Universidade de Aveiro Departamento de Eletrónica, Telecomunicações e Informática
2016

PURNACHAND NALLURI

ALGORITMO DE ESTIMAÇÃO DE MOVIMENTO E SUA ARQUITETURA DE HARDWARE PARA HEVC

A FAST MOTION ESTIMATION ALGORITHM AND ITS VLSI ARCHITECTURE FOR HIGH EFFICIENCY VIDEO CODING

Universidade de Aveiro
2016
Departamento de Eletrónica, Telecomunicações e Informática

PURNACHAND NALLURI

ALGORITMO DE ESTIMAÇÃO DE MOVIMENTO E SUA ARQUITETURA DE HARDWARE PARA HEVC

A FAST MOTION ESTIMATION ALGORITHM AND ITS VLSI ARCHITECTURE FOR HIGH EFFICIENCY VIDEO CODING

Tese apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Doutor em Engenharia Eletrotécnica, realizada sob a orientação científica do Doutor António José Nunes Navarro Rodrigues, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e do Doutor Luis Filipe Mesquita Nero Moreira Alves, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

This PhD thesis was supported by FCT (Fundação para a Ciência e a Tecnologia), Portugal. Grant ref. :SFRH/BD/73266/2010

dedicated to my family
my brother late Govind Nalluri
my dad and mom
and my wife

O Júri

Presidente: Doutor Fernando Manuel dos Santos Ramos, Professor Catedrático, Universidade de Aveiro

Vogais: Doutor Luciano Volcan Agostini, Pró-Reitor de Pesquisa e Pós-Graduação, Universidade Federal de Pelotas, Brasil

Doutor Marco Mattavelli, Maître D'enseignement et de Recherche, École Polytechnique Fédérale de Lausanne, Suiça

Doutor Leonel Augusto Pires Seabra de Sousa, Professor Catedrático, Instituto Superior Técnico, Universidade de Lisboa

Doutor Dinis Gomes de Magalhães dos Santos, Professor Catedrático, Universidade de Aveiro

Doutor António José Nunes Navarro Rodrigues, Professor Auxiliar, Universidade de Aveiro (supervisor/orientador)

agradecimentos

The research work for this thesis was carried at Institudo de Telecomunicações (IT) and Departamento de Electrónica, Telecomunicações e Informática (DETI), Universidade de Aveiro, Campus Universitário de Santiago, Aveiro, Portugal, during the years 2010-2015.

I would like to express my gratitude to my supervisors Prof. Luis Nero Alves and Prof. Antonio Navarro for their guidance, motivation and every possible effort to carry out my research work. Prof. Luis Nero taught me how to approach towards a problem (especially in VLSI circuits and systems) in research work, and motivated at many points. Prof. Navarro's experience in video coding helped how to solve the critical problems without which this thesis could not have been accomplished.

I would like to thank Prof. Manuel Almeida Valente for his great help and support given to me in the initial days of my arrival to Portugal, and because of whom I came to know about Aveiro and FCT scholarship.

I would like to thank all my colleagues in CSI lab of IT, Aveiro for their technical support and friendship, Nuno Lourenço, Domingos Terra, Miguel Bergano, Nelson Silva and Mónica Figueiredo all made the lab a comfortable place to work. I would also like to express my thankfulness to IT administrative, HR and security staff for their cooperation throughout my stay in IT.

I would like to thank my family, my wife Sirisha and my parents (dad Prof. Dr. N. Veeraiah and mom N.V. Kumari) for their wonderful support and love that helped me survive in difficult times. I would also like to thank my uncles Dr. V. Ravikumar, Dr. G. Sahaya Bhaskaran, Dr. Y. Gandhi, Dr. K.S.V. Sudhakar, Dr. G. Nagaraju and to my cousin Valluri Ravikumar for their support throughout my PhD period.

I would like to express my gratefulness to my loving brother late Govind Nalluri, whose very thoughts are foundation to my conscience.

Finally, I would like to thank and acknowledge my funding agency FCT (Fundação para a Ciência e a Tecnologia). This research work was supported by FCT grant reference SFRH/BD/73266/2010.

palavras-chave

Codificação de vídeo, Norma HEVC, Estimação de movimento, Arquitetura de hardware, FPGA.

resumo

A codificação de vídeo tem sido usada em aplicações tais como, vídeo-vigilância, vídeo-conferência, video *streaming* e armazenamento de vídeo. Numa norma de codificação de vídeo, diversos algoritmos são combinados para comprimir o vídeo. Contudo, um desses algoritmos, a estimação de movimento é a tarefa mais complexa. Por isso, é necessário implementar esta tarefa em tempo real usando arquiteturas de hardware apropriadas. Esta tese propõe um algoritmo de estimação de movimento rápido bem como a sua implementação em tempo real. Os resultados mostram que o algoritmo e a arquitetura de hardware propostos têm melhor desempenho que os existentes. A arquitetura proposta opera a uma frequência máxima de 241.6 MHz e é capaz de processar imagens de resolução 1080p@60Hz, com todos os tamanhos de blocos especificados na norma HEVC, bem como um domínio de pesquisa de vetores de movimento até ±64 pixels.

keywords Video Coding, HEVC standard, Motion Estimation, VLSI Architecture, FPGA.

abstract Video coding has been used in applications like video surveillance, video conferencing, video streaming, video broadcasting and video storage. In a typical video coding standard, many algorithms are combined to compress a video. However, one of those algorithms, the motion estimation is the most complex task. Hence, it is necessary to implement this task in real time by using appropriate VLSI architectures. This thesis proposes a new fast motion estimation algorithm and its implementation in real time. The results show that the proposed algorithm and its motion estimation hardware architecture out performs the state of the art. The proposed architecture operates at a maximum operating frequency of 241.6 MHz and is able to process 1080p@60Hz with all possible variables block sizes specified in HEVC standard as well as with motion vector search range of up to ±64 pixels.

Table of Contents

List of Figures

List of Figures

List of Figures

List of Figures

List of Figures

List of Acronyms

AGU	Address Generation Unit
AMP	Asymmetric Mode Partitions
ASIC	Application Specific Integrated Circuit
AVC	Advanced Video Coding
BBME	Block Based Motion Estimation
BD	Bjontegaard Delta
BDM	Block Difference Measure
BRAM	Block Random Access Memory
CABAC	Context Adaptive Binary Arithmetic Coding
CAGR	Compound Annual Growth Rate
CAVLC	Context Adaptive Variable Length Coding
CB	Coding Block
CBR	Constant Bit Rate
CIF	Common Intermediate Format
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
CTB	Coding Tree Block
CTU	Coding Tree Unit
CU	Coding Unit
DCT	Discrete Cosine Transform
DSR	Dynamic Search Range
DVD	Digital Versatile Disc
EPZS	Enhanced Predictive Zonal Search
ET	Early Termination
FPGA	Field Programmable Gate Array
FME	Fractional Motion Estimation
FR	Fine Refinement

List of Acronyms

GOP	Group Of Pictures
HEVC	High Efficiency Video Coding
HD	High Definition
HDTV	High Definition TeleVision
HDL	Hardware Description Language
HM	HEVC software reference Model
HVS	Human Visual System
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
IP	Intellectual Property
ITU-T	International Telecommunication Union - Telecommunication Standardization
JCT-VC	Joint Collaborative Team on Video Coding
JVT	Joint Video Team
LDB	Low Delay Bi-predictive
LDP	Low Delay Predictive
LUT	Look Up Table
MC	Motion Compensation
MCP	Motion Compensated Predication
ME	Motion Estimation
MMVSSP	Multiple Min-point Variable Step Size Prediction
MPEG	Moving Picture Experts Group
MSE	Mean Square Error
MV	Motion Vector
MVD	Motion Vector Difference
NTSS	New three step search
PB	Prediction Block
PMV	Predicted Motion Vector
PSNR	Peak signal-to-noise ratio
PU	Prediction Unit

List of Acronyms

QCIF	Quarter Common Intermediate Format
RA	Random Access
RAM	Random Access Memory
ROM	Read Only Memory
RDO	Rate Distortion Optimization
RTL	Register Transfer Level
SAD	Sum of Absolute Differences
SATD	Sum of Transformed Differences
SD	Standard Definition
SDTV	Standard Definition TeleVision
SEA	Successive Elimination Algorithm
SIMD	Single Instruction Multiple Data
SR	Search Range
SSD	Sum of Squared Differences
SUMHEXS	Simple Unsymmetrical-Cross Multi Hexagon-Grid Search
SW	Search Window
TB	Transform Block
TSS	Three step search
TU	Transform Unit
TZSearch	Test Zone Search
UMHEXS	Unsymmetrical-Cross Multi Hexagon-Grid Search
VBSME	Variable Block Size Motion Estimation
VCEG	Video Coding Experts Group
VoD	Video on Demand

1 INTRODUCTION

1.1 PROBLEM DEFINITION AND MOTIVATION

With the advent of digital revolution, there is a huge change from analog and mechanical technology to digital technology. This revolution created a huge impact on almost all types of industries in the areas of arts, entertainment, communications, marketing, media etc. and has marked the beginning of information age. With more and more advances in computational speed of digital computing devices, there is an ever growing increase in generation and demand for digital information including data, audio and video. Amongst all these multimedia data, the digital video is more complicated and challenging to process (which is the used in many applications including storage, surveillance, web streaming, broadcasting, video communications and conferencing) as the amount of data required for video is huge compared to data and audio information. Hence, there is an endless research actively going on from the past three decades to compress the digital video.

1.1.1 The Need of Video Compression

Compression of video is always necessary in streaming, broadcasting and storage applications. With the growing demand for mobile and fixed line internet video, the demand for compressing the video is also growing by including more sophisticated

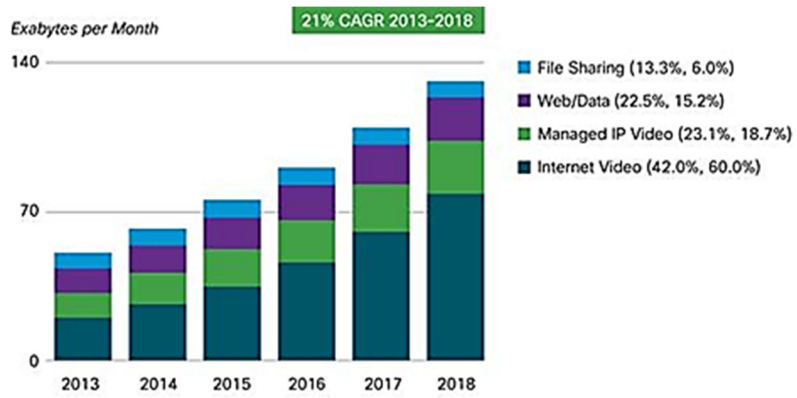


Fig. 1.1 Forecast of Global IP Traffic in Future by Application Category (Source Cisco VNI, 2014)

compression algorithms. Fig. 1.1 shows the trend and forecast of future IP (Internet Protocol) video, which includes internet streamed video, IP VoD (Video on Demand), video file sharing, video-streamed gaming and videoconferencing [1]. According to

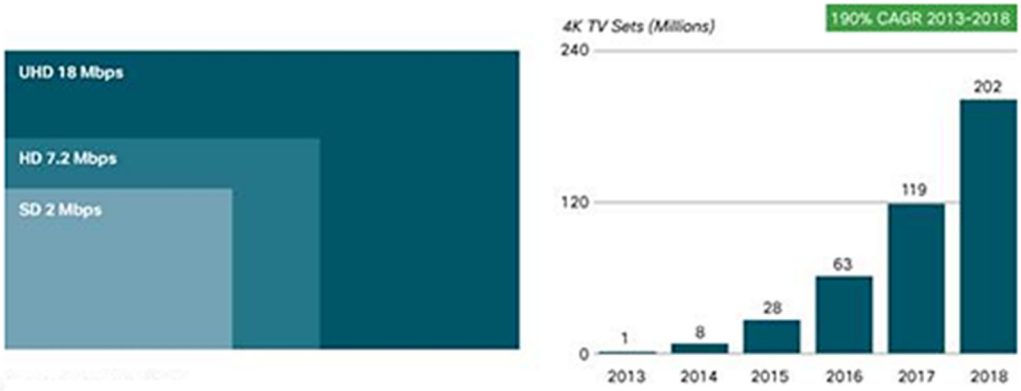


Fig. 1.2 Illustration of Increased trend in the usage of HD and UHD TVs (Source: Cisco VNI, 2014)

Cisco, this trend continues to grow and the expected internet video traffic by 2018 will be about 79% of total internet traffic. In the figure, the percentage values in the parenthesis next to the legend shows the relative traffic shares in the year 2013 and 2018 respectively. CAGR represents compound annual growth rate. From the figure it is clear that there is a growth in the rate for internet video (internet video + managed IP video) from 63% to 79% from the year 2013 to 2018, with a CAGR of 21%. The highlights of some of the predictions and forecasts about video service demands according to Cisco [1] are:

- Global IP video traffic will increase to 79% (of total IP traffic) by 2018 compared to 66% in 2013
- There is a rapid pace in the usage of Internet video to TV (via set top boxes).
- By 2018, the amount of VoD traffic will be equivalent to 6 billion DVDs per month.
- The number of consumers with 4k television sets will increase to 200 million by 2018 compared to 1 million in 2013 and 28 million in 2015, with a CAGR of 190%, as shown in Fig. 1.2.

From the above highlights, it is clear that there is always a growing trend in the video content generation and usage. Due to increase in 4k TV sets, there is also an increase in demand for compressing the video in broadcast services for providing video services at HD (High Definition) and UHD (Ultra HD). Further, the video content that is generated, need to be stored at data centers and cloud servers using hard disks or solid-

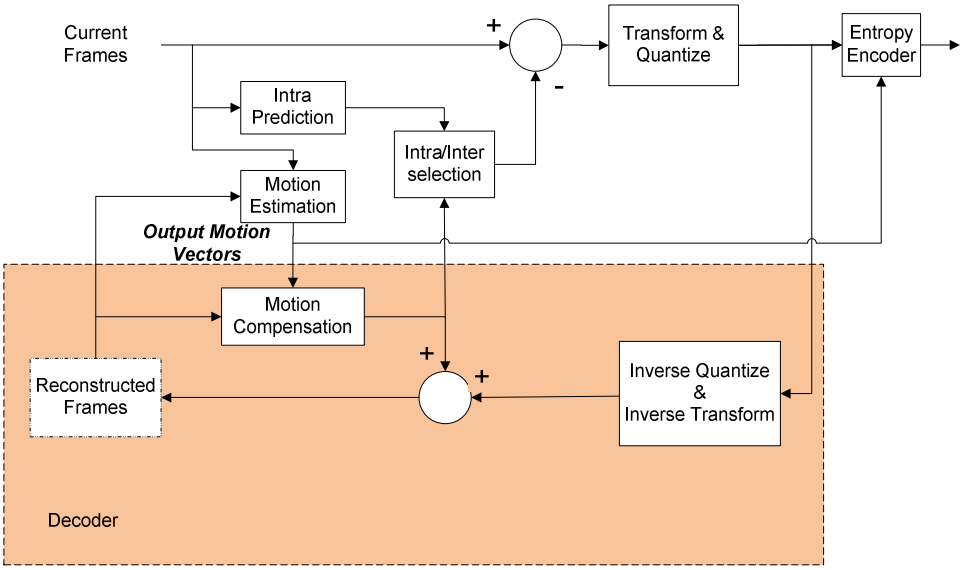


Fig. 1.3 Block Diagram of Block-based Video Encoder

state drives or using any other storage devices. The higher the compression of video, the lesser is the amount of memory used at these storage devices. Hence, there is always a growing demand for compressing the video.

1.1.2 Block-based Video Encoder System

Block based video coding is one of the mostly used compression technique for video. In block based video coding, each video frame is split into coding blocks. Each coding block is predicted, transformed. quantized and entropy encoded. The block diagram of a typical video encoder is shown in Fig. 1.3. Each frame is split into various coding blocks and then the blocks of first frame in a video frame sequence is intra predicted and encoded (prediction of image blocks within the frame) and the rest of the frames' blocks are either intra predicted or inter predicted (prediction between frames using Motion Estimation (ME) and motion compensation blocks). Nevertheless, in inter-prediction, in every frame the first block of slice (group of blocks) can be intra-coded and depends on the mode-decision algorithm. The ME block predicts and estimates motion between frames and generate the Motion Vectors (MVs). The MVs are entropy encoded and also sent to motion compensation block. The motion compensation block uses these MVs to generate motion compensated frames. These motion compensated frames are subtracted from the original frames (current frames) to generate residual frame blocks. This residual information is transformed, quantized and then entropy encoded. To generate identical predicted information in the decoder side, the encoder

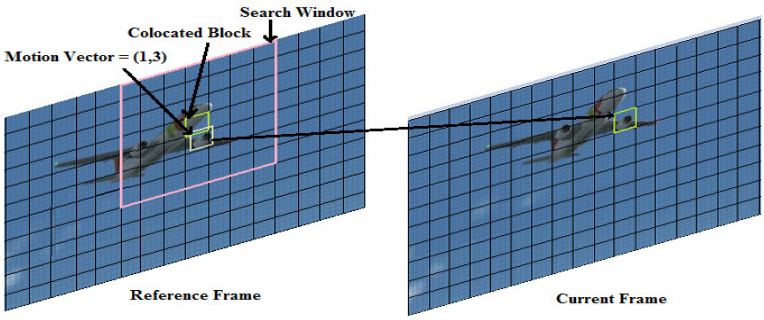


Fig. 1.4 Illustration of Motion Estimation Process

typically includes a decoding loop to reconstruct the original frames using predicted information. The decoding blocks are shown in dark color in Fig. 1.3. To do this, the decoder takes the quantized information and passes it through inverse quantization and inverse transformation blocks. Then, the obtained data is added to the predicted data to reconstruct the subsequent video frames.

1.1.3 The Motion Estimation Problem

Motion Estimation (ME) is the essential task in block based video encoders. It contributes to reduce the overall bitrate of a video signal by predicting and estimating the Motion Vectors (MVs) for each block in every frame. A good estimate of motion in a frame generates less entropy information (for residual frame blocks) and fewer bits to encode it and hence the compression ratio will be increased. So, for each block in every frame, the main task of ME is to estimate the motion content by finding the best matched block in the previously encoded frame (reference frame) region of interest (also called search window). The process of ME is shown in Fig. 1.4. For every block of the current frame, a new Search Window (SW) is defined and the ME algorithm searches for the best matched block using a predefined cost function. The final output of the ME are the coordinates of the optimal MV and its cost.

The ME problem can be formulated using (1.1) and (1.2), where MV(x,y) represents the optimal motion vector, SW represents the search window, J represents the Lagrangian cost function, D represents the distortion and R represents the bitrate required to encode the motion vector MV and λ_{MV} is the lagrangian multiplier. The distortion function usually employed is either SAD (Sum of Absolute Difference) or SSD (Sum of Squared difference). The SAD is widely used distortion function that can

$$MV(x_i, y_j) = \min_{i,j}\{J_{cost}(x+i, y+j)\} \quad (x+i, y+j)\ \epsilon\ SW \tag{1.1}$$

$$J_{cost} = D + \lambda_{MV} R \tag{1.2}$$

$$D = SAD(x,y) = \sum_{i=1}^{M}\sum_{j=1}^{N}\left|C(x_i, y_j) - R(x_i, y_j)\right| \tag{1.3}$$

be defined using (1.3), where C represents the current block, R represents the reference block and MxN represents size of the block in pixels.

Block based video encoders typically use Block Matching Algorithms (BMAs) to perform ME. In case, the ME algorithm searches each and every block in the SW, called Full Search (FS) algorithm. Searching every block in the entire SW increases the complexity of the encoder. Hence video encoders employ fast ME algorithms which skip most of the blocks that are unlikely to be the optimum MV. But by using fast search algorithms we may experience some degradation in the also decrease the output video quality as the estimated optimum MVs may not be accurate enough. Hence a good fast ME algorithm is necessary to decrease the ME complexity but with negligible loss in compression ratio and output video quality.

1.1.4 The Demand for Video Core Complexity

Although the fast ME algorithms reduce the ME complexity when compared to FS algorithm, they have higher complexity when compared with the other operations of the video encoder. Hence a good hardware architecture is necessary to reduce the complexity and exploit parallelism and pipelining at various levels of ME operation.

On the other hand, designing hardware for ME operation is challenging task as the architecture is constrained by many real time parameters like hardware resources usage, critical path delay, off-chip to on-chip memory bandwidth, on-chip memory buffer size, power consumption etc. Due to the increase in the complexity of advanced video coding standards (like the latest standard HEVC [2]), the hardware complexity of video core also increases. Fig. 1.5 shows the trend in relative complexity requirement of video encoder in mobile application processors, plotted over the years 2004 through 2020 [3]. The figure shows that the complexity of video core by 2020 is expected to increase

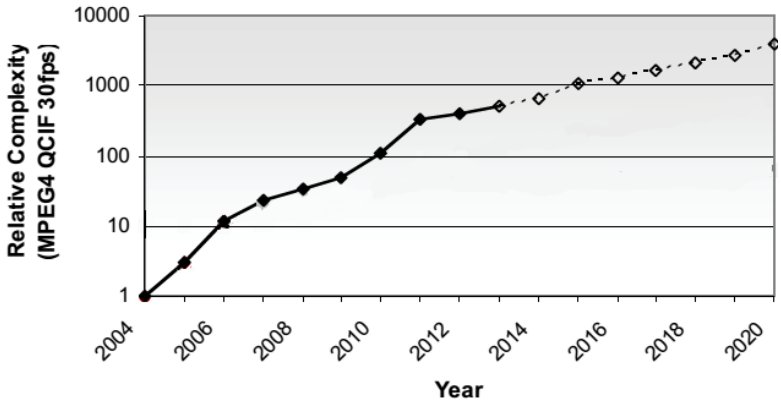


Fig. 1.5 Trend in Increase of Relative Hardware Complexity for a Mobile Video Processor from Years 2004 to 2020

almost exponentially compared to that of video core complexity in 2014. Hence it is very crucial and essential to design a dedicated hardware that meets the performance requirements as well as considering the hardware design costs like area, memory size, memory bandwidth etc.

1.2 OBJECTIVES OF THESIS

The problem of motion estimation is always challenging as the motion of objects in a video move randomly in both direction and magnitude. Many algorithms exist in the literature [62-104], where the origin of this problem dates back as old as 1981 [77]. When the video coding projects started standardizing and getting evolved to newer standards, the problem of ME was also getting more and more complicated. The latest video coding standard is the HEVC developed by JCT-VC which is a joint collaboration ITU-T and ISO/IEC [2, 4]. Compared to its prior standard H.264/AVC [5, 6], the coding block size increased from 16x16 to 64x64 pixels. Therefore the number of block modes also increased and so as the complexity of ME. Hence, the present thesis focused on reducing the computational complexity of ME through the proposal of novel hybrid ME algorithm. The algorithm is called hybrid because it comprises of various tools to reduce the computational complexity problem of ME. The algorithm is designed and verified using HEVC reference software [7, 8], which is the software for the latest video coding standard.

Although at the search window level, the algorithm for ME is independent of the block size, the hardware implementation of the architecture depends on the current

block size of the search process. Hence implementation of ME that is compatible with the coding sizes standardized in HEVC is another objective of this thesis. Hence, after designing and verifying the algorithm in software, a hardware architecture was designed at RTL (Register Transfer Level) using Verilog HDL (Hardware Description Language) [9]. The architecture is verified and synthesized using an FPGA (Xilinx Virtex-6 FPGA) [10].

The summary of objectives of this thesis are listed below:

- To design and verify a novel hybrid fast ME algorithm which outperforms the fast ME algorithm present in HEVC reference software.
- To implement the proposed ME algorithm in HDL that is compatible with the HEVC coding standard and to verify it using an FPGA.

1.3 SUMMARY OF ORIGINAL CONTRIBUTIONS

The summary of publications are listed below:

1. Nalluri, P; Alves, L. N.; Navarro, A.; "Complexity Reduction Methods for Fast Motion Estimation in HEVC", Elsevier Journal of Signal Processing: Image Communication (EURASIP), Vol. 39, Part A, pp. 280 - 292, November 2015.
2. Nalluri, P; Alves, L. N.; Navarro, A.; "High Speed Sad Architectures For Variable Block Size Motion Estimation In HEVC Video Coding", Proc. IEEE International Conf. on Image Processing – ICIP-2014, Paris, France, , Oct. 2014.
3. Nalluri, P; Alves, L. N.; Navarro, A.; “A novel SAD architecture for variable block size motion estimation in HEVC video coding”, Proc. IEEE International Symposium on System on Chip (SoC), 2013, pp.1-4, 23-24 Oct. 2013
4. Nalluri, P; Alves, L. N.; Navarro, A.; "Fast Motion Estimation Algorithm for HEVC Video Encoder", Proc Conf. on Telecommunications - ConfTele, Castelo Branco, Portugal, Vol. 1, pp. 1 - 4, May, 2013.
5. Nalluri, P; Alves, L. N.; Navarro, A.; "FPGA Based Synchronous Multi-Port SRAM Architecture for Motion Estimation", Proc Jornadas sobre Sistemas Reconfiguráveis - REC, Coimbra, Portugal, Vol. 9, pp. 89 - 92, February, 2013.
6. Nalluri, P; Alves, L. N.; Navarro, A.; "Fast Motion Estimation Algorithm for HEVC", Proc IEEE International Conf. on Consumer Electronics - ICCE, Berlin, Germany, Vol. 3, pp. 34 - 37, September, 2012.

7. Nalluri, P; Alves, L. N.; Navarro, A.; "Improvements to TZSearch Motion Estimation Algorithm for Multiview Video Coding", Proc. IEEE International Conf. on Systems, Signals and Image Processing - IWSSIP, Vienna, Austria, Vol. 19, pp. 388 - 391, April, 2012.

8. Nalluri, P; Alves, L. N.; Navarro, A.; "A Fast Motion Estimation Algorithm and its FPGA based hardware architecture for HEVC" (Submitted).

1.4 SUMMARY OF THESIS

The rest of the chapters in the thesis is organized as follows.

Chapter 2 describes the fundamentals and concepts of digital video coding. The chapter outlines a history of video coding standards, encoder block diagram of the latest video coding standard HEVC and describes the function of each block in the encoder. Further, this chapter explores various features and techniques involved in motion estimation.

Chapter 3 explains the state-of-the-art of ME algorithms. The chapter also explains ME hardware architecture and explains the state-of-the-art of ME hardware architectures.

Chapter 4 explains in detail each of the proposed ME method implemented and verified using HEVC reference software. The chapter explains in detail, each method that is used to design the algorithms and shows the results simulated in HEVC reference software.

Chapter 5 explains the proposed ME hardware architecture which is designed using HDL. This chapter explains in detail, the design methodology used in FPGA, state-diagrams and the simulated results and compares the synthesis results with recent works.

Chapter 6 concludes the thesis by summarizing the achieved results and presents future research directions.

2 FUNDAMENTALS OF DIGITAL VIDEO CODING AND MOTION ESTIMATION

2.1 INTRODUCTION

A video is a sequence of still images displayed at a fixed frame rate. The net after-effect is a video with objects moving in a background. If all the pictures (frames) in the video are encoded individually, then the size of the complete video will be equal to the sum of the bits obtained from each frame. This is practically not possible for large videos either in storage or communication applications. Hence there is a huge necessity to remove the redundant information in a video [11].

2.2 DIGITAL VIDEO CODING TERMINOLOGY

2.2.1 Block Based Video Coding

As explained in Chapter 1, in block based video coding, the sequence of images is compressed by dividing each frame into blocks. Each block is then motion compensated (predicted using previously coded neighboring blocks), transform coded, quantized and finally entropy coded (removes statistical redundant information). Each of these techniques is explained in the subsequent sections. The decoder receives these coded blocks and generates frames which are displayed at a fixed frame rate [12], [13].

Besides block based coding there are many other types of video coding like pixel-based video coding, content based video coding, fractal based video coding etc. but the block based video coding is the most widely used and efficient way of implementing a video codec either in software or in real-time due to its regularity in the coding structure. Some of examples of block based codecs include MPEG based codecs like MPEG-1, MPEG-2, MPEG-4, VCEG based codecs H.261, H.263 and their joint collaborated codecs like H.264, HEVC, etc.

2.2.2 Group of Pictures (GOP)

In block based video coding, the entire video is divided into a group of picture sequences so that the video encoder can further make the encoding process easy. Each Group of Pictures (GOP) contains a fixed number of frames [14]. Each frame is either intra coded or inter predicted depending upon the GOP pattern.

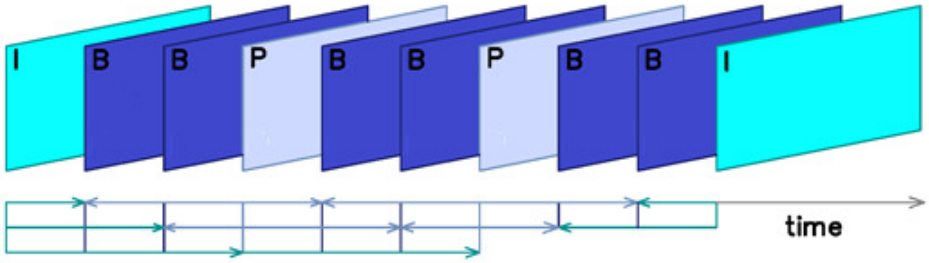


Fig. 2.1 Frames in a GOP with M=3 and N=10

A GOP typically contains three types of frames – Intra frame (or I-frame), Predictive frame (or P-frame) and Bi-predictive frame (or B-frame). The intra frame is coded independent of other frames and is coded just like a still image. The blocks in P-frame are coded using past frames that are already encoded. The blocks in a B-frame can use both past and future frames (stored in a frame buffer) for prediction. The GOP structure is typically represented by the order of these three types of frames I, P and B. The GOP structure is typically represented by two parameters, the GOP size and the maximum prediction depth. In MPEG based codecs, these values are represented by letters M (maximum prediction depth) and N (GOP size) [15]. For example M = 3 and N = 12 represents the GOP structure IBBPBBPBBPBBIBBP… and so on, which shows that the distance between two successive I frames is 12 (GOP size or N=12) and the distance between two successive P frames is 3 (M =3). The value N represents the number of frames in GOP in which the entire video is repeated with same pattern of frames with the first frame being Intra frame. The value M represent the maximum value of a reference frame index from a current frame. Fig. 2.1 represents the GOP structure of above example (IBBPBBPBBPI…, N=10). The arrow marks in the figure indicate the reference frames used to predict the corresponding frame.

2.2.3 Pictures, Frames and Fields

In digital videos, the next level of hierarchy after GOP are frames or fields. As mentioned in the above section, the frames can be either of types I, B or P. Depending on the type of sampling, the pictures can be called either frames or fields. If the scanning lines are progressive, then the picture is termed as a frame [16]. If the picture is scanned in interlaced order, then it is called a field. Each field is either odd-field or even-field depending on the line numbers index of scanned picture. Fig. 2.2 shows a QCIF size picture, top field and bottom field of foreman test sequence (2^{nd} frame) [17].

