are stored in a register to be compared with the next SADs.



(Figure 5.25)

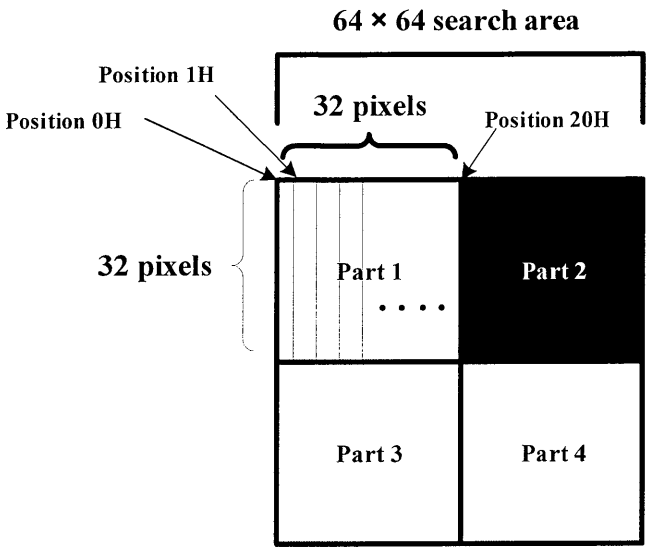
Simulation of Compare Unit

F. Full Processor Design Simulation

In this sub-section, we are going to elaborate the whole architecture of the ME co-processor given in Figure *5.4*. Six different cases are considered to verify the functionality of architecture-1. Followings are discussions about each case individually.

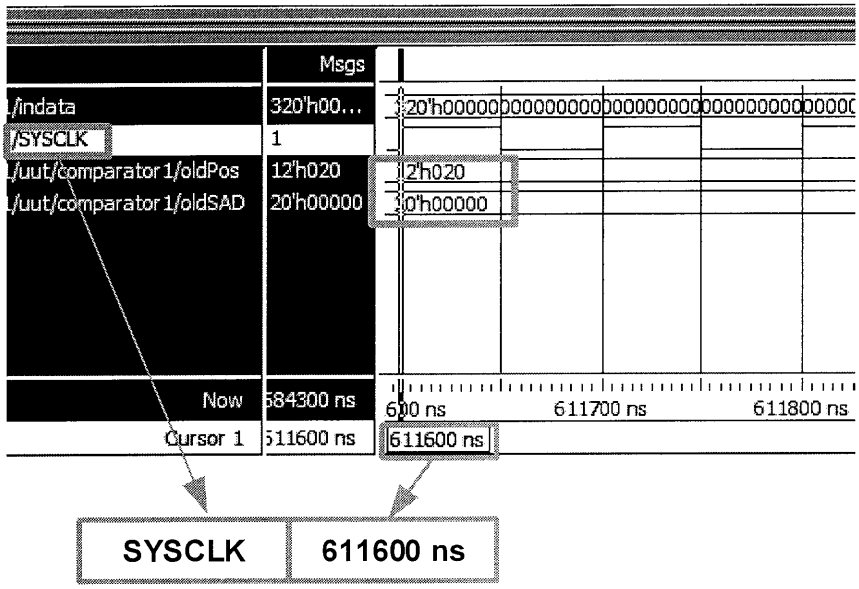
- **Full Design Simulation – Case 1**

In this case study, a simple test is performed in which the reference frame is all 0's (black background) except single 32×32 pixels block all 1's (a white block) located in position $(20)_{Hex}$ as see in Figure 5.26. The CB used for the ME process is equal to all 1's. So that the expected AVM should be allocated at position $(20)_{Hex}$ where the SAD value is $(00000)_{Hex}$ at clock cycle number 6116 as seen in Figure 5.27.



(Figure 5.26)

First reference frame with white background and 32×32 black block in position 20H

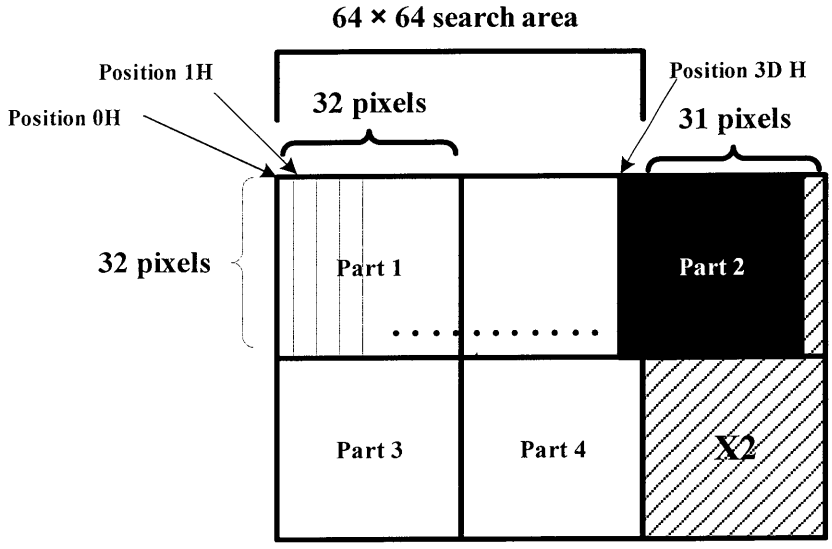


(Figure 5.27)

Simulation of complete ME process, case 1

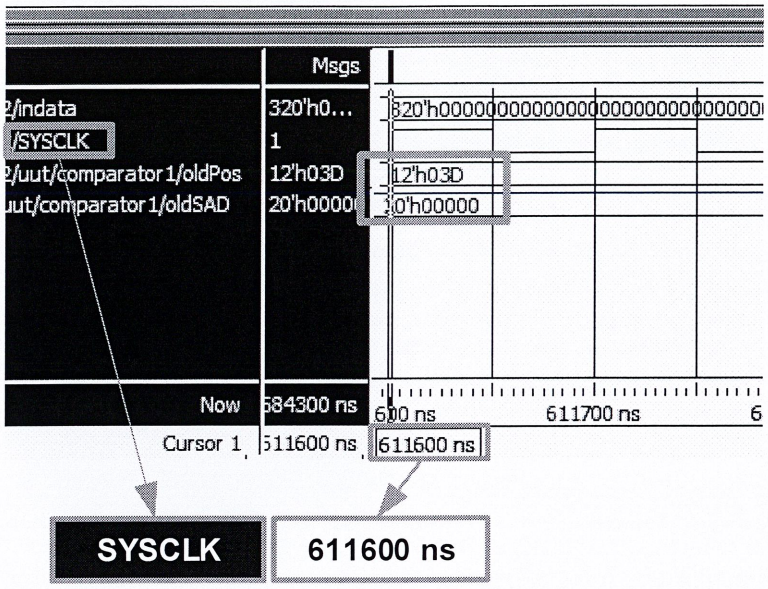
- **Full Design Simulation – Case 2**

This case is exactly same as the previous case with moving the 1's block in the reference frame to new location which is $(3D)_{Hex}$, as seen in Figure 5.28 and so that the expected AVM should be allocated at position $(3D)_{Hex}$ where the SAD value is $(00000)_{Hex}$ as seen in Figure 5.29.



(Figure 5.28)

First reference frame with white background and 32×32 black block in position 3DH



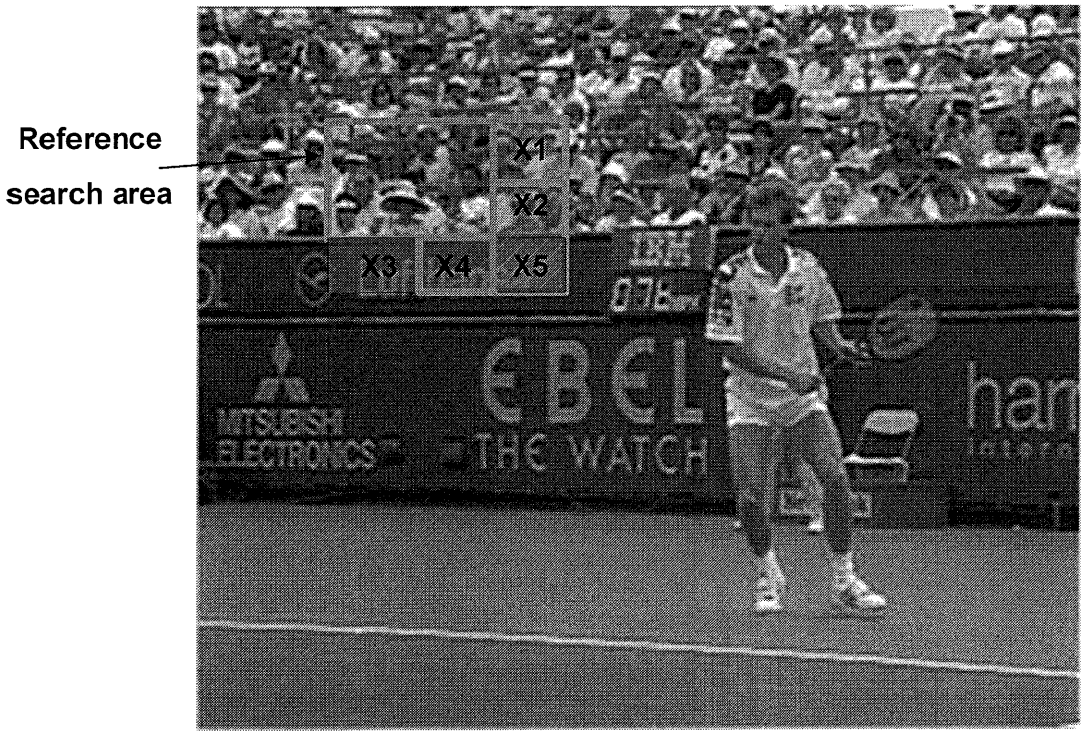
(Figure 5.29)

Simulation of complete ME process, case 2

- **Full Design Simulation – Case 3**

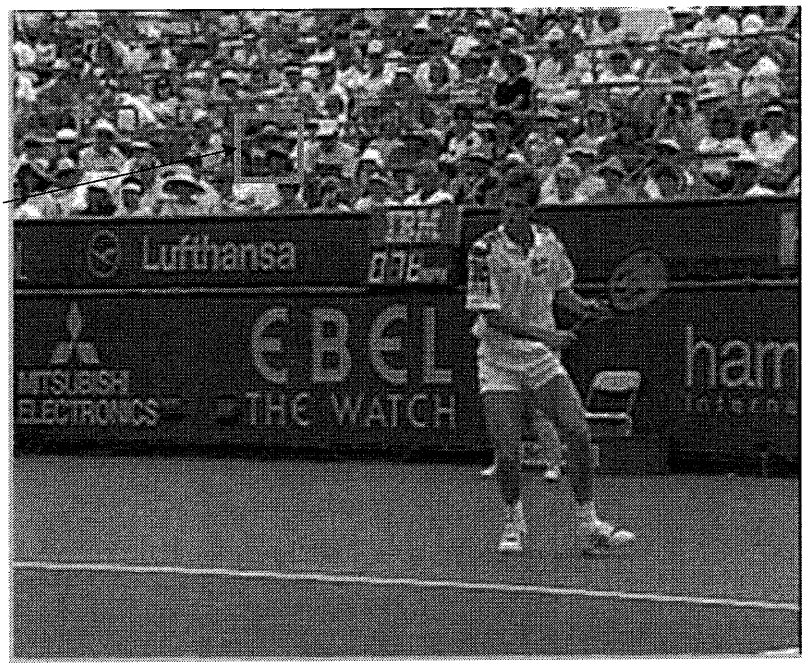
A real video (stefan.yuv with 288×352 pixels) is used in this case to show that the implemented architecture can work with real video sequences to perform the ME process. Figure 5.30 and Figure 5.31 show the reference frame and the current frame, respectively. Figure 5.32 and Figure 5.33 show the selected search area in the reference frame and current Block from the current frame, respectively. There are three steps done to get the final result. First, Matlab is used to get the input matrices (i.e. CB and RB) which are used in the Modelsim test bench. Second, VHDL is used to get the AMV as shown in Figure 5.34. Third, Matlab is used again to verify the obtained AMV by comparing the visual quality of the CB and obtained best match candidate block located at the calculated AMV as seen in Figure 5.35. The simulation

results shows that the best match is allocated at position number $(32)_{Hex}$ with a minimum SAD equal to $(1157B)_{Hex}$, see Figure 5.34. Figure 5.35 (a), the Current Block is matching the best match candidate block in Figure 5.35 (b) which is calculated using our ME co-processor. We also verified the obtained results by using FSBB-ME algorithm in Matlab 7.9.0 to calculate the best match block for the selected CB, and we found that it matches the VHDL results of our ME processor. It is worth mentioning that the FSBB-ME algorithm gives exact matching results as those seen in Figure 5.35. Figure 5.35 (c) shows best match block using FSBB-ME in Matlab, and as seen both Figure 5.35 (b) and (c) are almost the same.