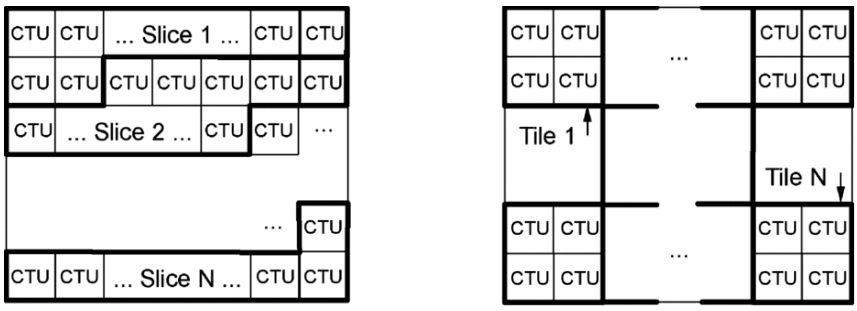
2nd frame | 2nd frame Top field | 2nd frame Bottom field

Fig. 2.2 Illustration of frame and fields in foreman video sequence

2.2.4 Slices and Coding Blocks

Each frame of video consists of two dimensional array of picture elements (or pixels). The pixel is the fundamental element of video frames. In block based video codecs each frame is divided into blocks of these pixels, and the maximum size of block depends on the video coding standard. For example in MPEG-2, H.264/AVC and HEVC the block sizes are 8x8, 16x16 and 64x64 respectively. Further, all the coding blocks in a frame need not be of same type (I/P/B type). Depending on the prediction requirement, some blocks may be of I-type, some may be P-type and some may be of B-type. I-frame contains all intra-predicted blocks (I-blocks). A P-frame contains both I-type and P-type blocks. A B-frame contains all the types of blocks (I, P and B blocks) [18].

A sequence of coding blocks can be grouped together and are called *slices*. A frame may contain one or more slices. The slice can be decoded independently from other slices in the same frame and hence useful for resynchronization of frame after data losses. Further slices can be encoded using I or P or B types. I-slice contains all I-type coding blocks. A P-slice can be encoded using coding blocks of P-type, in addition to I-



(a) (b)

Fig. 2.3 Subdivision of picture into (a) slices (b) tiles

type. A B-slice can be encoded using coding blocks of B-type, in addition to coding types available in P-slice. Fig. 2.3 (a) shows a frame that is divided into slices [19]. To enable parallelism in encoder and decoder architectures, HEVC defines another type of abstraction similar to slices which is termed as *tiles*. Tiles are typically rectangular blocks (but not necessarily) as shown in Fig. 2.3 (b). The entire frame can be partitioned into tiles, which further contain coding blocks. Each tile may contain slices and coding blocks. Further a group of tiles may also be ordered together to from slices [2].

2.2.5 Pixel Color space and Sampling Techniques

Each pixel in a video frame can be represented into luma component and chroma component. The luma component represents brightness and the chroma component represents color information. There are many color models such as RGB, CMYK, YC_bC_r, HSL (Hue Saturation and Lightness), HSV (Hue Saturation and Value), YP_bP_r, YIQ, YUV etc., each is used in a specific application, but broadly classifying, all these models can be categorized into two types. First type is to encode each value of primary colors (red, green, blue) or secondary colors (Cyan, Magenta, Yellow) separately and added to form various composite colors (like RGB model). The second type is to separate brightness (luminance) information from color (chrominance) information and to be encoded separately to get various combinations of luma-chroma values. The second type of color model is chosen in many video coding systems, since it has a flexibility to exploit the amount of chrominance information from luminance information. In principle, the Human Visual System (HVS) is more sensitive to luminance than to the color information. Hence, in color models like YC_bC_r, YUV etc, the color information is sub-sampled to reduce the bitrate or to save the memory required to store the pixel. In YC_bC_r model, the luminance value Y, chrominance-red (C_r) value and chrominance-blue (C_b) value can represented from basic RGB value as shown in (2.1).

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ C_b &= 0.564(B-Y) \\ C_r &= 0.713(R-Y) \end{aligned} \qquad 2.1$$

The sub-sampling scheme is usually notated in three part ratio R:H:V (like 4:2:2) to describe number of luminance and chrominance samples in a grid of R pixels wide and

2 pixels height. The first value (R) represents width of sampling reference pixels grid. Typically R is taken as 4. The next digit (H) indicates the number of chroma samples in the top row of Rx2 sample grid. The third digit represents number of chroma samples in the bottom row of Rx2 sample grid. For example 4:4:4 in YCbCr means for every 4x2 luma samples, it has four chroma samples in top and bottom row each. Similarly, 4:2:2 has two chroma samples in top row and two chroma samples in bottom row. The 4:2:0 format means it has only two chroma samples (Cb and Cr) in the top row but has no color samples in the bottom row of 4x2 sample grid [20]. Fig. 2.3 illustrates various chroma sub-sampling formats. H.264 and HEVC employs YC_bC_r color space model with 4:2:0 chroma sub-sampling technique.

(a) 4:4:4 (b) 4:2:2 (c) 4:1:1 (d) 4:2:0

Fig. 2.3 Illustration of various Pixel sub-sampling techniques

2.2.6 Video Formats

A video format defines the number of horizontal and vertical pixels (resolution) that are encoded or decoded. A video compression algorithm can compress many types of video formats. Depending upon the applications, the formats standardized by ITU and ISO/IEC can be categorized to three types – Intermediate formats, Standard Definition format, High Definition formats. The intermediate formats like CIF (Common Intermediate Format), and QCIF (Quarter CIF) are used in streaming applications like internet video and video conferencing. The Standard Definition (SD) Format was widely used in the older digital video televisions. The HD (High Definition) video formats are used in high-end display applications where high resolution is required. TABLE 2.1 shows various display formats with their resolution [16]. HEVC standard supports decoding and encoding upto 8k resolution [21].

2.2.7 Video Quality Measurement

To evaluate and compare the quality of video communication systems or codec, a good quality metric is essential. The most widely used quality measuring metric is PSNR (Peak Signal to Noise Ratio). The PSNR gives the relative measure for the error

TABLE 2.1 LIST OF VARIOUS DIGITAL VIDEO FORMATS WITH ITS RESOLUTION

Format	Resolution
QCIF	176 x 144
CIF	352 x 288
SD (PAL)	720 x 576
SD (NTSC)	720 x 480
HD – 720p	1280 x 720
HD – 1080i	1920 x 1080 (50 fields)
HD – 1080p	1920 x 1080 (25 frames)
2K (DCI-Digital Cinema Initiative)	2048 x 1080
UHD (Ultra High Definition)	3840 x 2160
4K (DCI)	4096 x 2160
8K	7680 x 4320

between original video and decoded video. PSNR is measured in logarithmic scale and it depends on the MSE (Mean Squared Error) between original and decoded video frames, relative to highest possible signal value in the image $(2^n\text{-}1)^2$, where n is the number of bits per image sample. The equation for PSNR is shown in (2.2) [22].

$$PSNR_{dB} = 10\log_{10}\frac{(2^n-1)^2}{MSE} = 20log_{10}\left(\frac{2^n-1}{\sqrt{MSE}}\right) \quad 2.2$$

PSNR is measured individually for luminance and chrominance components. For YUV format, there will be three PSNR values, PSNR-Y, PSNR-U, and PSNR-V. But for comparison, typically PSNR-Y (luminance) is only considered. For a 32-bit color video, there will be 8 bits allocated for luminance components. So, in (2.2) 'n' will be 8, making the numerator value equal to 255. Higher value of PSNR indicates high quality of video. Typically, a PSNR of 30-50 dB indicates that the quality of video is very good, 25-30 dB indicates an average quality, and below 25 dB indicates poor quality.

Apart from this objective quality measurement metrics, there are also subjective measurements, where the video quality is measured using the survey of viewers opinions on the decoded videos [23]. In HEVC, the objective measurement using PSNR is widely used.

2.2.8 Video Bitrate

The amount of bits at which the video encoder streams the compressed video (to a file or a communication channel) is measured in bits per second (bps) and is termed as bitrate. High bitrate video usually accommodates higher quality video (measured using

TABLE 2.2 COMMONLY USED BITRATES FOR VARIOUS STORAGE, STREAMING AND BROADCASTING STANDARDS

Target Bit rate	**Application**
16 kbps	videophone quality
1.15 Mbps (max)	VCD quality (using MPEG 1)
2.5 Mbps	480p (SD) Youtube video (using H.264/AVC)
3.5 Mbps (max)	SDTV (using MPEG 2)
5 Mbps	720p (Half HD) Youtube video (using H.264/AVC)
8 Mbps	1080p (Full HD) Youtube video (using H.264/AVC)
9.8 Mbps (max)	DVD (using MPEG 2)
8-15 Mbps	HDTV (using H.264/AVC)
29 Mbps	HD DVD
40 Mbps	1080p Blu-ray Disc (using MPEG4 AVC)

PSNR). Further, for a given bitrate constraint, a good encoder (like HEVC) can have better quality video compared to older codecs like H.264/AVC. Similarly, for a given video quality constraint, the new codec HEVC will use less bitrate compared to its predecessor H.264/AVC [2]. Some of the commonly used bitrates in various storage and streaming standards are listed in TABLE 2.2.

There are two types of bit rate schemes that a video encoder can use for video coding – Constant Bit Rate (CBR) and Variable Bit Rate (VBR). The CBR maintains a bit rate (set by user) over the entire video clip but limits the video quality over complex video segments. Live broadcasting media that are used via cable, satellite and terrestrial broadcasting require constant bitrate for their transmission. To achieve CBR the complex video frames are compressed either in real time or pre-encoded before they are transmitted.

The second type of coding scheme uses variable bit rate. In VBR scheme, all the video frames can have the same or targeted quality (PSNR), although the bitrate for each frame may vary (depending on the scene complexity). For more complex frame segments (blocks or slices), the VBR scheme allocates higher bitrate and for low complex frame segments it allocates less bitrate. The final average bitrate of encoded video is calculated by adding all the bitrates of individual frames and dividing it by the frames duration.

2.2.9 Rate Distortion Performance for Video Encoder

The Rate-Distortion (RD) performance is one of the methods that is used to evaluate the performance of different video (or image) encoders considering both the video quality (PSNR) and compression rate (bitrate). RD performance is usually depicted using graphs termed as RD curves as shown in Fig. 2.4 [24].

The figure shows RD curves simulated for test sequence Johnny (720p) using reference softwares of various video coding standards – H.262/MPEG-2 Main Profile, MPEG-4 Advanced Simple Profile, H.263, H.264/AVC High Profile and HEVC Main Profile. On x-axis, the bitrate in kbps is taken and on y-axis the PSNR (video quality) in dB is taken. The nearer the curve is towards the y-axis, the better is the RD performance for the corresponding standard. This is because, for a constant PSNR, the curve towards the y-axis takes lesser bitrate (better compression) compared to other curves. Further, for the same bitrate, the curve towards the y-axis achieve higher video quality. From the plots shown in Fig. 2.4, the highest RD performance is achieved by HEVC, followed by H.264/AVC, H.264, MPEG-4 and then the last, H.262/MPEG-2.

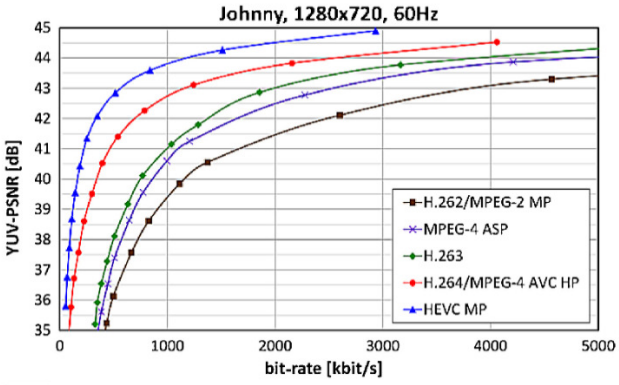
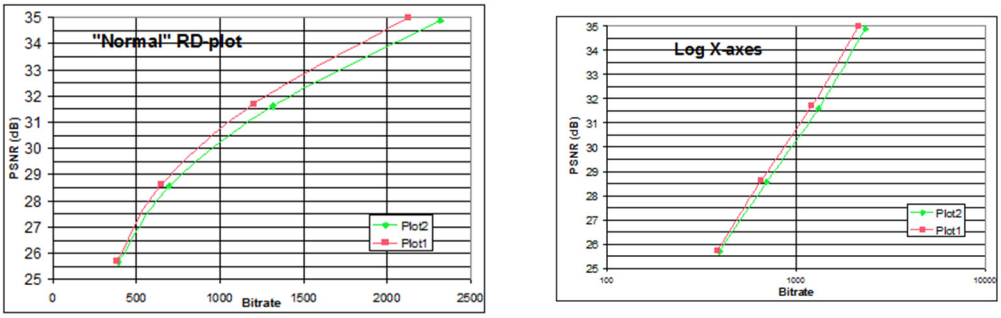


Fig. 2.4 Illustration of RD curves of test sequence Johnny (720p) using various video coding standards

2.2.10 Bjontegaard Delta Metrics for RD Performance Measurement

The RD performance difference between two curves can be measured using Bjontegaard Delta (BD) metrics [25, 26]. While taking one RD curve as a reference, the BD metrics (for the second curve) denotes the overall bitrate savings and the overall PSNR savings. There are two types BD measurements – BD-bitrate (or BD-rate) and



(a) (b)

Fig. 2.5 Illustration of RD curves used to calculate BD metrics. (a) Normal RD Plots (b) Logarithmically scaled plots

BD-PSNR. The BD-bitrate provides a number that denotes the overall bitrate savings, while the BD-PSNR denotes the overall PSNR difference between the two curves.

To calculate BD-bitrate and BD-PSNR, the RD performance of two encoder configurations for four different QP settings are taken and plotted with logarithmic scale of bitrate in x-axis and PSNR on y-axis. An example RD plots of two curves with their logarithmically scaled version of x-axis (bitrate values) is shown in Fig. 2.5. The curves are interpolated over the measured points using either (2.3) or (2.4), and then integrated over the x and y-axis respectively. The interpolation of PSNR as a function of bitrate is shown in (2.3) and (2.4) shows interpolation of bitrate as a function of PSNR. The difference over the integrated value over x-axis is taken as BD-bitrate value and the difference over y-axis is taken as BD-PSNR. In this way, the average PSNR difference in DB over the whole range of bitrates is calculated and similarly, the average bitrate difference in % over the whole range of PSNR is calculated.

$$PSNR = a + (b \times bitrate) + (c \times bitrate^2) + (d \times bitrate^3) \quad 2.3$$

$$bitrate = a + (b \times \mathrm{PSNR}) + (c \times \mathrm{PSNR}^2) + (d \times \mathrm{PSNR}^3) \quad 2.4$$

2.3 Types of Redundancies in Digital Video

Broadly classifying, there are two types of redundant information, one is the duplicate data and the other is irrelevant data. Duplication of data in videos mostly occurs due to correlation of objects within frames and between frames. There are three types of redundancies that occur due to duplicate data - spatial redundancy, temporal redundancy, and statistical coding redundancy. Irrelevant data is the information that the

Human Visual System (HVS) cannot perceive. This is usually termed as psycho-visual redundancy. Each of the redundancy type is explained briefly in the following sub-sections [16], [19].

(a) Spatial Redundancy

Neighboring pixels are highly correlated amongst themselves in a frame of video sequence. This redundant information can be exploited to achieve video compression. In block based video encoders like H.264/AVC, HEVC intra-prediction method is used to exploit this spatial redundancy which predicts the neighboring pixel blocks within the frame.

(b) Temporal Redundancy

Successive frames in a video are highly correlated since most of the video sequences consist of objects moving on a still background. This redundancy can be exploited using motion compensated coding technique.

(c) Psycho-Visual and Spectral Redundancy

This redundancy occurs due sensitivity variations of Human-Visual-System (HVS) for luminance and chrominance components in a video frame. Human visual system is less sensitive to details of pixel differences in an image. Hence the finer details of an image are quantized to achieve compression. The quantization based coding is a lossy coding. Further, the human eye is more sensitive to luminance component than chrominance component. Hence, video frames are typically encoded using chroma sub-sampling techniques (like 4:2:0) to remove the redundant color information.

(d) Statistical Coding Redundancy

The statistical redundancy occurs due to redundancy in neighboring bits of video information. After transforming the video frames (using transforms like Discrete Cosine Transform), the frame pixels information are arranged in according to their frequencies and hence the redundant data is easily removed. The statistical redundant data is exploited using entropy encoder.

2.4 DIGITAL VIDEO COMPRESSION TECHNIQUES

For exploiting the aforementioned redundancies, the digital video encoders use many algorithms and techniques. Altogether, these techniques can be classified into four types. Each of these types of coding techniques is explained in the following subsections [19].

2.4.1 Intra Frame Coding

Intra frame coding technique reduces the spatial redundant information (redundant information that occurs between pixels in the same frame). Typically for doing temporal coding in a video coding process, there should be at least one reference frame that should not be encoded by using blocks of neighboring frames. Hence, the first frame out of n frames is always intra-coded. The rest of the frames are coded using temporal prediction (exploiting temporal redundant information). The number 'n' is the GOP (Group of Pictures) size and denotes the number of frames in a sequence.

Although there is only one frame (intra-frame) in a GOP, the number of bits that are produced by intra-frames is significantly larger. Hence to compress this information, spatial prediction techniques were used. This is similar to still image coding, but used as part of video coding. Some standards like H.264/AVC and HEVC have a direct intra coding extensions (profiles) which use only intra frame coding and without any temporal (or inter-frame) coding.

2.4.2 Inter Frame Coding

To exploit the temporal redundant information, motion compensated coding (inter frame coding) technique is used. In motion compensated coding, each block of a video frame is predicted with neighboring blocks in past frames using block matching algorithms. This process is called Motion Estimation (ME). The output of the ME process is the Motion Vector (MV) of the predicted block, which is sent to the decoder.

After the prediction process, the motion compensated (or predicted) frames are generated using these predicted blocks. The MC frames are subtracted from the current frames to get the residual frames. Typically, the subsequent stages after motion compensation (MC) coding are the transformation stage, quantization stage and entropy coding stage. The final output of entropy coding are the bits that represent the motion compensated residual frames. The output of the motion compensated coding is not just

MC residual frames, but also Motion Vectors (MVs) that represent predicted blocks obtained from Motion Estimation (ME). These MVs are also entropy encoded. Hence the necessary condition that should be satisfied in MC coding is that the output entropy coded bits of MVs and MC residuals (after transformation and quantization) is less than the entropy coded bits of difference images without MC coding (also after transformation and quantization). This is shown in (2.5), where J represents entropy coded bits and difference image represents residual frame without MC.

$$J(MC_{residual}) + J(MV) < J(difference\_frame) \quad 2.5$$

2.4.3 Transform Coding and Quantization

The transform coding is a method to exploit irrelevant information in video frames. This is typically a lossy compression technique. In transform coding, the image or video frame (motion compensated residual frame) is transformed into frequency domain which represents distribution of frequencies of pixel values within block. Technically, the transformed blocks indicate low to high frequencies in original residual frame blocks. After the transformation, the transformed blocks are quantized to remove irrelevant information that cannot be visualized in detail by human eye. The quantization process leads to a lossy data.

Broadly classifying, there are two types of transforms - block-based and image based. Block based transforms operate on block of pixels and are suitable for block based video coding standards. They have low memory requirements and have low complexity compared to image based transforms. Image based transforms apply on an entire image or a large portion of image. Hence, they require high memory and are computationally expensive. Block based transforms suffer from blocking artifacts and hence require de-blocking filtering. Some of the examples for block based transforms are KL-transforms, Singular Value Decomposition, Discrete Cosine Transform (DCT), integer transforms, Hadamard transforms, etc. The most widely used block transform is Discrete Cosine Transform. The most commonly used image transform is the Discrete Wavelet Transform (DWT).

2.4.4 Entropy Coding

The entropy coders exploit the statistical redundancy in the quantized transform coefficients of residual information and in other information like motion vector

differences, frame headers, block headers etc. The entropy encoder encodes the input information into minimum number of bits by allocating more bits to low frequent data and less number of bits to high frequent data. The most widely used entropy coding schemes in block based video encoders are Context Adaptive Variable Length Coder (CAVLC) [27] which is based on variable length coding and Context Adaptive Binary Arithmetic Coder (CABAC) [28] which is based on arithmetic coding.

2.5 DIGITAL VIDEO CODING STANDARDS

A video coding standard is a language that contains syntaxes and other elements so that the decoder can understand and decode it, besides achieving a goal of compressing the video. A video coding standard, gives some flexibility in the implementation of the encoder but constraints it to follow a common format that every other decoder complying with the specified standard can understand the encoded bitstream. There are mainly two major working groups that standardize video codecs. One is the VCEG (Video Coding Experts Group) led by ITU-T (International Telecommunications Union - Telecommunication Standardization Sector) and the other is the MPEG (Moving Pictures Experts Group) led by ISO/IEC - JTC 1 (International Organization for Standardization / International Electrotechnical Commission - Joint Technical Committee 1) [19].

VCEG is more focused on conventional (esp. low-delay) video coding goals (e.g. good compression and packet-loss/error resilience). The VCEG standardized (and maintain) the H.26x line of video coding standards. MPEG is larger and takes on more ambitious goals (e.g. “object oriented video”, “synthetic-natural hybrid coding”, and digital cinema). The MPEG standardized (and maintain) MPEG-x line of video coding standards. Sometimes these major organizations team up and create a standard (e.g. ISO, IEC and ITU teamed up for both MPEG-2, JPEG, H.264/AVC and HEVC) [29].

2.5.1 History of video coding standards

The first video coding standard was H.120 published by CCITT (Comité Consultatif International Téléphonique et Télégraphique) which was renamed to ITU-T in 1984. There were very few implementations with very low quality, but it then gave a good initiative to its successors like H.261. The first practical coding standard from

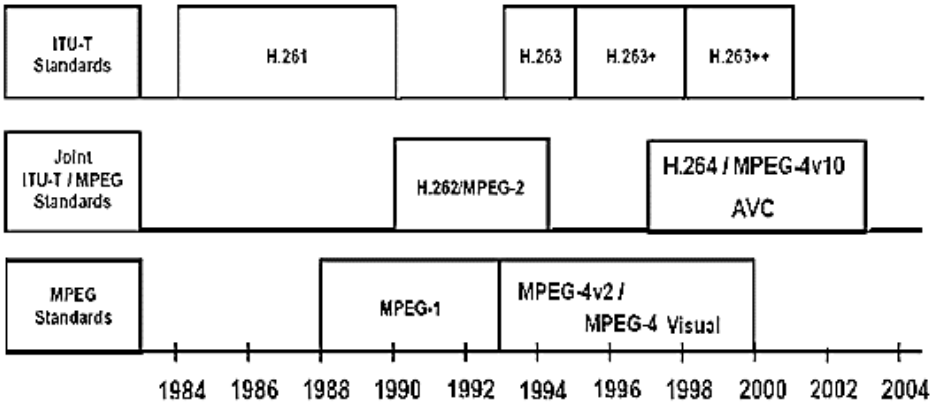


Fig. 2.6 Progression of ITU-T and MPEG works

VCEG group was H.261 standardized in 1988 and from MPEG it was MPEG-1 standardized in 1993. After these two standards, there are many successor individual standards and collaborative standards within (and in between) these groups. The timeline of these standards are shown in Fig. 2.6.

2.5.2 ITU/VCEG standards

Video Coding Expert Group (VCEG) has been handling the responsibility of standardizing and maintaining the video compression formats, and corresponding standards.

H.120: It is the first standard in digital video compression techniques [30] standardized in 1984 but like most firsts in many fields of study, is not matured enough and had poor quality and fewer implementations. It featured conditional replenishment, variable-length coding, scalar quantization, and differential PCM. Although its encoder offered good spatial resolution, it had very poor temporal quality, and thus could not be of much practical use.

H.261: Although H.120 preceded H.261 in late 1990, the latter is clearly the first video coding standard to be of practical use, in view of the conforming implementations, and the subsequent coding standards that emerged with H.261 as their base design [31]. H.261 is the first video coding standard that pioneered the concept of a basic processing unit, titled macroblock. H.261 only specifies the guidelines for decoding a video, the encoding process can employ any algorithm as long as the output can be decoded

according this standard. H.261 is published by ITU-T VCEG. The popular MPEG-1 video coding standard was derived from H.261 and JPEG standards.

H.262: An enhancement of MPEG-1 (based on H.261 & JPEG) video coding standard, and jointly developed by VCEG and MPEG groups (process completed in late 1994), H.262/MPEG-2 offers support for interlaced video and is optimized for high bit rates above 3 Mbits/sec [32], [33]. H.262/MPEG-2 allows the tools to implement only a subset of the standard by defining various profiles and levels within the specification to accommodate diverse needs of the applications.

H.263: H.263 was an evolutionary development based on the learnings from the previous video coding standards H261, MPEG-1, and MPEG-2 (standardized in late 1995). It was originally intended for H324 communications (PSTN, video conferencing and video telephony), but found applicable even for H320 (ISDN based video conferencing), H323 (RTP/IP based video conferencing), SIP (IP based video conferencing) and RTSP (streaming media) solutions [34].

H.263V2 : H.263V2/H.263+ is the enhanced version of H.263 video coding standard that provides additional features as appendices to the original H.263 standard, thus retaining every aspect of the parent yet improving encoding efficiency and reducing data loss in transmission channels.

H.264: H.264/MPEG-4 is often called "Advanced Video Coding" (AVC) standard, and is by far the most widely used one in the industry. It was collectively developed by a committee of experts from VCEG and MPEG groups, titled Joint Video Team (JVT). The first version of its draft is released in May 2003 [5]. H.264 is lossy, motion compensated, and block-oriented video compression standard that offers good quality and high compression ratio at low bit rates. H.264 offers several advancements in video compression techniques such as Scalable Video Coding (SVC), Multiview Video Coding (MVC), Entropy Coding Design including binary arithmetic coding (CABAC) and variable length coding (CAVLC), loss resilience features such as Network Abstraction Layer (NAL), Flexible Macroblock Ordering (FMO), Data Partitioning (DP), etc [6], [35].

H.265: H.265 or High Efficiency Video Coding (HEVC) is a successor to the H.264/MPEG-4 AVC standard and was developed by a Joint Collaborative Team on Video Coding (JCT-VC), a collaboration effort by MPEG, and VCEG groups. Its first version of draft was completed and released in January 2013 [4]. H.265 takes advantage of the advancements in computational power of the hardware devices in recent times to achieve higher compression at lower bit rates without compromising much on quality, offering at least 50% improvement over H.264/MPEG-4 [2].

2.5.3 ISO/MPEG Standards

The Moving Picture Experts Group (MPEG) is a standards body setup by ISO and IEC to standardize the compression and transmission techniques for Digital Audio and Video. MPEG team actively collaborates with other such expert groups to formulate worldwide standards in digital video compression. The most notable outcomes of such collaboration are with the VCEG team that culminated in the specification of H.264/MPEG-4 AVC, and H.265/HEVC video coding standards. MPEG standards are segregated into several parts, each part describing a certain aspect of the whole specification. MPEG team has formulated the following digital video coding standards:

MPEG-1 (1993): It specified the compression mechanism for moving pictures, and accompanying audio at bit rates lesser than 1.5 Mbits/sec [36] [37]. This specification also includes the MPEG-1 Audio Layer III (MP3) audio compression format. MPEG-1 decimates images to meet the low bit rate requirement, and thus results in comparatively poor quality. MPEG-1 was primarily used to store videos in CD until MPEG-2 has arrived on stage.

MPEG-3: MPEG-3 was intended for high definition television with features like scalable and multi-resolution compression but was found redundant, and was merged with MPEG-2 [15]. There is no MPEG-3 standard now.

MPEG-4 (1998): MPEG-4 achieves higher compression ratios at lower bit rates without compromising on the quality, and advances in depicting computer graphics with three dimensional shapes and surface texture. MPEG-4 also supports intellectual property management and protection (IPMP). There were several parts included in MPEG-4, of which two are highly used – MPEG-4 part 2 and MPEG-4 part 10. The MPEG-4 part 10

is just another name for H.264/AVC Joint Video Team (JVT). The MPEG-4 part 2, also known as MPEG-4 visual is developed and maintained by MPEG [38].

MPEG-7 (2002): Multimedia Content Description Interface, a mechanism to allow additional information such as the composer, lyrics, author, publisher, and other such details along with the compressed content to facilitate easier lookup of such metadata, once the content is in hand. MPEG-7 is not another standard like MPEG-1 or MPEG-4 but a mechanism to standardize sharing of such metadata along with the content compressed with any of the above standards [39].

2.5.4 Other standards

Apart from MPEG and VCEG based standards, there are many other video codec standards of which Microsoft's VC-1 and Google VP9 based codecs are most widely used in both web and physical devices.

VC-1: VC-1 is a proprietary video standard which was initially released by Microsoft in 2006 as SMPTE video codec (Society of Motion Picture and Television Engineers) [40]. It is described as alternative to H.264/AVC. VC-1 supports both interlaced and progressive encoding. VC-1 is an attractive codec for video broadcasting industry because it supports direct interlaced video coding without converting the video first to progressive format. VC-1 is supported in windows media, Microsoft Silverlight framework, Blu-ray discs, Slingbox [41].

VP9: VP9 is an open standard developed by Google with the aim to reduce bitrate by 50% compared to its predecessor VP-8 [42]. Web browsers like Firefox, Opera, and Chrome, uses VP9 with HTML5 video tag. VP9 is being used in smart TVs with 4k resolution and in some YouTube web streaming videos with 4k resolution.

AVS: Audio Video Standard (AVS) is a compression standard for digital audio and digital video, which was meant to compete with AAC audio and H.264/MPEG-4 AVC video to potentially replace MP3 audio and MPEG-2 video. The audio and video files have an .avs extension as a container format. From the year 2013, its working group set a new target to compete with H.265/HEVC. Some of the open-source implementations of an AVS video decoder were found in the OpenAVS project and in the libavcodec library [29].

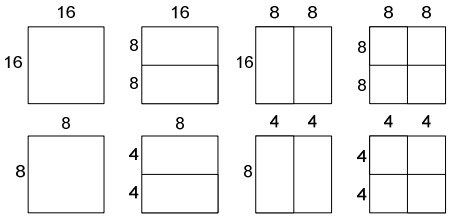


Fig. 2.7 Illustration of Macroblock and its sub-block sizes in H.264/AVC

2.6 H.264/AVC CODING STRUCTURE

H.264/AVC is the most widely used standard prior to the latest video coding standard HEVC. H.264/AVC is developed by JVT (Joint Video Team) which is a collaboration team formed by ISO/IEC MPEG and ITU-T VCEG [6]. In H.264/AVC each frame is divided into 16x16 size coding blocks termed as macroblocks. Each macroblock can be again subdivided into blocks of size 16x8, 8x16, 8x8, 8x4, 4x8 and 4x4 as shown in Fig. 2.7. In H.264/AVC, new features like multiple reference frames and variable block size motion estimation were introduced. With multiple reference frames, more than one reference frame is used for estimating the motion vector. In prior standards like MPEG-2 only one reference frame is used for ME. H.264/AVC allows up to 16 reference frames (or 32 reference fields for interlaced scanning). With variable block size motion estimation, the motion estimation is performed on all block sizes of coding block from 4x4 to 16x16 as shown in Fig. 2.7.

2.7 HEVC CODING STRUCTURE

In HEVC, the block size for coding is increased to 64x64, in-order to increase the coding efficiency at the cost of increase in coding complexity. The structure of HEVC block coding hierarchy is generalized into quadtree-based coding tree units (CTUs) or coding units (CUs). The maximum size of each CTU is 64x64 and each CTU is further sub-divided recursively into square blocks down to 8x8 sizes [2]. Each CTU is a generalized structure of block coding hierarchy, where it is assigned quadtree-based prediction units (PUs) of different types, either intra or inter or skip. Each PU is further assigned into quadtree-based Transform Units (TUs) with a specific transform size. The representation of quadtree-based CTU with their PU types is shown in Fig. 2.8.

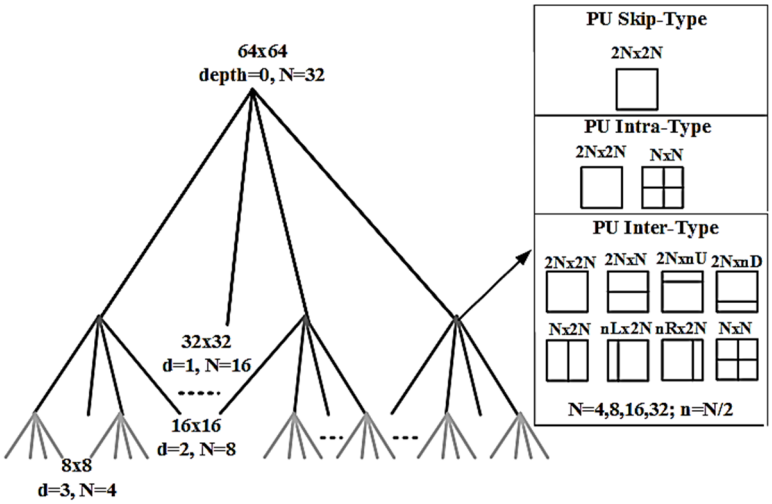


Fig. 2.8 Quadtree coding structure in HEVC

2.8 REFERENCE SOFTWARES FOR VIDEO CODING STANDARDS

To verify and test the performance of various encoding and decoding algorithms for the specified standard, reference software is usually implemented for some of the standards including H2.64/AVC and HEVC. The reference software is normative and any decoder implementation should be able to decode the bitstream encoded using reference software encoder. The reference software also includes a decoder software which is used to decode the bitstream encoded using the complying standard encoder. One of the main goals of the reference software is to provide a platform to conduct experiments in order to determine which coding tools provide the desired coding performance.

2.8.1 H.264/AVC Reference Software JM

A reference software was implemented for H.264/AVC by JVT of ISO/IEC MPEG and ITU-T VCEG that complies with the standard. The software is technically termed as Joint Model (JM), which was initially released in August 2004 [43] and the latest version is JM 18.0 released in March 2011. JM consists of both encoder and decoder softwares. JM software is supported in MS Visual studio .NET platform (for Windows operating system) and gcc (GNU Compiler Collection) platform (for UNIX and Windows operating system) [44].

2.8.2 HEVC Reference Software HM

Like JM for H.264/AVC, reference software for HEVC was implemented by JCT-VC regrouping experts from ITU-T SG 16 and ISO/IEC SC29 WG11, known as HM (HEVC Model) [7]. The initial version of HM (HM 1.0) was released in 2010 and the

TABLE 2.3 VARIOUS CONFIGURATIONS FOR HEVC HM ENCODER

Configuration	Internal Bit Depth	
	Main	High Efficiency (HE)
Intra (I)	8	10
Low Delay-B (LB)	8	10
Low Delay-P (LP)	8	10
Random Access (RA)	8	10

current version of HM is HM 16.2 [8]. Like JM, HM also consist of both encoder software and decoder software. The supported environments for HM are MS Visual Studio 8, MS Visual Studio 9, Xcode and Linux GCC compiler.

2.8.3 Various Configurations in HM

There are eight default configurations provided with HM reference software out of which four configurations belong to 10-bit internal bit depth (bit depth for luma and chroma both set to 10 bit) and the remaining four belong to 8-bit internal bit depth. The internal bit depth here specifies the number of bits used to represent a pixel sample with 4:2:0 chroma sub-sampling. The four modes provided are intra-mode, low-delay mode, low-delay P mode and random access mode. Each mode had two types of configurations – one with 8-bit internal bit depth and the other with 10-bit internal bit depth making a total of eight configurations. The 8-bit mode is technically termed as main mode and 10-bit mode is termed as High Efficiency (HE or HE10 or Main10) mode. In each mode, the first frame in a GOP sequence is encoded as I-frame or IDR (Instantaneous Decoder Refresh) picture. In IDR picture all the slices are encoded as I-slices. TABLE 2.3 lists out the summary of various configurations available for HM encoder [45, 46].

Intra-only Configuration:

In intra-mode, all the frames are encoded as IDR pictures (encoded using intra-prediction) only. There are no temporal reference pictures and there is no motion compensated coding. The quantization parameter (QP) does not change between and within each picture. This mode has lowest coding efficiency and coding complexity compared to other modes. The graphical representation of this configuration is shown in

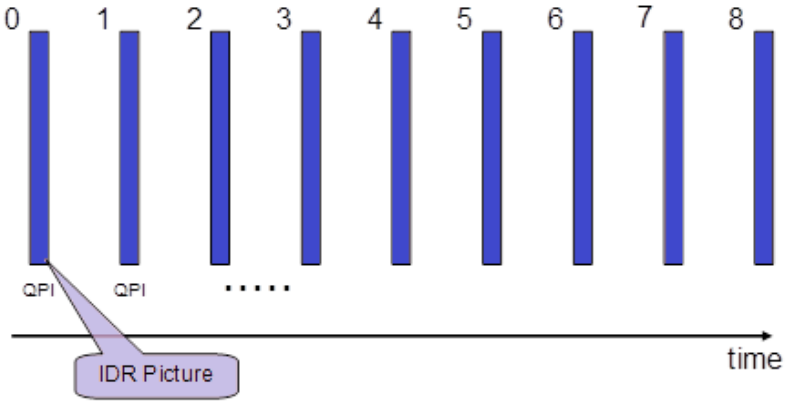


Fig. 2.9 Illustration of encoding order of pictures in HEVC Intra-only configuration

Fig. 2.9. The number associated for each picture represents the encoding order. The QPI represents QP for IDR picture which is same for all pictures.

Low-delay Configurations:

In low-delay configuration, only the first frame is encoded as IDR picture. There are two low-delay configurations that are supported by HEVC. One is low-delay configuration (or low-delay B configuration) and the other is low-delay-P configuration, which is treated as optional configuration. The difference between low-delay configuration and low-delay P configuration is, in low-delay P mode all the frames in a GOP are taken as P-pictures only while in low-delay mode all the frames in a GOP are taken as Generalized P and B pictures (GPB) only. In both these configurations the first frame is encoded as IDR picture. A graphical representation for low-delay B configuration is shown in Fig. 2.10. The number shown for each picture represents encoding order. The QP for each inter coded picture is derived by adding an offset to

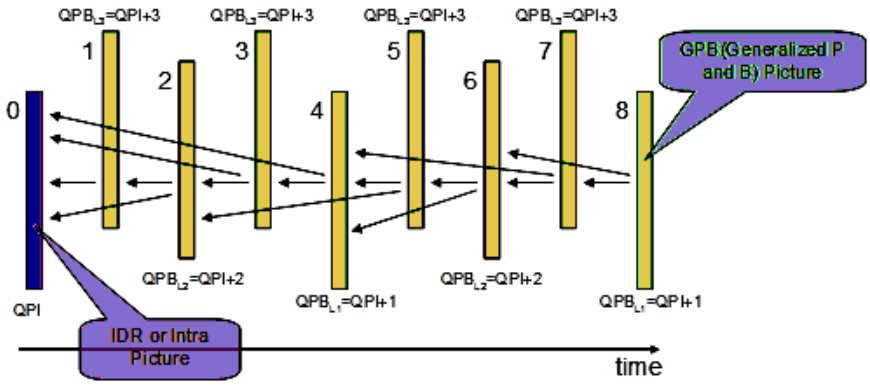


Fig. 2.10 Illustration of encoding order of pictures in HEVC low-delay B configuration

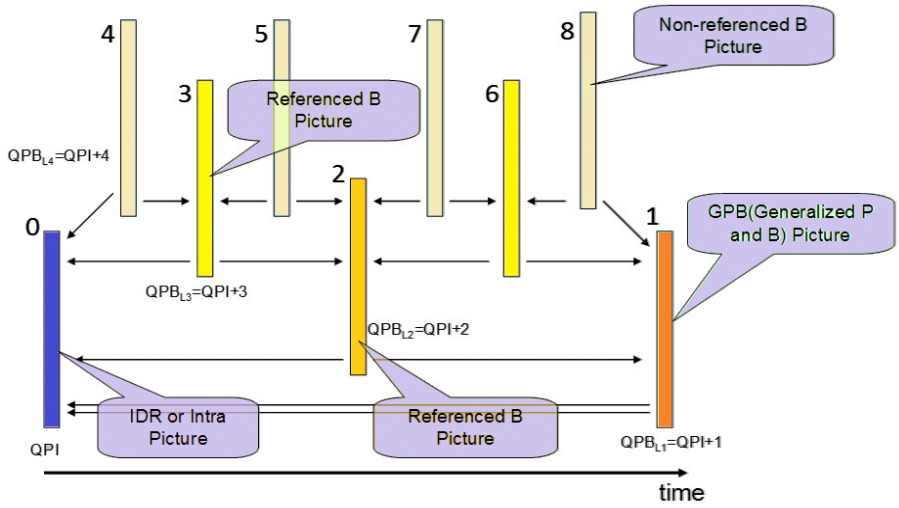


Fig. 2.11 Illustration of picture encoding order in HEVC Random Access configuration

the QP of intra-coded picture. The QP offset depends on temporal layer of inter coded picture.

Random Access Configuration:

In random access configuration, a hierarchical B-structure is used for encoding as shown in Fig. 2.11. As shown in the figure the frames are divided into different layers – L1 to L4. The first picture is encoded as IDR picture. The second picture in the following first pictures is encoded as GPB picture, that can refer (for inter prediction) to I-frames or any other GPB pictures. The pictures in the rest of the layers are B-pictures. The pictures in the last layer are non-referenced B-pictures (that are not used as reference frames for any other frames). Depending on the temporal layer, the offset of QP is derived and added to the QP of IDR picture, to get the final QP for the corresponding inter picture.

2.8.4 Reference Test Sequences for HM

For testing the HM encoder software, standard test sequences were recommended by JCT-VC. These test-sequences are available in [47]. The test sequences are grouped into classes from A through F, depending on their frame size or format. TABLE 2.4 shows the names of these test sequences with their frame size, frame count, frame rate and supported configurations in the subsequent columns of the table [45]. All the

TABLE 2.4: SUMMARY OF TEST SEQUENCES RECOMMENDED FOR HEVC HM ENCODER

Class	Sequence name	Frame size	Frame count	Frame rate	Bit depth	Intra mode	Random Access mode	Low-delay mode
A	Traffic	4k (2560x 1600)	150	30fps	8	Main/HE10	Main/HE10	NA
	PeopleOnStreet		150	30fps	8	Main/HE10	Main/HE10	NA
	Nebuta		300	60fps	10	Main/HE10	Main/HE10	NA
	SteamLocomotive		300	60fps	10	Main/HE10	Main/HE10	NA
B	Kimono	1080p (1920x 1080)	240	24fps	8	Main/HE10	Main/HE10	Main/HE10
	ParkScene		240	24fps	8	Main/HE10	Main/HE10	Main/HE10
	Cactus		500	50fps	8	Main/HE10	Main/HE10	Main/HE10
	BQTerrace		600	60fps	8	Main/HE10	Main/HE10	Main/HE10
	BasketballDrive		500	50fps	8	Main/HE10	Main/HE10	Main/HE10
C	RaceHorses	WVGA (832x480)	300	30fps	8	Main/HE10	Main/HE10	Main/HE10
	BQMall		600	60fps	8	Main/HE10	Main/HE10	Main/HE10
	PartyScene		500	50fps	8	Main/HE10	Main/HE10	Main/HE10
	BasketballDrill		500	50fps	8	Main/HE10	Main/HE10	Main/HE10
D	RaceHorses	WQVGA (416x240)	300	30fps	8	Main/HE10	Main/HE10	Main/HE10
	BQSquare		600	60fps	8	Main/HE10	Main/HE10	Main/HE10
	BlowingBubbles		500	50fps	8	Main/HE10	Main/HE10	Main/HE10
	BasketballPass		500	50fps	8	Main/HE10	Main/HE10	Main/HE10
E	FourPeople	720p (1280x720)	600	60fps	8	Main/HE10	NA	Main/HE10
	Johnny		600	60fps	8	Main/HE10	NA	Main/HE10
	KristenAndSara		600	60fps	8	Main/HE10	NA	Main/HE10
F	BaskeballDrillText	832x480	500	50fps	8	Main/HE10	Main/HE10	Main/HE10
	ChinaSpeed	1024x768	500	30fps	8	Main/HE10	Main/HE10	Main/HE10
	SlideEditing	1280x720	300	30fps	8	Main/HE10	Main/HE10	Main/HE10
	SlideShow	1280x720	500	20fps	8	Main/HE10	Main/HE10	Main/HE10

sequences are not recommended for all modes – class-A sequences do no support low-delay modes and class-E sequences are not recommended for random access modes.

2.9 BLOCK DIAGRAM OF HEVC ENCODER

A brief outlook of HEVC encoder with individual blocks is shown in Fig. 2.12 [48], [49]. Each frame is first split into Coding Tree Units (CTUs) conforming HEVC

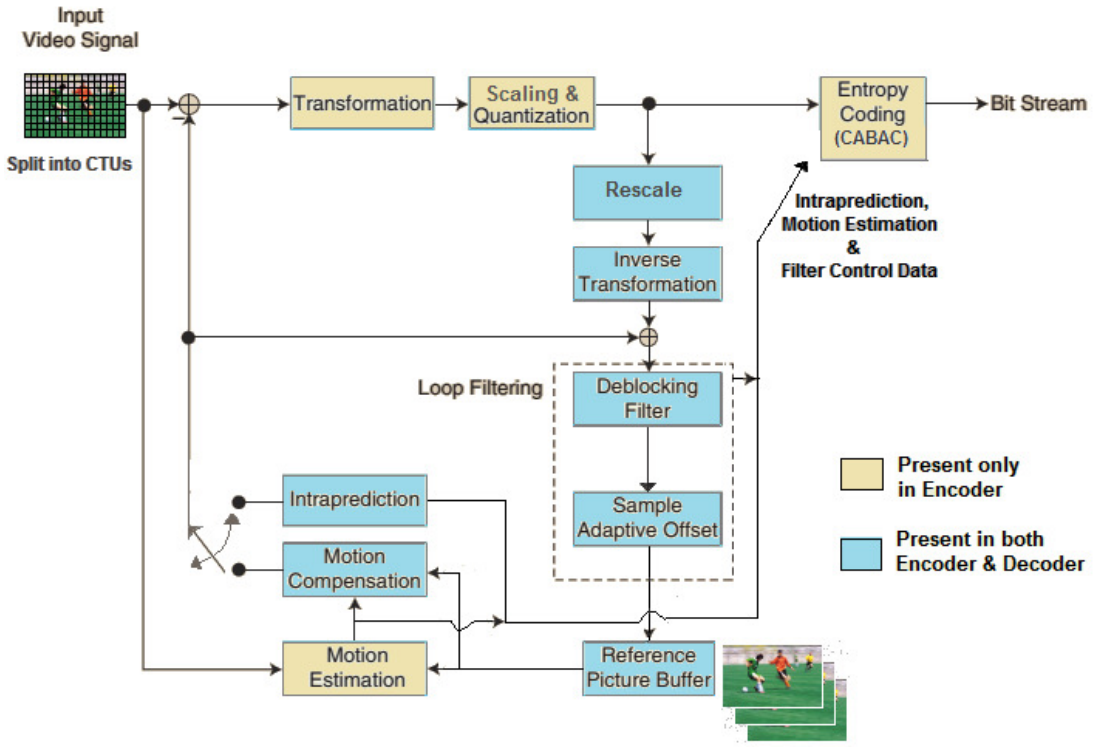


Fig. 2.12 Block Diagram of HEVC Encoder

standard. Each CTU is converted into motion vectors (or intra prediction information) and predicted blocks. Then the predicted information is subtracted from original frame data to get residual information. The residual information is transformed and quantized. The quantized information along with prediction information is entropy encoded using arithmetic encoder CABAC (Context Adaptive Binary Arithmetic Coder).

The encoder also includes a decoder processing loop (shown in blue color) to ensure that it will generate identical prediction information for the subsequent processed data. Hence the quantized transform coefficients are inverse-scaled and then inverse-transformed to generate a decoded approximation of residual data. This residual will then be added to the predicted information to get a decoded approximation of original frame data. The block-wise processing and quantization process usually introduces some artifacts in the reconstructed frame and hence it is fed to loop filters – deblocking filter, Sample Adaptive Offset (SAO) filter and Adaptive Loop Filters (ALF) to smooth these artifacts. The output of these filters are passed through a memory buffer called Decoded Picture Buffer (DPB) which stores the decoded frames which are later used for prediction of subsequent pictures.

***Intra-Prediction Unit*:** The first picture in each GOP sequence is encoded using the intra prediction unit, apart from the first CTU in each slice of other frames. If the encoding mode is of intra type, all blocks are encoded in intra prediction mode. HEVC supports 35 luma directional modes (including DC intra prediction and planar prediction modes) compared to H.264/AVC with 8 directional modes and hence the complexity of intra-prediction is increased while achieving a huge intra-coding efficiency. The intra-prediction modes for HEVC and H.264/AVC are shown in Fig. 2.13. In planar prediction mode, the predicted block is generated by averaging the horizontal and vertical interpolated blocks [50]. The intra-prediction can be performed at different block sizes ranging from 4x4 to 64x64.

Apart from luma intra-prediction, there are also chroma intra-prediction modes. In HEVC, the chroma intra-prediction modes are increased to six compared to four intra-prediction modes in H.264/AVC. The various chroma intra-prediction modes in HEVC are direct mode (DM), linear mode (LM), vertical (mode 0), horizontal (mode 1), DC (mode 2), and planar (mode 3). The modes DM and LM are used to exploit correlation between luma and chroma components [51]. The DM and LM modes are frequently used in intra-coding of chroma component due to existing correlation between luma and chroma components of an image [52].

***Motion Estimation and Motion Compensation*:** The motion estimation unit together with motion compensation unit performs inter-picture prediction by converting the frames into motion vectors and motion predicted blocks. The motion estimation unit estimates the motion vectors of each block in a frame (except in intra frames) while the motion compensation unit uses these motion vectors and generates motion compensated (predicted) frames. These motion compensated frames are then subtracted from the original video frames to get the residual frames and processed further. Typically, the motion estimation block uses block matching algorithms to find the motion vector of each block in a frame. The motion compensation unit performs interpolation (using functions like weighted-prediction) on reference picture to form motion compensated frame for every current frame.

The ME is performed on various block sizes called Variable Block Size ME (VBSME). In H.264/AVC, there are 7 modes (4x4 to 16x16) for inter-prediction with an output of 41 MVs (in maximum) for each macroblock. Since the block size in HEVC is

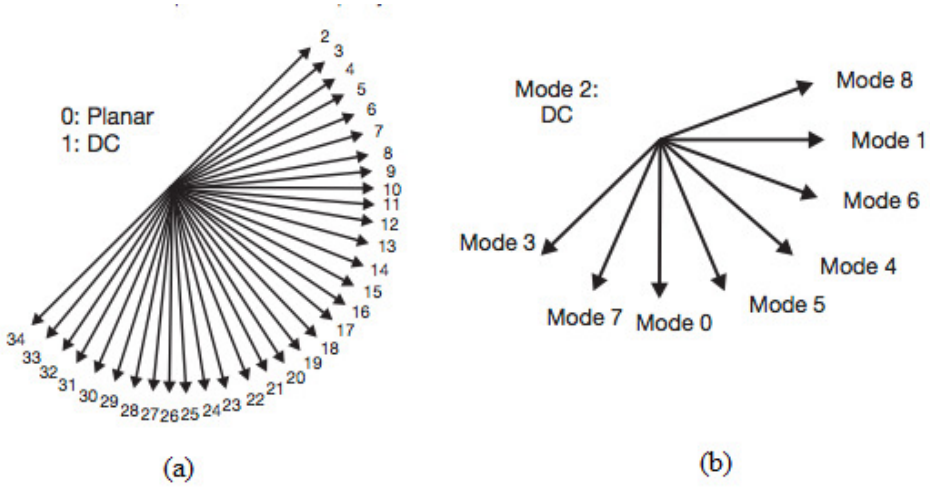


Fig. 2.13 Comparison of various luma intra-prediction modes in (a) HEVC (b) H.264/AVC

increased until 64x64, there are 13 modes for inter-prediction. The details of various modes in HEVC are explained in section 2.10.2. Further, the ME is performed in sub-pixel accuracy (details explained section 2.10.5) using interpolation filters. Like in H.264/AVC, the HEVC performs ME and MC up to quarter pixel accuracy, but with improved interpolation filters [53], [54].

***Transform*:** The transform unit transforms the residual information using integer transforms. In HEVC, the transform size is increased until 32x32 (compared to 8x8 in H.264/AVC). Just like DCT (Discrete Cosine Transform), the integer basis functions are defined and can be applied on various transform block sizes ranging from 4x4 to 32x32 pixels (4x4, 8x8, 16x16 and 32x32) [55]. As mentioned above, the TU is the basic unit for transforms and quantization in HEVC with its tree structure having root at CU level. The size and shape of TU depends on PU size. Further, HEVC supports rectangular transforms (for non-square PUs) with row and column transforms having different sizes. For 4x4 luma intra-prediction modes, integer transform derived from Discrete Sine Transform (DST) is alternatively used in HEVC. The DST is used (for some 4x4 intra-mode sample) because they fit better for residual samples near boundaries (which tend be of large in amplitude for pixels away from boundaries) [56]. The DST contribute up to 1% reduction in bitrate for intra prediction data with almost same computationally complex compared to DCT - based integer transforms [2].

***Scaling and Quantization*:** The transformed coefficients of the residual data are scaled and passed through quantizer which make the coefficients to select from a limited set of

discrete finite values. This is one of the lossy compression technique, where the number of transform coefficients are reduced and any insignificant coefficients are reduced to zero. Hence the quantizer is considered as one of the primary sources of compression in a video encoder [57]. HEVC supports quantization scaling matrices for different transform block sizes.

To control a tradeoff between compression ratio and video quality, a parameter called Quantization Parameter (QP) is used and is set before the HEVC encoder starts encoding. Larger QP value increases step size and increases the quantization step size and increases the compression ratio but decrease the output video quality. On the other hand, smaller QP can be set and increase the output video quality but it increases the encoded video stream bitrate (or reduces the compression ratio). Hence the QP is set depending on the bandwidth constraints and video quality requirements. Like in H.264/AVC the QP values can be set ranging from 0 to 51 and the mapping of QP values to step size is logarithmic. Hence for every increment in QP by 6 almost doubles the quantization step size.

***Inverse Transform and Rescaling*:** As explained above, the quantized coefficients are rescaled (or inverse quantized) and inverse transformed to get a decoded approximation of residual data. Each quantized coefficient is multiplied by an integer value to restore it to its original scale. The inverse transform apply inverse DCT operation which is a weighted coefficient matrix applied to a rescaled information. The reconstruct residual data will be similar but not identical to the original residual data, due to the loss of information in forward quantization process. A larger quantization step value (due to larger QP) will produce a larger difference between original and reconstructed data.

***Loop Filters*:** As explained above, the loop filters are applied after the picture is reconstructed and before they are used for motion compensated prediction. Apart from deblocking filter (which is also used in H.264/AVC), HEVC includes one in-loop processing filters - SAO filter. These in-loop filters are used to compensate the distortion introduced by the encoding steps – prediction, transformation and mostly by quantization. The more these in-loop filters are used the better is the quality of the reconstructed frame which are used as reference pictures for motion compensated prediction unit.

The deblocking filters in HEVC is similar to that of in H.264/AVC. The blocking artifacts in HEVC is due to several kinds of block boundaries such s CUs, PUs and TUs. For each block boundary and based on the artifact introduced, the encoder applies a decision to turn the filter on or off and to apply a weak filter or strong filter [48]. The SAO filter classifies the reconstructed pixels into either intensity or edge properties. It then adds offset value - either Band Offset (BO) or Edge Offset (EO) to these classified pixels to reduce the distortion [58], [59].

Entropy Coding Unit: The entropy encoder encodes the data by exploiting any statistical redundant data if exist. The entropy encoder is applied to quantized transformed coefficient data, MV data, and loop-filter coefficients data and to various high level syntax elements of HEVC. In H.264/AVC, only CAVLC is used in base profile and CABAC is optionally used in main and high profiles. In HEVC CABAC is used as it is more efficient than CAVLC due to its arithmetic coding engine and more sophisticated context modeling [60]. The CABAC increases the coding efficiency but at the cost of increase in coding complexity. Hence, to increase the throughput in HEVC an alternative mode called High Throughput Binarization (HTB) mode is used which utilizes the best features of both CAVLC and CABAC [61]. The first mode which uses only CABAC is termed as High Efficiency Binarization (HEB) mode. In HTB, the quantized transformed residual coefficients are encoded using CAVLC while the rest of the data like syntax elements, MV data, filter coefficients etc. are encoded using CABAC.

2.10 MOTION ESTIMATION FEATURES

2.10.1 Motion Estimation Objective

As explained in Chapter 1, the objective of the ME is to find the best matched block in past (or buffered future frame) frame's search window for each block of current frame. First, the current block is defined after the frame is divided into blocks of size specified by the standard (maximum of 64x64 pixels in HEVC). Then the reference frame is defined using neighboring frames. The reference frame may also be a future frame, where the current frame is virtually a current frame and in reality a buffered past frame. The next stage is to identify the region of interest where search has to be performed. This region is called Search Window (SW). If the ME performs search on all the blocks

in the reference frame then the search complexity will be very high. Hence the search window is defined around the best predicted search point.

After defining the SW, the search has to be done, based on the ME algorithm. This is shown in Fig. 1.4. Within the search window, if the search operation is done on all the search points, then it is called Full Search (FS) algorithm. The full search gives the best compression efficiency with best output video quality at the cost of highest coding complexity. To reduce the search complexity, most of the blocks are skipped which are less likely to be the final MV. These algorithms are called fast search algorithms.

2.10.2 Variable Block Size Motion Estimation

To achieve higher compression efficiency, the motion estimation is performed in many dimensions, one of them being the variation in block size. This is done by splitting current block into smaller sized sub-blocks and performing ME for each sub-block. This feature is called Variable Block Size ME (VBSME). Hence the output of ME operation for one current block will be a MV for each of the sub-blocks. In the motion compensation stage, the motion compensated frame is generated by considering all the modes (block sizes) of the current block, and choosing the best mode.

As explained in Section 2.6, Fig. 2.7 shows the variable block sizes in H.264. As seen from the figure, the maximum block size in H.264 is 16x16. Each block can be partitioned into seven modes which are 16x16, 16x8, 8x16, 8x8, 8x4, 4x8 and 4x4. The number of blocks in modes 16x8, 8x16, 8x8 is 2, 2, and 4 respectively. Each 8x8 block is again subdivided in two 8x4, two 4x8 and four 4x4 sized blocks. For each 16x16 block (also called macroblock in H.264), there are 41 sub-blocks (including 16x16 mode). The search operation of ME has to be performed for each of all these 41 blocks giving an output of 41 MVs. Thus the complexity is increased dramatically when VBSME feature is added. Due to VBSME, the compression efficiency will be increased as the encoder can select the best mode that gives the lowest bitrate. Hence at the cost of increase in computational complexity, the compression efficiency is increased. Furthermore, this complexity can also be reduced using fast mode-decision algorithms [62]- [63]. The fast mode-decision algorithm skips some modes for ME operation that are unlikely to be the best mode.

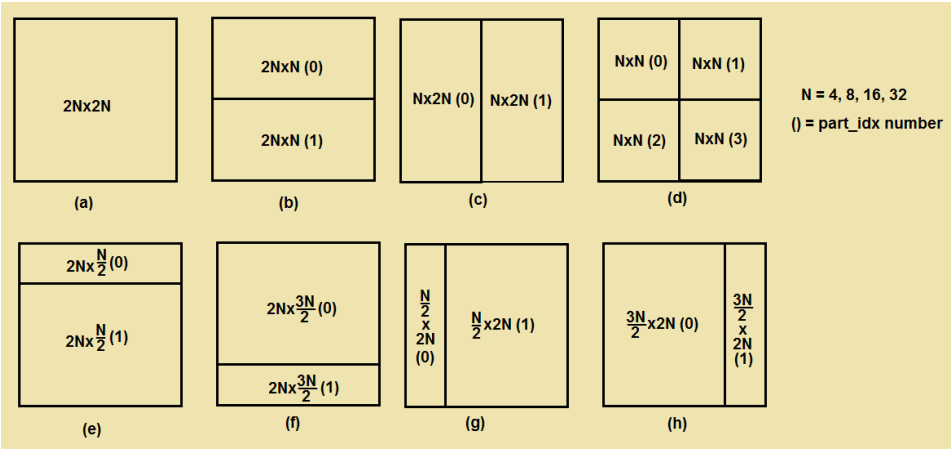


Fig. 2.14 Sub-partition sizes of each PU for variable block size ME in HEVC

The VBSME feature in HEVC is much more complex. In HEVC, the maximum block size is 64x64. Each current frame is divided into blocks of Coding Tree Units (CTUs) or Coding Units (CUs) of maximum size 64x64. As explained in section 2.7 - Fig. 2.8, each CTU is sub-divided into Prediction Units (PUs) for inter-prediction. There are three variables that a PU can have - partition depth (d), partition size (s) and partition index (idx). The partition depth takes values from 0 to 3 corresponding to each CU size as shown in TABLE 2.5. Each CTU has different PU partition sizes with values 2Nx2N, 2NxN, Nx2N, NxN. The value N can have values 32, 16, 8 and 4 which corresponds to half of the CTU. For example, the CTU with partition size 32x32 (N=16) can have PU sizes 32x32, 32x16, 16x32 and 16x16. These modes are called symmetric partition modes. Apart from these, HEVC supports motion estimation for Asymmetric Partition Modes (AMPs) with PU sizes 3N/2 x 2N, N/2 x 2N, 2N x 3N/2 and 2N x N/2. Hence the CU size 32x32 can also have PU sizes 32x8, 32x24, 8x32 and 24x32.

Further each PU sub-partition is denominated by partition indices starting from 0. The various PU sizes that each CU can have are shown in TABLE 2.5, 3rd column. This is also illustrated in Fig. 2.14. The figure also indicates the partition index values for each PU sub-partition. The total number of MVs that each 64x64 PU (including its sub-partitions) can have are shown in TABLE 2.5 last column. For 8x8 CU size, the PU modes 8x6, 8x2, 6x8, 2x8 and 4x4 are usually not used in HEVC reference software [7], as the additional computational cost that has to be spent on encoder is huge even after considering their improved RD performance. Hence, the total number of variable block

TABLE 2.5: LIST OF VARIOUS PU PARTITIONS AND ITS SUB-PARTIONS IN A 64X64 CTU

CU depth (d)	**CU Size**	**PU and its Sub-block sizes**	**Number of PU partitions at each CU size**	**Total MVs for a 64x64 PU**
0	64x64	64x64, 64x16, 64x32, 64x24, 16x64, 32x64, 48x64, 32x32	32x32 – 4 partitions, rest of the sizes – 2 partitions each	64x64 – 1 MV, 32x32 – 4 MVs, rest of the sizes – 2 MVs each
1	32x32	32x8, 32x16, 32x24, 8x32, 16x32, 24x32, 16x16	16x16 – 4 partitions, rest of the sizes – 2 partitions each	16x16 – 16 MVs, rest of the sizes – 8 MVs each
2	16x16	16x4, 16x8, 16x12, 4x16, 8x16, 12x16, 8x8	8x8 – 4 partitions, rest of the sizes – 2 partitions each	8x8 – 64 MVs, rest of the sizes – 32 MVs each
3	8x8	8x4, 4x8	8x4 – 2 partitions, 4x8 – 2 partitions.	8x4 – 128 MVs, 4x8 – 128 MVs.
Total MVs				**593**

size MVs that are to be calculated in HEVC are 593 (as shown in the TABLE 2.5), which is huge in number compared to 41 MVs in H.264/AVC.