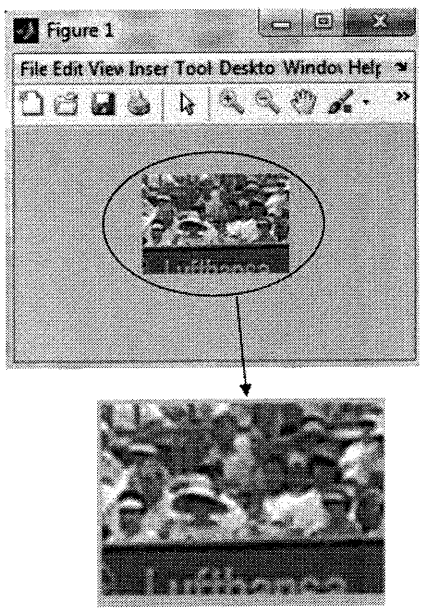
(Figure 5.30)

Reference frame (stefan.yuv frame number 0)



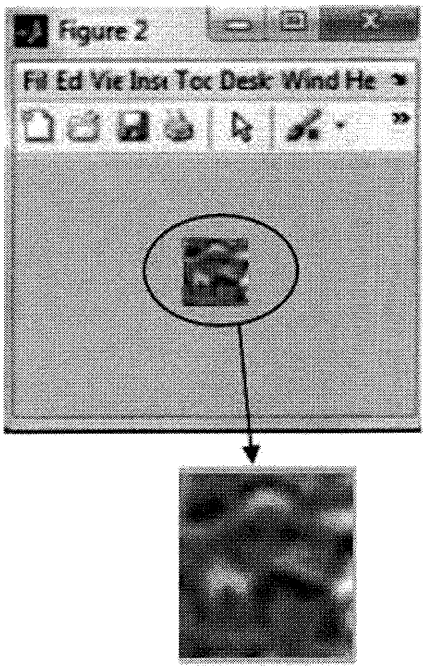
(Figure 5.31)

Current frame (stefan.yuv frame number 1)



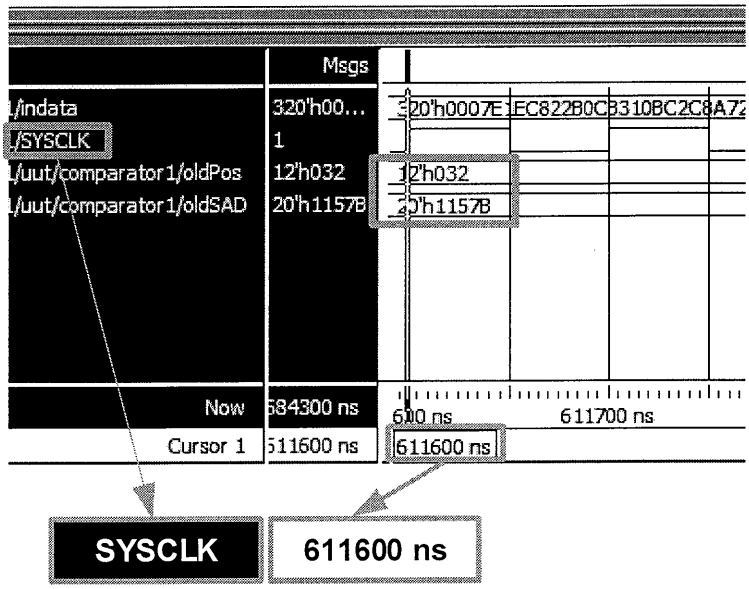
(Figure 5.32)

Reference search area



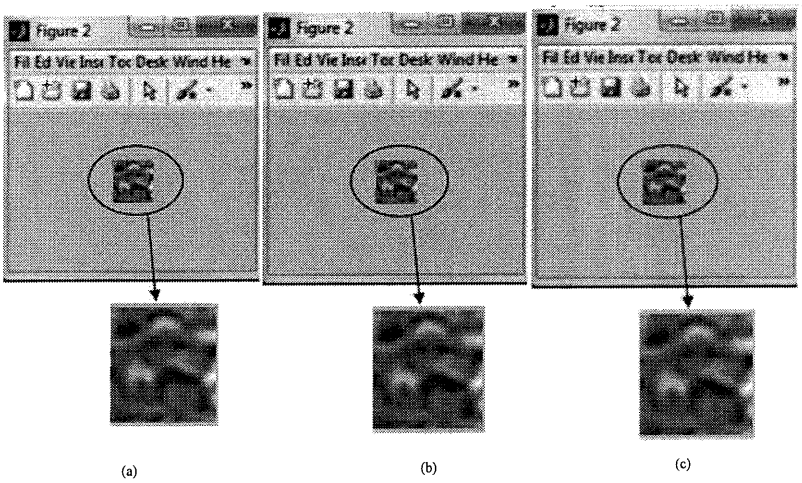
(Figure 5.33)

Current block



(Figure 5.34)

Simulation results of ME process

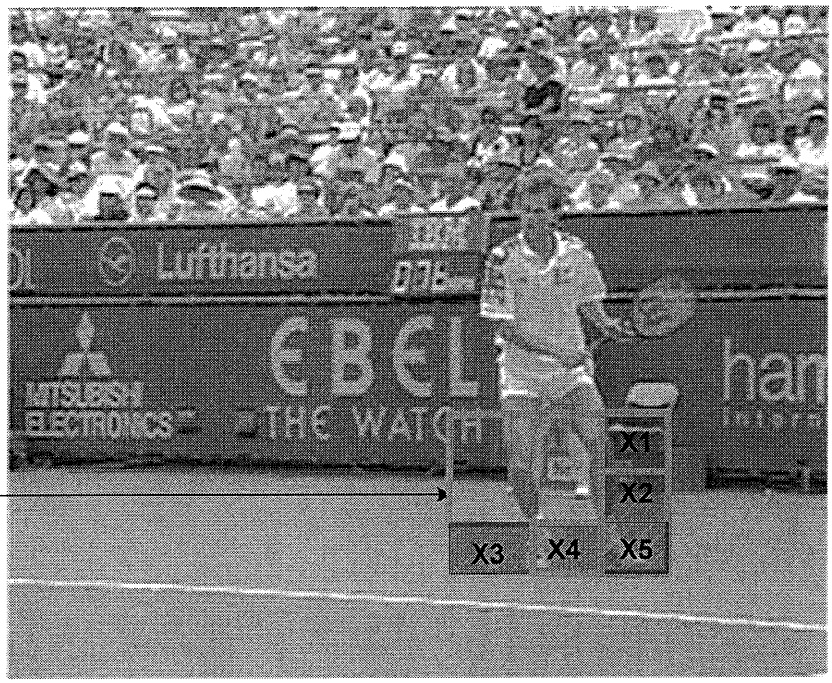


(Figure 5.35)

Comparing the result of the implemented architecture. (a) Current block. (b) Best match in the search area. (c)Best match using FSBB-ME in Matlab

- **Full Design Simulation – case4**

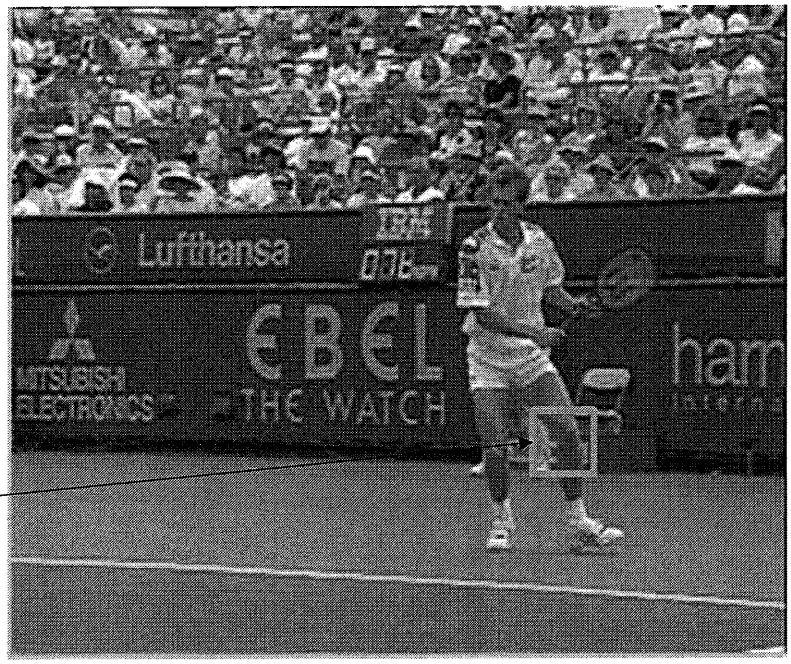
The same real video (stefan.yuv with 288×352 pixel) is also used in this case. Figure 5.36 and Figure 5.37 show the reference frame and the current frame, respectively along with reference search area and current block. Same three steps were done in this case to calculate and verify the ME co-processor results. The simulation results shows that the best match is allocated at position number (27)Hex with a minimum SAD equal to (08DAC)Hex, see Figure 5.38. Figure 5.39 (a), the Current Block is matching the best match block in (b) which is calculated using the ME co-processor. We also used FSBB-ME algorithm in Matlab 7.9.0 to calculate the best match candidate block for the selected CB and we found that it matches the VHDL results seen in Figure 5.38 and Figure 5.39. Figure 5.39 (c) shows best match block using FSBB-ME in Matlab, and as seen both Figure 5.39 (b) and (c) are almost the same.



Reference search area

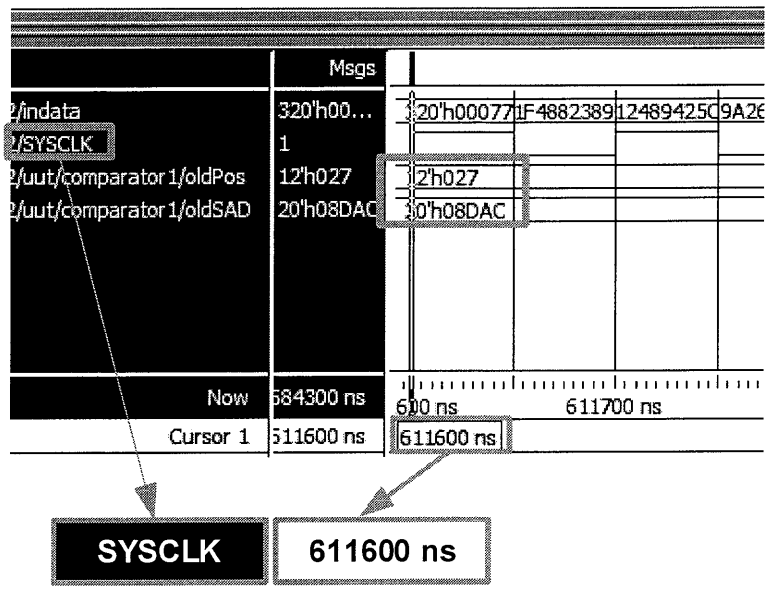
(Figure 5.36)

Reference frame (stefan.yuv frame number 0)



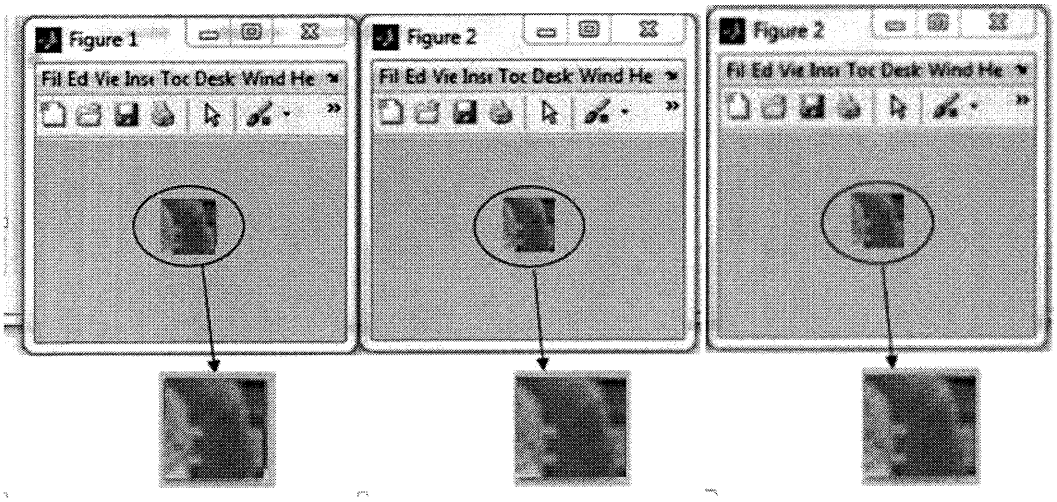
Current block

(Figure 5.37)
Current frame (stefan.yuv frame number 1)



(Figure 5.38)

Simulation results of ME process



(a) (b) (c)

(Figure 5.39)

Comparing the result of the implemented architecture. (a) Current block. (b) Best match in the search area. (c) Best match using FSBB-ME in Matlab

- **Full Design Simulation – Case 5**

A real HD video (ducks_take_off.yuv with 720×486 pixels) is used in this case to show that the implemented architecture can work with real video sequences to perform ME process. Figure 5.40 and Figure 5.41 show the reference frame and the current frame, respectively. There are three steps done to get the final result. The simulation results shows that the best match is allocated at position number $(BFE)_{Hex}$ with minimum SAD equal to $(10ACB)_{Hex}$, see Figure 5.42. Figure 5.43 (a) shows the selected Current Block and Figure 5.43 (b) shows the obtained best match candidate block using the architecture of our ME co-processor. Also, using FSBB-ME algorithm verify the results in Figure 5.43. Figure 5.43 (c) shows best match block using FSBB-ME in Matlab, and as seen both Figure 5.43 (b) and (c) are almost the same.



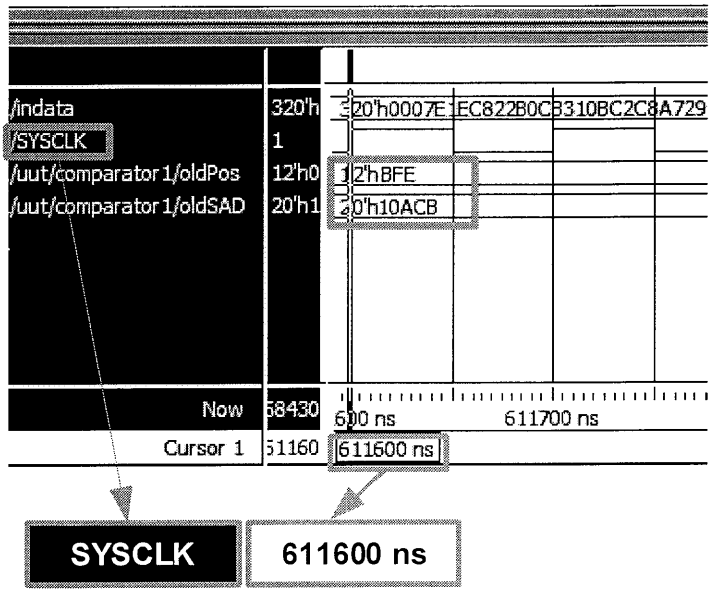
(Figure 5.40)

Reference frame (ducks_take_off.yuv frame number 0)



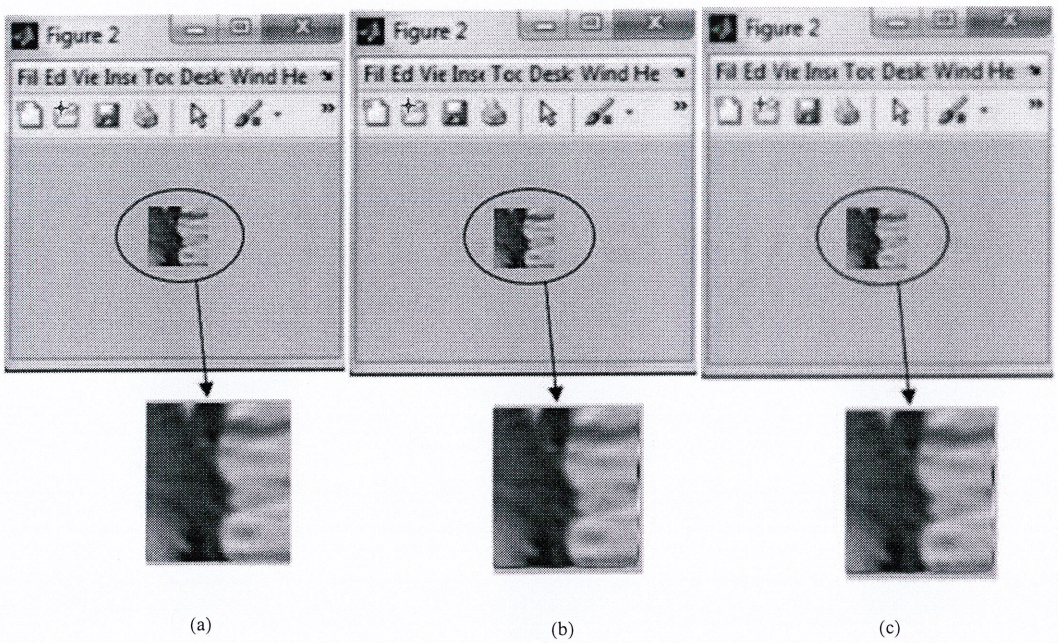
(Figure 5.41)

Current frame (ducks_take_off.yuv frame number 1)



(Figure 5.42)

Simulation results of ME process



(a) (b) (c)

(Figure 5.43)

Comparing the result of the implemented architecture. (a) Current block. (b) Best match in the search area. (c)Best match using FSBB-ME in Matlab

- **Full Design Simulation – Case 6**

A real HD video (ducks_take_off.yuv with 720×486 pixels) is used in this case study. Figure 5.44 and Figure 5.45 show the reference frame and the current frame, respectively. The simulation results shows that the best match candidate block is allocated at the position number $(7FF)_{Hex}$ with a minimum SAD equal to $(01CA2)_{Hex}$, see Figure 5.46.. Figure 5.47 (a), the Current Block is matching the best match block in Figure 5.47 (b) which is calculated using the implemented ME co-processor. Figure 5.47 (c) shows best match block using FSBB-ME in Matlab, and as seen both Figure 5.47 (b) and (c) are almost the same.