2.10.3 Multiple Reference Frames for Motion Estimation

The reference frame used for motion estimation can be more than one. Each PU (or macroblock in H.264/AVC) searches for MV in more than one previous frame, as illustrated in Fig. 2.15. As a result, the PU can choose the best matched block from more than one reference frame (in its corresponding SW) and hence the final compression efficiency can be improved but at the cost of increase in computational complexity.

In H.264/AVC, the maximum number of reference frames allowed is 16 [44] while in the latest standard HEVC, it is 4 [7]. Different PUs in the same current frame can have MV from different reference frames. Furthermore, different partition sizes in the same PU can have MV from different reference frames. To reduce the complexity, some of the reference frames are skipped using heuristic approaches. These are called reference frame skip algorithms [64].

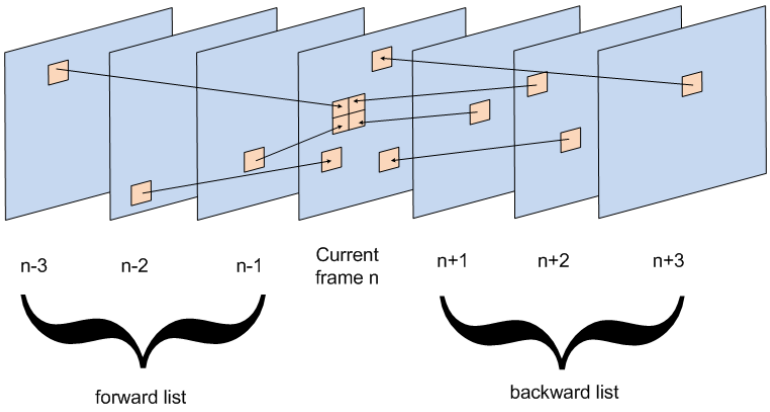


Fig. 2.15 Illustration or multiple reference frames and bi-directional ME

During decoding process, the reference frames are stored in temporary reference frame buffers. The usage of more reference frames demands increase in memory buffer size and its usage. Hence apart from increase in reference frame buffer size, the memory bandwidth also increases. Furthermore, due to increase in number of reference frames there is also more possibility of same reference memory locations being accesses multiple times. This problem is known as *locality of reference* which impacts speed of decoder. Efficient data reuse algorithms need to be designed as the number of reference frames increase.

2.10.4 Bi-directional Motion Estimation

As explained in section 2.2.2, the motion estimation and prediction can be done using both past and/or future reference frames. Prediction using past reference frames is called *forward prediction*, while the one using future reference frames is called *backward prediction*. This is illustrated in Fig. 2.15. Both the forward or backward prediction is done by storing the reference frames (past or future frames) in reference frame buffers. Using bi-directional ME, the compression efficiency can be increased at the cost of increase in complexity.

Technically the list of reference frames corresponding to forward prediction is called forward list, and corresponding to backward prediction is called backward list. Each frame (and block) that is being encoded is categorized as either of type I, P or B, which stands for Intra-type, Prediction type (forwarded only) or Bi-directional type (both forward and backward prediction). In an I-frame, all the coding blocks are only intra coded. A P-frame can have coding blocks of type P apart from I-type. A B-frame

can have all the types of coding blocks – I, P, and B type. A P-type coding block uses reference frames only from the forward list. A B-type coding block combines the reference frame indices from forward and backward list (technically called combined list) and uses this list for ME operation [7].

2.10.5 Fractional Motion Estimation

As explained in section 2.9, the motion estimation can be carried with sub-pixel accuracy. Just like in H.264/AVC, the accuracy of motion estimation in HEVC is carried out until ¼ pixel for luma samples. To obtain the subpixel samples, interpolation filters are used. These interpolated samples are later used to estimate the MVs using full search or fast search algorithms. In HEVC, the interpolation is usually done using 8-tap digital filter for half-pixel luma samples and 7-tap filters for quarter-pixel luma samples (horizontally and vertically) [53], [48].

An illustration of half-pixel and quarter-pixel samples around integer pixels is shown in Fig. 2.16. The letters with uppercase (A) in the figure shows integer sample locations, whereas the lower case letters represent fractional sample locations that will be generated using interpolation. The filter coefficients that are used in half-pixel and quarter-pixel interpolation for luma samples are shown in TABLE 2.6 [48].

2.10.6 Rate Distortion Optimized Motion Estimation

The cost function that is associated in the search process of motion estimation involves both distortion metric and the bitrate of the Motion Vector Difference (MVD). This is shown in (2.6). Hence, the final encoded output is optimized with video quality loss and the amount of data bits required by the final video (bitrate). Further it is easy to control the ME process if the encoding process is of constant bitrate type or constant quality type.

$$J_{MV} = D + \lambda . R \qquad 2.6$$

As shown in (2.6), the Rate Distortion Optimization (RDO) works by including Lagrange multiplier (λ) in the cost function. For a given target bitrate and QP (Quantization Parameter), the λ can be calculated using empirical relationship shown (2.7), where λ_{MODE} is the Lagrange multiplier when the distortion function is the SSD (Sum of Squared Difference).

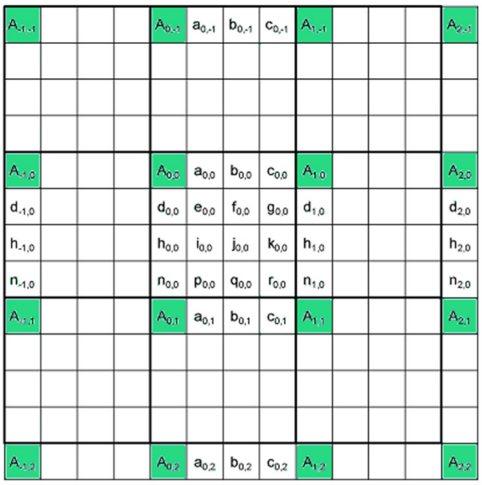


Fig. 2.16 Illustration of luma fractional interpolated samples around integer samples.

TABLE 2.6 SUB-PIXEL INTERPOLATION FILTER COEFFICIENTS IN HEVC FOR LUMA SAMPLES

Index *i*	**-3**	**-2**	**-1**	**0**	**1**	**2**	**3**	**4**
¼ position	-1	4	-11	40	40	-11	4	1
½ position	-1	4	-10	58	17	-5	1	0
¾ position	1	-5	17	58	-10	4	-1	0

If the distortion function used is SAD then the Lagrange multiplier can be denoted by λ_{Motion} and can be calculated using empirical relationship shown in (2.8). The value 'α' in (2.7) is equal to 1 for a non-referenced hierarchical B-picture. But for a referenced B-picture, the value depends on number of referenced B pictures (num_of_B_Pictures) used for that picture and can be calculated using (2.9), where *clip3(*a,b,v*)* function clips the value 'v' between 'a' and 'b'. The value 'W_k' in (2.7) is a weighting factor and depends on QP offset hierarchy level of the current picture within a GOP. Various values of 'W_k' with its corresponding QP offset, hierarchical level and slice type is shown in TABLE 2.7 [46].

$$\lambda_{Mode} = \alpha \times W_k \times 0.85 \times 2^{\frac{QP-12}{3}} \tag{2.7}$$

$$\lambda_{Motion} = \sqrt{\lambda_{Mode}} \tag{2.8}$$

$$\alpha = \begin{cases} 1.0 - \text{clip3}(0.0, 0.5, 0.05 \times num\_of\_B\_Pictures), & for\ referenced\ picture \\ 1.0, & for\ non-referenced\ picture \end{cases} \tag{2.9}$$

TABLE 2.7: DERIVATION OF 'W_K' VALUES USED FOR CALCULATING λ

Number of referenced Pictures, k	**QP offset Hierarchical Level**	**Slice type**	**Referenced**	W_k
0	0	I	-	0.57
1	0	GPB	1	RA: 0.442 LD: 0.578
2	1, 2	B or GPB	1	RA: 0.3536 x Clip3(2.0, 4.0, (QP-12)/6.0) LD: 0.4624 x Clip3(2.0, 4.0, (QP-12)/6.0)
4	3	B	0	RA: 0.68 * Clip3(2.0, 4.0, (QP-12)/6.0)

2.10.7 Distortion Metrics for Motion Estimation

There are many distortion functions that can be used in the cost function shown in (2.6) like SAD (Sum of Absolute Difference), SSD (Sum of Squared Difference), SATD (Sum of Transformed Difference), MSE (Mean Squared Error) etc. The distortion functions trade-off between search accuracy and complexity. The most commonly used functions are SAD and SSD. For a given current and reference blocks of pixels with equal size MxN, the SAD and SSD can be calculated using (2.10) and (2.11), where 'C' and 'R' are current and reference pixel blocks respectively, *(x, y)* are MV coordinates of reference block.

$$SAD(x,y) = \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} |C(i,j) - R(x+i, y+j)| \qquad 2.10$$

$$SSD(x,y) = \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} \big(C(i,j) - R(x+i, y+j)\big)^2 \qquad 2.11$$

The SSD metric has high prediction accuracy but has high computational complexity. For a given MxN current and reference blocks of pixels, the SAD has 3MN operations (subtract, absolute and add) whereas SSD requires MN additions and MN multiplications. Since multiplications require huge complexity over subtract and absolute operations, SAD is more often used than SSD.

3 MOTION ESTIMATION ALGORITHMS AND THEIR VLSI ARCHITECTURES

3.1 INTRODUCTION

As explained in Chapter 2, motion estimation is the most time consuming task in a video encoder. The main objective of the motion estimation is to find the optimized motion vector for each of the current encoding block. The brute way to do this is to search each point in the entire search window and find the motion vector of the block that gives a global minimum error. This is called Full-Search method. Full search method is very time consuming, however it gives the best video quality and the lowest bitrate (highest compression ratio). Instead of searching all the blocks in the search window, the motion estimation algorithms can be designed to choose a certain fixed or varying pattern of blocks which gives the closest matching block, or sometimes the best block. As mentioned earlier, these types of methods are called fast search methods. The fast search methods are usually very fast compared to full search, with only a slight reduction in encoded video quality (PSNR) and slight increase of bitrate.

The ME problem explained so far is a block based technique. But there are many other types of ME methods (or algorithms) which are used in various applications apart from video compression. Various types of ME methods and their classification are explained in the following section.

3.2 CLASSIFICATION OF MOTION ESTIMATION ALGORITHMS

There are many ways to classify ME algorithms. Broadly classifying, they can be categorized into two types - direct methods and indirect methods. In direct methods, the MVs are estimated directly from measurable image quantities at each pixel in the image (such as image brightness, brightness based cross correlation) [65]. In indirect methods (also called feature-based methods), distinct features (based on corners of the image, geometry of the objects such as epi-polar and focal geometry, photometric invariance of the image) from each image are extracted separately and then reconstructed and examined for their resemblances to find the motion and shape of objects [66]. Indirect methods typically match correspondence by using a statistical function applied over a local or global area in the image. In summary, feature-based

methods minimize the error surface based on the distances between some corresponding features while the direct methods minimize an error measurement based on direct image information from all pixels in the image.

Direct methods can be further classified into two types – time domain methods and frequency domain methods [67]. Frequency domain techniques are based on relationship between transformed coefficients of shifted images, and they are not widely used for image sequence coding. In these methods, the motion estimation is done by taking the transform of the block in frequency domain. Some of the methods in frequency domain are phase correlation using DFT, matching in DCT and wavelet domain. Time domain methods match the correspondences in the spatial domain of two different frames of a video. Time domain methods can be again classified into two types – pixel based methods and block based methods.

The pixels based methods (also called optical flow methods) determine MVs for each pixel in the image. They are designed with an assumption that the brightness or intensity of a pixel remains constant when they are shifted. They also add additional constraints like smoothness for the displaced motion vectors to make the algorithms interactive. Optical flow methods are used in many applications like object detection and tracking, image dominant plane extraction, movement detection and robot navigation. The optical flow methods require huge computation time which make this impractical for video compression applications. Hence, an alternative approach for video compression are the block based methods. In block based methods, the entire frame is divided into non-overlapping blocks (of sizes such 64x64, 16x16, 8x8) and for each block the optimal MV (MV of block which has least distortion) is searched in the reference frame. Although this is done with an assumption that the entire block undergoes a translational motion, it is practically valid except for introduction of blocking artefacts, which can be removed through de-blocking filters [16]. Because of its low complexity in implementation (compared to optical flow methods), block based methods are used in most of the video coding standards including MPEG-2, H.264/AVC and the latest standard, HEVC.

Further, there are two approaches in estimating the MVs using block based methods. The first type is full search and the others are fast ME methods (or algorithms)

as explained in Section 3.1. The fast search algorithms can be of many types. Broadly classifying, they can be categorized into two types: basic and hybrid algorithms. The basic algorithms are based on a single idea or concept. The hybrid algorithms combine two or more basic algorithms with a mixture of concepts. Some of the basic and hybrid fast ME algorithms are explained in Section 3.4 and Section 3.6, respectively.

3.3 THE FULL SEARCH ALGORITHM

The full search method searches every possible location in the entire search window. As a result the algorithm finds the best matching block for every block in all the frames of video and gives the highest PSNR. But the computational time is very high. Let R be the search range set for the ME algorithm. Then the maximum possible number of search points for the SW are $(2R+1)^2$. Let W_{CB} x H_{CB} be the width and height of the current block and let W_f x H_f be the frame width and height. Then there will be N_{CB} coding blocks for each frame that can be calculated using (3.1). Let N_p be the number of sub-partitions for each current block CTU. Let N_f be the number of reference frames for each block that the ME has to be performed. Then the total number of search points per CTU $N_{SP\text{-}CTU}$ can be calculated using (3.2).

$$N_{CB} = \frac{W_f \times H_f}{W_{CB} \times H_{CB}} \tag{3.1}$$

$$N_{SP-CTU} = N_f \times N_p \times N_{CB} \times (2R+1)^2 \tag{3.2}$$

For example, in H.264/AVC, the default maximum coding block size 16x16, with seven variable block size modes varying from 16x16 to 4x4 (one 16x16, two 16x8, two 8x16, four 8x8, eight 8x4, eight 4x8, sixteen 4x4 blocks). The total N_p is equal to 41, the default N_f is equal to 5. For an HD frame (1280x720), there are 3600 coding blocks (N_{CB}). Hence the $N_{sp\text{-}CTU}$ is equal to 738k blocks/frame. For a five minutes video with 30fps frame rate, the total number of frames is equal to 9k, and hence the total search points will be 6.642×10^9. For HEVC the number is even higher, since the coding block size is 64x64. While this number is very huge, the fast search algorithm rely on reducing the N_f and/or N_p and/or $N_{CB.}$ Algorithms that reduce N_{CB} are called block skip algorithms, and that reduce N_f are called reference-frame skip algorithms. Algorithms that reduce search points N_p which is at SW level are typically called fast ME algorithms. The present thesis proposes a fast ME algorithm for HEVC.

3.4 TYPES OF FAST MOTION ESTIMATION ALGORITHMS

The fast ME algorithms can be classified into many types. Based on the loss of output video quality, the algorithms can be classified into lossless fast ME algorithms and lossy fast ME algorithms. In lossless fast ME algorithms, the ME algorithms achieve gain in speed with same PSNR (video quality) and bitrate (compression ratio) compared to FS algorithm. The Successive Elimination Algorithm (SEA), MLSEA (Multi level Successive Elimination Algorithm), PDE (Partial Distortion Elimination) etc. are some examples. In lossy fast ME algorithms, there will be huge gain in speed but slight decrease in PSNR and bitrate compared to FS algorithm. Some of the examples are search area sampling techniques, pixel decimation techniques, hybrid algorithms (include more than one fast ME algorithms) etc. Each of these techniques is explained briefly in the following sub-sections.

3.4.1 Successive Elimination Algorithms

The Successive Elimination Algorithm (SEA) is a two stage algorithm [68]. In the first stage, the algorithm calculates the absolute difference between sum of intensities of all pixels between current and reference blocks as shown in (3.3), where ADS represents Absolute Difference between Sums, C represents current frame block, R represents Reference frame block and with MxN taken as the size of the block. In the second stage, the algorithm eliminates the search points based on an inequality equation shown in (3.4).

$$ADS = \left| \sum_{i}^{M} \sum_{j}^{N} C(i,j) - \sum_{i}^{M} \sum_{j}^{N} R(i,j) \right| \tag{3.3}$$

$$ADS \leq SAD = \sum_{i}^{M} \sum_{j}^{N} |C(i,j) - R(i,j)| \tag{3.4}$$

For any two blocks of equal size MxN, the ADS is always less than or equal to their SAD. Based on this inequation, the search blocks which has larger ADS then the current minimum distorted search point, can be omitted from the search process (and SAD calculation is not done). Though there is an additional cost of adding the intensity values, the omitted search points account more for reduction in total complexity. By

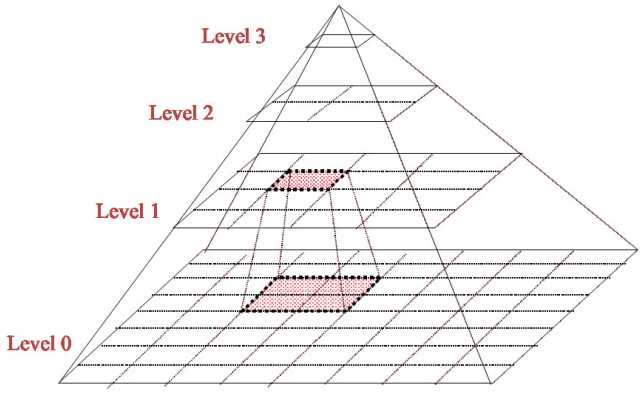


Fig. 3.1 Hierarchical pyramid structure for ME

using SEA in [68], the computational complexity of ME is reduced by 75% to 85% compared to full-search algorithm.

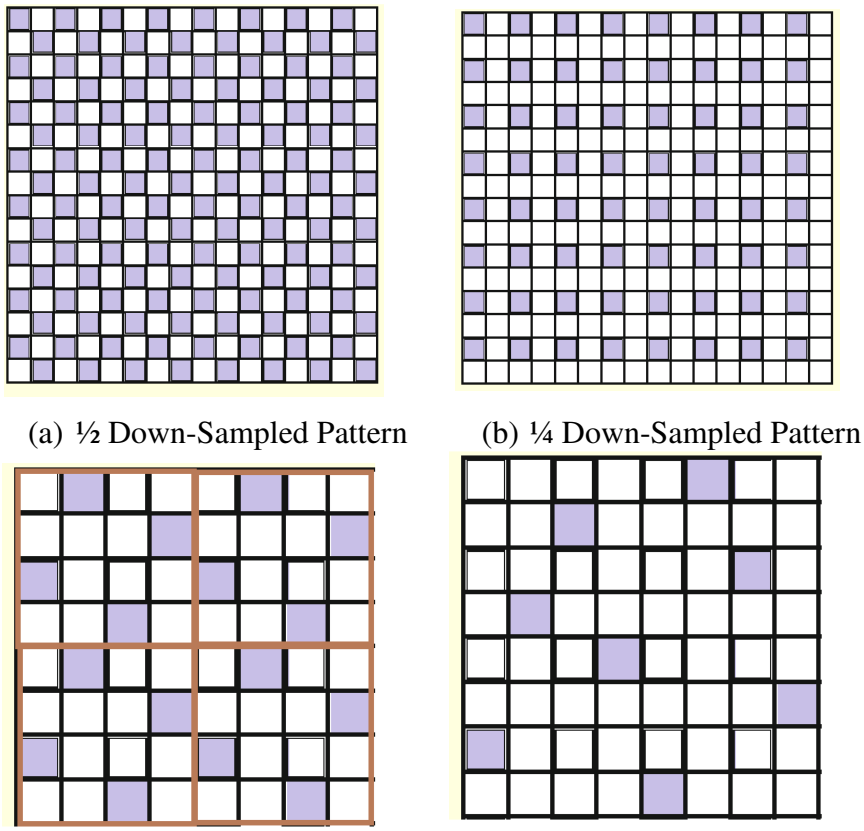
In [69], the MSEA (Multilevel SEA) algorithm was proposed which extends the concept of SEA in a multilevel hierarchical way. Each level is a subsampled version of original current/reference block, as illustrated in Fig. 3.1. In each level l, the pixel values are calculated by adding all the values of corresponding neighbouring pixels at level l+1. Then, the Subsampled-SAD (SSAD) at each level between current and reference blocks is calculated. Based on the inequality shown in (3.5), the complete SAD is calculated only if the corresponding block satisfies the criteria in all the levels. The MSEA is stated to reduce the computational complexity up to 95% compared to FS algorithm.

$$SSAD_0 \leq SSAD_1 \leq SSAD_2 \leq \cdots \leq SSAD_l \cdots \leq SAD \tag{3.5}$$

Another important lossless technique is PDE (Partial Distortion Elimination) [70, 71]. The PDE algorithm eliminates the non-possible candidates before the complete calculation of matching error. Here the calculation of matching error is considered as sequential calculation and accumulation of partial distortions. Hence, the non-possible candidate can be judged if the accumulated partial distortion for the candidate exceeds minimum distortion or minimum cost value. Like SEA algorithms, the PDE algorithms also reduce huge computation time compared to FS algorithm.

3.4.2 Hierarchical and Multiresolution Algorithms

In hierarchical search algorithms (also called search area sampling algorithms), the given frame or SW is down-sampled to lower resolution. Typically this is done by



(a) ½ Down-Sampled Pattern (b) ¼ Down-Sampled Pattern

(c) 4-Queen Tiles in 8x8 Block (d) 8-Queen Pattern

Fig. 3.2 Pixel-decimation Pattern for Motion Estimation

forming a multi-level pyramidal structure, with each higher level containing a down-sampled version of SW (or frame) for previous level, as shown in Fig. 3.1. Usually, these multiresolution levels are formed using sampling filters like low-pass filters [72, 73]. In the hierarchical SW technique [74], the ME is first coarse performed at higher level of pyramid hierarchy and for fine-refinement the ME is performed in the lower levels (high resolution level).

The hierarchical motion estimation algorithms are not only used in video compression, but also widely used in in frame interpolation for frame rate up conversion applications [72, 75]. In [73], a hierarchical stochastic fast ME algorithm was proposed which uses Kalman filter to get the low resolution hierarchical levels, and then the final MVs are obtained by using block matching ME algorithm. The total computational gain is about 5% of the computations taken for full-search algorithm. In [76], a hierarchical motion estimation method using adaptive image down-sampling method was proposed. Based on the motion analysis in the frames the sampling position of the pixel in the

frame is changed adaptively. The proposed algorithm achieves more accurate MVs than that of traditional hierarchical ME methods.

3.4.3 Pixel Decimation Block Matching Algorithms

The block matching algorithms uses matching criteria on all pixels of a block with an assumption that all the pixels in the block are moved by the same amount of displacement (and direction). But, if only few pixels in the block are used, then the accuracy of estimated MV will be degraded. However, if some pre-defined pattern of pixels in the blocks is used, the degradation in the accuracy of MV estimate may be controlled and reduced. Hence by reducing the complexity for each search point, there will be a huge savings in the computational time of the total ME algorithm. Some of these patterns are shown in Fig. 3.2. Fig. 3.2 (a) shows the ½ down-sampled version and Fig. 3.2 (b) shows ¼ down-sampled version of a 16x16 block originally proposed in [77] and the ¼ down-sampled version is analysed and improved in [74].

There are many other approaches like hexagonal pattern, spiral patterns etc. and one of the efficient and successful approach was by using N-Queen pattern proposed in [78], as shown in Fig. 3.2 (c) and (d). The name is derived from a famous problem in chess with an objective of placing 'N' queens in an NxN chessboard such that no two queens threaten each other. Fig. 3.2 (c) shows the lattice structure for 4-Queen pattern of pixels tiled in an 8x8 block, while Fig. 3.2 (d) shows 8-Queen pattern. To evaluate the efficiency of the patterns objectively, the spatial homogeneity and directional coverage are calculated. The spatial homogeneity is measured using average (μ) and variance (σ^2) of spatial distances from each skipped pixel to its nearest skipped pixel as shown in (3.6) and (3.7), where S(x,y) represent co-ordinates of pixels selected nearest to the position P(x,y), K represents number of selected pixels and N represents the dimension of the block. For calculating the directional coverage, an edge is defined which a line is passing through the pixel point taken in any of the directions horizontal (0^0), vertical (90^0) and diagonal (45^0 and 135^0). The directional coverage is measured as percentage of edges with at least one of the selected points exist on the edge,

$$\mu = \frac{1}{(N^2 - K)} \sum_{(x,y)}^{N} \|P(x,y) - S(x,y)\| \tag{3.6}$$

$$\sigma^2 = \frac{1}{(N^2 - K)} \sum_{(x,y)}^{N} (\|P(x,y) - S(x,y)\| - \mu_d)^2 \quad (3.7)$$

In [78], a comparison for sampling lattices for an 8x8 block is performed and showed that the 8-Queen pixel-decimation pattern has highest spatial coverage compared to other patterns. The directional coverage of 8-Queen pattern has full directional coverage which is equivalent to non-decimated block's directional coverage. In [79], a new pixel decimation algorithm for ME was proposed based on boundary region matching and genetic algorithms for finding the optimal length pattern in an NxN block. The algorithm improved coding efficiency almost similar video quality compared to existing pixel-decimation patterns.

3.4.4 Search Points Reduction Algorithms

The search point reduction algorithms aim to reduce the complexity of ME process by reducing the number of search points in the search window. Some of these algorithms are explained in the following sub-section.

3.4.4.1 Three Step Search (TSS) Algorithm

The TSS algorithm was one of the oldest algorithm that was proposed (1981) for ME [77]. Later, many modifications were proposed to improve the performance and efficiency of the algorithm. The general idea of the algorithm is shown in Fig. 3.4. The three step search starts with searching the centre point, which is the collocated point of current block in reference frame. Then, with step size 4, it starts searching the surrounding eight locations, shown in circles. The point which has the minimum error (minimum SAD or SSD), is taken as reference centre point to the next step. If the lowest cost is at the centre, then the motion search is stopped and the centre point is taken as the motion vector. Otherwise, the algorithm proceeds to second step. In the second stage, the step size is reduced to 2, and the search is performed around the surrounding eight positions, as shown in triangles in the figure. The best matching point is again taken as reference, to the third step and the search is performed with step size one, as shown with square dots in the figure. The step is the final step, since there is no further search possible with step size less than one. The best matched point in the third step is the final best matching motion vector.

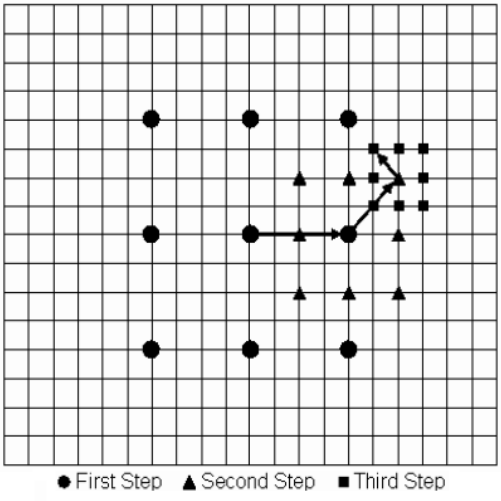


Fig. 3.3 Illustration of Three-Step Search Algorithm for Motion Estimation

3.4.4.2 Block Based Gradient Descent (BBGDS) Algorithm

The Block Based Gradient Descent Search algorithm applies the gradient descent algorithm (or steepest descent algorithm) that is widely used in optimization theory [80]. The steepest descent algorithm basically optimizes a function for minima, in the steepest direction. A similar approach is followed for block matching algorithm, where the search is performed based on the minimal point obtained in previous steps. This is similar to the TSS algorithm with a step size of one at each stage of algorithm. The search pattern used in BBGDS algorithm is also a square pattern with step size one. Initially, the search is performed for the eight points around the centre point, along with the centre point location. The minimal point is taken as the origin for the next step and the pattern checked in the next step is also square pattern with step size one. Depending on the location of the minimal point, the number of search points in the next step will be either three or five points. If the location is at the corner of the square pattern, the number of search points will be five in the next step, otherwise the number of search points will be three. If the minimal point is at the centre point, the algorithm stops, and the centre point is taken as the final motion vector location. The algorithm continues in unlimited steps, until the minimal point is found to be at the centre point location. The algorithm is illustrated graphically in Fig. 3.4.

The example shown in Fig. 3.4 has a solution in eight steps. The centre point location for the eight step, which is taken from the optimal point obtained in the seventh

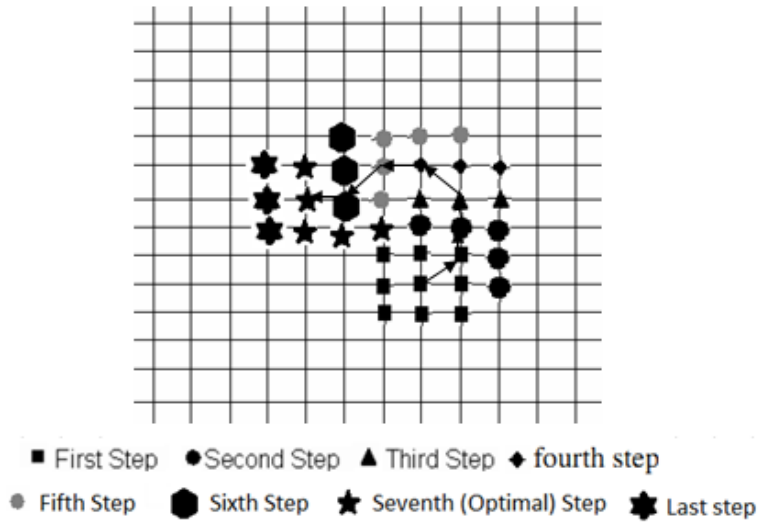
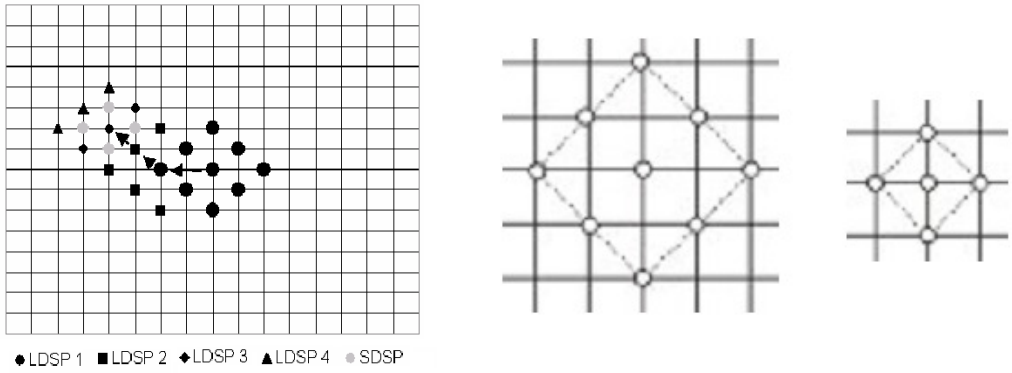


Fig. 3.4 Illustration of Block Based Gradient Descent Algorithm

step (indicated by *) is the final optimal motion vector. The BBGDS algorithm gives the fastest and the most optimal motion vector since the minimal distorted blocks are continuously converging towards the minima. The main disadvantage is that this algorithm gets trapped in the local minima point, just like the gradient descent algorithm in optimization theory. A conjunction of this algorithm with another global search algorithm, will give the most optimal motion vector in fewer search locations.