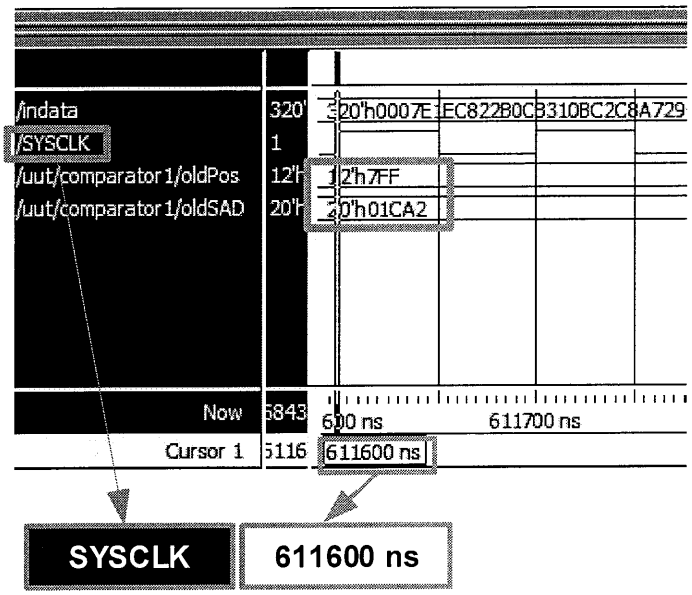
(Figure 5.44)

Reference frame (ducks_take_off.yuv frame number 0)



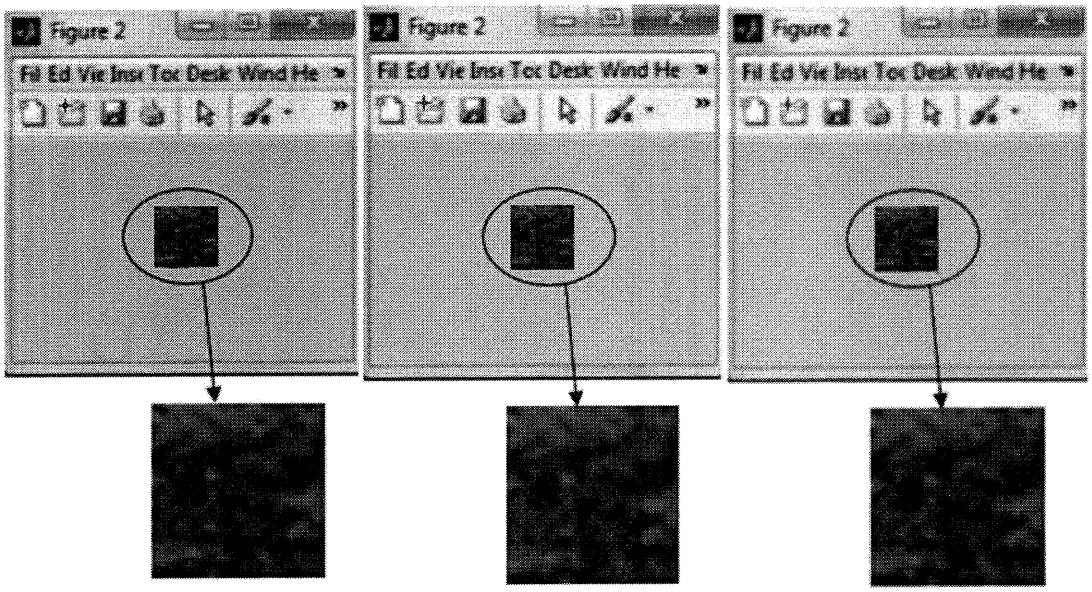
(Figure 5.45)

Current frame (ducks_take_off.yuv frame number 1)



(Figure 5.46)

Simulation results of ME process



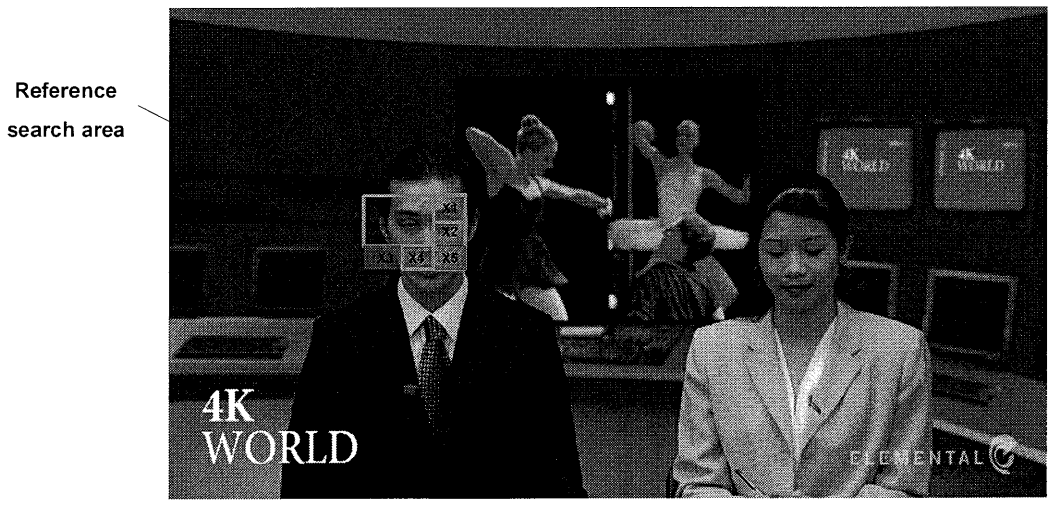
(a) (b) (c)

(Figure 5.47)

Comparing the result of the implemented architecture. (a) Current block. (b) Best match in the search area. (c)Best match using FSBB-ME in Matlab

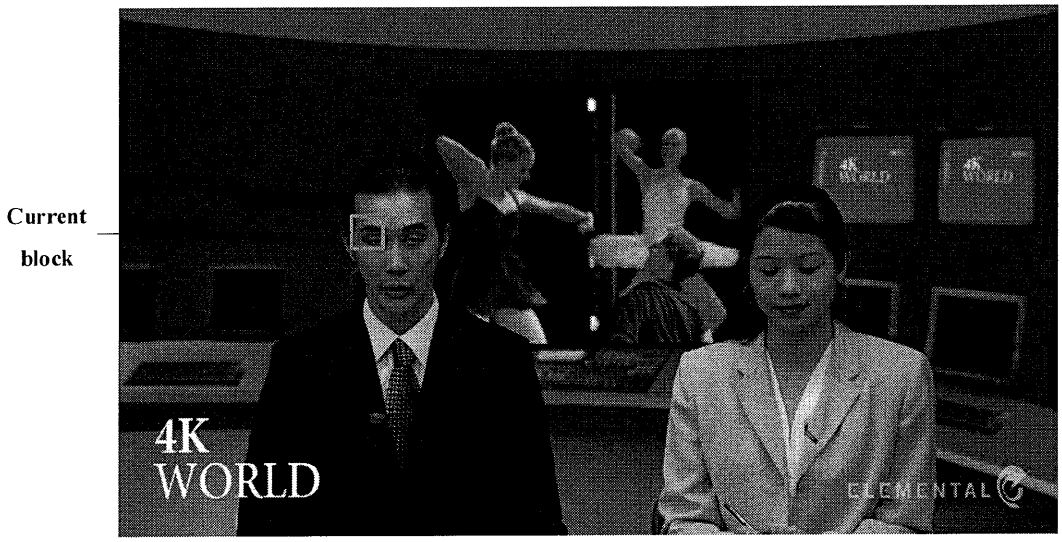
- **Full Design Simulation – Case 7**

A real 4k video (News_ProRes.MOV with 3840×2160 pixels) is used in this case study. Figure 5.48 and Figure 5.49 show the reference frame and the current frame, respectively. The simulation results shows that the best match candidate block is allocated at the position number $(822)_{Hex}$ with a minimum SAD equal to $(1107E)_{Hex}$, see Figure 5.50. Figure 5.51 (a), the Current Block is matching the best match block in Figure 5.51 (b) which is calculated using the implemented ME co-processor. Figure 5.51 (c) shows best match block using FSBB-ME in Matlab, and as seen both Figure 5.51 (b) and (c) are almost the same.



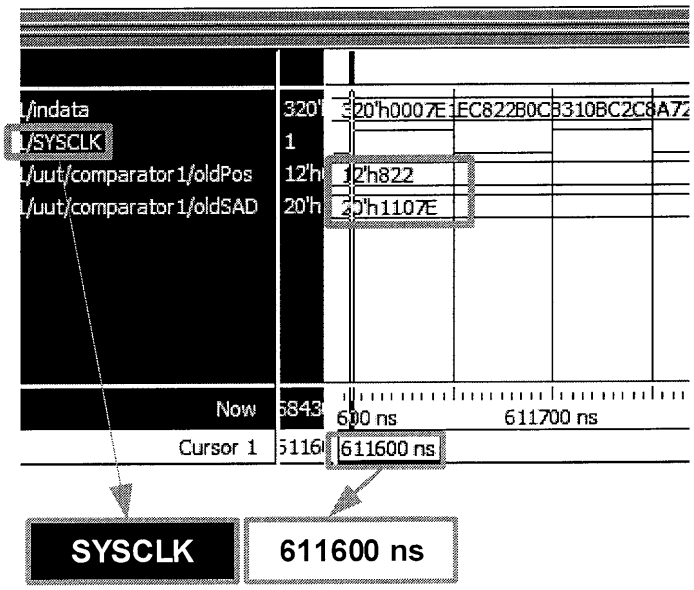
(Figure 5.48)

Reference frame (News_ProRes.MOV frame number 0)



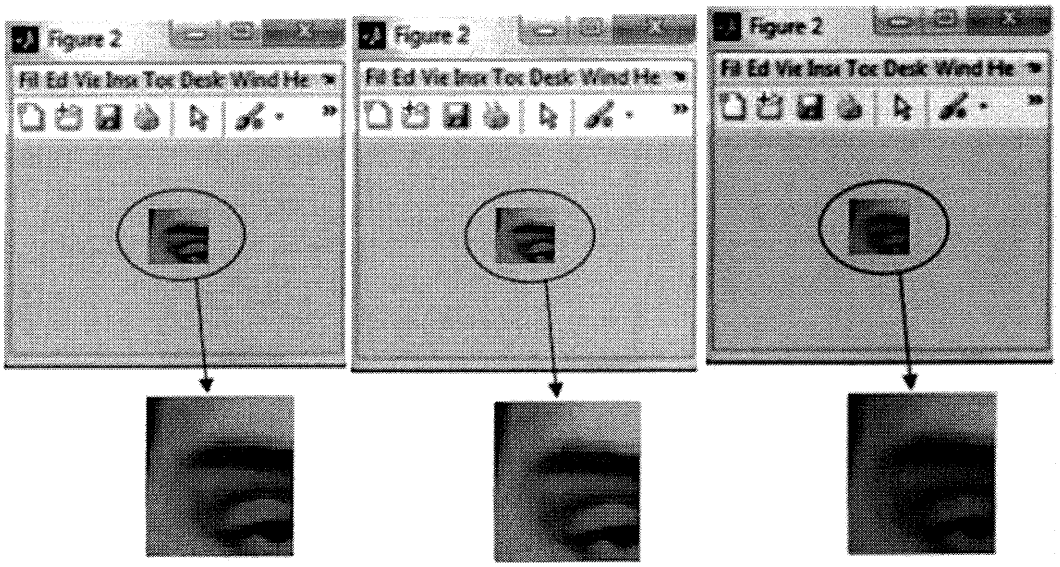
(Figure 5.49)

Current frame (News_ProRes.MOV frame number 1)



(Figure 5.50)

Simulation results of ME process



(Figure 5.51)

Comparing the result of the implemented architecture. (a) Current block. (b) Best match in the search area. (c)Best match using FSBB-ME in Matlab

5.3.2 FPGA Implementation

In FPGA implementation, we have implemented and synthesized the VHDL code using Xilinx ZYNQ-7 ZC706 Evaluation Board with (7z045ffg900-2) FPGA chip. Table 5.2 summarizes the key features of (7z045ffg900-2) FPGA chip (Xilinx Inc. 2016). Table 5.3 summarizes the hardware cost of architecture-1. Using a search area and CB of sizes 64×64 and 32×32 pixels, respectively, the maximum calculated hardware frequency is 545 MHz. This allows the ME co-processor to process the following video resolutions using the shown frame rates in Table 5.4 (i.e. 4k video resolution with 9 fps). It is clear from the Table 5.4 that our co-processer can be used for real time high resolution video applications. Figure 5.52 shows hardware utilization for FPGA Xilinx (7z045ffg900-2) chip using ISE Design Suite 14.5 tool, as seen in Figure 5.53.

(Table 5.2)
Zynq-7000 XC7Z045 FFG900-2 key features

Key Features	• Optimized for quickly prototyping embedded applications using Zynq-7000 SoCs • Hardware, design tools, IP, and pre-verified reference designs • Demonstrates a embedded design, targeting video pipeline • Advanced memory interface with • 1GB DDR3 Component Memory • 1GB DDR3 SODIM Memory • Enabling serial connectivity with PCIe Gen2x4, SFP+ and SMA Pairs, USB OTG, UART, IIC • Supports embedded processing with Dual ARM Cortex-A9 core processors • Develop networking applications with 10-100-1000 Mbps Ethernet (RGMII) • Implement Video display applications with HDMI out • Expand I/O with the FPGA Mezzanine Card (FMC) interface
Memory	• DDR3 Component Memory 1GB (PS) • DDR3 SODIM Memory 1GB (PL) • 2X16MB Quad SPI Flash (config) • IIC - 1 KB EEPROM
Power	• 12 V wall adaptor • Current measurement capability of supplies

```
Device utilization summary:
---------------------------

Selected Device : 7z045ffg900-2


Slice Logic Utilization:
 Number of Slice Registers:           62550  out of  437200     14%
 Number of Slice LUTs:                71520  out of  218600     32%
    Number used as Logic:             71520  out of  218600     32%

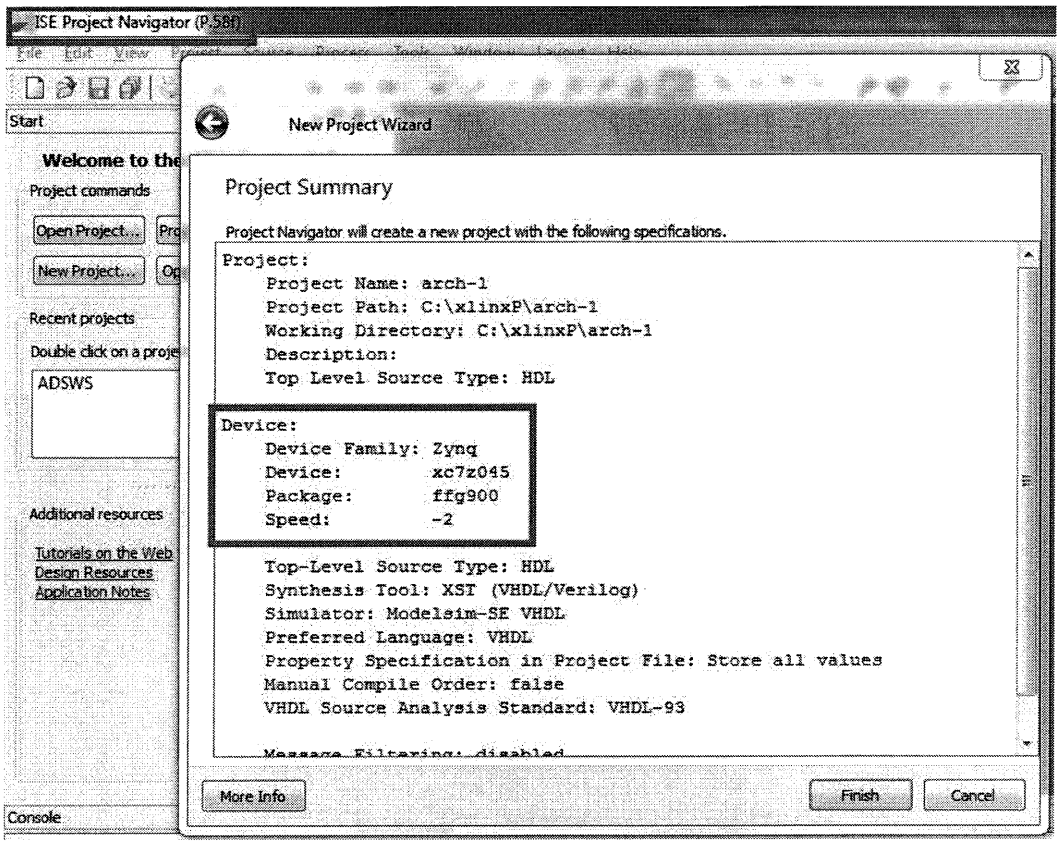
Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:  132918
   Number with an unused Flip Flop:   70368  out of  132918     52%
   Number with an unused LUT:         61398  out of  132918     46%
   Number of fully used LUT-FF pairs:  1152  out of  132918      0%
   Number of unique control sets:      3090

IO Utilization:
 Number of IOs:                        355
 Number of bonded IOBs:                355  out of    362     98%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRL/BUFHCEs:        2  out of    200      1%
```

(Figure 5.52)

Device utilization summary using ISE Design Suite 14.5 tool



(Figure 5.53)

Device type using ISE Design Suite 14.5 tool

(Table 5.3)

A summary of implementations for first architecture results using FPGA

FPGA Xilinx ZYNQ-7 ZC706	
# of PEs	1024
Search area size	64×64 pixels
Block size	32×32 pixels
# of slice LUTs	71.52 K
# of slice registers	62.55 K
# of LUT flip flop	132.918 K
Max. freq.	545 MHz
Throughput	1/6116
Data reuse level	Levels A and B

(Table 5.4)
Calculated frame rate for different formats for video transmission

Resolution	Pixels/line	Lines/frame	Calculated frame rate for HD videos (fps)
UHD 4k	3840	2160	9
HDTV broadcast	1920	1080	43
SDTV broadcast (D1)	720	486	124

5.4 Summary

The implemented architecture in this chapter can perform the FSBB-ME algorithm in an accurate and efficient way. The time verification was performed using VHDL tool. A total of 6116 clock (611600 ns) which are required to perform one complete ME process using CB and search area of sizes 32×32 pixels and 32×32 pixels, respectively. The implemented ME co-processor can perform ME at 9 frame per second using 4K (3840×2160) at 545 MHz with 100% unitization of all the hardware components. A smart data reuse is used in order to reduce the I/O memory BW. Regular data flow is used to move the data of the search area both horizontally (inside the PE array) and vertically (inside the local memory). This guarantees the simplicity of the hardware implementation of the ME co-processor in this chapter.

Architecture-1 has many advantages that help in speeding up the ME process of the H.265/HEVC CODEC. Pipelining and 100% PE utilization are the main advantage of architecture-1 which means there is no stall at all. Parallel operations that are appeared in the design of the Adder Tree can be considered as another advantage. In addition, reducing the I/O memory BW by using the data reuse principle is another advantage.

Finally, regular data flow that guarantees simple hardware design. The architecture-1 can be efficiently used in real time video applications that target high speed transmission and high visual quality. The computations of the FSBB-ME that are implemented using architecture-1 can be greatly decreased if we remove redundant operations from ME process. This can be achieved using the implemented second architecture (architecture-2) in the following chapter.

Chapter 6

MOTION ESTIMATION CO-PROCESSOR ARCHITECTURE-2

6.1 Introduction

6.1.1 Principle of Prediction Technique

6.2 Adaptive Search Window Size Technique (ASWS)

6.2.1 Architecture-2: An Overview

6.2.2 Architecture Components

6.2.3 Challenges and Motivation of Architecture-2

6.3 Simulation and Implementation Results

6.3.1 VHDL Verification

6.3.2 FPGA Implementation

6.4 Summary

Chapter 6 MOTION ESTIMATION CO-PROCESSOR ARCHITECTURE-2

6.1 Introduction

Full Search Block Based-Motion Estimation computational complexity was attacked in many techniques. One technique is dependent on minimizing the number of search points needed to find the best match candidate block in the search area. The Motion Estimation process is the most exhaustive time consuming part in the H.265/HEVC encoder. It consumes up to 84% of the total encoding time. The Total Computational Complexity (TCC) required for obtaining the best match candidate block in the reference frame can be expressed using the following general Equation 6.1:

$$TCC=[(n_1 \times n_2) .(Sub+Abs+Add)].M \qquad (6.1)$$

Where, n_1 and n_2 are the number of rows and columns in the Current Block. Sub, Abs, and Add represent the computational complexity of subtraction, absolute value and addition operations, respectively, that are required for calculating the matching criteria defined by the SAD operation. M is the total number of checked points in the search area in the reference frame. It is clear from the previous equation that the computational complexity of the ME process per block depends on the number of Sub, Abs, Add operations, and the total number of search points (M) inside the search area. In this chapter, the value of TCC is greatly reduced by decreasing the number of search points (M) in the search area. This can be achieved using two different techniques. The first

technique is the Adaptive Search Window Size (ASWS) technique. It diminishes the size of the search area according to the motion activity of the CB. The second technique is the predictive technique that is used to move the search center to a point near to the expected best match candidate block in the search area. Following sub-sections will describe the two algorithms in details.

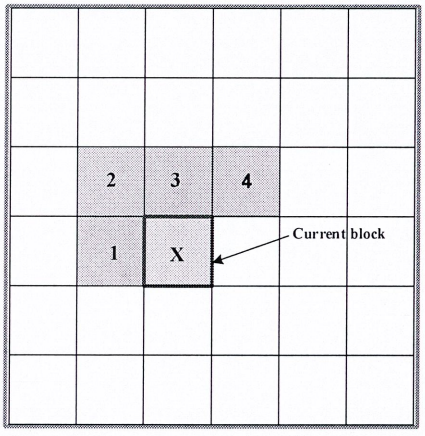
6.1.1 Principle of Prediction Technique

We can benefit from the homogeneous property of a video sequence to move the search center of the search area to a near point of the best match candidate block. As seen in Figure 6.1, the AMVs of the surrounding blocks to the CB can be used to expect the AMV of the CB. As an example, the AMVs of the pre-encoded blocks (blocks 1, 2, and 3 in (Figure 6.1) can be used to estimate the search center of the CB. The new search center is called Initial Search Center (ISC). It is expected to be near to the best match candidate block. The more the accuracy of decided ISC is, the less the number of search points M that are required in the search area. Two types of prediction techniques, or a combination of them, are generally used: spatial prediction and temporal prediction (Ohm et al., 2012). In this chapter, we consider the spatial prediction to achieve more savings in computations. As seen in Figure 6.1, in spatial prediction, it is assumed that the neighboring blocks which cover the same video object undergo a similar translational motion between two adjacent frames. Consequently, the AMVs of the neighboring blocks can be used to predict the ISC of the CB (ITU-T & Int., 2003). More accurate predictions can be obtained by using the temporal prediction which assumes that the object region moves with a similar motion over consecutive frames. The more accurate the prediction is, the more the

accuracy of the ISC position. In turn, this will increase the accuracy of the calculated AMV as well as the probability of considering the ISC as the final solution. The most usable prediction technique by most video standards such as H.264/AVC and H.265/HEVC is the median prediction. Equations (6.2) and (6.3) shows the median predictor used in X and Y directions where PMV is the Predicted Motion Vector (PMV).

$$PMV_X = \text{Median}\ (AMV_{X1}, AMV_{X2}, AMV_{X3}) \qquad (6.2)$$

$$PMV_Y = \text{Median}\ (AMV_{Y1}, AMV_{Y2}, AMV_{Y3}) \qquad (6.3)$$



(Figure 6.1)

Four current blocks

6.2 Adaptive Search Window Size Technique (ASWS)

The implemented algorithm is called Adaptive Search Window Size (ASWS). The algorithm is efficient enough to meet the following requirements: (a) Less computations, (b) Regular data flow, and (c) Data reuse to minimize I/O memory BW. ASWS is used to adaptively change the search area size for each CB by calculating a new search area size. Full search algorithm is also used with the ASWS algorithm but with

reduced search area size and less number of computations. ASWS can be used to diminish the size of the search area according to the motion activity of the CB. The implemented algorithm is suitable for hardware implementation as determining the search window size doesn't require significant computations.

The implemented algorithm is highly dependent on the accuracy of the predicted PMV or ISC. Three AMVs of previously encoded blocks are used to predict the PMV, see Figure 6.1. The three AMVs are the immediate neighbors of the CB. These neighboring blocks are known as Region-Of-Support (ROS). A normal prediction technique is used, which is the median of the AMVs of the immediate neighboring blocks seen in Figure 6.1. This prediction is commonly used in the most of the new video coding standards. Median predictor has enough accuracy to maintain almost the same visual quality of the video sequence and it is recommended by H.265/HEVC standard.

The size of the new search area is calculated based on the value of the PMV. The smaller the PMV, the more chance that the actual AMV of the CB to be very close to the ISC. Consequently, less search area size is required. Otherwise, if the PMV is large, then the AMV of the CB is expected to be far from the ISC point. As a result, larger search area is required. The selection of the search area size will be as follow: (1) PMV is calculated using ROS blocks, this step is skipped if there are no enough neighboring blocks in the ROS, full search is used for these blocks, (2) Determine the search window size using PMV using Equation 6.4, and (3) Consider the new search origin to be the calculated PMV and perform full search in the selected search window to find the best match.

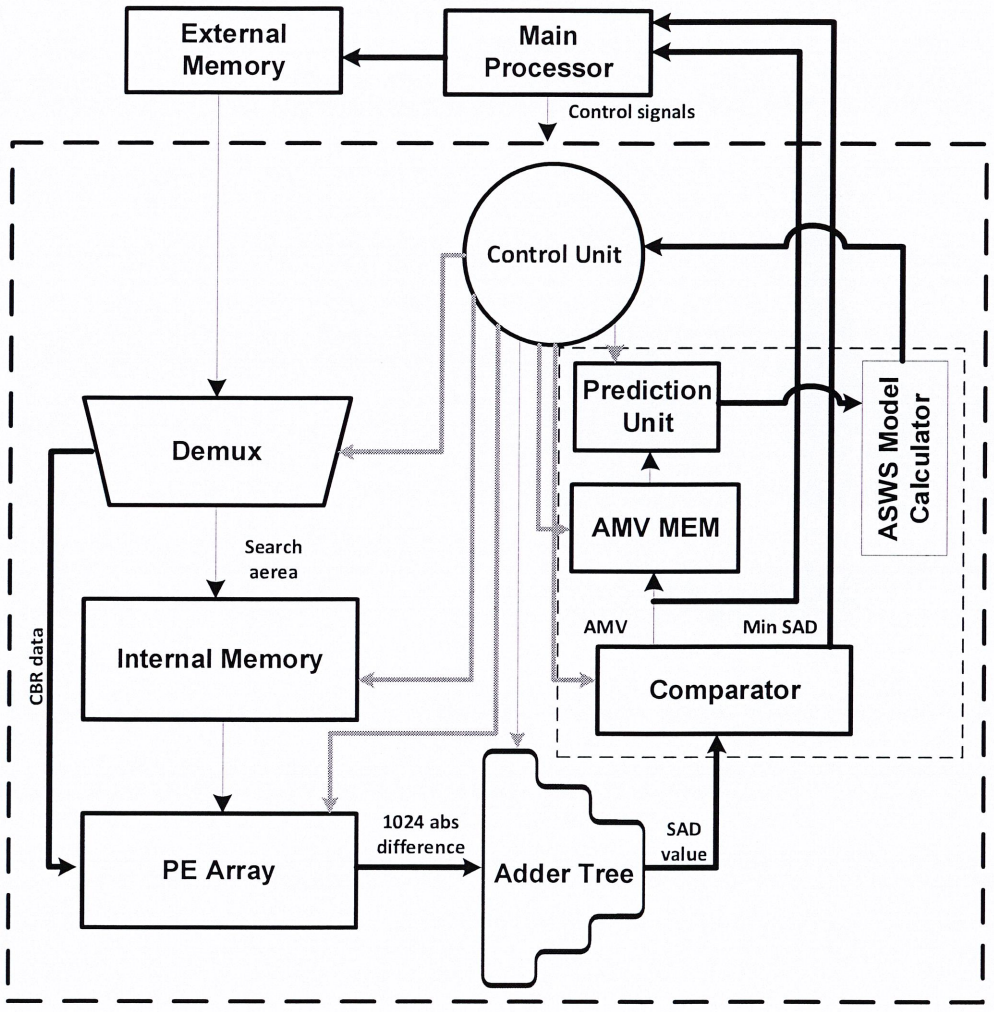
$$\text{Search window}= \begin{cases} 16 \times 16\ pixels, if\ |MVp| \leq L1 \\ 32 \times 32\ pixels, if\ L1 \leq |MVp| < L2 \\ 48 \times 48\ pixels, if\ L2 \leq |MVp| < L3 \\ 64 \times 64\ pixels, if\ |MVp| > L3 \end{cases} \quad (6.4)$$

The selection of the values L_1, L_2 and L_3 is based primarily on analyzing the Motion Activity Field (MAF) of video sequences using exhaustive offline simulations of six HD video sequences. The selected values of L_1, L_2 and L_3 are based on the following notions using our offline Matlab experiments. If the Absolute Difference between both the PMV and AMV (shown in Figure 6.1) is small, then the expected best match is very close to the ISC point. Consequently, small search area size is required. If the Absolute Difference is high, the AMV is far from the ISC and larger search area size is required. Based upon these two assumptions, the Absolute Difference plane is divided into three regions: maximum, medium, and small. Keeping in mind that the PMV is a good indicator for the homogeneity of the blocks surrounding both the current and the reference blocks, we can find the range of the corresponding PMV for each region. The chosen global values L_1, L_2 and L_3, were designated to be 7, 15 and 23 respectively.