3.4.4.3 Diamond Search (DS) Algorithm

The Diamond Search algorithm [81] is similar to BBGDS algorithm except that the search is performed in diamond shaped pattern instead of square pattern. In principle, the algorithm takes two types of fixed patterns, one is Large Diamond Search Pattern (LDSP), and the other is Small Diamond Search Pattern (SDSP), shown in Fig. 3.5 (b) and (c) respectively. The first steps of the algorithm are initiated with the LDSP and the last step uses the SDSP, as illustrated in Fig. 3.5(a). If the least cost point is at the origin in the initial steps, the algorithm jumps to last step. The algorithm continues using LDSP, until the minimal point comes out to be at the origin of the LDSP. If the least weight is not at the origin, then in the algorithm checks three or five locations depending on the location of the minimal point in the previous step. If the minimal point is at the corner of the LDSP, then the next step needs to be checked 5 locations and if the minimal point is at the side of the diamond, then the next step needs to be checked



(a) (b) (c)

Fig. 3.5 Illustration of (a) Diamond Search Algorithm (b) Large Diamond Search Pattern (LDSP) (c) Small Diamond Search Pattern (SDSP)

only 3 locations. Since there is no limitation to the number of steps, the algorithm gives a good PSNR which is close to that of Full Search algorithm [81].

3.4.4.4 2D-Logarithmic Search Algorithm

The Two Dimensional Logarithmic Search is designed by extending the TSS Algorithm [82]. It starts searching with five locations along the centre of the edges of the search window, and the centre point. The minimal point is taken as the reference point for the next step and the search area is reduced by a factor of two. The algorithm continues to search until the search area is reduced to 3x3, and in the last step the search is performed in the entire nine locations. The algorithm is illustrated graphically in Fig. 3.6.

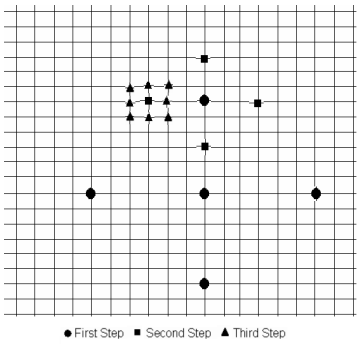


Fig. 3.6 Illustration of 2D Logarithmic Search Algorithm

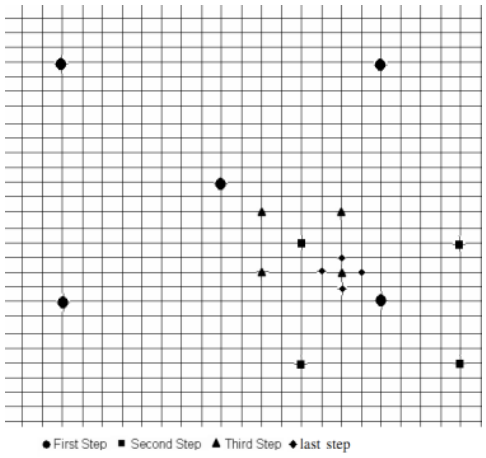


Fig. 3.7 Illustration of Cross Search Algorithm

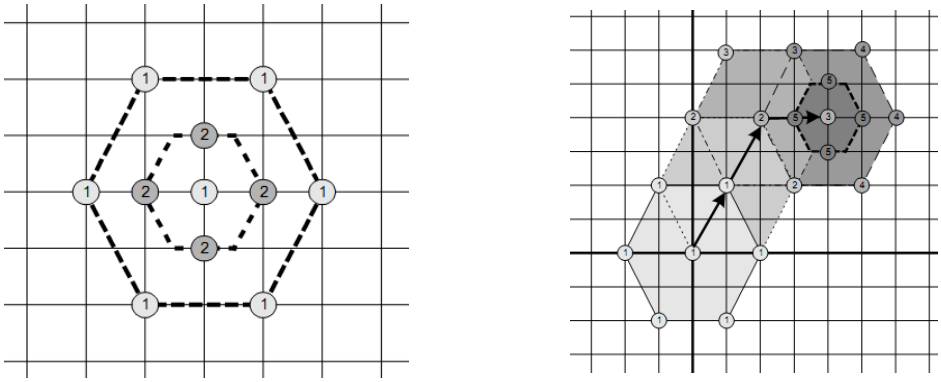
3.4.4.5 Cross Search (CS) Algorithm

The cross search algorithm [11] is similar to Logarithmic search, except that the search is performed in cross ('X') pattern instead of diamond (plus '+') pattern. The algorithm starts with the corners of the search window, along with the centre point. The minimal point obtained is taken as the reference centre to the next step, with search window size reduced by a factor of two. The algorithm continues until the search window size is reduced to 3x3. In the last step with search window size 3x3, the pattern is either cross pattern or SDSP (Small Diamond Search Pattern), depending upon the location of the minimal point obtained in the previous step. If the location of the minimal point obtained in the previous of the last step is at the bottom left or top right, then the search is performed in a SDSP pattern in the last step, otherwise it is performed in a cross pattern in the last step also. The algorithm is illustrated graphically in Fig. 3.7.

3.4.4.6 Hexagonal Search (HS) Algorithm

The hexagonal Search is one of the most efficient fast-block-matching-algorithms [83]. The algorithm searches for the minimal point by considering a hexagonal pattern. The algorithm can be explained by classifying into three stages: Starting, Searching and Ending.

Starting: The large hexagon (shown in Fig. 3.8(a)) along with centre point (0,0) making seven points are checked initially. If the minimal point is found to be at the centre, then the algorithm skips to final stage, the ending step.



(a) (b)

Fig. 3.8 Illustration of (a) Hexagonal Search Pattern (b) Hexagonal Search Algorithm

Searching: The minimal point (minimal distorted block), obtained in the starting step is taken as centre, and a large hexagon is again formed by adding only three new search locations. With the obtained hexagonal pattern, the minimal point is again searched. If the minimal point is found to be at the centre, then the algorithm skips to the third step 'ending', otherwise this step is repeated continuously.

Ending: In this final step, the search pattern is switched from large hexagon to small hexagon that needs to be covered only four points, as shown in Fig. 3.8 (a). The minimal point obtained in this small hexagon is taken as the final motion vector block.

An example based on Hexagonal search is shown in Fig. 3.8 (b), which converges towards the minima in five steps. The hexagon search is similar to BBGDS algorithm, except that the search is performed in hexagonal pattern instead of square pattern. Changing the pattern to hexagon makes the convergence towards the optimal point more fast, since in every successive step (except initial and final steps), the number of new search points added is only three, while in the BBGDS algorithm the number of points added are three or five based on the location obtained in the previous step. On the other hand, the 2DLGS and Cross Search Algorithms also have three locations added in every subsequent step, but the probability of converging towards the local minima is more compared to Hexagonal Search Algorithm.

The aforementioned algorithms search for MV with an assumption that the optimal point in monotonically converging. But the objects in videos move randomly in

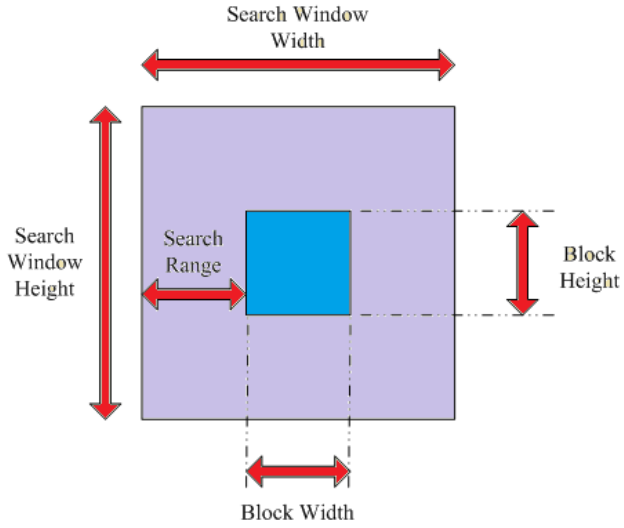


Fig. 3.9 Relation of Search Range with Search Window Size

directions and the same block may contain more than one object that move in different directions. Hence these algorithms are to be pre-processed with one or more techniques for searching an efficient MV while the search is performed to minimum number of points. These are called hybrid ME algorithms. The Hybrid ME algorithms embed more than one technique to reduce the complexity of the ME process and to maintain almost similar video quality and bitrate compared to FS ME algorithm. The details of some of the techniques used in hybrid ME algorithms are explained in the following section.

3.5 COMPLEXITY REDUCTION TECHNIQUES IN SEARCH POINT REDUCTION ALGORITHMS

There are many ways to reduce the complexity of SW points. But the complexity reduction should not decrease the output video quality and compression efficiency compared to full search algorithm. Hence all the complexity reduction techniques are typically compared against the results of Full Search algorithm.

3.5.1 Dynamic Search Range for Motion Estimation

Before starting the search operation, the Search Window (SW) size or its range (distance from search centre to outer edge of SW) has to be defined. The search window size depends on search range and search block size as shown in (3.8), where SW_W and SW_H represents SW width and SW height respectively, SB_W, SB_H represents search

block width and search block height respectively, SR_x and SR_y denotes the search range in horizontal and search range in vertical direction respectively. This is also illustrated in Fig. 3.9. The number of search points only depends on search range in horizontal (SR_x) and vertical directions (SR_y) as shown in (3.10). The search range is typically same in horizontal and vertical directions and is equal to maximum block size defined in the codec. For H.264 its default value is equal to 16 [44], and for HEVC it is equal to 64 [8].

$$SW_{W,H} = \left(2 \times SR_{x,y}\right) + SB_{W,H} \tag{3.8}$$

$$SP = \left((2 \times SR_x) + 1\right) \times \left(\left(2 \times SR_y\right) + 1\right) \tag{3.9}$$

For reducing the complexity of ME operation, the SW size can be reduced dynamically based on statistics of previously coded neighbouring MVs. There are many dynamic search algorithms that were proposed in the literature [9]-[12]. HEVC reference software [4] uses an adaptive search range algorithm, which changes SW size in according to temporal difference between current and reference frames as shown in (3.10), where POC represents Picture Order Count or corresponding current and reference frames, 'Round' represents rounding function, SR_{Max} denotes the maximum search range configured initially for ME, ASR_SCALE denotes adaptive search range scaling factor with default value 1 and 'GOP_Size' denotes the GOP size. According to (3.10), for near reference frames, the search range increases and for farther reference frames the search range decreases

$$SR = \frac{Round(SR_{Max} \times ASR\_SCALE \times \left|POC_{Cur} - POC_{Ref}\right|}{GOP\_Size} \tag{3.10}$$

In [84, 85], the authors use Cauchy and Laplace distribution functions to model previously encoded neighbouring motion vector differences (MVDs) and vary the search range according to their probabilities. Compared with full search algorithm, the achieved reduction in complexity was more than 90%. In [86], an adaptive search range algorithm working at block level based on quantization parameter was proposed. The ME computational time reduction achieved was about 16% on average compared to H.264/AVC fast ME algorithm [87]. In [88], the authors describe an adaptive search

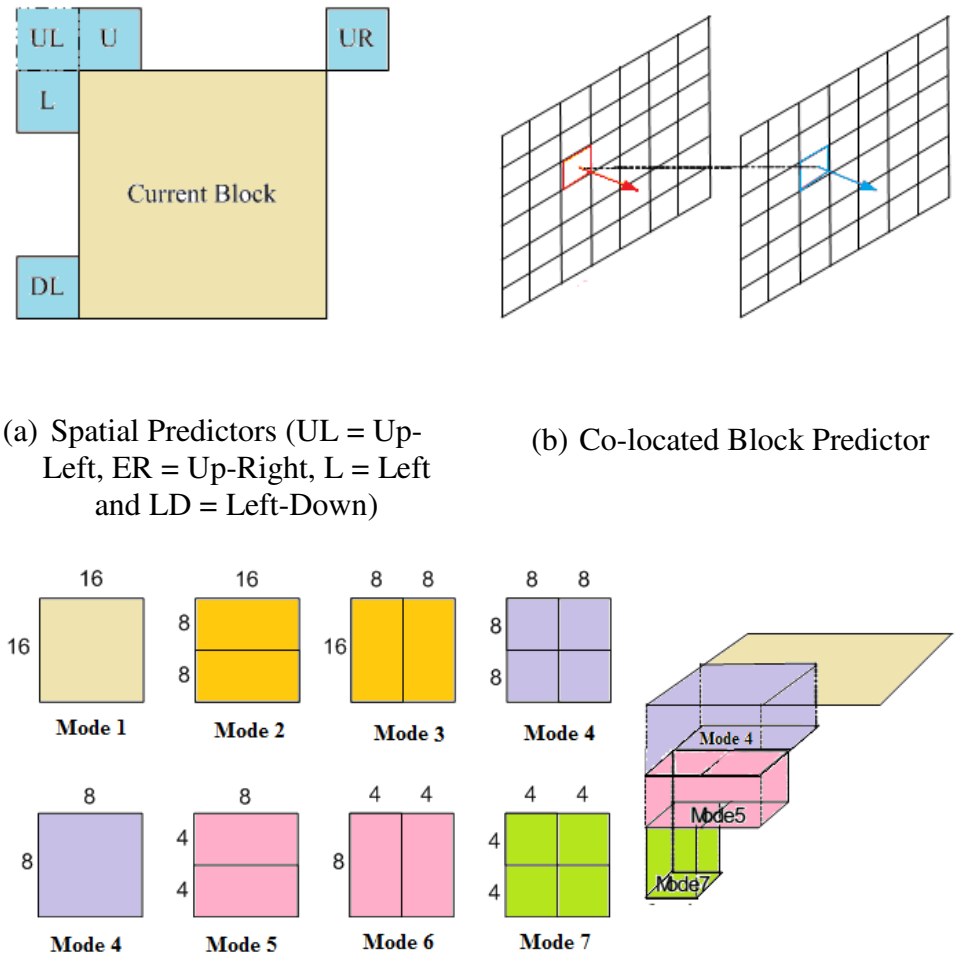
range algorithm working at block level based on motion vector differences (MVDs). The algorithm increases the search range by one pixel and performs ME for the new search points, if the MVD (after motion estimation) lies on the edge of search window. The achieved reduction in computation load is about 97% compared with full search algorithm.

3.5.2 Predictive Based Motion Estimation

One of the simplest techniques to reduce the ME complexities is to start the search process with initial point at nearest possible position from final MV. This is achieved by predicting the MV using prediction algorithms and terminating the ME algorithm using early termination algorithm. A good prediction algorithm provides a good initial search point close to the optimal MV and reduces the number of iterations in ME, thus reducing the complexity. To do this, a ME algorithm can incorporate more than one prediction algorithm and take the least cost point as the initial search point. One of the earliest and efficient prediction algorithms is the median predictor, which is the median of left, right and up-right (or up-left) block's MV. Fig. 3.10 shows the position of candidate blocks left, up, up-right and up-left and their MVs are used for spatial prediction. Since these blocks are already encoded, their MVs are used for predicting the current block MV. The MVs of these blocks are called predictors - left predictor for MV of left block, up predictor for MV of up block and up-right predictor for MV of up-right block. The median predictor is defined using (3.11), where MV_L, MV_U and MV_{UR} are left predictor, up predictor and up-right predictors. If the up-right predictor is not available (for blocks in right side border of the frame) then up-left (MV_{UL}) predictor is used in (3.11).

$$MV_{Pred} = median(MV_L, MV_U, MV_{UR}) \tag{3.11}$$

Some of the prediction algorithms used in H.264/AVC ME algorithms are median prediction, up-layer prediction, co-located block prediction and neighbouring reference frame prediction [87, 43]. The co-located block predictor is the MV of the co-located (same position block in the previous frame) block, as shown in Fig. 3.10 (b). The up-layer predictor (or upper-mode predictors) is the MV of upper-mode variable-size block of the same block (macroblock in H.264/AVC) which is already encoded.



(a) Spatial Predictors (UL = Up-Left, ER = Up-Right, L = Left and LD = Left-Down)

(b) Co-located Block Predictor

(c) Upper Mode Predictors

Fig. 3.10 Various Positions of Blocks for Prediction in Motion Estimation

Fig. 3.10 (c) shows the upper mode blocks for various block sizes in H.264/AVC. Since the ME is performed from higher block size to lower block sizes, the MVs of its immediate higher size blocks (or upper mode blocks) can be used to predict the MV of the current block. HEVC uses left, up, up-right, up-left, down left and median predictors for ME [8]. The down-left predictor is the MV of the block located left down-side of the current block as shown in Fig. 3.10.

In [89, 90], the authors proposed Simulated Annealing Adaptive Search (SAAS) algorithm which predicts initial search point based on statistical analysis of previous frame's Motion Vector Correlation. The search pattern is adjusted for each block according to Predicted Motion Vector and the search region is adaptively divided and

Simulated Annealing (SA) mechanism is adopted to select search power for each region to avoid trapping into local minima. It was reported that the algorithm offers considerable improvement in computing time and motion search points at the same rate-distortion performance compared to the conventional fast motion estimation algorithms. In [91], the initial search point is predicted based on directional asymmetry search, by taking a small diamond pattern with five points as the initial predictors. It was reported that method significantly reduces the number of search points for locating a motion vector and contribute to speed up the search process with reasonable PSNR values.

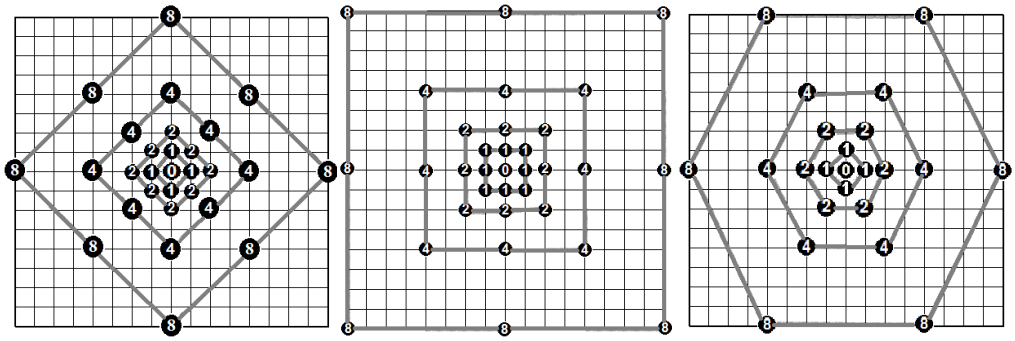
3.5.3 Early Termination Algorithms for Motion Estimation

After the prediction step, some of the prediction blocks may get the optimal motion vectors without the necessity to perform further searches. For this reason, fast ME algorithms use early termination algorithms, defining a threshold to terminate the search process. If the threshold value is fixed to a constant, the encoded video may lose quality because some of the MVs may have better MV than the threshold limit. On the other hand if the threshold is too lower, the algorithm may lose performance in terms of speed. Hence early termination (ET) algorithms uses adaptive threshold, which changes threshold value dynamically depending on parameters such as SADs or MVDs of previously encoded neighboring blocks.

Adaptive ET algorithms for ME can be applied at frame level and block level. In our work, for determining the threshold in ME algorithm for HEVC, the average of all costs in the first frame of GOP after intra-frame is taken since there is a close relationship between average costs of each frame of GOP. This is shown in (3.12), where 'N(d,p)' represents the total number of coding units in the first frame of GOP for the depth 'd' and partition size 'p', and cost is the distortion cost used in the algorithm.

$$Th(d,p) = \sum_{i=1}^{N(d,p)} \frac{cost(i)}{N(d,p)} \tag{3.12}$$

In H.264/AVC, fast ME algorithms - UMHexS, SUMHexS [92], the threshold is defined at block level using SADs of previously encoded spatial neighbouring blocks, upper mode SAD, collocated block SAD. In [93], the authors proposed an early-level



(a) Diamond Pattern (b) Square Pattern (c) Hexagonal Pattern

Fig. 3.11 Search Patterns for Motion Estimation with stride length 8

termination method suitable for hierarchical ME structure which terminates high-level redundant motion searches by establishing thresholds based on current block mode and motion search level. It also applies the early refinement termination in order to avoid unnecessary refinement for high levels of hierarchical block structure. In [94], an adaptive early termination strategy was developed based on statistical characteristics of rate-distortion (RD) cost regarding current block and previously processed blocks and modes. It was reported that by using their method, most motion searches can be stopped early, with a large number of search points saved.

3.5.4 Grid Patterns for Finding Motion Vector

If the motion vector does not satisfy the early termination condition, the ME algorithm starts searching for the best matched block. In the searching process, there is a possibility that the MV gets trapped into local minima. To avoid this, the ME process uses a global optimization algorithm. The global search patterns are usually grids (diamond or square or hexagon grids) with stride-lengths (length from start point to grid pattern) smaller near initial search point and with stride-lengths large as the grid moves away from start point. Fig. 5.5 shows some of the most commonly used grid patterns. Fig. 3.11(a) shows square patterns, Fig. 3.11 (b) shows diamond patterns and Fig. 3.11 (c) shows hexagon patterns. These patterns are evaluated based on the number of search points used in a given search window and based on the directional coverage. A good pattern must have highest directional coverage with lowest number of search points used. From the patterns in Fig. 5.5 the hexagon has better directional coverage and lower number of search points compared to others, and hence many fast algorithms use hexagon patterns to find the coarse refined MV [87, 92]. The minimum cost block in

these grids can be taken as the initial MV for further processing steps in ME algorithm. After finding the minimum point, the ME algorithm can check with a threshold value for early termination again.

Some hybrid ME algorithms like EPZS [95] and PMVFast [96] skip this global search stage since they rely on several prediction algorithms to predict the initial search point (explained in detail in Section 3.6.1). Fast ME algorithms in H.264/AVC reference software like UMHexS and SUMHexS [92, 87, 43] use hexagonal grids to find the coarse optimum point (explained in detail in Section 3.6.2 and Section 3.6.3). TZSearch fast ME algorithm in H.264 JMVC (Joint Multi-view Video coding) [97] and HEVC [7, 8] use diamond and square patterns to find the global optimum (explained in detail in Section 3.6.4). In [98], a new fast ME algorithm was proposed by replacing multi-hexagonal grids with multi octagonal grids in UMHexS algorithm [87]. It was reported that the algorithm can reduce five to ten percent of the computational complexity of the UMHexS algorithm without loss of its accuracy. In [99], a new ME algorithm based on multi-octagonal grids was proposed. The algorithm takes adaptive search strategies by using the distribution characteristics of motion vector. The reported improvement in speed is 91% compared to FS algorithm with little negligible loss in PSNR and compression ratio.

3.5.5 Fine Refinement Patterns for Finding Optimal Motion Vector

After finding the global minima, the ME algorithm can be designed to use early termination algorithm. If the threshold for MV cost is not met by global minima, then the ME algorithm refines the MV obtained from the previous step using one of the gradient descent based algorithms explained in Section 3.4.4. As explained, the gradient descent based ME algorithm starts with an initial search point and finds the cost of the search points of the pattern formed around the search center.

The fast ME algorithms in H.264/AVC like UMHexS, SUMHexS [92, 87, 43] use hexagonal patterns for fine refinement. The EPZS algorithm uses LDSP and SDSP patterns for fine refinement [95]. In [100, 101], the authors propose a pattern switching mechanism that adaptively switches between 3-step search and block-based gradient descent search algorithms. The algorithm classifies the motion content of a block using a simple and efficient motion content classifier called error descent rate where the

classifier requires only the searching of a few points in the search window and then a division operation. It was reported and verified that the algorithm is very robust. In [102], a new Hybrid Hexagon Kite Cross Diamond Search (HYBHKS) algorithm based on adaptive switching of the Hexagon Search (HEXS) and Kite Cross Diamond Search (KCDS) patterns was proposed. It was reported that that HYBHKS performed better than KCDS and HEXS in terms of number of search points while maintaining similar compression ratio and PSNR.

3.6 H.264/AVC AND HEVC HYBRID MOTION ESTIMATION ALGORITHMS

The encoder in reference software for H.264/AVC (JM) and for HEVC (HM) uses hybrid ME algorithms, which combines more than one technique (as mentioned in Section 3.2). Some of these hybrid ME algorithms are explained in detail in the following sub-sections.

3.6.1 EPZS (ENHANCED PREDICTIVE ZONAL SEARCH) Algorithm

The Enhanced Predictive Zonal Search (EPZS) algorithm [95, 103] is a prediction based hybrid algorithm and mainly comprises of 3 steps. The first step is the initial predictor selection which selects the best MV predictor from a set of potentially likely predictors. The second step is the adaptive early termination which terminates the motion estimation process if some predefined conditions are satisfied. The last step is the prediction refinement stage which employs a refinement pattern around the best predictor to essentially improve the final prediction. All these features add up to improve the performance of the algorithm.

3.6.1.1 Predictor Selection

The predictor selection stage is the key feature in EPZS algorithm, as the accuracy and speed of converging optimal MV depends on how best the predicted MVs are. The prediction algorithms used here are, spatial predictors (left, top, top-right), median predictor, co-located block predictor and temporal predictors. The spatial, median and collocated predictors are explained in Section 3.5.2. In temporal predictors, the MVs of neighbouring blocks of collocated block are taken, as shown in Fig. 3.12. If there is more than one reference frame, then for the new reference frame which is at temporal distance TD_N from current frame, the new motion vector (MV_N) can be predicted using

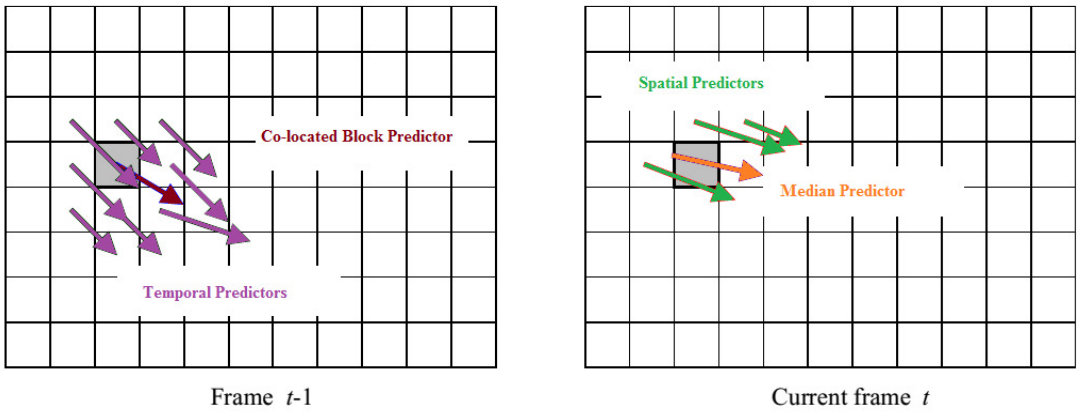


Fig. 3.12 Various Predictors Used in EPZS Algorithm

previous coded MV information using other reference frames as shown in (3.13). MV_C in (3.13) denotes already coded MV for current block in a reference frame which is at temporal distance TD_C from the current frame.

$$MV_N = \frac{TD_N \times MV_C}{TD_C} \tag{3.13}$$

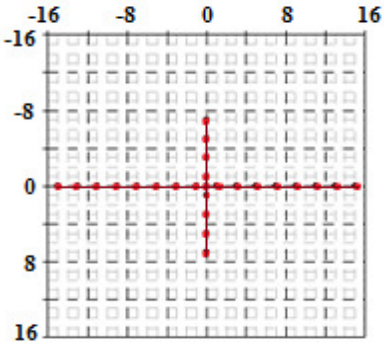
3.6.1.2 Adaptive Early Termination

The threshold cost value for early termination used in EPZS algorithm is adaptive to the costs obtained for previously encoded neighbouring blocks (of same block size) of current block. Initially, a threshold value T_1 (which is equal to number of pixels of the current block type) is used. Then for sub-sequent blocks, the threshold T_2 is changed according to (3.14), where 'a' and 'b' are constants (with a=1.1 and b=T_1) and $minJ_1$, $minJ_2$… $minJ_n$ are minimum cost values obtained for previous blocks.

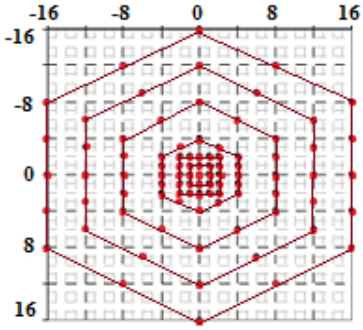
$$T_2 = a \times min(minJ_1, minJ_2, \ldots minJ_n) + b \tag{3.14}$$

3.6.1.3 Motion Vector Refinement

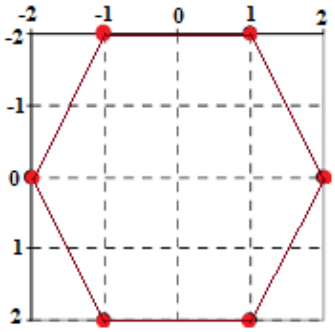
If the early termination condition is satisfied for any predicted point, then the predicted point is taken as final MV. Otherwise, the algorithm searches for optimal MV around the least cost point in predictors set, using LDSP pattern explained in Section 3.4.4.3, Fig. 3.5 (b). If block size is smaller than 8x8, then the algorithm searches using square patterns shown in Fig. 3.13. Further, to get a better MV even after the termination condition is met, the algorithm searches using a SDSP pattern shown in Fig. 3.13 (c).



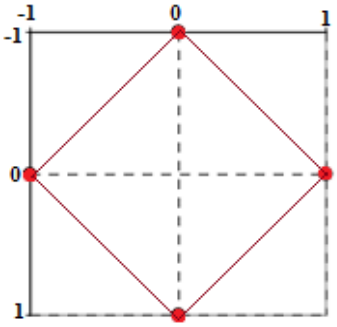
(a) Unsymmetrical-cross Pattern



(b) Uneven Multi-hexagon Grid Pattern



(c) Extended Hexagon Pattern



(d) Small Diamond Search Pattern

Fig. 3.13 Various Search Patterns used in UMHexS Algorithm

3.6.2 UMHEXS (UNSYMMETRICAL-CROSS MULTI HEXAGON-GRID SEARCH) Algorithm

The UMHexS [87, 44] Search algorithm aims to use efficient search pattern schemes and reduce the complexity. The algorithm starts with a prediction step for finding the best predicted search point. The prediction algorithms used are median predictor, up-layer predictor and collocated block predictors. In the second step, the UMHexS algorithm uses two advanced pattern schemes – unsymmetrical cross search and uneven multi hexagon grid search, shown in Fig. 3.13 (a) and (b). These patterns are found very efficient in searching the least cost point without getting trapped in local minima [87], and hence is the name of the algorithm. Though the UMHexS is mainly based on these advanced search patterns, it also employs early termination criteria after the prediction stage. Hence, if the early termination criterion is not met, then the algorithm uses the search patterns and if it is met, the algorithm terminates. The threshold values taken to terminate the algorithm are based on the costs of neighboring blocks (of same size) for current block. Further the threshold for early termination is multiplied by modulating factor β which depends on QP. In the final stage, the algorithm goes for a final

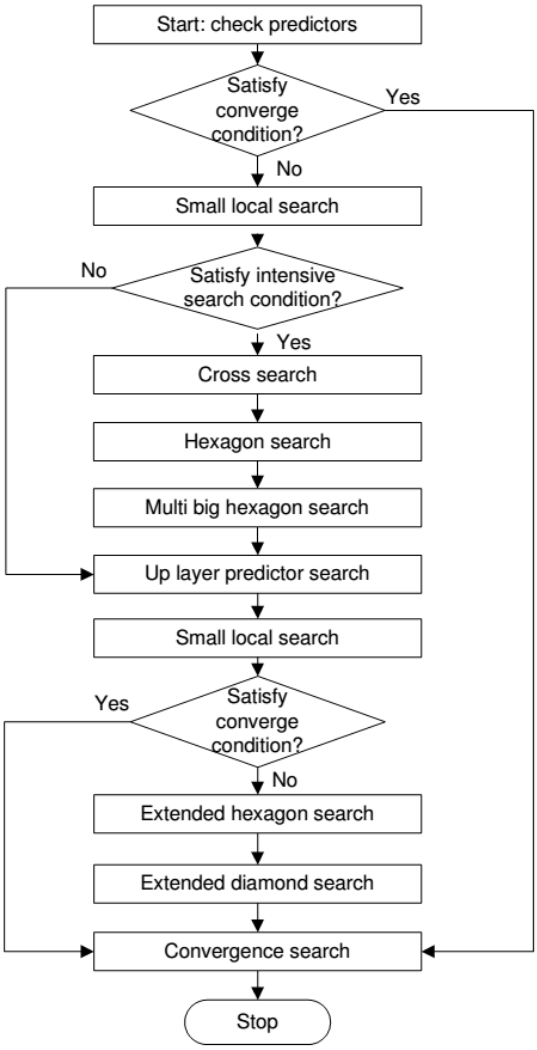
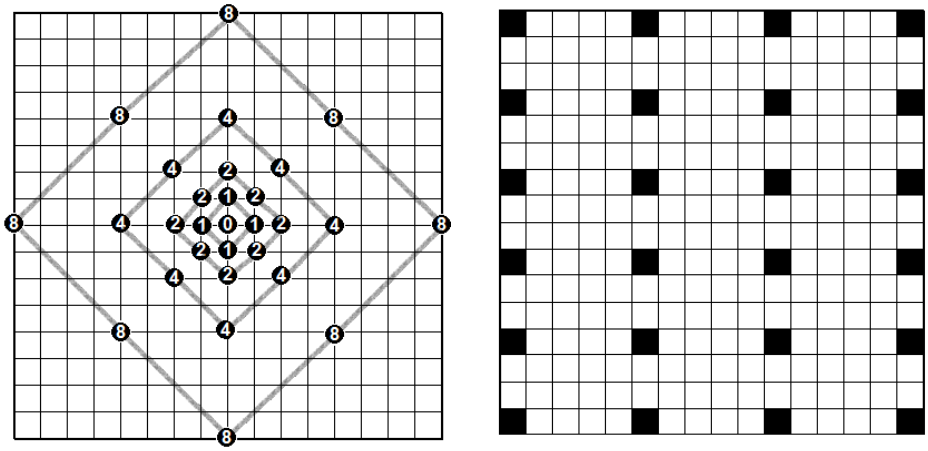


Fig. 3.14 Flowchart of SUMHexS Algorithm

refinement based on two patterns extended hexagon based search and Small Diamond Search (SDSP) as shown in Fig. 3.13 (c) and Fig. 3.13 (d) respectively.

3.6.3 SUMHEXS (SIMPLE UMHEXS) Algorithm

The SUMHexS algorithm is similar to UMHexS, except for a few modifications [104, 92], to make the algorithm simple. The temporal predictors in UMHexS algorithm consume much memory due to variable block sizes and multiple reference frames. Hence the temporal predictors are removed and only spatial median and up-layer predictors are used in SUMHexS algorithm. The complicated early termination



(a) (b)

Fig. 3.15 Search Patterns used in TZSearch Algorithm (a) Diamond Grid Patterns (b) Raster Search Pattern

condition in UMHexS algorithm is replaced with simple convergence and intensive search conditions. The detailed flowchart of the algorithm is shown in Fig. 3.14.

3.6.4 TZSearch (TEST ZONE SEARCH) Algorithm

The TZSearch algorithm is used in encoder of HEVC reference software HM [7, 8]. It combines the concept of grid patterns (used in UMHexS and SUMHexS algorithms) and search area sampling technique (explained in Section 3.4.2) which is also termed as raster search. It consists of four stages:

1. ***Motion Vector Prediction***: TZS algorithm employs median predictor, left predictor, up predictor and upper right predictor. The minimum of these predictors is selected as a starting location for further search steps.

2. ***Initial Grid Search***: In this step, the algorithm searches the search window in diamond or square patterns with different stride lengths ranging from 1 through 64, in multiples of 2. Each grid pattern contains 8 search points. The patterns used are either 8-point diamond search or 8-point square search that can be selected. A sample diamond grid pattern with stride length 8 is shown in Fig. 3.15 (a). The motion vector with minimum cost is taken as the centre search point for further steps. The stride length for this minimum distortion point is stored in variable ‘uiBestDistance’.

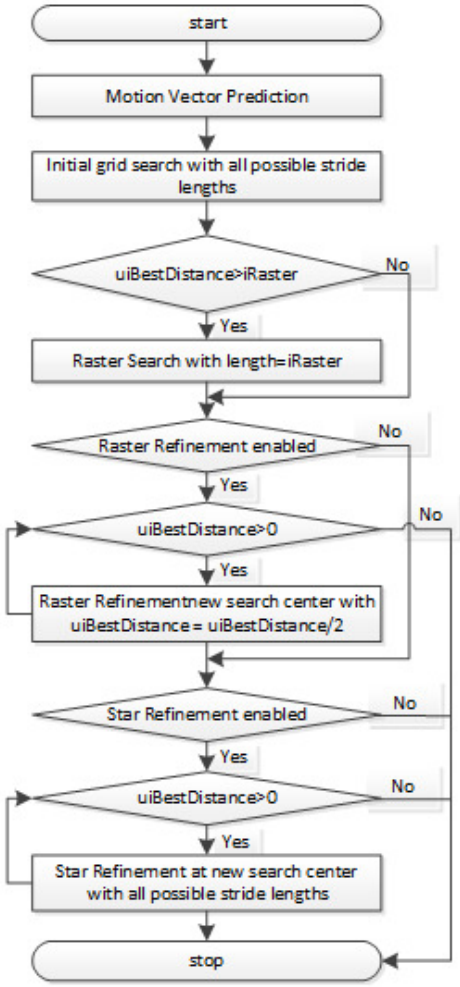


Fig. 3.16 Flowchart of TZSearch Motion Estimation Algorithm

3. ***Raster Search***: As mentioned, the raster search is a simple full-search on a down-sampled version of the search window. A predefined value 'iRaster' for raster scan is set (with default value '5') before compilation of the code [7]. This value is used as a sampling factor for search window. The search window (for 16x16 search window) for raster scan with *iRaster* value '5' is shown in Fig. 3.15 (b). As shown in flowchart of Fig. 3.16, the condition for performing this raster search is that *uiBestDistance* (obtained from previous step) must be greater than *iRaster*. If this condition is not satisfied, the algorithm will skip this step. If this step is processed, then *uiBestDistance* is changed to *iRaster* value.

4. ***Raster/Star Refinement***: This step is a fine refinement of the motion vectors obtained from the previous step. As shown in flowchart in Fig. 3.16, either raster

refinement or the square/diamond (star refinement) pattern refinement can be enabled. In general, only one of the refinement methods is enabled for fast computation. In both of these refinements, either 8-point square pattern or 8-point diamond pattern is used. The two refinement methods differ in their search operation. The raster-refinement will search by down-scaling the *uiBestDistance* value (obtained from raster search) by 2 in every step of the loop, till *uiBestDistance* equals to zero. The star refinement is similar to step 2 except for change in starting search location. The whole refinement process will only start if *uiBestDistance* is greater than zero. After every loop, the new stride length is stored in variable *uiBestDistance*. The loop breaks (or the search stops) when *uiBestDistance* equals to zero, which means that the obtained search point is the optimal MV.

3.7 MOTION ESTIMATION ARCHITECTURES

The design metrics used to evaluate performance of a system in software applications are different from that of hardware implementation. Typically, the complexity analysis in software is measured in terms of number of search operations (for motion estimation) or total motion estimation time taken. But in hardware, this design metric is defined considering the I/O bandwidth and silicon area also, apart from processing speed. So, the algorithm verified in software is only for functionality and the algorithm should be properly modified with these design considerations for designing the suitable architecture of the motion estimation accelerator module.

Fig. 3.17 shows a basic video encoder unit with motion estimation accelerator interfacing with the system buses. Either the motion estimation unit can be embedded into total encoder system, with encoder being as a single module, or the motion estimation module can be separated as a single unit with advanced configurations. The present proposal considers designing the motion estimation accelerator as a separate module with its interfaces to the SOC buses.

3.7.1 Classification of Motion Estimation Architectures

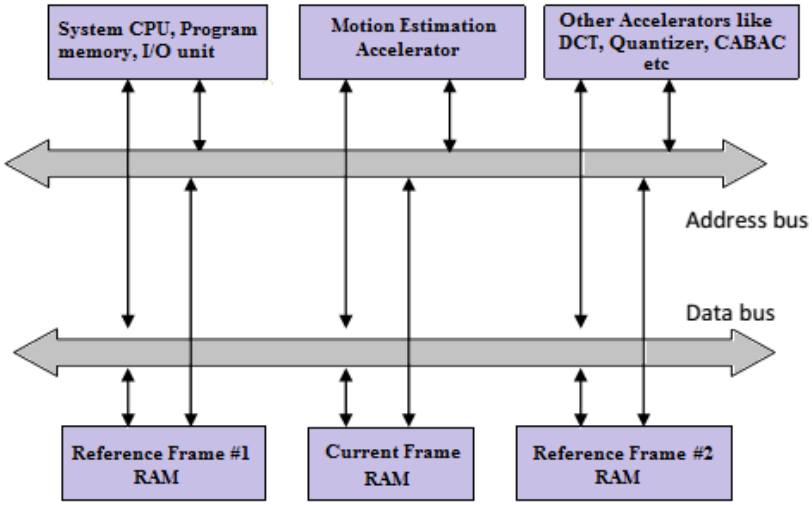


Fig. 3.17 System-on-Chip Architecture of Video Encoder with Motion Estimation Accelerator

There are many ways of designing architecture for ME module. Broadly classifying there are two type of architectures, one is to design a state machine that is suitable for only one algorithm (algorithm specific architecture explained in Section 3.7.4) and the other type is to add some programmability to the architecture (programmable architectures explained in Section 3.7.6), so that the end user can configure the algorithm and architecture based on the video coding requirements. Some architectures are designed for more than one pre-defined algorithm without much programmability and the end user can select/configure one from these pre-defined algorithms. These are

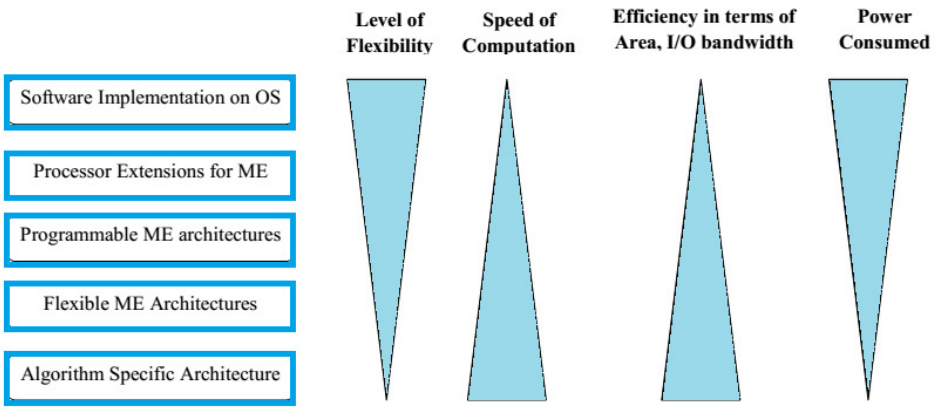


Fig. 3.18 General Classification of Motion Estimation Architectures

called flexible or configurable architectures (explained in Section 3.7.5).

Another important type for accelerating the ME process is by adding ME specific instructions to the general purpose CPUs (explained in Section 3.7.6). The processor extensions may accelerate the ME process but are not much efficient compared to real time encoding. Each of these techniques with some of the recent works are explained in the following sub-sections. As we observe from algorithm specific architectures to programmable architectures, the flexibility of handling more rates of video data (like fast motion, intermediate motion and slow motion videos) increases, but at the cost of reduced performance and increased complexity and hardware costs (like power consumed, area occupied etc.). Fig. 3.18 shows the general classification of ME architectures with their performance and cost comparisons. In the figure, the tip of the triangle indicates the lowest value while the base of the triangle indicates the highest value.

3.7.2 Design Considerations of Motion Estimation

In the hardware implementation stage the main limitations is the amount of memory required to process and the clock speed of the embedded processor that executes the motion estimation core. The maximum clock speed constraint can be met, by using a highly parallel architecture, which can reduce the total motion estimation time. Due to high requirement of the memory, the data that is required to travel from the motion estimation core and the processor also increases. This is a main limitation of the hardware bus architecture, which cannot be increased arbitrarily. To reduce the data traffic in the data bus, the ME architecture design employs data-reuse schemes, since there are lot of overlapping pixels between consecutive search windows and reference blocks. The Variable Block Size Motion Estimation (VBSME) for H.264/AVC and HEVC standard impels the motion estimation process to compute the motion vectors in various different modes for each coding macro block (4x4 to 16x16 in H.264/AVC and 4x4 to 64x64 in HEVC). By using this VBSME, there is a huge improvement in Rate-Distortion Performance, but it makes motion estimation implementation highly computational intensive and hardware expensive. Hence, for real time applications, the ME architectures are composed of PE (Processing Element) arrays, where each PE is responsible for calculating the SAD between one current block pixel and candidate block pixel.

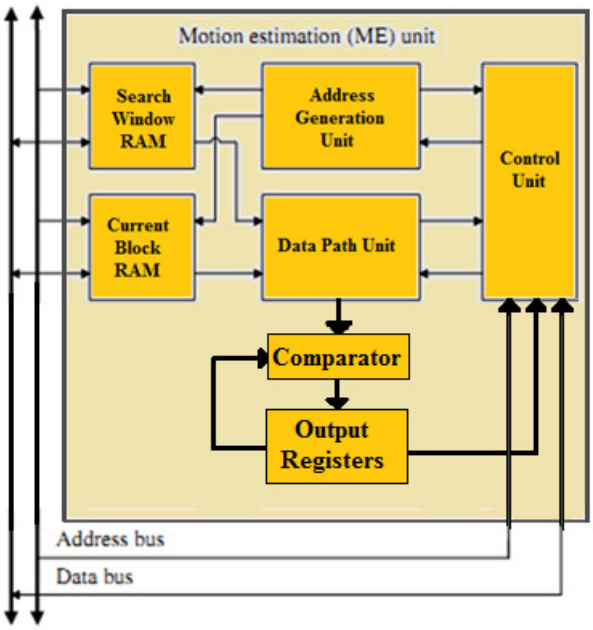


Fig. 3.19 Internal Architecture of Motion Estimation Module

3.7.3 Internal Architecture of Motion Estimation Accelerator

Fig. 3.19 shows a typical internal architecture of motion estimation module. The structure consists of a search area buffer, and the current area buffer, a data path unit (DPU) module (for implementing Processing element array and optimization logic), and an address generation unit (AGU) and control unit. The entire ME module is controlled using a state machine in control unit, with addresses of search points in the search window stored in AGU using a program counter. When the control unit triggers the AGU, the AGU sends the address to the SW RAM which in turn sends the reference block data to the DPU. The DPU receives the reference block data along with the current block data (from current block RAM) and calculates the SAD and rate-distortion cost. The costs are compared using a comparator module, and the least distortion cost along with its MV address is stored in final register

3.7.4 Algorithm Specific Architectures

As mentioned, algorithm specific architectures are optimized for one single algorithm and usually provide high throughput and low VLSI costs (in terms of area, power etc.). But these kind of architectures are not flexible enough to adapt the architectures to the changing VLSI demands.

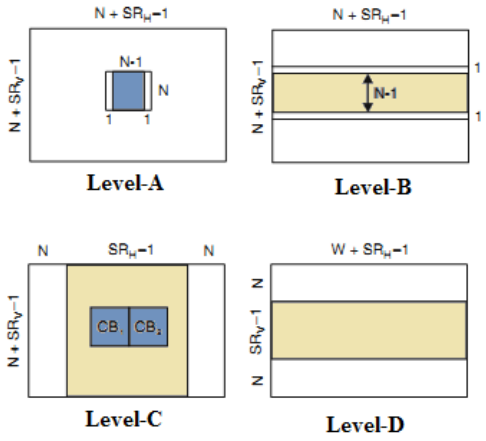


Fig. 3.20 Data Reuse schemes for Full Search Motion Estimation Architectures

3.7.4.1 Full Search Algorithm Architectures

Full search algorithm has many advantages over fast search in VLSI implementations like regular structures, simple control overhead, and highest PSNR over fast search algorithms. As mentioned, the basic element of a motion estimation architecture is the Processing Element (PE) block, which is responsible for calculating the SAD between one current and one reference pixel. There are many architectures implemented for full search algorithm in the video coding history, and can be broadly classified into two categories based on their topology of PE arrays: 1D array architectures and 2D architectures. These architecture implementations has two basic requirements, one is to reduce computational overhead by designing an appropriate parallel architecture and the other is to reduce data traffic by reusing the overlapping pixels in the consecutive reference blocks of search window (and in between consecutive search windows).

In [105], the authors Huang et. al. proposed a 2D Motion Estimation architecture. It consists of 4x4 arrays of 4x4PEs with 1D data broadcasting and 1D partial data reuse scheme. The reference pixels are sent simultaneously to PEs through multiplexers, and the initial reference pixels in a 4x4 block are propagated through delay elements. Totally it takes 4 clock cycles to calculate one SAD.

In [106], the authors propose data reuse schemes for full search ME architecture which are categorized into 4 levels of schemes – level-A, level-B, level-C and level-D. The level-A scheme reuses the overlapped reference pixels between adjacent reference

blocks of one current macro block. The level-B scheme reuses overlapped reference pixels of adjacent horizontal block strips of one current macroblock. The level-C reuses for adjacent search windows of one current macroblock and level-D for entire adjacent search window strips. Fig. 3.20 shows these four level data reuse schemes, where N denotes macroblock size, W denotes frame width, SRV denotes the vertical search range and SRH denotes horizontal search range. The level-A scheme has the smallest local memory requirement but highest off-chip memory traffic, while the level-D has the smallest off-chip memory and largest local memory. Level-C has a balance between the local memory and off chip memory data traffic and hence recommended in most designs.

In [107], the authors Chen et. al. proposed 2D array architecture for motion estimation. The architecture is a multiple 8-candidate parallel architecture and uses a shared reference buffer. Each array consists of a 16x16 PE and computes 41 SADs of a reference block in every cycle. In the architecture, eight horizontally adjacent reference blocks are processed in parallel. By using this multiple candidate data reuse scheme, the design considerably reduced on-chip memory traffic and power-consumption.

In [108], the authors Kao et. al. proposed a memory efficient and highly parallel variable block size ME architecture for full search. The architecture consists of 16 2-D arrays and each array consists of 16x16 processing elements (PEs). In the architecture four groups of 2D array PEs perform block matching for four current blocks in a pipelined fashion. To reduce memory access the architecture uses memory data reuse scheme by taking advantage of overlapping pixels and saves 98% of on-chip memory access compared to level-C data reuse scheme proposed in [106] and with only 25% of local memory overhead. The architecture was synthesized into a TSMC 180-nm CMOS cell library and is capable of processing full HD at 30 fps video when running at 130 MHz.

3.7.4.2 Fast Search Algorithm Architectures

There are many architectures that were proposed for fast ME algorithms [109]-[129]. Each of the fast architectures typically modifies the basic search pattern in the algorithm according to the hardware constraints. For instance, if the search points in the search

pattern are irregular, some of them are removed to make memory access simple and the obtained algorithm is designed and verified for RD performance [109, 110].

In [111], the authors Porto et. al. proposed architecture for diamond search algorithm. In the architecture, the nine candidate blocks of the LDSP pattern are sent to nine processing units in parallel where computation for SADs is done and then a comparator compares the SADs. If the minimum cost point obtained is at the centre of the pattern, then a final search with SDSP is performed. The architecture can perform ME for blocks of size 16x16 in a maximum search range of ±100. On an average, the architecture was reported to process 120 full HD frames per second at a maximum clock frequency of 185.7 MHz.

In [110, 112, 109], the authors Ndili et.al proposed an architecture for hardware oriented, modified diamond search algorithm. The algorithm had a better RD performance and is comparable to full search algorithm and has high speed up gain compared to full search algorithm. The architecture is verified and prototyped on an FPGA, and supports encoding of QCIF to HD (720p) sized frames in all variable block sizes of H.264/AVC (4x4 to 16x16) and with a search range of ±16. The architecture can process four 16x16 blocks in parallel and uses SAD adder tress architecture to compute variable block size SADs. The maximum operating clock frequency of the architecture obtained was 246.5 MHz.

In [113], the authors Ndili et.al proposed architecture for modified SUMHexS algorithm. The SUMHexS algorithm was changed with hardware oriented modifications. The architecture uses SAD adder tress architecture to compute variable block size SADs (from 4x4 to 16x16). The architecture can process at maximum frequency of 145.2 MHz and can support full HD frames.

In [114], the authors Rahman et. al. proposed VLSI architecture based on UMHexagonS algorithm. The architecture emphasizes on providing a trade-off between gate count and throughput. It uses dual port SRAMs to store current and previous frame outside the chip and uses 6 processing units to compute SADs of six 16x16 blocks in parallel. The architecture also uses six buffers to store. The architecture required 32 bytes of memory bandwidth, and approximately 30 MHz of minimal clock speed for calculating ME with 5 reference frames and 16 pixels search range.

In [115], the authors Juri et. al. compared and proposed and improvement in the throughput of UMHexS based architectures proposed in [114, 112, 116]. The architecture uses 256 processing elements (PEs) where each PE calculates SAD for one current block pixel and one reference block pixel. The architecture uses adder tree architecture to calculate variable block size SADs from 4x4 to 16x16. The architecture improved throughput of about 91% compared to full search architecture and about 20% improvement in throughput compared to UMHexS algorithm based architecture [116].

3.7.5 Flexible and Configurable Architectures

Flexible architectures are designed to perform motion estimation using more than one single algorithm. Flexibility in the designed architecture reduces the efficiency, since they require additional logic for memory management, setting up of data paths and address generation units (AGUs). But the flexible architectures has the advantage of providing high quality and high processing speed for a mixture of slow, medium and fast moving videos.

Lee. et. al. [117] proposed an integer ME algorithm that can realize more than one fast ME algorithm and including FS algorithm. The algorithm consists of customizable search centres that can be more than one. Then the local search is performed using either full-search or gradient-descent based fast search algorithms. The architecture consists of 5 parallel search units, each able to perform for one search point. Each search unit consists of a 16x16 PE (Processing Element) array and adder trees to support variable block size ranging from 16x16 to 4x4. The authors showed that in real-time, the architecture can perform ME for a full HD frame (with 2 reference frames) at 60fps at a maximum clock frequency of 266 MHz.

In [118], the authors Xiong et. al. proposed a flexible architecture that can support full-search and three fast search algorithms – 4-step search algorithm, diamond search algorithm and hexagonal search algorithm. The architecture uses 2-D RAMs and 2-D PE array to support the fast ME algorithms. The architecture increased system throughput by up to 85.47% and decreased power consumption by up to 13.83% compared to conventional baseline ME architecture, with an area increase of up to 65.53% in worst case scenario.

In [119], the authors Verma et. al. proposed a reconfigurable ME architecture that supports full-search and fast search algorithms that use patterns including diamond, hexagon, big-hexagon and spiral. The architecture uses a 2-D hybrid PEs which can reuse reference frame blocks and the routing architecture is designed using NOC routers. The architecture also reduced the gate count up to 7x compared to its ASIC counterpart.

In [120], the authors Vanne et. al. proposed a configurable ME architecture that can support wide range of block matching algorithms including – BBGDS (Block Based Gradient Descent Search), DS (Diamond Search), CDS (Cross Diamond Search), Hexagonal Based Search (HEXBS) and TSS (Three Step Search). The architecture by maps the memory blocks to SAD unit in accordance to the search path generated from the input parameters. The architecture provided high performance than the conventional ME architecture in real time for full HD videos.

3.7.6 Programmable Architectures and Processor Extensions

Programmable ME architectures has application specific instruction sets to program any of the fast ME algorithms. Depending on the application of video compression, the fast ME algorithm parameters like search range, search pattern shapes early termination criteria and others can be programmed. The application specific instructions can also be used to exploit parallelism at search point level like computing matching criteria (SADs or SSDs).

In [121] the authors Drolapas et. al. proposed a programmable ME architecture with low hardware cost. The architecture uses a speculative execution technique which compares the current SAD value with the best SAD value discovered so far, instead of waiting for the new SAD value itself. Hence using the speculation calculator, the partially accumulated SADs are forwarded to control module which continues the program execution in parallel with SAD module. Hence the total number of effective clock cycles for search process is decreased. The architecture is programmable with an instruction set common to most of the block matching algorithms like MVFAST and PMVFAST algorithms. The architecture is implemented in an FPGA.

In [122], the authors Nunez-Yanez et. al. proposed a programmable ME processor that can process HD videos of H.264/AVC standard. The processor can be

programmed using c-like syntaxes in the developed tools to describe almost all the fast ME algorithms. These syntaxes are later compiled into custom instruction set which can then be executed by the ME processor. Further the processor architecture is scalable and configurable which is able to select the desired number of execution units and which is determined by algorithm and throughput requirements. The architecture was implemented and verified in an FPGA.

The processor extensions are architecture/assembly-language level extensions to an existing general purpose CPU ISAs (Instruction Set Architectures) for accelerating the execution of a required application. For multimedia applications, there are two types of processor extensions – Sub-word parallelism and arithmetic instruction extensions. Sub-word parallelism is typically referred as Single Instruction Multiple Data (SIMD) which exploits the parallel processing of data by splitting a high precision ALU to a number of low precision ALUs and by controlling them together. For example, a 64 bit addition is performed by adding eight 8-bit additions, and adding them again with the carry bits. There are many SIMD ISAs implemented and traded under various companies like INTEL's MMX and SSIE-2, AMD's 3DNow!, Apple's AltiVec etc [123]. Though these processor extensions perform the required instruction in parallel, the output data bytes should be consecutively addressable in the memory for every search position. Typically for the motion estimation task, the required operation is the distortion measure criteria operation like SAD (Sum of Absolute Differences) or MAD (Mean of Absolute Differences). Some ISAs like SPARC's VIS (Visual Instruction Set), DEC's Alpha MVI (Motion Video Instructions) etc. has special instruction for performing SADs [67].

In [124], the authors S.Kim et. al. proposed a Motion Estimation Specific Instruction Set (MESIP) Processor with has high data reusability. The authors introduced a novel search scan order called center biased search scan which exploits the symmetry of the search pattern and reduce redundant data loading on MESIP by about 26.9% and 16.1% compared with raster scan and snake scan. The authors report that their architecture is suitable for low power and high performance ME implementations.

3.7.7 Motion Estimation Architectures for HEVC Standard

There are very few number of ME architectures complying HEVC standard [125]-[129], that were implemented and published, as of now. Though, for HEVC ME algorithm at search window level has the same logic with H.264/AVC at functional level, it will have a huge difference in implementation level. This is because the SAD calculation and the memory access have to perform for 64x64 blocks. Further complexity lies when the architecture supports variable block size ME with all the block sizes present in HEVC standard (from 8x4/4x8 to 64x64).

In [125], the authors Sinangil et. al. proposed a cost and coding efficient ME engine for HEVC standard. The authors analyse 11 different configurations of the ME engine by quantifying coding efficiency and hardware costs including on-chip bandwidth, off-chip bandwidth, core area and on-chip memory area. Based on their analysis one configuration is chosen and algorithm improvements are presented to further reduce hardware implementation cost of the selected configuration. The search algorithm used for the architecture is a two stage search strategy, where the first stage consists of a coarse search process performed by sub-sampling the SW and in second the stage, the algorithm performs a localised 3-step search by using TSS algorithm (explained in Section 3.4.4). It is reported, that in overall, the architecture provided 56x on-chip bandwidth, 151x off-chip bandwidth, 4.3x core area and 4.5x on-chip memory area savings when compared to the hardware implementation of the HM-3.0 encoder design.

In [126], the authors Jou et. al. proposed an architecture for fast ME algorithm complying HEVC. To reduce complexity of ME process, the authors proposed a joint algorithm and architecture optimization, with a predictive integer ME (IME) algorithm to select the most probable search directions and steps through a statistical analysis which reduced the number of search points by 90.5%. The architecture uses a 16x16 processing unit to compute the partial matching cost of all PUs with the same 16x16 current block in an interlaced order and share their common reference block to reduce the on-chip buffer size and off-chip memory bandwidth. The bandwidth is further reduced by a cache with double Z-scan indexed addressing to simplify the cache controller. The architecture was reported with implementation in TSMC 90-nm CMOS process and to support real-time encoding of 4Kx2K (QFHD) at 60 frames per second operated at 270 MHz with 778.7K logic gates and 17.4 KB of on-chip memory.

In [127], the authors Byun et. al. proposes a full search ME architecture for HEVC standard which supports asymmetric motion-partitioning (AMP) mode. In the architecture, two new structures, one for a memory read controller and the other for sum of absolute difference (SAD) summation block were proposed. The memory read controller reduces the internal memory read time, and the SAD summation block structure supports the recursive quad-tree coding unit structure and the asymmetric motion-partitioning mode. It is reported that the proposed design was implemented in Verilog HDL and synthesised using the 65 nm CMOS technology and the obtained gate count is 3.56 M with an internal static random access memory of about 20 kbyte. The maximum operation frequency obtained was 250 MHz for a 4K-Ultra high definition (UHD) (3840 × 2160P at 30 Hz) sized video.

In [128], the authors Sanchez et. al. proposed a hardware friendly Multi Point Diamond Search (MPDS) ME algorithm for HEVC with its low power hardware architecture. The MPDS algorithm was implemented in HEVC reference software and its efficiency is compared with the standard HEVC fast algorithm. The evaluation result, in average, shows loses of only 1.7% in the compression rate and 0.05% in PSNR. The main advantage of the MPDS algorithm is its hardware friendly aspect and hence the authors present its hardware design with focus on real time processing HD 1080p videos. It was reported that the designed architecture is synthesized for TSCM 90nm technology and is capable to process HD 1080p videos in real time at 30 frames per second with an operational frequency of only 41.3 MHz and maintains a good trade-off among quality, compression rate and hardware costs.