6.2.1 Architecture-2: An Overview

In addition to the same main components of the first implemented architecture in chapter 5, there are three main components are added for a great reductions in computations. Prediction unit (PU), Adaptive Search Window Size (ASWS) unit, and AMV memory are the main three components of the second ME co-processor architecture as seen in Figure 6.2. The prediction unit is used for calculating the median predictor for the current block. This unit takes 3 AMVs as inputs which

are taken from the AMV memory and outputs the predicted PMV which is used as the new ISC for the ME process. The ASWS unit computes the new search window size according to the estimated PMV by the PU.



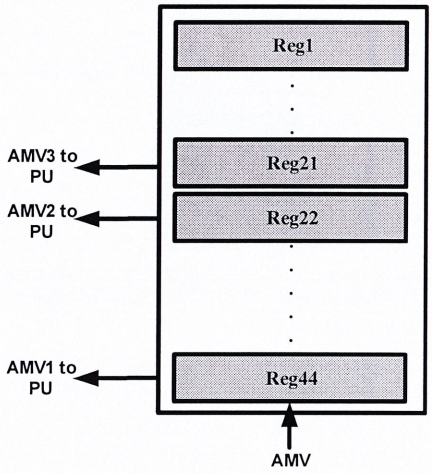
(Figure 6.2)
Top level architecture of the second implemented ME co-processor using prediction and ASWS techniques

6.2.2 Architecture Components

In addition to the main units of the first implemented ME co-processor in chapter 5, two more units are added and those are summarized as follows:

A. Actual Motion Vector Memory (AMV MEM)

The purpose of the AMV memory is to store the final AMVs for previously encoded two rows of blocks in the current frame. These AMVs will be used to calculate the PMV for each CB. AMV memory consists of 44 registers to cover 2 rows of the current frame of size (720×486). The size of AMV MEM can be upgraded according to the resolution of the processed video sequences. The median predictor is calculated using the 3 AMVs found in the AMV memory. These three AMVs are always found in the same position (reg21, reg22, and reg44) for a given video resolution of size 720×486. Figure 6.3 shows the used structure of the AMV memory.



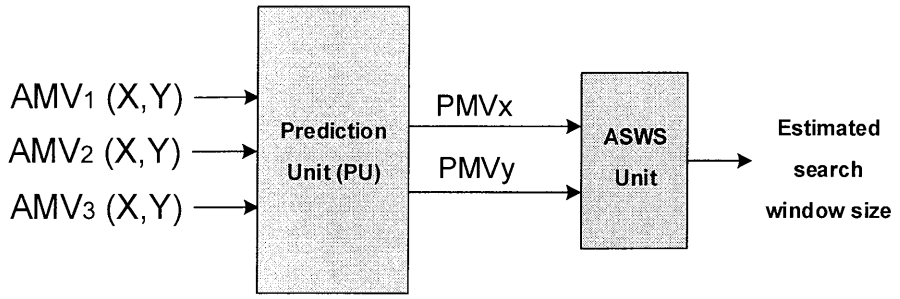
(Figure 6.3)

AMV Memory used with Prediction unit

B. Adaptive Search Window Size Module (ASWS)

The PMV is calculated as follows using (Figure 6.4: (1) PU takes three AMVs of previously encoded surrounding blocks of the CB in X and Y directions, (2) The PU outputs a PMV_X and PMV_Y using Equations 6.2

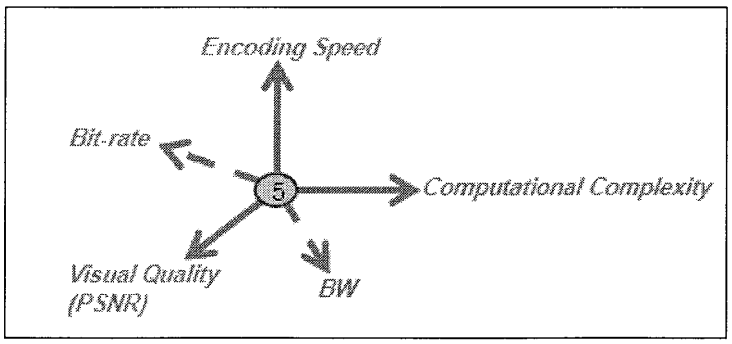
and 6.3, and (3) The ASWS unit calculate search area size based on maximum absolute value between PMV_X and PMV_Y using Equation 6.4, and finally outputs the required search area size to the control unit in order to adjust the ME process based on the estimated search area size.



(Figure 6.4)
Adaptive Search Window Size (ASWS) unit combined with the PU

6.2.3 Challenges and Motivation of Architecture-2

The main motivation of the implemented architecture is to reduce the computational complexity. Adjusting any of the below five parameters will affect negatively or positively on the others, see Figure 6.5. We are concerning with decreasing the Computational Complexity (CC) of the encoding process and studying its effects on the other parameters.



(Figure 6.5)

The parameters which affect the performance of any video coder

6.3 Simulation and Implementation Results

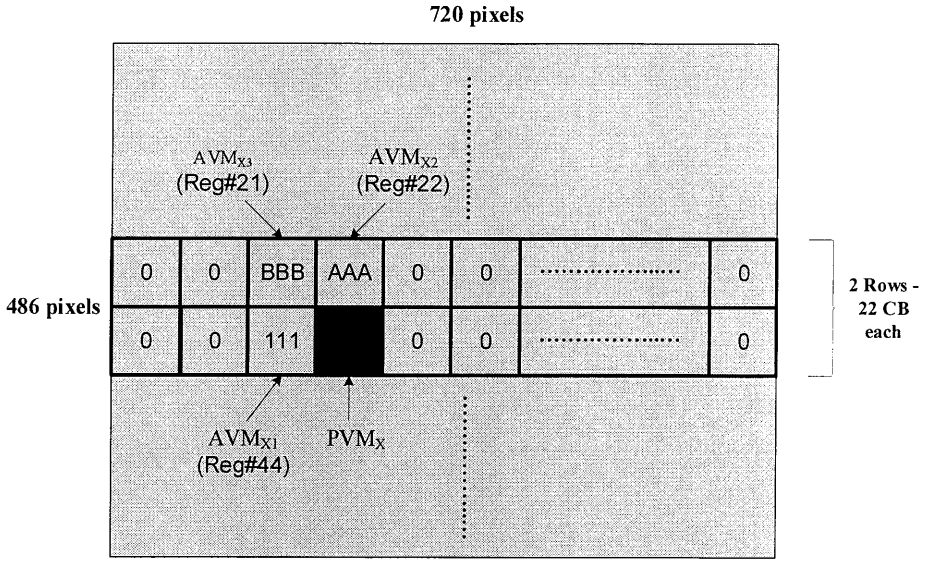
The second implemented ME co-processor architecture is elaborated in this section using the VHDL and FPGA tools. The frames are divided into 32×32 block sizes. The used search area size is 64×64 pixels. Modelism (version10.4a) tool is used for full functional verification using real frames as test benches. The implemented co-processor is also implemented using FPGA chip (Xilinx ZYNQ-7 ZC706) to calculate the hardware cost and frequency. The simulation results will be summarized in below subsections.

6.3.1 VHDL Verification

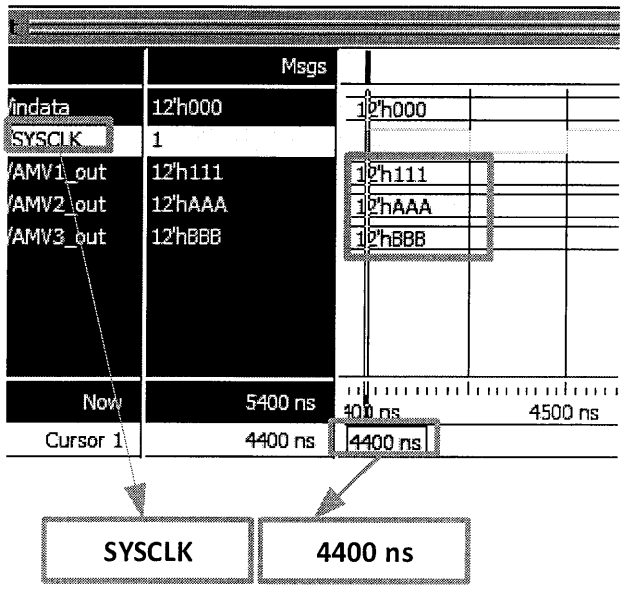
In this subsection we evaluated the validity and the time verification of the new components of the implemented ME second co-processor using VHDL tool.

A. Actual Motion Vector Memory (AMV MEM)

As mentioned before, AMV memory consists of 44 registers to cover 2 rows of the current frame of size (720×486) as seen in Figure 6.6. Figure 6.7 shows the outputs of the AMV memory, which are "111", "AAA", and "BBB" for AVM_{X1}, AVM_{X2}, and AVM_{X3}, respectively using the input data shown in Figure 6.6. And same simulation can be done for AVM_{Y1}, AVM_{Y2}, and AVM_{Y3}, respectively.



(Figure 6.6)
Sample frame of size 720×486 for AVM memory

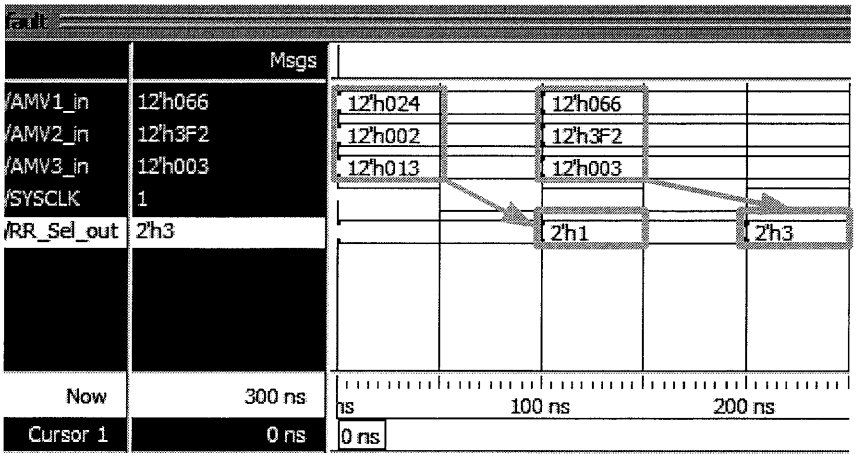


(Figure 6.7)

Simulation of AMV memory

B. Adaptive Search Window Size Unit (ASWS)

Figure 6.8 shows two groups of AMVs highlighted below used to calculate the new search window size using the ASWS unit. The first group is $(024)_{Hex}$, $(002)_{Hex}$, and $(013)_{Hex}$ .So, the median AMV is $(013)_{Hex}$ and according to Equation 6.4, the new search window size will be 32×32 pixels. Also, the second group is $(066)_{Hex}$, $(3F2)_{Hex}$, and $(003)_{Hex}$. Accordingly, the median AMV is "066 H" and again according to Equation 6.4 the new search window size will be 64×64 pixels. Additionally, an internal signal "RR_Sel_sig" is used in the VHDL code to indicate the search area size. This signal has four different cases, "0", "1", "2", and "3" which represents 16×16, 32×32, 48×48, and 64×64, respectively. It is worth mentioning that one clock cycle is needed to get an output.



(Figure 6.8)

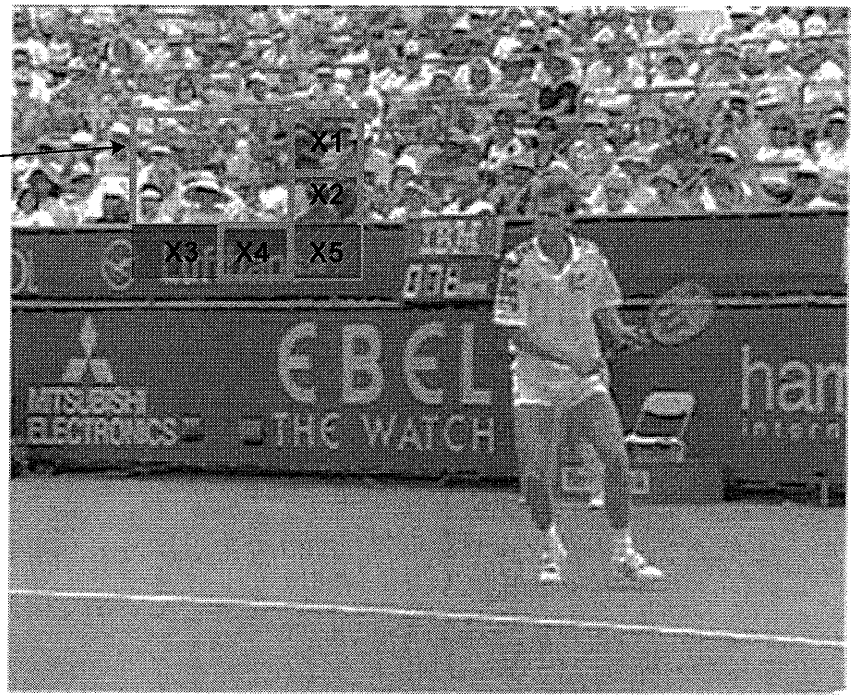
ASWS simulation

C. Full Processor Design Simulation

In this sub-section, we are going to elaborate the whole architecture of the ME co-processor given in Figure 6.2. Four different cases were considered to verify the functionality of architecture-2 using real video sequences. Following are discussions about each case.