In [129], the authors Xu et. al. proposed a high performance VLSI architecture for integer motion estimation in HEVC. The architecture supports coding tree block (CTB) structure with the AMP mode. The architecture contains two parallel sub-architectures to meet 1080p at 30fps real-time video coding. The size CTB in the architecture is set to 32x32 pixels by default, and it can be extended 64x64 pixels. A serial mode decision module to find optimal partition mode for the architecture was also implemented. The architecture was designed for full search algorithm with level-D data reuse scheme (explained in Section 3.7.4) [106].

4 PROPOSED MOTION ESTIMATION ALGORITHM

4.1 INTRODUCTION

The proposed algorithm in the thesis consists of novel techniques for each of the motion estimation tools explained in Section 3.5 of Chapter 3. The enhanced tools are then integrated to form a hybrid fast ME algorithm. This algorithm is then compared with the fast search algorithm (TZSearch) included in the HEVC reference software HM [130]. Each of these enhanced techniques is explained in the following sections.

4.2 PROPOSED DYNAMIC SEARCH RANGE ALGORITHM

The proposed algorithm reduces search window size using the search range of the previously coded blocks. This is done by calculating the Euclidean distance between the search center and the collocated MV as shown in (4.1). The actual search center for the SW is the Predicted MV (PMV) and the coordinates of PMV and the final MV are taken with the co-located block as origin (with coordinates (0,0)). This is illustrated in Fig. 4.1. The difference between PMV and MV_{opt} is the Motion Vector Difference (MVD) which is shown in (4.2). The new origin of the MVD is shifted to the PMV from the co-located block resembling the actual search range of the previously coded neighboring blocks. Hence the euclidean length of the MVDs of previously coded blocks is taken for SW size prediction. For predicting the search range, three sets of prediction points are taken into consideration - spatial predictors, up-layer predictors and temporal predictors.

$$r = \sqrt{(x_p)^2 + (y_p)^2} \tag{4.1}$$

$$MVD(x, y) = PMV(x_p, y_p) - MV_{opt}(x_o, y_o) \tag{4.2}$$

4.2.1 Spatial Predictors

The MV of the current block has a probability to be near to the MVs of already encoded spatially neighboring blocks. This is due to fact that the moving object in the current block could be shared with one or many of the encoded neighboring blocks. This means the search range and direction for the current block can be predicted using the MVDs of

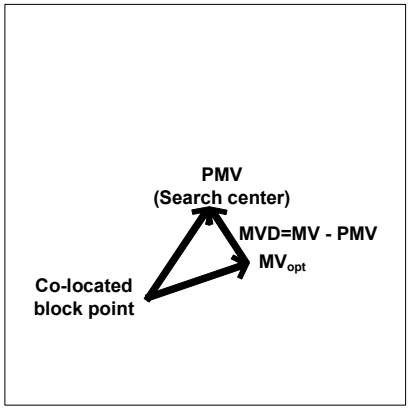


Fig. 4.1 Illustration of Relation between Predicted Motion Vector, Co-located Motion Vector and Optimal Motion Vector.

these predictors. The spatial predictors considered for search range prediction are left predictor, up, up-right and up-left predictors. The maximum value of these MVDs is calculated and used for SR prediction as formulated in (4.3), where L, Up, UR, represents the MVD co-ordinates of the spatial left-down, up-left and up-right blocks.

$$DSR_{spat} = max\left\{(x_L^2 + y_L^2)^{1/2}, \left(x_{Up}^2 + y_{Up}^2\right)^{1/2}, (x_{UR}^2 + y_{UR}^2)^{1/2}\right\} \qquad (4.3)$$

4.2.2 Upper Mode Block Predictors

For any current block in HEVC, the ME starts from the upper mode block size to the lower mode block sizes. Since the upper mode is already encoded, its MVD can be used to predict the search range of the current block. In HEVC there are seven inter-prediction mode block sizes for each partition depth as shown in TABLE 4.1. For a

TABLE 4.1: PROPOSED UP-LAYER PREDICTION MODES FOR HEVC (d = 0,1,2,3; N = 32,16,8,4; n=N/2;)

Current depth	Current partition size	Sub-partition Size and Mode	Upper mode depth	Upper mode partition size
d	2Nx2N	2Nx2N	d-1, d>0	min(2NxN, Nx2N), d>0
		2NxnD	D	2Nx2N
		2NxN		
		2NxnU		
		nRx2N		
		Nx2N		
		nLx2N		

block with maximum size of 64x64 there are four depths (d=0 to 3) and for the last partition depth (d=4 and block size 4x4) there is no further splitting. For rectangular blocks, their immediate upper mode square block MVD is taken for SR prediction. For square blocks, the MVDs of two immediate upper mode rectangular blocks are considered. The maximum value of these two MVDs is taken and used for predicting the SR of the current block.

4.2.3 Temporal Predictors

The accuracy of spatial predictors may decrease for sequences with fast moving objects. The spatial predictors are also erroneous if the moving objects in the neighboring blocks share different boundaries from that of the current block. In such cases, the correlation of objects in the temporal domain can be taken into account. Unlike spatial predictors, the temporal predictors depend on search window reference frame index. Furthermore, the MV gives better information about the search range than the MVD (of co-located block), as the MVD itself depends on spatial predictors of the collocated block. Hence, the proposed approach takes the MV of collocated block and uses its reference frame index to predict the search range.

4.2.4 Dynamic Search Range Prediction Algorithm

The present paper uses the aforementioned predictors and defines the search range. The maximum value of the Euclidean radius of all the predicted points is taken as the new Search Range (SR). The detailed steps of the proposed algorithm are:

Step 1: Take the MVDs of the spatial neighboring blocks and calculate their euclidean radius. The maximum value is taken as the predicted search range due to spatial predictors as shown in (4.4).

$$DSR_{spat} = max\left\{(x_L^2 + y_L^2)^{1/2}, \left(x_{Up}^2 + y_{Up}^2\right)^{1/2}, (x_{UR}^2 + y_{UR}^2)^{1/2}\right\} \qquad (4.4)$$

Step 2: Take the MVD of upper mode block and calculate its Euclidean radius. This is taken as the predicted search range due to upper mode block as shown in (4.5).

$$DSR_{UM} = (x_{UM}^2 + y_{UM}^2)^{1/2} \tag{4.5}$$

Step 3: Take the MV of the collocated block and calculate its euclidean radius. Divide it by the reference frame number as shown in (4.6).

$$DSR_{temp} = \frac{(x_{col}^2 + y_{col}^2)^{1/2}}{ref\_frame\_number} \tag{4.6}$$

Step 5: Take the maximum of all the above predicted DSR values as shown in (4.7).

$$DSR_{pred} = max\{DSR_{spat}, DSR_{UM}, DSR_{temp}\} \tag{4.7}$$

Step 6: Multiply the obtained search range obtained with (4.7) by the factor 2 to get the DSR value. The final DSR value is the clipped version of the previous result as shown in (4.8)

$$DSR = 2 \times DSR_{Pred} \begin{cases} \geq \dfrac{input\_search\_range}{4} \\ \leq input\_search\_range \end{cases} \tag{4.8}$$

The proposed DSR algorithm was compared with the TZSearch algorithm of the HEVC reference software HM. TABLE 4.2 shows the summary of simulation results. The test conditions for the simulation are shown in Table 2.3 of Section 2.7. The results show that on an average, there is 47.2% decrease in ME time (ΔT) or 48.5% reduction in search points (ΔN), when compared to TZSearch algorithm, with negligible reduction in PSNR (0.004 dB average) and negligible increase in bitrate (0.12 %). The variation (in time savings and complexity) between various test sequences is due to the fact that some sequences have high motion content while some have low.

TABLE 4.2: SIMULATION RESULTS FOR PROPOSED DSR ALGORITHM

Class	Sequence	ΔT (%)	ΔN (%)	ΔPSNR (dB)	Δbit-rate (%)
B (1920x1080)	Kimono	60.4	62.8	0.005	-0.13
	ParkScene	43.7	47.1	0.001	-0.08
	Cactus	56.5	57.5	0.003	-0.31
	BasktbalDrve	32.7	33.6	0.002	-0.05
	BQTerrace	65.1	68.7	0.003	-0.09
C (832x480)	RaceHorses	63.4	69.8	0.001	-0.14
	BQMall	51.3	53.8	0.003	-0.11
	PartyScene	48.8	50.0	0.006	-0.02
	BasktbalDril	61.4	64.2	0.012	-0.36
D (416x240)	RaceHorses	64.3	67.6	0.007	-0.25
	BQSquare	17.6	17.8	0.001	-0.19
	BlwngBubles	38.5	38.5	0.006	-0.04
	BasktbalPass	62.5	65.5	0.007	-0.04
E (1280x720)	FourPeople	32.8	30.9	0.002	-0.07
	Johnny	20.0	17.0	0.004	-0.05
	KristnAndSar	37.5	32.4	0.002	-0.06
Average		**47.2**	**48.5**	**0.004**	**-0.12**

4.3 PROPOSED INITIAL SEARCH POINT PREDICTION ALGORITHM

The proposed algorithm uses a median predictor and the aforementioned spatial, temporal, upper-mode predictors for finding the initial search point. The median predictor is calculated by taking the average of the spatial left, up and upper right coordinates. The cost of each predictor is calculated. The predictor with least cost is taken as the best starting point for ME process.

A set of predicted MV points called Initial Search Points set (ISP set) is used to store the initial search points. Initially, the least cost point is added to this set. Though the least cost point can lead to fast convergence when compared with other predictors, there is also a possibility that the other predictors may finally converge to a better MV. Hence, for each of the predicted points, the Euclidean distance from the least cost point is calculated. If this distance is greater than a threshold value (taken as 16) then the local minima of the predicted point is calculated and added to the ISP set. The least cost point in the ISP set is taken as the final ISP (Initial Search Point) for the ME process. The complete ISP algorithm is depicted in Fig. 4.2.

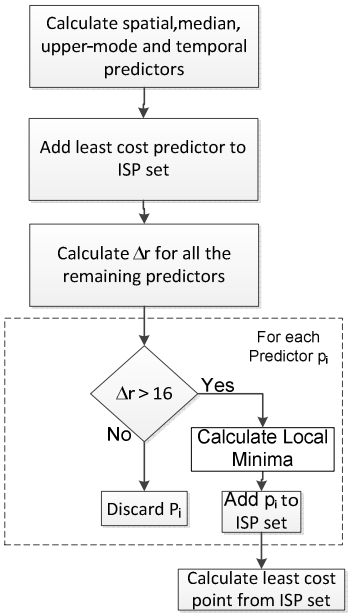


Fig. 4.2 Flowchart of proposed ISP algorithm

TABLE 4.3: SIMULATION RESULTS FOR PROPOSED ISP ALGORITHM

Class	Sequence	ΔT (%)	ΔN (%)	ΔPSNR (dB)	Δbit-rate (%)
B (1920x1080)	Kimono	18.2	18.6	0.000	-0.07
	ParkScene	15.9	15.4	-0.001	0.01
	Cactus	14.1	15.0	0.000	-0.01
	BasktbalDrve	8.3	8.3	0.000	0.00
	BQTerrace	15.6	15.9	0.001	-0.03
C (832x480)	RaceHorses	17.6	18.3	0.001	-0.01
	BQMall	14.8	15.2	-0.002	-0.09
	PartyScene	14.2	15.3	0.001	0.03
	BasktbalDril	17.3	17.5	0.003	-0.07
D (416x240)	RaceHorses	20.7	20.8	0.005	-0.07
	BQSquare	2.8	4.4	0.000	0.02
	BlwngBubles	14.9	14.4	-0.005	-0.06
	BasktbalPass	17.1	17.2	0.003	0.00
E (1280x720)	FourPeople	10.8	11.0	-0.002	-0.07
	Johnny	7.5	7.2	0.000	0.08
	KristnAndSar	12.3	11.8	-0.003	0.02
Average		**13.9**	**14.1**	**0.001**	**-0.02**

TABLE 4.3 shows the summary of simulation results after adding the proposed ISP algorithm to the TZSearch algorithm. The simulation results show that there is on average 13.9% reduction in ME time or 14.1% reduction in search points compared with original TZSearch algorithm. The PSNR loss is 0.001 dB (average) and bitrate increase is 0.02% (average) which is negligible.

4.4 PROPOSED EARLY TERMINATION ALGORITHM

For the aforementioned spatial neighboring blocks and temporally co-located block the previous cost values are stored and the least cost in these values is taken as the threshold for early termination. If the cost between the current block and ISP is less than this threshold then this point is skipped. This is possible due to the correlation of the cost of spatial neighbors and temporal blocks (of same size) with the cost of current block. This step can reduce the complexity. Even after skipping the global search stage, fine refinement stage is carried out and hence the quality loss can be compensated. TABLE 4.4 shows the summary of simulation results after adding this algorithm to the TZSearch algorithm. The simulation results show that there is on average 8.2%

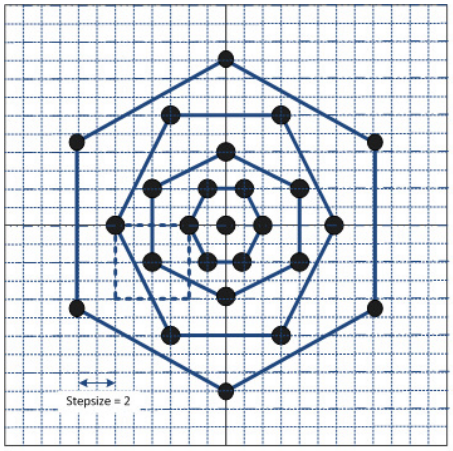


Fig. 4.3 Illustration of Rotating-hexagonal search pattern

reduction in ME time or 5.8% reduction in search points compared with TZSearch algorithm without early termination. There is negligible PSNR loss - 0.001 dB and negligible bitrate increase is 0.03%.

TABLE 4.4: SIMULATION RESULTS FOR PROPOSED GLOBAL SEARCH SKIP ALGORITHM

Class	Sequence	ΔT (%)	ΔN (%)	ΔPSNR (dB)	Δbit-rate (%)
B (1920x1080)	Kimono	2.1	4.3	0.001	-0.07
	ParkScene	5.4	5.1	0.000	-0.08
	Cactus	7.3	6.0	0.002	-0.04
	BasktbalDrve	5.1	4.3	0.000	-0.04
	BQTerrace	8.7	6.6	0.001	-0.02
C (832x480)	RaceHorses	20.3	6.9	0.000	-0.05
	BQMall	10.5	7.7	0.000	-0.06
	PartyScene	9.4	7.3	0.001	0.01
	BasktbalDril	8.8	7.1	0.003	-0.09
D (416x240)	RaceHorses	9.3	7.0	0.003	-0.06
	BQSquare	7.8	5.7	0.002	0.02
	BlwngBubles	8.6	6.4	-0.003	-0.06
	BasktbalPass	9.1	7.4	0.004	-0.03
E (1280x720)	FourPeople	6.8	4.5	0.000	-0.01
	Johnny	4.7	2.3	0.000	0.02
	KristnAndSar	7.7	3.7	-0.003	0.08
Average		**8.2**	**5.8**	**0.001**	**-0.03**

4.5 PROPOSED GRID PATTERN ALGORITHM

After the early termination stage, if the threshold condition is not satisfied, the algorithm searches for the global minimum point. This step is used in order to prevent the MV from getting trapped into local minimum. Once the coarse point is found, it can be later refined to get final the optimal MV. The parameters that affect the convergence speed and accuracy of the global search point stage are the search pattern shape and the step size. The search pattern shape is the grid pattern shape taken and the step size is the distance between successive grids. In our published work [P5], it was demonstrated that rotating hexagonal grids provide better performance than square or diamond patterns. The proposed algorithm improves the results by advancing a variable step-size for the rotating-hexagonal search pattern.

As explained in chapter 3, the fast ME algorithm used in HEVC has two types of grids, diamond and square. Both of these types of grids have 8 search points per grid. If these grids are replaced with hexagonal grids, computational time can be saved as the

hexagonal grids have only 6 points per grid. Although hexagonal grids save computation time, they might lose output video quality for vertical motion estimation as the pattern orientation favor horizontal motion. This is due to the fact that the horizontal hexagons have more horizontal search points and thus provide better estimates for horizontal moving objects. On the other hand, vertical hexagons have more vertical search points and provide better estimates for vertically moving objects, losing performance for horizontal motion. Hence, the proposed algorithm adopts a rotating hexagonal pattern for balancing performance between horizontal and vertical motion as shown in Fig. 4.3. The total number of search points for each pattern for a given search range can be computed with (4.9) and (4.10), where N_D, N_S, N_H, N_{RH} represents the number of search points for diamond, square, hexagon and rotating-hexagon patterns respectively (*floor* rounds to the least integer).

$$N_D = N_S = 8 \times floor(log_2 R) \tag{4.9}$$

$$N_H = N_{RH} = 6 \times floor(log_2 R) \tag{4.10}$$

The proposed algorithm uses variable step size for the search pattern to increase the accuracy of the global search point. Since the MV probability density is high until the stride length (radius of the search pattern) value reaches 16, the proposed algorithm uses a constant step-size of 2. After this stride length, the proposed algorithm uses a logarithmic step size with initial value 2 and proceeding to values 4, 8, 16 and so on. This is due to the center-biased nature of MVs, whose distribution is concentrated more near the origin and decreases as it moves away [131].

Though the global minimum search point algorithm is actually used to solve local minima problems, a single minimum search point may not be always enough. Some search points which have cost slightly higher than the minimum point may converge to a better optimum in a fine refinement stage. Hence, all the points in the global search stage with a cost less than a threshold cost value are identified and added to GSP (Global Search Point) set in sorted order. The cost threshold value that is considered is the minimum cost value of previously coded spatial neighboring blocks and co-located block. Then, for each point in GSP set, the Euclidean distance (Δr) and the cost difference (ΔJ) to the least cost point are calculated. Points with Δr less than 16

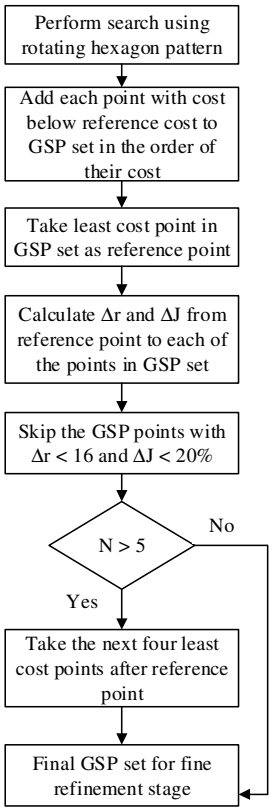


Fig. 4.4 Flowchart of the proposed grid pattern algorithm

are deleted from the GSP set. The value 16 is taken since, the fine-refinement stage in the proposed ME algorithm is performed in the reduced SW size of 16 and thus any point with a distance less than 16 from least cost point will be searched in the fine refinement stage.

Furthermore, if the number of points (N) in GSP set is too large, the complexity of the total ME algorithm increases, as the fine refinement has to be performed for each point in the GSP set. Hence, a tolerance limits for the cost difference and the total number of points are set, and the redundant points are removed from the GSP set to get a better trade-off between complexity and PSNR loss. The tolerance limit for ΔJ is set to 20% and for 'N', it is set to 5 points. Any point which has ΔJ less than 20% is first removed and then checked for the total number of remaining points in the GSP set. If there are more than 5 points, than these points are sorted in descending cost (ΔJ) order. All the points above the fifth position are removed from the GSP set. The detailed flowchart of the proposed algorithm is shown in Fig. 4.4.

TABLE 4.5: SIMULATION RESULTS FOR PROPOSED MMVSSP ALGORITHM

Class	Sequence	ΔT (%)	ΔN (%)	ΔPSNR (dB)	Δbit-rate (%)
B (1920x1080)	Kimono	12.8	15.0	0.003	-0.13
	ParkScene	16.3	17.9	0.001	-0.13
	Cactus	14.2	15.9	0.002	-0.04
	BasktbalDrve	17.7	19.6	0.000	-0.01
	BQTerrace	15.3	17.1	0.002	-0.08
C (832x480)	RaceHorses	11.9	13.7	0.003	-0.11
	BQMall	16.0	17.1	0.001	-0.10
	PartyScene	15.6	17.0	0.004	-0.08
	BasktbalDril	12.1	12.8	0.006	-0.08
D (416x240)	RaceHorses	13.2	14.2	0.010	-0.20
	BQSquare	22.6	24.1	0.001	-0.02
	BlwngBubles	17.2	17.0	0.002	-0.14
	BasktbalPass	14.4	15.5	0.003	-0.19
E (1280x720)	FourPeople	16.1	17.0	0.000	-0.09
	Johnny	16.9	18.0	0.002	-0.01
	KristnAndSar	16.9	18.3	-0.002	-0.03
Average		**15.6**	**16.9**	**0.002**	**-0.09**

The proposed algorithm is implemented with the TZSearch fast ME algorithm present in the HEVC reference software. TABLE 4.5 shows the summary of simulation results. The results show that there is 15.6% decrease in ME time or 16.9% decrease in ME search points, with negligible loss in PSNR (0.002 dB) and bitrate (0.09%).

4.6 PROPOSED FINE REFINEMENT ALGORITHM

After searching the global minimum point, the next task is to find the final optimal MV. Most often, the SAD error function does not decrease monotonically from the global minimum point. This is illustrated in Fig. 4.5, for class C RaceHorses sequence for a search window in 7th frame. The graph illustrates that there are many local minimum points. Hence, to find the optimal point, first the local search window is constructed with search range 16. Then the rotating hexagon search pattern (with constant step-size two) is used again within this local SW to find a local sub-optimal search point. After finding the local sub-optimal point, the ME algorithm starts refining monotonically using a gradient descent search based hexagonal pattern shown in Fig 4.6. In each step of the refinement, the minimum point is taken as the center point of the new hexagonal search pattern. Hence for a hexagon, there are only three new points to be searched in

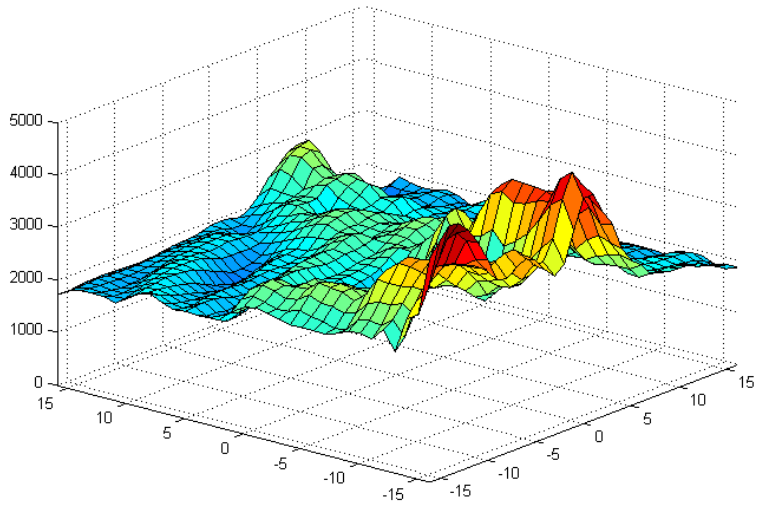


Fig. 4.5 Surface plot of ME cost for class C RaceHorses sequence

each step, unlike in a square pattern which has either three or five new points in each step. The diamond pattern also has three new points in each step. The hexagon pattern is however able to cover wider search area than the diamond and hence it converges accurately. The fine refinement stops when the minimum cost point is the center point. Then, there are ten points that were not covered by the convergence process. The proposed ME algorithm checks these last ten points around that search point, as shown by the grey points in Fig. 4.6.

The native fine refinement algorithm embedded in the TZSearch ME Algorithm was replaced by the proposed algorithm, in order to verify its performance. The summary of the simulation results are shown in TABLE 4.6. The results show that there is 15.9% reduction in ME time or 13.7% reduction with negligible loss in PSNR (0.003 dB) and negligible increase in bitrate (0.05 %), when compared to the TZSearch

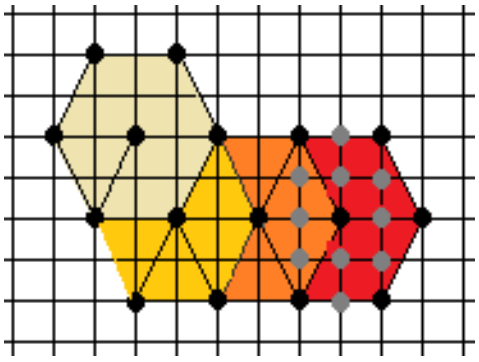


Fig. 4.6 Fine refinement patterns using hexagons

TABLE 4.6: SIMULATION RESULTS FOR PROPOSED FINE REFINEMENT ALGORITHM

Class	Sequence	ΔT (%)	ΔN (%)	ΔPSNR (dB)	Δbit-rate (%)
B (1920x1080)	Kimono	14.4	15.0	0.001	0.08
	ParkScene	19.2	16.8	0.001	0.06
	Cactus	14.5	13.1	0.002	0.01
	BasktbalDrve	15.8	13.6	0.002	0.08
	BQTerrace	15.6	14.2	0.002	0.02
C (832x480)	RaceHorses	24.1	14.5	0.001	0.04
	BQMall	17.4	15.4	0.003	0.05
	PartyScene	14.4	13.5	0.002	0.02
	BasktbalDril	13.8	12.8	0.006	0.10
D (416x240)	RaceHorses	17.7	16.0	0.003	0.01
	BQSquare	12.3	10.6	0.003	0.01
	BlwngBubles	17.4	15.6	0.002	0.08
	BasktbalPass	16.0	15.3	0.005	0.06
E (1280x720)	FourPeople	12.2	10.2	0.003	0.05
	Johnny	13.2	10.5	0.005	0.07
	KristnAndSar	15.6	12.4	0.008	0.12
Average		**15.9**	**13.7**	**0.003**	**0.05**

ME Algorithm.

4.7 SIMULATION RESULTS OF OVERALL PROPOSED ALGORITHM

The proposed algorithms were integrated to form a hybrid ME algorithm. This algorithm was implemented and tested with the HEVC reference software HM [130]. TABLE 4.7 to TABLE 4.20 show the simulation results for full search, TZSearch algorithm, and the proposed complete algorithm. TABLE 4.21 to TABLE 4.24 show the comparison results of proposed algorithm with full search and TZSearch algorithm. In each experiment, the average results of all sequences for each class are shown. ΔT, ΔN, and ΔE represents the percentage difference of total ME time, number of search points and total encoding time between the original TZSearch algorithm and the proposed algorithm, respectively. BD-PSNR and BD-Bitrate denotes the Bjontegard-Delta PSNR and Bjontegard-Delta bitrate [25] between TZSearch and the proposed algorithm. The results show that there is a 48.3% to 66.1% reduction in ME complexity or 50.3% to

65.6% reduction in ME time with negligible loss in bitrate (0.12 to 0.85 BD-bitrate) and PSNR (0.004 to 0.03 dB BD-PSNR).

TABLE 4.25 shows the summary of the comparison results with the full search algorithm and the TZSearch algorithm for all configurations. The results show that on an average there is 98.6 to 99.4% decrease in complexity or 97.1 to 98.8% reduction in ME time compared to FS algorithm. Similarly, compared to TZSearch algorithm the decrease in complexity and ME time are 34.5 to 55.7% and 32.4 to 55.1% respectively. The decrease in total encoding time compared to FS and TZSearch algorithm ranges from 78.7 to 92.1% and 10.3 to 33.9% respectively. The BD-PSNR loss and BD-bitrate compared to FS range from -0.023 to -0.055 dB and 0.53 to 0.84 % respectively. Similarly, the BD-PSNR loss and BD-bitrate compared to TZSearch range from -0.015 to -0.048 dB and 0.39 to 0.51 % respectively. Hence the results show that in any given configuration the overall gains in complexity and encoding speeds are significant with negligible loss in PSNR and bitrate. Annex A shows the RD curves of each sequence for the full search, TZSearch and the proposed algorithm. These RD curves demonstrate that the complexity reduction is achieved with negligible loss in bitrate and output video quality (PSNR).

The proposed algorithm is compared with some of the latest works of HEVC motion estimation in the literature [132] and [133]. When comparing to other papers, our proposed algorithm outperforms the related results. For instance, for BasketballPass video sequence, the proposed algorithm reached ΔT = 81.75% (QP = 27) and ΔT = 80.29% (QP = 32) whereas in [132], ΔT = 76.81% (QP = 27) and ΔT = 74.70% (QP = 32) and in [133], ΔT = 21.37% (QP = 27) and ΔT = 18.08% (QP = 32) were reached.