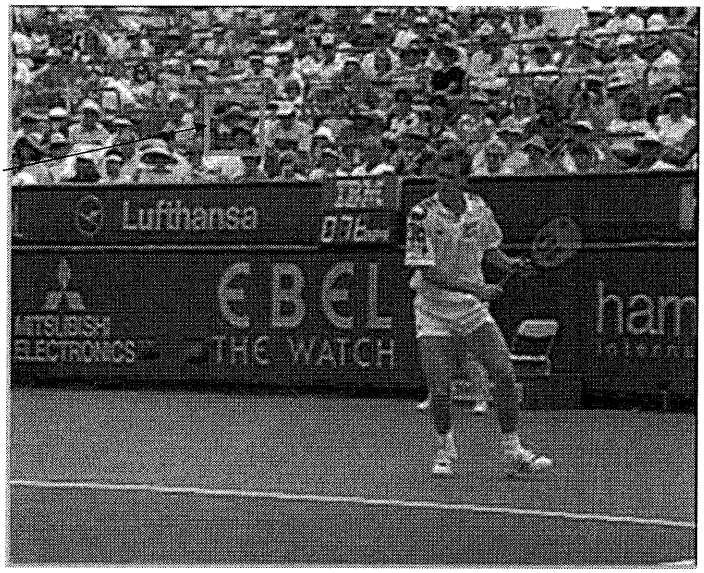
- **Full Design Simulation – Case 1**

A real video (stefan.yuv with 288×352 pixels) is tested in this case using the ME co-processor in Figure 6.2. Figure 6.9 and Figure 6.10 show both of the reference frame and the current frame, respectively, of Stefan video sequences. Figure 6.11 and Figure 6.12 show the selected search area and the Current Block, respectively. The simulation results shows that the best match candidate block is allocated at position number $(32)_{Hex}$ with a minimum SAD equal to $(1157B)_{Hex}$, and "RR_Sel_sig" equal to $(0)_{Hex}$ which means 16×16 search area, as seen in Figure 6.13. The total number of clock cycles needed for 16×16 search area is 762. Figure 6.14 shows the similarity between the CB and the obtained best match candidate block using the implemented ME co-processor architecture 2. It is clear from Figure 6.14 that the implemented architecture is very accurate in deciding the best match candidate block that is almost same as the CB. It is worth mentioning that FSBB-ME algorithm gives same result as in Figure 6.14. This is an indicator for the high accuracy of the implemented architecture. Figure 6.14 (c) shows best match block using FSBB-ME in Matlab, and as seen both Figure 6.14 (b) and (c) are almost the same.



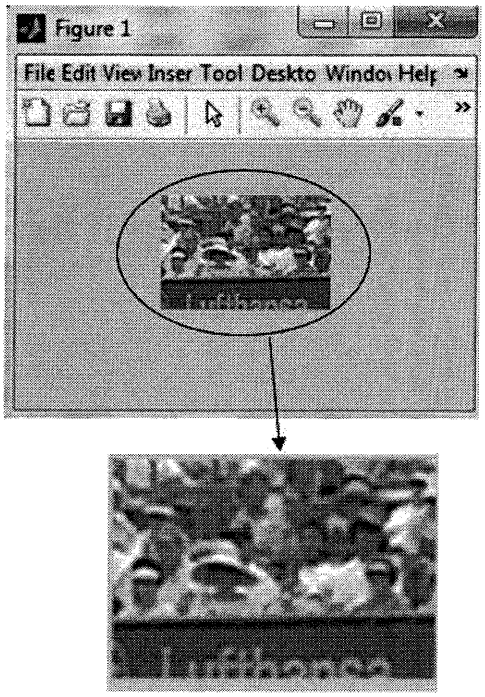
(Figure 6.9)

Reference frame (stefan.yuv frame number 0)



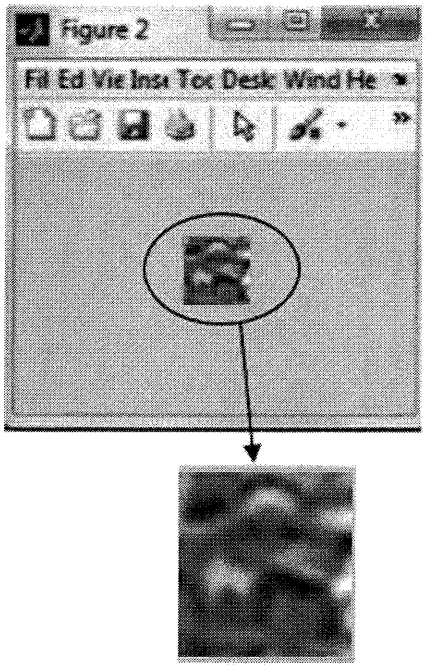
(Figure 6.10)

Current frame (stefan.yuv frame number 1)



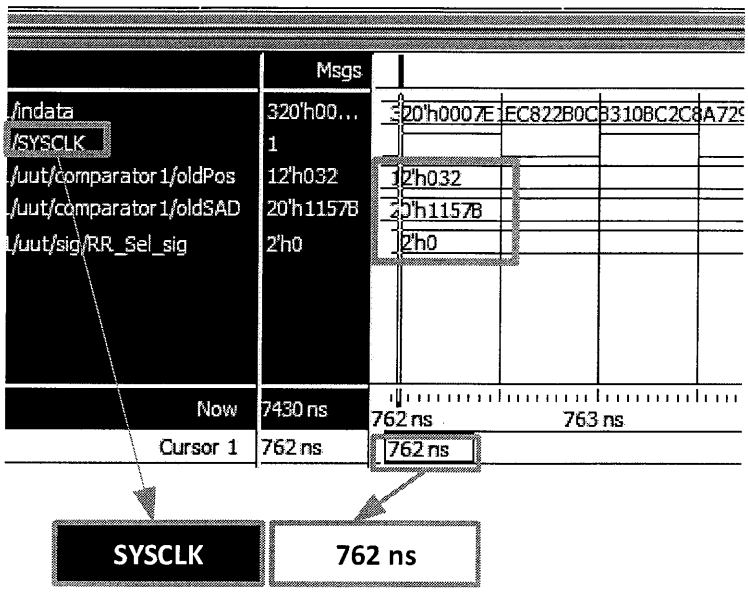
(Figure 6.11)

Reference search area



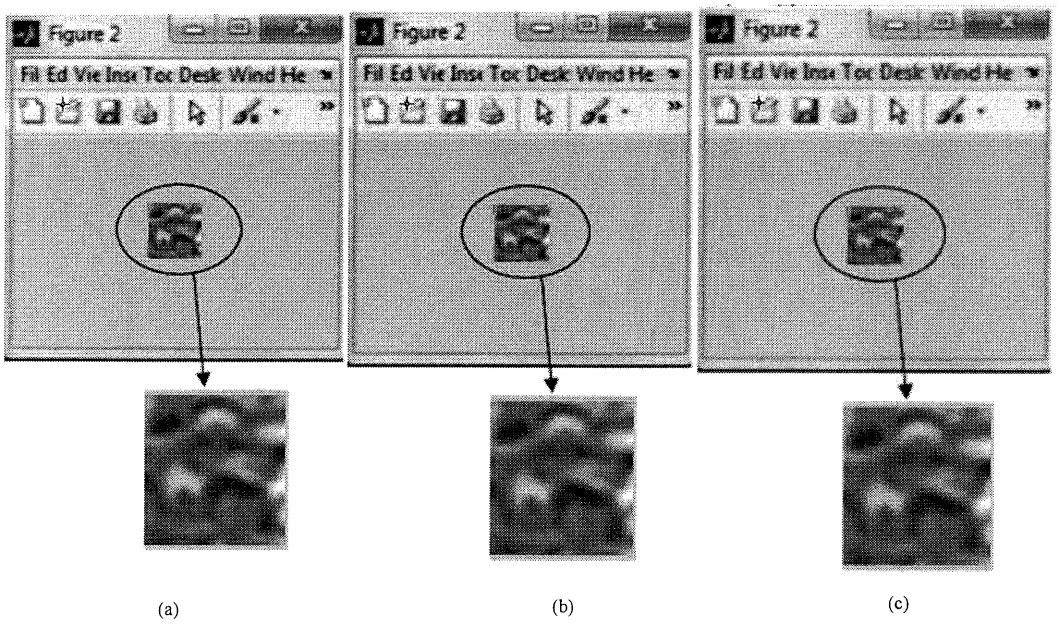
(Figure 6.12)

Current Block



(Figure 6.13)

Simulation results of ME process



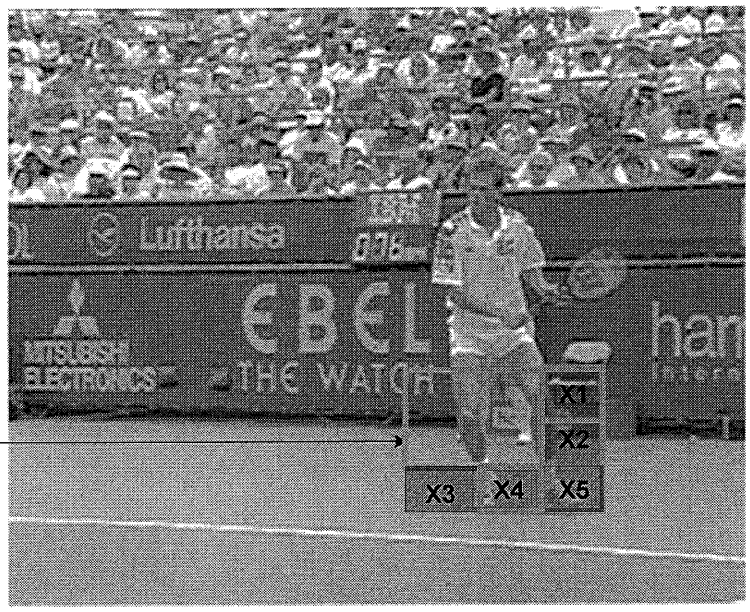
(a) (b) (c)

(Figure 6.14)

Comparing the result of the implemented architecture. (a) Current block. (b) Best match in the search area. (c) Best match using FSBB-ME in Matlab

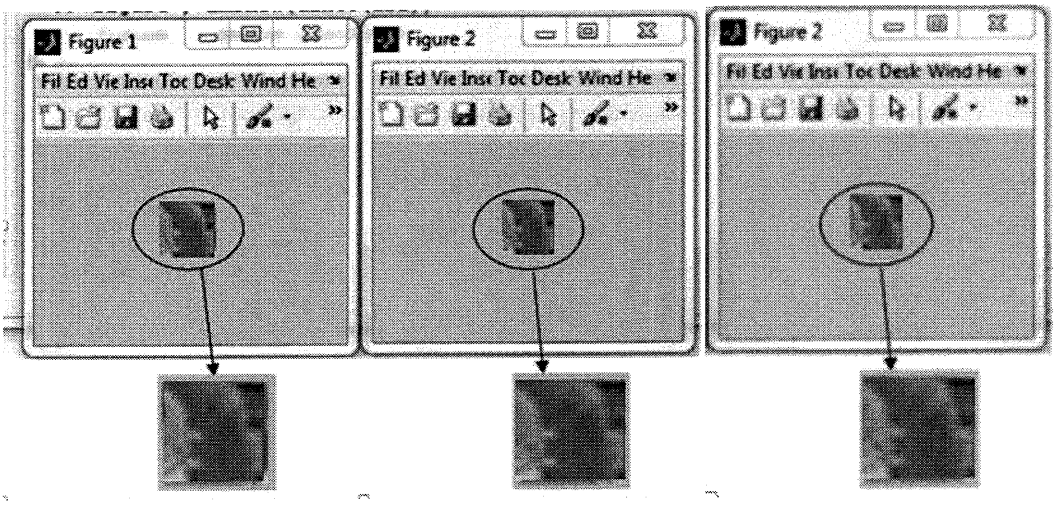
- **Full Design Simulation – case2**

The same real video (stefan.yuv with 288×352 pixels) is also used in this case. Figure 6.15 and Figure 6.16 show the reference frame and the current frame, respectively, along with the reference search area and the Current Block. The simulation results shows that the best match is allocated at position number $(23)_{Hex}$ with a minimum SAD equal to $(08DAC)_{Hex}$, and "RR_Sel_sig" equal to $(0)_{Hex}$ which means 16×16 search area, see Figure 6.17. Figure 6.18 (a), the Current Block is matching the best match block in Figure 6.18 (b) which is calculated using the ME co-processor architecture 2. FSBB-ME gives same results as in Figure 6.18. Figure 6.18 (c) shows best match block using FSBB-ME in Matlab, and as seen both Figure 6.18 (b) and (c) are almost the same.



(Figure 6.15)

Reference frame (stefan.yuv frame number 0)



(a) (b) (c)

(Figure 6.18)

Comparing the result of the implemented architecture. (a) Current block. (b) Best match in the search area. (c) Best match using FSBB-ME in Matlab

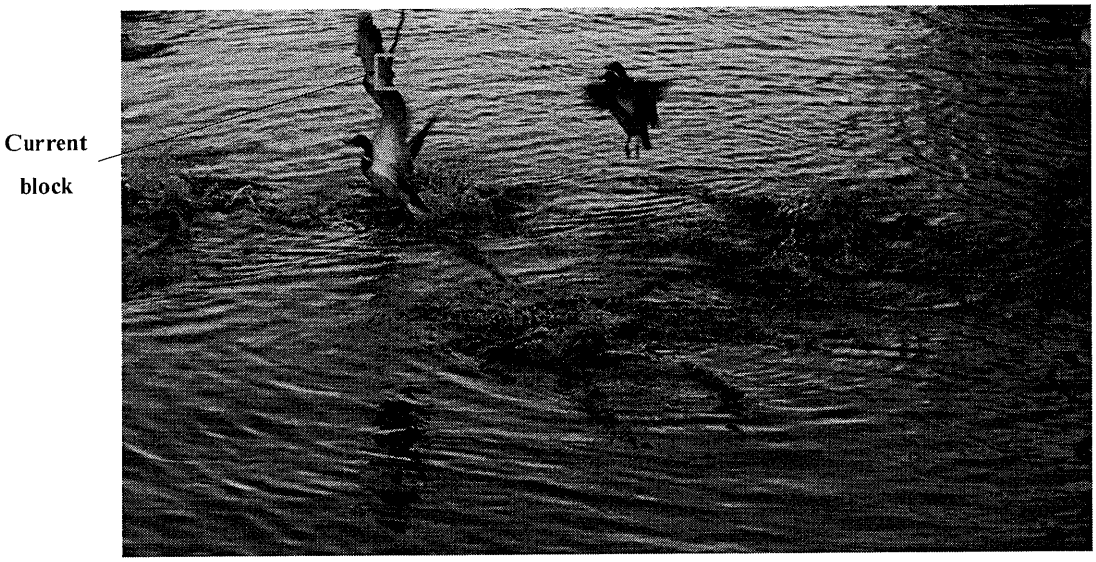
- **Full Design Simulation – Case 3**

A real HD video (ducks_take_off.yuv with 720×486 pixels) is tested in this case using the ME co-processor in Figure 6.2. Figure 6.19 and Figure 6.20 show a reference frame and the current frame, respectively of Stefan video sequences. The simulation results shows that the best match is allocated at position number $(32)_{Hex}$ with a minimum SAD equal to $(10ACB)_{Hex}$, and "RR_Sel_sig" equal to $(0)_{Hex}$ which means 16×16 search area, as seen in Figure 6.21. we used Full Search algorithm in Matlab 7.9.0 to calculate the best match block and we found that it matches the VHDL results of the ME processor as seen in Figure 6.22. Figure 6.22 shows a similarity between the CB and the obtained best match candidate block using the implemented ME co-processor architecture 2. It is clear from Figure 6.22 that the implemented architecture is very accurate in deciding the best match candidate block that is almost same as the CB.



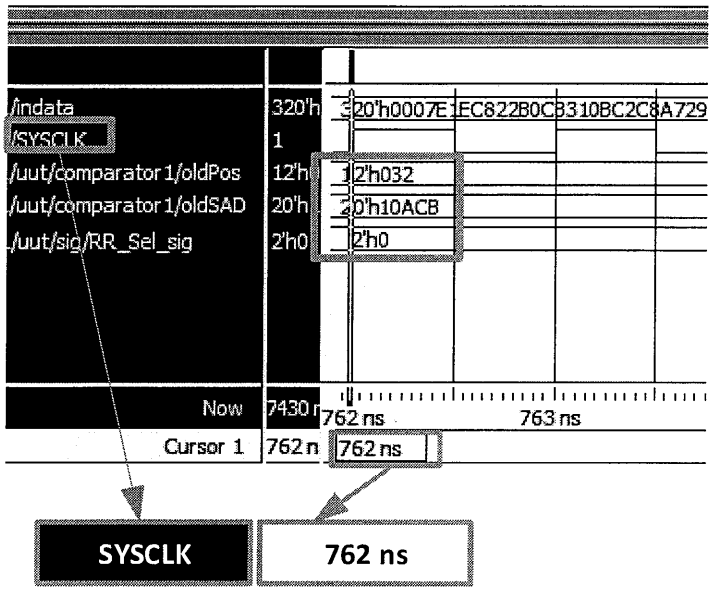
(Figure 6.19)

Reference frame (ducks_take_off.yuv frame number 0)



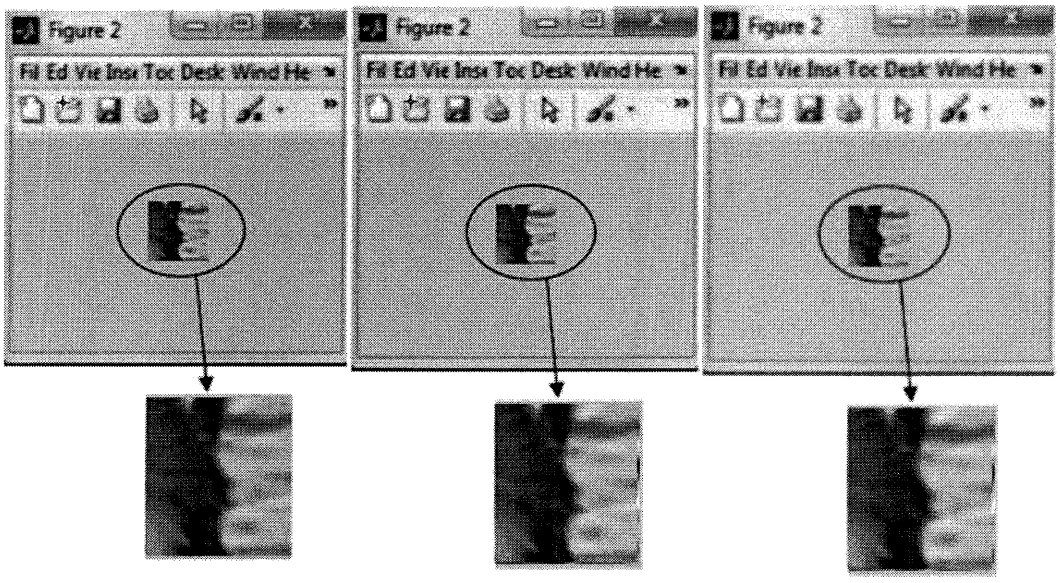
(Figure 6.20)

Current frame (ducks_take_off.yuv frame number 1)



(Figure 6.21)

Simulation results of ME process



(a) (b) (c)

(Figure 6.22)

Comparing the result of the implemented architecture. (a) Current block. (b) Best match in the search area. (c) Best match using FSBB-ME in Matlab

- **Full Design Simulation – Case 4**

A real HD video (ducks_take_off.yuv with 720×486 pixels) is tested in this case using the ME co-processor in Figure 6.2. Figure 6.23 and Figure 6.24 show a reference frame and the current frame, respectively of Stefan video sequences. The simulation results shows that the best match is allocated at position number $(28)_{Hex}$ with a minimum SAD equal to $(01CA2)_{Hex}$, and "RR_Sel_sig" equal to $(0)_{Hex}$ which means 16×16 search area, as seen in Figure 6.25. we used Full Search algorithm in Matlab 7.9.0 to calculate the best match block and we found that it matches the VHDL results of the ME processor as seen in Figure 6.26. Figure 6.26 shows a similarity between the CB and the obtained best match candidate block using the implemented ME co-processor architecture 2. It is clear from Figure 6.26 that the implemented architecture is very accurate in deciding the best match candidate block that is almost same as the CB. Figure 6.26 (c) shows best match block using FSBB-ME in Matlab, and as seen both Figure 6.26 (b) and (c) are almost the same.



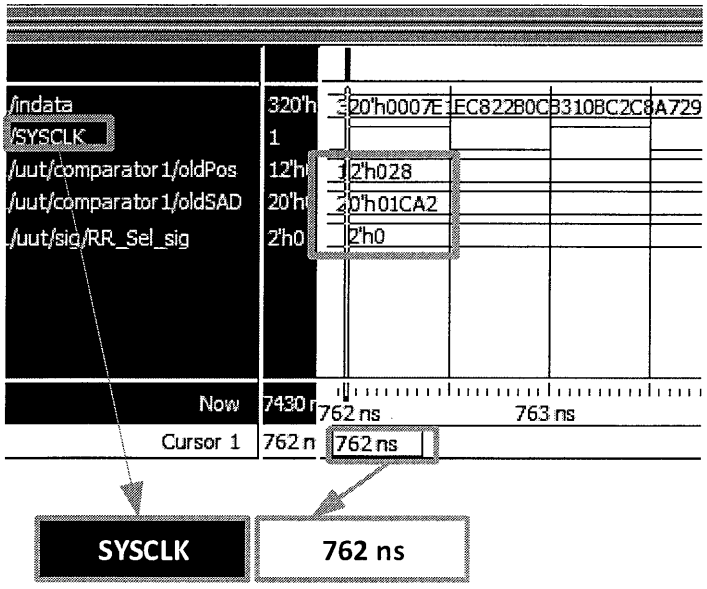
(Figure 6.23)

Reference frame (ducks_take_off.yuv frame number 0)



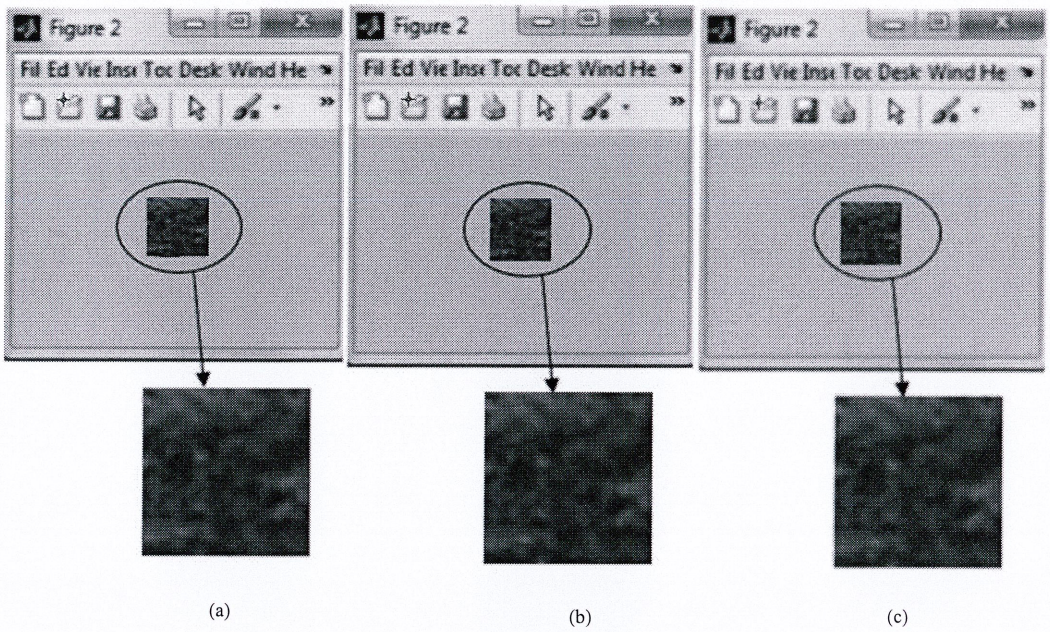
(Figure 6.24)

Current frame (ducks_take_off.yuv frame number 1)



(Figure 6.25)

Simulation results of ME process



(a) (b) (c)

(Figure 6.26)

Comparing the result of the implemented architecture. (a) Current block. (b) Best match in the search area. (c) Best match using FSBB-ME in Matlab

6.3.2 FPGA Implementation

In FPGA implementation, we have implemented and synthesized the VHDL code using Xilinx ZYNQ-7 ZC706 Evaluation Board with (7z045ffg900-2) FPGA chip. The implemented architecture takes the Current Block of size 32×32 pixels and the variable search area as inputs and finds the PMV of the CB. Table 6.1 shows a summary of the ME co-processor hardware cost including PE array, adder tree, comparator, the control unit, AMV memory, prediction unit, and ASWS module. The maximum hardware frequency calculated by the FPGA chip is 527 MHz. This allows the ME co-processor to process the HD video resolutions in Table 6.2 with the indicated frame rates. This allows the designed ME co-processor to be used in high resolution real time video applications.

The implemented ME coprocessor provide maximum efficiency with help of data reuse, pipelining, parallelism to achieve 100% utilization of each component. The maximum operational frequency is 527 MHz this grantees real ME for 4K (3840×2160) video sequences applications with 10 fps. Figure 6.27 shows hardware utilization for FPGA Xilinx (7z045ffg900-2) chip using ISE Design Suite 14.5 tool.

```
Device utilization summary:
---------------------------

Selected Device : 7z045ffg900-2


Slice Logic Utilization:
 Number of Slice Registers:           62587  out of  437200    14%
 Number of Slice LUTs:                71601  out of  218600    32%
    Number used as Logic:             71601  out of  218600    32%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:  133035
   Number with an unused Flip Flop:   70448  out of  133035    52%
   Number with an unused LUT:         61434  out of  133035    46%
   Number of fully used LUT-FF pairs:  1153  out of  133035     0%
   Number of unique control sets:      3093

IO Utilization:
 Number of IOs:                        386
 Number of bonded IOBs:                355  out of     362    98%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRL/BUFHCEs:        2  out of     200     1%
```

(Figure 6.27)

Device utilization summary using ISE Design Suite 14.5 tool

(Table 6.1)
A summary of implementations results for second architecture using FPGA

FPGA Xilinx ZYNQ-7 ZC706	
# of PEs	1024
Search area size	16×16, 32×32, 48×48, and 64×64
Block size	32×32 pixels
# of slice LUTs	71.61 K
# of slice registers	62.59 K
# of LUT flip flop	133.04 K
Max. freq.	527 MHz
Throughput:	1/6116
Data reuse level	Levels A and B

(Table 6.2)

Calculated frame rate for different formats for video transmission

Resolution	**Pixels/line**	**Lines/ frame**	**Calculated frame rate for HD videos (fps)**
UHD 4k	3840	2160	10
HDTV broadcast	1920	1080	42
SDTV broadcast (D1)	720	486	144

6.4 Summary

The target of the architecture in this chapter is to reduce the computational complexity of the ME process. As a result, the overall encoding time of HD video sequences will be greatly decreased with an acceptable degradation in the coding efficiency (i.e., video quality). This will be achieved by using high speed ME algorithm which is ASWS as well as prediction technique. The implemented implementation ignore the unneeded operations in ME process. Also, the search area in this architecture is dynamic as it depends on the predicted motion vector PMV which will be most of the time indicating smaller search area size (i.e. less than 64×64). The second implemented architecture not only reduces the computation complexity of the ME, but it also reduces the ME overall time consumption. Additionally, same features of first implemented architecture are applied here such as re-using old data that have been already fetched from the main memory. As a result, both I/O BW and redundant computations will be reduced. Consequently, a huge savings in the video encoding time can be achieved.

Chapter 7

CONCLUSION AND RECOMMENDATIONS

Chapter 7 CONCLUSION AND RECOMMENDATIONS

This dissertation has presented several efficient techniques and algorithms that reduce the I/O memory BW and speed up the Motion Estimation (ME) process of H.265/HEVC CODEC. There are two main problems when implementing the hardware of the ME unit in H.265/HEVC standard: the huge number of data required for performing the ME process and the high computational complexity compared to previous standards. In this thesis, two architectures were implemented to solve the previous two problems.

The first architecture which using some previous work (Ismail, El-Medany, Al-Junaid, & Abdelgawad, 2014) solves the issue of huge data required from the external memory to perform the ME process. This is achieved by using a smart data flow that efficiently use level A and level B data reuse principle. Data reuse principle is mainly used to reduce the I/O memory BW. Consequently, the overall time required to fetch the data from the external memory is greatly decreased. The implemented ME co-processor in first architecture can perform ME at 9 frame per second using 4K (3840×2160) at 454 MHz.

The second architecture provides reductions in the computations of the ME process. This is achieved by using prediction and Adaptive Search Window Size (ASWS) techniques. The second architecture can perform ME at 10 frame per second using 4K (3840×2160) at 527 MHz with 100% unitization of all hardware components. The aim of the two

implemented architectures is to process real-time video coding for 4K and Ultra High Definition TV (UHD-TV) resolutions.

Table 7.1 shows a comparison between the two implemented architectures.

(Table 7.1)
Hardware complexity comparison for the two implemented architectures

FPGA Xilinx ZYNQ-7 ZC706	**Arch-1**	**Arch-2**
# of PEs	1024	1024
Search area size	64×64 pixels	16×16, 32×32, 48×48,and 64×64
Block size	32×32 pixels	32×32 pixels
# of slice LUTs	71.52 K	71.61 K
# of slice registers	62.55 K	62.59 K
# of LUT flip flop	132.918 K	133.04 K
Frequency	454 MHz	527 MHz
Throughput	1/6116	1/6116
Data reuse level	Levels A and B	Levels A and B
Area	Lesser Area than Arch-2	Increased Area but result in speeding up the ME process

The recent standard H.265/HEVC offers low bit-rate transmission by implementing many new features. Implementing all of the introduced features of H.265/HEVC will dramatically increase the complexity of the H.265/HEVC encoder. In this work, many solutions are provided and implemented to reduce the complexity of the H.265/HEVC ME unit as

mentioned above. For real time video coding, hardware implementation becomes more essential due to the limitations of software. Full search algorithms are the most suited due to their attributes. The main problem of such algorithms is their high computations which is the main bottleneck that prevents using such algorithms in real time video applications. In this thesis, many algorithms were used overcome the high computations problem such as prediction algorithm to move the search origin to a new search origin that is very close or equal to the AMV, and Adaptive Search Window Size (ASWS) algorithm which is used to decide the size of the search area according to the expected motion activity of a block.

There are three main categories that can be considered in the future work as follows:

A. Prediction algorithms: Different prediction algorithms can be used to move the search origin to a new search origin that is very close or equal to the AMV such as modified average, modified median, median temporal SAD, and weighted average temporal SAD.

B. Adaptive algorithms: Such as Adaptive Early Search Termination (AEST) which is used to check if the predicted new search origin can be considered as a best match AMV.

C. Dynamic Algorithms: The previous adaptive algorithm is not efficient as well as the dynamic algorithms since they have fixed thresholds. Dynamic algorithms are more mature and smart since they adaptively adjust the thresholds according to pre-estimated parameters that efficiently detect the motion activity of a block. Dynamic Early Stop Search Termination (DESST), Dynamic Padding Pixels Motion Estimation (DPP-ME), Dynamic Internal

Stop Search (DISS), and Dynamic Early Stop Search (DESS) are examples of dynamic algorithms which are used to improve the resolution as well as the bit-rate with a little increase in the computations compared with the previous adaptive algorithm.

Combining the previous categories together in one system can greatly reduce the computations. Also, data reuse principle level C and Level D can be implemented to minimize memory I/O Bandwidth, but this will increase the hardware complexity.

References

Ahmed, A., Shahid, M. U., Martina, M., Magli, E., & Masera, G. (2013). *VLSI Architecture for Low-Complexity Motion Estimation in H.264 Multiview Video Coding.* Paper presented at the Euromicro Conference on Digital System Design (DSD).

Alkanhal, M., Turaga, D., & Chen, T. (1999). Correlation based search algorithms for motion estimation. *Picture Coding Symposium*, 99-102.

Bailo, G., Bariani, M., Barbieri, I., & Raggio, M. (2004). *Search window size decision for motion estimation algorithm in H.264 video coder.* Paper presented at the International Conference on Image Processing (ICIP).

Bossen, F., Bross, B., Suhring, K., & Flynn, D. (2012). HEVC Complexity and Implementation Analysis. *IEEE Transactions on Circuits and Systems for Video Technology, 22*(12), 1685-1696. doi: 10.1109/tcsvt.2012.2221255.

Chen, X., Liu, P., Zhou, D., Zhu, J., Pan, X., & Goto, S. (2010). A high performance and low bandwidth multi-standard motion compensation design for HD video decoder. *IEICE Transactions on Electronics, E93-C*(3), 253-260.

Chun-Ho, C., & Lai-Man, P. (2002). A novel cross-diamond search algorithm for fast block motion estimation. *IEEE Transactions on Circuits and Systems for Video Technology, 12*(12), 1168-1177. doi: 10.1109/tcsvt.2002.806815.

De Cock, J., Notebaert, S., Lambert, P., & Van de Walle, R. (2008). *Advanced bitstream rewriting from H.264/AVC to SVC.* Paper presented at the 15th IEEE International Conference on Image Processing (ICIP).

Feng, J., Lo, K. T., & Karbowiak, A. E. (1995). Adaptive block matching motion estimation algorithm for video coding. *Electronic Letters, 31*(18), 1542-1543. doi: 10.1049/el:19951047.

Goel, S., Ismail, Y., & Bayoumi, M. A. (2005). *Adaptive search window size algorithm for fast motion estimation in H.264/AVC standard.*

Paper presented at the 48th Midwest Symposium on Circuits and Systems.

Goel, S., Ismail, Y., Devulapalli, P., McNeely, J., & Bayoumi, M. A. (2006). *An Efficient Data Reuse Motion Estimation Engine.* Paper presented at the IEEE Workshop on Signal Processing Systems Design and Implementation (SIPS).

Grellert, M., Shafique, M., Karim Khan, M. U., Agostini, L., Mattos, J. C. B., & Henkel, J. (2013). *An adaptive workload management scheme for HEVC encoding.* Paper presented at the 20th IEEE International Conference on Image Processing (ICIP).

Huong, H., Klepko, R., Nam, N., & Demin, W. (2011). A high performance hardware architecture for multi-frame hierarchical motion estimation. *IEEE Transactions on Consumer Electronics, 57*(2), 794-801. doi: 10.1109/tce.2011.5955224.

Ismail, Y. (2010). *Efficient Smart Algorithms and Architectures for Real-time Video Transmission in Pixel and Frequency Domains.* Doctor of Philosophy, University of Louisiana at Lafayette, Graduate Faculty.

Ismail, Y., El-Medany, W., Al-Junaid, H., & Abdelgawad, A. (2014). High performance architecture for real-time HDTV broadcasting. [journal article]. *Journal of Real-Time Image Processing*, 1-12. doi: 10.1007/s11554-014-0430-1.

Ismail, Y., McNeely, J. B., Shaaban, M., Mahmoud, H., & Bayoumi, M. A. (2012). Fast Motion Estimation System Using Dynamic Models for H.264/AVC Video Coding. *IEEE Transactions on Circuits and Systems for Video Technology, 22*(1), 28-42. doi: 10.1109/tcsvt.2011.2148450.

ITU-T, & Int. (2003). Advanced video coding for generic audiovisual services. *Int. Telecommun. Union-Telecommun. (ITU-T) and Int. Standards Org./Int. Electrotech. Comm. (ISO/IEC) JTC 1.*

ITU-T, & ISO/IEC-JTC1. (1994). Generic Coding of Moving Pictures and Associated Audio Information - Part 2: Video.

ITU-T, & JVT. (2003). Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T Rec.

H.264/ISO/IEC 14 496-10 AVC. *Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, JVTG050.*

ITU-T, & JVT. (2012). High Efficiency Video Coding (HEVC) Text Specification Draft 6. *Joint Video Team of ISO/IEC MPEG and ITU-T VCEG.*

Jen-Chieh, T., Tian-Sheuan, C., & Chein-Wei, J. (2002). On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture. *IEEE Transactions on Circuits and Systems for Video Technology, 12*(1), 61-72. doi: 10.1109/76.981846.

Kai, S. (1998). *Adaptive motion estimation based on statistical sum of absolute difference.* Paper presented at the International Conference on Image Processing (ICIP).

Kim, J., Jun, D., Jeong, S., Cho, S., Choi, J. S., Kim, J., & Ahn, C. (2012). An SAD-Based Selective Bi-prediction Method for Fast Motion Estimation in High Efficiency Video Coding. *ETRI Journal, 34*(5), 753-758. doi: 10.4218/etrij.12.0112.0186.

Koga, T., Iinuma, K., & Iijima, A. (1981). Motion-compensated interframe coding for video conferencing. *Proc. NTC81*, C9.6 - 9.6.5.

Lai-Man, P., & Wing-Chung, M. (1996). A novel four-step search algorithm for fast block motion estimation. *IEEE Transactions on Circuits and Systems for Video Technology, 6*(3), 313-317. doi: 10.1109/76.499840.

Malvar, H. S., Hallapuro, A., Karczewicz, M., & Kerofsky, L. (2003). Low-complexity transform and quantization in H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology, 13*(7), 598-603. doi: 10.1109/tcsvt.2003.814964.

Marlow, S., Ng, J., & McArdle, C. (1997). *Efficient motion estimation using multiple log searching and adaptive search windows.* Paper presented at the Sixth International Conference on Image Processing and Its Applications.

Minocha, J., & Shanbhag, N. R. (1999). *A low power data-adaptive motion estimation algorithm.* Paper presented at the IEEE 3rd Workshop onMultimedia Signal Processing.

Nie, Y., & Kai-Kuang, M. (2002). Adaptive rood pattern search for fast block-matching motion estimation. *IEEE Transactions on Image Processing, 11*(12), 1442-1449. doi: 10.1109/tip.2002.806251.

Ohm, J., Sullivan, G. J., Schwarz, H., Thiow Keng, T., & Wiegand, T. (2012). Comparison of the Coding Efficiency of Video Coding Standards 2014 including High Efficiency Video Coding (HEVC). *IEEE Transactions on Circuits and Systems for Video Technology, 22*(12), 1669-1684. doi: 10.1109/tcsvt.2012.2221192.

P., D., & Marikkannan, S. (2014). Implementation of Motion Estimation Algorithm for H.265/HEVC. *International Conference on Signal Processing, Embedded System and Communication Technologies and their applications for sustainable and renewable energy (ICSECSRE)*.

Pastuszak, G., & Jakubowski, M. (2013). Adaptive Computationally Scalable Motion Estimation for the Hardware H.264/AVC Encoder. *IEEE Transactions on Circuits and Systems for Video Technology, 23*(5), 802-812. doi: 10.1109/tcsvt.2012.2223791.

Renxiang, L., Bing, Z., & Liou, M. L. (1994). A new three-step search algorithm for block motion estimation. *IEEE Transactions on Circuits and Systems for Video Technology, 4*(4), 438-442. doi: 10.1109/76.313138.

Saponara, S., & Fanucci, L. (2004). Data-adaptive motion estimation algorithm and VLSI architecture design for low-power video systems. *IEE Proceedings - Computers and Digital Techniques, 151*(1), 51-59. doi: 10.1049/ip-cdt:20030858.

Shan, Z., & Kai-Kuang, M. (1997). *A new diamond search algorithm for fast block matching motion estimation.* Paper presented at the Proceedings of International Conference on Information, Communications and Signal Processing (ICICS).

Sugito, Y., Iguchi, K., Ichigaya, A., Chida, K., Sakaida, S., Shishikui, Y., . . . Sekiguchi, S.-i. (2013). *Development of the Super Hi-Vision HEVC/H.265 Real-Time Encoder.* Paper presented at the Annual Technical Conference & Exhibition, SMPTE.

Sullivan, G. J., Ohm, J., Woo-Jin, H., & Wiegand, T. (2012). Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE*

Transactions on Circuits and Systems for Video Technology, 22(12), 1649-1668. doi: 10.1109/tcsvt.2012.2221191.

Sung-Fang, T., Chung-Te, L., Hsuan-Hung, C., Pei-Kuei, T., Kuan-Yu, C., & Liang-Gee, C. (2013). *A 1062Mpixels/s 8192x4320p High Efficiency Video Coding (H.265) encoder chip.* Paper presented at the Symposium on VLSI Circuits (VLSIC).

Timhillone Software Co.,Ltd. (2015). Retrieved from http://www.h264soft.com/introduce.html

Video coding for low bit rate communication. (2000). *Int. Telecommun. Union-Telecommun. (ITU-T), Recommendation H.263, version 1, 1995; version 2,1998; version 3.*

Wiegand, T., Sullivan, G. J., Bjontegaard, G., & Luthra, A. (2003). Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology, 13*(7), 560-576. doi: 10.1109/tcsvt.2003.815165.

Woong, I. L. C., Byeungwoo, J., & Jechang, J. (2003). *Fast motion estimation with modified diamond search for variable motion block sizes.* Paper presented at the International Conference on Image Processing (ICIP).

Yeong-Kang, L., & Lien-Fei, C. (2003). *A high data-reuse architecture with double-slice processing for full-search block-matching algorithm.* Paper presented at the Proceedings of the 2003 International Symposium on Circuits and Systems (ISCAS).

Zhi, Y., Jiajun, B., Chun, C., & Xu, L. (2005). *Fast predictive variable-block-size motion estimation for H.264/AVC.* Paper presented at the IEEE International Conference on Multimedia and Expo (ICME).

Zhou, J., Zhou, D., he, G., & Goto, S. (2011). Cache based motion compensation architecture for quad-HD H.264/AVC video decoder. *IEICE Transactions on Electronics, E94-C*(4), 439-447. doi: 10.1587/transele.E94.C.439.

ملخص الدراسة

وفر نظام التشفير H.265/HEVC ما يصل إلى 50٪ في معدل إرسال إشارات الفيديو بالمقارنة مع نظام التشفير السابق H.264/AVC. ونتيجة لذلك، يمكن تحقيق جودة عالية في عمليات التشفير وإرسال إشارات الفيديو عالية الوضوح مثل 4K و(UHD-TV). تحقيق هذا الحد المرتفع من معدل إرسال إشارات الفيديو يتطلب إزالة كل الأجزاء المكرره في إشارات الفيديو قبل إرسالها. في هذه الأطروحة نحن نركز على إزالة الأجزاء المكرره عن طريق استخدم عملية تقدير الحركة (ME) (Motion Estimation). وحدة تقدير الحركة (ME) هي واحدة من أكثر المعالجات استهلاكاً للوقت في وحدة التشفير H.265/HEVC لأنها تستهلك ما يقرب من 84٪ من الوقت الكلي للتشفير. هناك نوعان من المشاكل الرئيسية لنظام التشفير H.265/HEVC في وحدة تقدير الحركة (ME): العدد الهائل من البيانات اللازمة لتنفيذ عملية تقدير الحركة (ME) والعمليات الحسابية الكثيرة اللازمة لإجراء عملية تقدير الحركة (ME) بالمقارنة مع انظمة التشفير السابقة. في هذه الأطروحة، تم تنفيذ إثنين من المعالجات لحل المشكلتين السابقتين. المعالج الأول الذي تم تنفيذه بالاستعانه بعمل سابق (Ismail, El-Medany, Al-Junaid, & Abdelgawad, 2014) يحل مشكلة العدد الهائل من البيانات اللازمة لتنفيذ عملية تقدير الحركة (ME). ويتحقق ذلك باستخدام تدفق البيانات بطريقة ذكية عن طريق استخدام البيانات المستخدمة سابقاً وذلك باسخدام مستوى A ومستوى B من مبدأ إعادة استخدام البيانات. ويستخدم مبدأ إعادة استخدام البيانات للتقليل من كمية إدخال وإخراج البيانات من وإلى الذاكرة. وبالتالي فإن الوقت الكلي المطلوب لجلب البيانات من الذاكرة الخارجية ينخفض إلى حد كبير. نفذت عملية التشفير باستخدام وحدة تقدير الحركة (ME) في المعالج الأول معدل 9 إطار فيديو في الثانية الواحدة باستخدام دقة 4K (3840 × 2160) وبتردد 454 ميغاهرتز. أما المعالج الثاني تم تنفيذه بالاستعانه ببرنامج سابق (Goel, Ismail, & Bayoumi, 2005) فهو يقلل الكثير من التعقيد الحسابي في عملية تقدير الحركة (ME) وعليه فإن وحدة تقدير الحركة (ME) في المعالج الثاني يمكنها تنفيذ 10 إطار فيديو في الثانية الواحدة باستخدام دقة 4K (3840 × 2160) وبتردد 527 ميغاهرتز مع 100٪ استهلاك لجميع مكونات المعالج.

جامعة البحرين

كلية تقنية المعلومات



معالج سريع لتقدير حركة إشارات فيديو وحدة التشفير H.265/HEVC

اطروحه مقدمة كجزء من متطلبات الحصول على درجة الماجستير في تقنية المعلومات

إعداد

محمد نبيل حماد

20082351

إشراف

د. ياسر علي إسماعيل

(أستاذ مساعد)

جامعة البحرين

مملكة البحرين

أبريل 2016