TABLE 4.7: SIMULATION RESULTS OF FULL SEARCH ALGORITHM FOR CLASS B AND CLASS C SEQUENCES IN LOW DELAY P MODE

Sequence	*QP*	*Bitrate (Kbps)*	*Y-PSNR (dB)*	*Total encoding time (sec)*	*Total ME Search Points*	*Total ME Time (sec)*
Kimono (class B) 1920x1080	**22**	7363.0976	42.0089	30703	3.18029E+11	29036
	27	3562.88	40.0787	29929	3.16192E+11	28584
	32	1771.616	37.3917	29352	3.15173E+11	28230
	37	892.864	34.6794	28843	3.13112E+11	27870
Park Scene (class B) 1920x1080	**22**	9616.864	39.9021	15603	1.55357E+11	14063
	27	3920.5888	37.19	14663	1.50695E+11	13512
	32	1710.4064	34.5179	14184	1.48398E+11	13227
	37	756.544	32.0279	13406	1.40301E+11	12548
Cactus (class B) 1920x1080	**22**	26892.1867	38.5293	9469	85704110312	7800
	27	7668.12	36.5113	8520	81706099384	7377
	32	3471.1067	34.4457	8117	79868816982	7179
	37	1724.1067	32.1464	8172	80965171300	7317
Basketball Drive (class B) 1920x1080	**22**	67358.704	39.0987	17181	1.66816E+11	15145
	27	15127.344	35.3152	14176	1.43992E+11	12943
	32	4595.856	33.1709	11104	1.1423E+11	10183
	37	1827.104	30.913	8772	88491644009	7951
BQ Terrace (class B) 1920x1080	**22**	14950.6667	39.8842	37039	3.92948E+11	35430
	27	5354.2133	38.4426	35882	3.88643E+11	34669
	32	2603.1733	36.7536	35123	3.84485E+11	34092
	37	1375.0267	34.7734	34806	3.78846E+11	33862
Race Horses (class C) 832x480	**22**	7089.584	39.5332	6309	63737199410	5854
	27	2901.288	35.5399	6256	64375362098	5913
	32	1320.352	32.2388	6153	64241037765	5880
	37	621.656	29.3896	6030	63845471709	5803
BQ Mall (class C) 832x480	**22**	5527.728	39.7072	1491	13130164079	1196
	27	2541.024	36.7977	1402	12873734814	1173
	32	1271.12	33.7717	1386	13156620907	1195
	37	656.48	30.7793	1340	12992349959	1171
Party Scene (class C) 832x480	**22**	13861.8267	38.2909	1500	11377168059	1048
	27	6313.2	34.0909	1417	11764317384	1086
	32	2902.0267	30.49	1415	12638747206	1160
	37	1330.2	27.313	1395	13022545748	1188
Basketball Drill (class C) 832x480	**22**	3945.8933	40.4765	2018	18637282920	1704
	27	1831	37.3603	2090	20232013917	1843
	32	884.3467	34.4784	2181	21810973597	1979
	37	458.3733	31.9658	2350	24052779670	2175

TABLE 4.8: SIMULATION RESULTS OF FULL SEARCH ALGORITHM FOR CLASS D AND CLASS E SEQUENCES IN LOW DELAY P MODE

Sequence	*QP*	*Bitrate (Kbps)*	*Y-PSNR (dB)*	*Total encoding time (sec)*	*Total ME Search Points*	*Total ME Time (sec)*
Race Horses (class D) 416x240	**22**	1700.08	39.5203	1062	10319522416	949
	27	830.52	35.3431	1065	10554651413	975
	32	404.872	31.7321	1039	10405557275	966
	37	203.112	28.7918	1009	10575393851	951
BQ Square (class D) 416x240	**22**	2921.136	38.8745	218	1389023841	124
	27	1138.736	34.6021	191	1391405615	125
	32	514.24	31.4292	177	1362340327	125
	37	244.032	28.4483	167	1431340558	126
Blowing Bubbles (class D) 416x240	**22**	2226.7867	38.399	282	2040904656	184
	27	952.08	34.8528	284	2342016121	211
	32	426.28	31.7578	254	2169090209	198
	37	194.4933	29.0058	248	2288982356	202
Basketball Pass (class D) 416x240	**22**	921.1467	41.3041	184	1442628035	126
	27	462.64	37.6572	191	1602193844	141
	32	232.2667	34.3564	196	1700333098	152
	37	119.4267	31.4207	191	1750457401	152
Four People (class E) 1280x720	**22**	3047.056	42.484	1681	14109467777	1260
	27	1311.264	40.377	1519	13066090000	1166
	32	710.704	37.8422	1462	12758729494	1135
	37	403.504	34.9432	1483	13144551438	1167
Johnny (class E) 1280x720	**22**	2976.176	42.7202	2147	18940012850	1697
	27	916.896	40.9513	1870	16920269420	1511
	32	420.496	39.0079	1649	14858222257	1320
	37	230.992	36.6697	1553	13952985209	1240
Kristen And Sara (class E) 1280x720	**22**	2842.112	43.1495	1857	15796594631	1418
	27	1136.272	41.0986	1645	14256984642	1273
	32	576.064	38.7833	1693	15133389627	1353
	37	315.408	36.1355	1739	15922624158	1416

TABLE 4.9: SIMULATION RESULTS OF FULL SEARCH ALGORITHM FOR CLASS B AND CLASS C SEQUENCES IN LOW DELAY B MODE

Sequence	*QP*	*Bitrate (Kbps)*	*Y-PSNR (dB)*	*Total encoding time (sec)*	*Total ME Search Points*	*Total ME Time (sec)*
Kimono (class B) 1920x1080	**22**	6989.7	42.1573	32125	3.29E+11	29969
	27	3459.3	40.2322	30933	3.23E+11	29117
	32	1704.5	37.5649	30061	3.18E+11	28500
	37	851.56	34.8612	29292	3.14E+11	27901
Park Scene (class B) 1920x1080	**22**	9193.2	39.9779	17748	1.73E+11	15733
	27	3826.3	37.237	16289	1.63E+11	14687
	32	1680.1	34.5439	15304	1.56E+11	13919
	37	741.39	32.0333	14596	1.5E+11	13343
Cactus (class B) 1920x1080	**22**	24399	38.6072	11436	1.02E+11	9352
	27	7398.9	36.5712	9886	9.21E+10	8354
	32	3408.6	34.4943	9524	9.1E+10	8205
	37	1700.5	32.1786	9401	9.12E+10	8188
Basketball Drive (class B) 1920x1080	**22**	56327	39.0548	19214	1.84E+11	16767
	27	12453	35.6032	15077	1.5E+11	13470
	32	3944.1	33.3929	12521	1.25E+11	11208
	37	1680.1	31.0729	10273	1.02E+11	9068
BQ Terrace (class B) 1920x1080	**22**	13976	39.9553	37575	3.94E+11	35523
	27	5141.2	38.5337	36468	3.89E+11	34811
	32	2513.5	36.8806	35691	3.85E+11	34241
	37	1340.7	34.9053	34934	3.79E+11	33600
Race Horses (class C) 832x480	**22**	6691	39.5571	6762	6.74E+10	6212
	27	2824.9	35.6457	6585	6.68E+10	6147
	32	1300.2	32.303	6383	6.56E+10	6017
	37	614.69	29.4174	6138	6.49E+10	5823
BQ Mall (class C) 832x480	**22**	5198.6	39.8052	1711	1.45E+10	1339
	27	2429.9	36.8791	1593	1.41E+10	1291
	32	1231.4	33.8592	1525	1.38E+10	1261
	37	643.52	30.8433	1475	1.38E+10	1237
Party Scene (class C) 832x480	**22**	12504	38.4023	1792	1.36E+10	1262
	27	5999.5	34.2471	1746	1.41E+10	1322
	32	2823	30.5495	1701	1.46E+10	1354
	37	1305.9	27.3367	1670	1.53E+10	1383
Basketball Drill (class C) 832x480	**22**	3688.5	40.5948	2250	2.02E+10	1857
	27	1753.9	37.4717	2343	2.21E+10	2016
	32	852.24	34.5658	2367	2.33E+10	2091
	37	443.17	32.0506	2428	2.44E+10	2182

TABLE 4.10: SIMULATION RESULTS OF FULL SEARCH ALGORITHM FOR CLASS D AND CLASS E SEQUENCES IN LOW DELAY B MODE

Sequence	*QP*	*Bitrate (Kbps)*	*Y-PSNR (dB)*	*Total encoding time (sec)*	*Total ME Search Points*	*Total ME Time (sec)*
Race Horses (class D) 416x240	**22**	1660	39.6077	1165	1.14E+10	1030
	27	824.81	35.4252	1134	1.13E+10	1022
	32	401.78	31.7454	1085	1.1E+10	991
	37	201.15	28.8194	1091	1.12E+10	1011
BQ Square (class D) 416x240	**22**	2244.9	38.9145	247	1.52E+09	137
	27	970.02	34.9272	217	1.47E+09	133
	32	472.48	31.6392	196	1.41E+09	127
	37	232.64	28.5271	184	1.4E+09	124
Blowing Bubbles (class D) 416x240	**22**	2106.5	38.479	353	2.56E+09	233
	27	924.49	34.9171	356	2.93E+09	263
	32	421.92	31.7923	340	2.95E+09	264
	37	194.92	29.0344	299	2.63E+09	234
Basketball Pass (class D) 416x240	**22**	900.19	41.3494	213	1.57E+09	138
	27	456.36	37.7131	209	1.61E+09	142
	32	230.33	34.398	203	1.64E+09	143
	37	118.79	31.398	208	1.74E+09	152
Four People (class E) 1280x720	**22**	2896.3	42.5569	1899	1.49E+10	1323
	27	1289.5	40.4316	1760	1.42E+10	1257
	32	706.13	37.8646	1676	1.37E+10	1204
	37	402.67	34.9589	1717	1.43E+10	1263
Johnny (class E) 1280x720	**22**	2587.9	42.8426	2586	2.22E+10	1977
	27	856.02	41.0844	2224	1.93E+10	1709
	32	413.2	39.1141	1972	1.69E+10	1493
	37	228.82	36.7544	1921	1.66E+10	1456
Kristen And Sara (class E) 1280x720	**22**	2701.8	43.2114	2218	1.8E+10	1609
	27	1113.3	41.1433	2066	1.73E+10	1537
	32	565.6	38.8118	2096	1.81E+10	1605
	37	314.4	36.1756	2128	1.83E+10	1643

TABLE 4.11: SIMULATION RESULTS OF TZSEARCH ALGORITHM FOR CLASS B AND CLASS C SEQUENCES IN LOW DELAY P MODE

Sequence	*QP*	*Bitrate (Kbps)*	*Y-PSNR (dB)*	*Total encoding time (sec)*	*Total ME Search Points*	*Total ME Time (sec)*
Kimono (class B) 1920x1080	**22**	7357.53	42.0075	2375	6.196E+09	736
	27	3561.786	40.076	1944	5.017E+09	624
	32	1777.504	37.3727	1607	3.86E+09	512
	37	889.792	34.6967	1350	2.824E+09	409
Park Scene (class B) 1920x1080	**22**	9622.176	39.8995	1778	2.142E+09	263
	27	3922.08	37.1851	1361	1.769E+09	228
	32	1710.906	34.5171	1130	1.446E+09	196
	37	757.9264	32.0245	996	1.188E+09	168
Cactus (class B) 1920x1080	**22**	26935.2	38.5296	1964	2.826E+09	325
	27	7678.147	36.5099	1397	2.315E+09	276
	32	3475.613	34.4353	1161	1.872E+09	236
	37	1731.107	32.1414	1029	1.493E+09	201
Basketball Drive (class B) 1920x1080	**22**	67392.53	39.0977	2267	2.321E+09	277
	27	15129.17	35.3067	1421	1.677E+09	212
	32	4596.88	33.1538	1080	1.267E+09	174
	37	1828.016	30.8963	937	997718451	142
BQ Terrace (class B) 1920x1080	**22**	14929.88	39.8812	2259	5.622E+09	688
	27	5356.493	38.4422	1739	4.263E+09	554
	32	2603.987	36.7461	1441	3.164E+09	445
	37	1376.947	34.759	1250	2.358E+09	358
Race Horses (class C) 832x480	**22**	7118.28	39.5403	632	1.554E+09	182
	27	2917.112	35.5395	496	1.31E+09	160
	32	1326.4	32.2238	403	1.038E+09	135
	37	623.6	29.3662	336	794031692	112
BQ Mall (class C) 832x480	**22**	5543.456	39.7067	341	446815190	50
	27	2549.552	36.7953	269	388602040	45
	32	1273.392	33.771	227	326834209	39
	37	656.944	30.7602	200	272688700	34
Party Scene (class C) 832x480	**22**	13890.95	38.2977	513	620446227	68
	27	6329.373	34.0949	388	545585109	62
	32	2914.333	30.4886	306	450311595	54
	37	1333.107	27.3043	250	358812891	45
Basketball Drill (class C) 832x480	**22**	3956.667	40.4824	387	670078838	75
	27	1837.333	37.3527	312	569456541	67
	32	886.6267	34.4763	257	461882086	58
	37	458.0533	31.9501	222	368540515	49

TABLE 4.12: SIMULATION RESULTS OF TZSEARCH ALGORITHM FOR CLASS D AND CLASS E SEQUENCES IN LOW DELAY P MODE

Sequence	*QP*	*Bitrate (Kbps)*	*Y-PSNR (dB)*	*Total encoding time (sec)*	*Total ME Search Points*	*Total ME Time (sec)*
Race Horses (class D) 416x240	**22**	1702.336	39.5124	147	326350758	36
	27	835.904	35.3394	120	287298379	33
	32	407.816	31.7276	99	235870365	29
	37	203.92	28.7732	81	184092883	24
BQ Square (class D) 416x240	**22**	2914.464	38.866	98	44107053	4
	27	1136.672	34.6002	69	41073161	4
	32	513.744	31.4165	53	38747966	4
	37	243.584	28.444	45	36576430	4
Blowing Bubbles (class D) 416x240	**22**	2233.107	38.4098	109	129333066	14
	27	953.9333	34.8495	84	111632416	12
	32	428.28	31.7539	65	90117509	10
	37	195.92	28.9916	54	72658664	9
Basketball Pass (class D) 416x240	**22**	924.1067	41.3012	65	74604633	8
	27	462.8667	37.6504	56	66739132	7
	32	231.1867	34.345	49	57244446	7
	37	119.2267	31.4102	44	49413123	6
Four People (class E) 1280x720	**22**	3033.488	42.4883	466	425547001	49
	27	1312.496	40.3803	389	382700843	42
	32	710.848	37.8325	361	348783802	40
	37	401.12	34.9331	345	321518005	38
Johnny (class E) 1280x720	**22**	2983.616	42.7226	497	458236419	56
	27	915.168	40.9506	398	380403494	45
	32	422.976	38.9891	362	333470300	40
	37	230.08	36.6498	345	304617434	38
Kristen And Sara (class E) 1280x720	**22**	2845.248	43.1457	501	583931233	71
	27	1133.344	41.0897	422	479206214	58
	32	574.688	38.7774	385	405412172	51
	37	316.448	36.1441	364	357662162	49

TABLE 4.13: SIMULATION RESULTS OF TZSEARCH ALGORITHM FOR CLASS B AND CLASS C SEQUENCES IN LOW DELAY B MODE

Sequence	*QP*	*Bitrate (Kbps)*	*Y-PSNR (dB)*	*Total encoding time (sec)*	*Total ME Search Points*	*Total ME Time (sec)*
Kimono (class B) 1920x1080	**22**	6987.44	42.157	2983	6.52E+09	884
	27	3455.96	40.237	2521	5.42E+09	763
	32	1701.70	37.564	2165	4.24E+09	649
	37	851.30	34.860	1907	3.14E+09	548
Park Scene (class B) 1920x1080	**22**	9197.84	39.979	2341	2.26E+09	389
	27	3833.63	37.236	1904	1.88E+09	350
	32	1682.75	34.545	1645	1.55E+09	308
	37	741.00	32.033	1525	1.29E+09	284
Cactus (class B) 1920x1080	**22**	24394.92	38.608	2474	2.97E+09	447
	27	7418.97	36.575	1877	2.41E+09	384
	32	3415.41	34.496	1624	1.95E+09	342
	37	1698.83	32.176	1509	1.55E+09	308
Basketball Drive (class B) 1920x1080	**22**	56350.30	39.051	2771	2.45E+09	404
	27	12499.15	35.592	1876	1.72E+09	318
	32	3959.44	33.369	1548	1.29E+09	276
	37	1677.97	31.045	1433	1.04E+09	247
BQ Terrace (class B) 1920x1080	**22**	13979.64	39.955	2797	5.81E+09	809
	27	5162.84	38.535	2265	4.39E+09	668
	32	2516.11	36.866	1944	3.27E+09	552
	37	1335.71	34.898	1766	2.42E+09	470
Race Horses (class C) 832x480	**22**	6703.48	39.558	736	1.54E+09	204
	27	2841.68	35.654	606	1.31E+09	182
	32	1303.71	32.283	512	1.04E+09	157
	37	615.05	29.396	439	7.94E+08	131
BQ Mall (class C) 832x480	**22**	5195.76	39.804	434	4.62E+08	73
	27	2431.76	36.884	362	4.02E+08	64
	32	1237.78	33.856	317	3.35E+08	60
	37	644.85	30.838	289	2.79E+08	54
Party Scene (class C) 832x480	**22**	12502.95	38.392	609	6.3E+08	93
	27	6012.80	34.251	495	5.53E+08	86
	32	2831.73	30.547	413	4.6E+08	74
	37	1305.48	27.324	352	3.68E+08	67
Basketball Drill (class C) 832x480	**22**	3696.37	40.595	480	6.7E+08	95
	27	1759.20	37.460	407	5.73E+08	88
	32	850.25	34.558	350	4.63E+08	78
	37	441.77	32.036	312	3.72E+08	70

TABLE 4.14: SIMULATION RESULTS OF TZSEARCH ALGORITHM FOR CLASS D AND CLASS E SEQUENCES IN LOW DELAY B MODE

Sequence	*QP*	*Bitrate (Kbps)*	*Y-PSNR (dB)*	*Total encoding time (sec)*	*Total ME Search Points*	*Total ME Time (sec)*
Race Horses (class D) 416x240	**22**	1665.18	39.605	175	3.28E+08	43
	27	830.34	35.416	147	2.89E+08	38
	32	405.15	31.749	125	2.37E+08	34
	37	200.94	28.786	107	1.86E+08	28
BQ Square (class D) 416x240	**22**	2242.69	38.912	116	44997116	10
	27	971.47	34.933	90	41624063	10
	32	472.83	31.636	76	38992282	9
	37	234.45	28.529	67	36967461	9
Blowing Bubbles (class D) 416x240	**22**	2107.79	38.479	136	1.34E+08	19
	27	930.53	34.915	110	1.15E+08	18
	32	419.92	31.767	90	93907847	15
	37	193.20	29.017	77	75976792	14
Basketball Pass (class D) 416x240	**22**	898.49	41.326	87	76637077	13
	27	458.37	37.724	77	68235827	12
	32	229.64	34.392	70	58349909	11
	37	117.92	31.409	65	50987854	11
Four People (class E) 1280x720	**22**	2894.26	42.561	663	4.45E+08	94
	27	1290.13	40.431	586	3.98E+08	89
	32	707.02	37.868	553	3.61E+08	84
	37	402.58	34.964	535	3.3E+08	81
Johnny (class E) 1280x720	**22**	2576.40	42.833	705	4.76E+08	103
	27	857.47	41.083	603	3.99E+08	93
	32	412.08	39.105	563	3.49E+08	87
	37	228.21	36.733	541	3.13E+08	82
Kristen And Sara (class E) 1280x720	**22**	2693.94	43.208	730	6.52E+08	127
	27	1111.31	41.148	635	5.17E+08	107
	32	566.00	38.810	591	4.31E+08	100
	37	315.28	36.177	562	3.72E+08	92

TABLE 4.15: SIMULATION RESULTS OF PROPOSED ALGORITHM FOR CLASS B AND CLASS C SEQUENCES IN LOW DELAY P MODE

Sequence	*QP*	*Bitrate (Kbps)*	*Y-PSNR (dB)*	*Total encoding time (sec)*	*Total ME Search Points*	*Total ME Time (sec)*
Kimono (class B) 1920x1080	**22**	7362.57	42.008	1814	1.36E+09	177
	27	3570.14	40.073	1485	1.21E+09	162
	32	1766.65	37.390	1240	1.07E+09	142
	37	890.80	34.685	1076	9.33E+08	131
Park Scene (class B) 1920x1080	**22**	9644.87	39.896	1639	9.31E+08	122
	27	3928.15	37.181	1245	8.53E+08	112
	32	1710.71	34.510	1043	7.86E+08	103
	37	756.33	32.020	933	7.28E+08	97
Cactus (class B) 1920x1080	**22**	26987.25	38.521	1766	9.07E+08	115
	27	7723.00	36.503	1240	8.24E+08	105
	32	3487.80	34.429	1035	7.69E+08	101
	37	1733.76	32.137	936	7.24E+08	92
Basketball Drive (class B) 1920x1080	**22**	67531.95	39.094	2131	9.39E+08	120
	27	15153.23	35.302	1323	8.2E+08	106
	32	4582.14	33.148	1005	7.4E+08	97
	37	1830.91	30.892	888	6.9E+08	90
BQ Terrace (class B) 1920x1080	**22**	14946.49	39.881	1734	1.22E+09	164
	27	5368.09	38.441	1330	1.05E+09	145
	32	2610.72	36.741	1128	9.17E+08	128
	37	1378.71	34.752	1010	8.18E+08	117
Race Horses (class C) 832x480	**22**	7142.51	39.541	487	2.98E+08	38
	27	2924.09	35.522	373	2.71E+08	35
	32	1330.76	32.211	300	2.39E+08	32
	37	625.83	29.357	252	2.09E+08	29
BQ Mall (class C) 832x480	**22**	5587.98	39.698	315	1.66E+08	19
	27	2560.99	36.776	248	1.56E+08	20
	32	1287.86	33.754	210	1.46E+08	17
	37	661.01	30.761	188	1.37E+08	17
Party Scene (class C) 832x480	**22**	13962.89	38.289	473	1.91E+08	24
	27	6373.64	34.083	351	1.82E+08	23
	32	2921.97	30.472	275	1.69E+08	21
	37	1333.68	27.299	225	1.54E+08	19
Basketball Drill (class C) 832x480	**22**	3978.64	40.463	337	1.94E+08	24
	27	1849.56	37.345	269	1.77E+08	22
	32	887.52	34.453	222	1.61E+08	20
	37	459.39	31.937	194	1.48E+08	19

– TABLE 4.16: SIMULATION RESULTS OF PROPOSED ALGORITHM FOR CLASS D AND CLASS E SEQUENCES IN LOW DELAY P MODE

Sequence	*QP*	*Bitrate (Kbps)*	*Y-PSNR (dB)*	*Total encoding time (sec)*	*Total ME Search Points*	*Total ME Time (sec)*
Race Horses (class D) 416x240	**22**	1713.21	39.513	119	70639388	8
	27	840.52	35.316	95	66144311	7
	32	409.26	31.698	76	59863410	7
	37	204.69	28.772	63	52756039	7
BQ Square (class D) 416x240	**22**	2941.01	38.867	97	35786878	4
	27	1145.90	34.576	68	33813930	4
	32	512.88	31.402	53	32123884	4
	37	243.54	28.416	45	30889806	4
Blowing Bubbles (class D) 416x240	**22**	2243.16	38.393	102	48990277	6
	27	959.33	34.840	77	45924563	5
	32	428.41	31.741	60	41833241	5
	37	198.04	28.994	50	37782697	4
Basketball Pass (class D) 416x240	**22**	929.28	41.289	62	34984216	4
	27	467.71	37.656	53	34061723	4
	32	233.73	34.348	47	32925241	3
	37	119.32	31.388	43	31780297	3
Four People (class E) 1280x720	**22**	3041.50	42.478	462	3E+08	37
	27	1317.97	40.369	392	2.85E+08	35
	32	712.26	37.837	365	2.76E+08	34
	37	404.38	34.945	350	2.69E+08	33
Johnny (class E) 1280x720	**22**	2994.13	42.712	490	3.23E+08	39
	27	920.90	40.934	398	2.99E+08	38
	32	419.76	38.980	365	2.82E+08	34
	37	227.81	36.650	350	2.72E+08	33
Kristen And Sara (class E) 1280x720	**22**	2850.56	43.141	484	3.31E+08	42
	27	1134.75	41.086	411	3.08E+08	39
	32	573.78	38.777	379	2.92E+08	36
	37	318.50	36.143	361	2.81E+08	36

TABLE 4.17: SIMULATION RESULTS OF PROPOSED ALGORITHM FOR CLASS B AND CLASS C SEQUENCES IN LOW DELAY B MODE

Sequence	*QP*	*Bitrate (Kbps)*	*Y-PSNR (dB)*	*Total encoding time (sec)*	*Total ME Search Points*	*Total ME Time (sec)*
Kimono (class B) 1920x1080	**22**	6992.24	42.159	2406	2.27E+09	303
	27	3458.41	40.235	2062	2.08E+09	283
	32	1703.93	37.567	1796	1.9E+09	259
	37	850.44	34.851	1606	1.74E+09	242
Park Scene (class B) 1920x1080	**22**	9210.20	39.978	2229	1.86E+09	247
	27	3834.25	37.232	1817	1.72E+09	233
	32	1685.58	34.539	1580	1.6E+09	217
	37	741.36	32.027	1449	1.52E+09	207
Cactus (class B) 1920x1080	**22**	24442.68	38.609	2268	1.58E+09	207
	27	7432.48	36.570	1699	1.35E+09	177
	32	3427.24	34.486	1483	1.27E+09	170
	37	1708.80	32.166	1368	1.2E+09	158
Basketball Drive (class B) 1920x1080	**22**	56401.20	39.050	2649	1.86E+09	247
	27	12488.70	35.592	1801	1.64E+09	219
	32	3951.87	33.369	1496	1.52E+09	202
	37	1677.86	31.038	1382	1.44E+09	193
BQ Terrace (class B) 1920x1080	**22**	13987.25	39.956	2288	2.07E+09	278
	27	5164.25	38.528	1879	1.87E+09	258
	32	2523.69	36.867	1656	1.71E+09	241
	37	1337.80	34.911	1525	1.6E+09	226
Race Horses (class C) 832x480	**22**	6733.18	39.560	606	5.01E+08	66
	27	2851.66	35.653	491	4.64E+08	63
	32	1307.35	32.267	416	4.22E+08	57
	37	618.74	29.390	364	3.82E+08	54
BQ Mall (class C) 832x480	**22**	5200.86	39.800	402	2.63E+08	34
	27	2441.06	36.873	333	2.46E+08	32
	32	1245.17	33.847	293	2.31E+08	30
	37	647.79	30.830	269	2.16E+08	27
Party Scene (class C) 832x480	**22**	12526.59	38.399	573	3.75E+08	48
	27	6037.48	34.252	463	3.55E+08	46
	32	2839.31	30.544	385	3.27E+08	42
	37	1309.99	27.327	329	2.97E+08	39
Basketball Drill (class C) 832x480	**22**	3714.40	40.597	429	2.96E+08	37
	27	1761.31	37.453	363	2.79E+08	36
	32	854.89	34.551	312	2.61E+08	34
	37	444.31	32.033	282	2.47E+08	33

TABLE 4.18: SIMULATION RESULTS OF PROPOSED ALGORITHM FOR CLASS D AND CLASS E SEQUENCES IN LOW DELAY B MODE

Sequence	***QP***	***Bitrate (Kbps)***	***Y-PSNR (dB)***	***Total encoding time (sec)***	***Total ME Search Points***	***Total ME Time (sec)***
Race Horses (class D) 416x240	**22**	1670.08	39.594	150	1.23E+08	15
	27	831.42	35.386	125	1.17E+08	15
	32	405.57	31.724	106	1.08E+08	14
	37	201.94	28.761	91	97015348	13
BQ Square (class D) 416x240	**22**	2244.66	38.913	116	74235128	9
	27	972.18	34.922	91	71254287	9
	32	472.74	31.630	77	66801454	8
	37	233.81	28.531	68	62261607	8
Blowing Bubbles (class D) 416x240	**22**	2114.56	38.475	131	95485714	11
	27	930.63	34.925	104	89160582	11
	32	418.61	31.746	85	81078083	10
	37	194.04	28.992	73	73230996	9
Basketball Pass (class D) 416x240	**22**	903.89	41.325	80	51936540	7
	27	459.79	37.709	72	51108934	6
	32	230.33	34.368	66	51162345	6
	37	118.53	31.412	62	52260397	6
Four People (class E) 1280x720	**22**	2901.89	42.554	640	4.44E+08	57
	27	1291.89	40.427	563	4.08E+08	53
	32	707.81	37.863	532	3.85E+08	49
	37	403.62	34.959	513	3.72E+08	48
Johnny (class E) 1280x720	**22**	2582.30	42.837	684	5.34E+08	70
	27	858.90	41.076	585	4.85E+08	64
	32	415.46	39.100	546	4.5E+08	57
	37	227.52	36.732	526	4.24E+08	54
Kristen And Sara (class E) 1280x720	**22**	2694.29	43.205	688	5.63E+08	74
	27	1115.47	41.143	606	5.22E+08	69
	32	568.18	38.802	567	4.96E+08	63
	37	316.00	36.170	542	4.75E+08	64

TABLE 4.19: SIMULATION RESULTS OF TZSEARCH ALGORITHM FOR CLASS A SEQUENCES IN RANDOM ACCESS MODE

Sequence	*QP*	*Bitrate (Kbps)*	*Y-PSNR (dB)*	*Total encoding time*	*Total ME Search Points*	*Total ME Time (sec)*
Traffic (class A) 2560x1600	**22**	13628.03	41.620	3006.014	1.84E+09	342
	27	5461.34	39.057	2471.496	1.6E+09	309
	32	2607.24	36.481	2180.01	1.41E+09	281
	37	1350.86	33.814	2041	1.28E+09	265
People on Street (class A) 2560x1600	**22**	32858.05	40.193	4614	6.8E+09	980
	27	15824.14	37.165	3842	5.72E+09	865
	32	8313.78	34.184	3318	4.7E+09	749
	37	4678.12	31.429	2994	3.91E+09	667
Nebuta (class A) 2560x1600	**22**	917165.90	38.832	11532	1.91E+10	2358
	27	679629.39	32.511	10587	1.7E+10	2120
	32	472875.45	26.790	9721	1.35E+10	1863
	37	298457.71	21.120	8646	9.46E+09	1531
Stream Locomotive (class A) 2560x1600	**22**	778091.58	38.500	10629	1.38E+10	1668
	27	561708.59	32.570	9763	1.16E+10	1453
	32	373970.14	27.070	8809	8.88E+09	1241
	37	270132.99	22.840	7759	6.36E+09	1036

TABLE 4.20: SIMULATION RESULTS OF PROPOSED ALGORITHM FOR CLASS A SEQUENCES IN RANDOM ACCESS MODE

Sequence	*QP*	*Bitrate (Kbps)*	*Y-PSNR (dB)*	*Total encoding time (sec)*	*Total ME Search Points*	*Total ME Time (sec)*
Traffic (class A) 2560x1600	**22**	13823.15	41.610	2925	2.06E+09	274
	27	5584.13	39.040	2430	1.95E+09	257
	32	2692.29	36.460	2197	1.88E+09	249
	37	1406.48	33.790	2071	1.83E+09	239
People on Street (class A) 2560x1600	**22**	33109.93	40.190	3980	2.79E+09	372
	27	15960.13	37.150	3330	2.62E+09	354
	32	8397.89	34.160	2920	2.47E+09	329
	37	4722.05	31.400	2649	2.33E+09	317
Nebuta (class A) 2560x1600	**22**	920653.80	38.820	9380	3.25E+09	419
	27	683683.20	32.562	9069	3.25E+09	424
	32	477046.04	26.838	8385	3.24E+09	432
	37	301275.70	21.140	7508	3.12E+09	420
Stream Locomotive (class A) 2560x1600	**22**	780416.40	38.540	9247	3.21E+09	408
	27	564626.37	32.612	8913	3.15E+09	411
	32	377392.85	27.031	8141	3.06E+09	403
	37	271941.98	22.777	7122	2.89E+09	387

TABLE 4.21: COMPARISON OF RESULTS OF PROPOSED ALGORITHM WITH FULL SEARCH ALGORITHM FOR CLASS B SEQUENCES

Sequence Name	**QP**	***Num of srch points ΔN (%)***	***ME time T (ΔT %)***	***Total Enc Time (ΔE %)***	***BD-PSNR***	***BD-bitrate***
Kimono (class B) 1920x1080	**22**	99.571	99.390	94.092	3.53E-05	-0.03952
	27	99.617	99.433	95.038		
	32	99.662	99.497	95.775		
	37	99.702	99.530	96.269		
ParkScene (class B) 1920x1080	**22**	99.401	99.132	89.496	-0.01158	0.374027
	27	99.434	99.171	91.509		
	32	99.471	99.221	92.647		
	37	99.481	99.227	93.040		
Cactus (class B) 1920x1080	**22**	98.942	98.526	81.350	-0.02384	1.025484
	27	98.991	98.577	85.446		
	32	99.037	98.593	87.249		
	37	99.106	98.743	88.546		
Basketball_Drive (class B) 1920x1080	**22**	99.437	99.208	87.597	-0.01745	0.786849
	27	99.431	99.181	90.667		
	32	99.352	99.047	90.949		
	37	99.220	98.868	89.877		
BQTerrace (class B) 1920x1080	**22**	99.689	99.537	95.318	-0.01216	0.604316
	27	99.729	99.582	96.293		
	32	99.762	99.625	96.788		
	37	99.784	99.654	97.098		
Average						

TABLE 4.22: COMPARISON OF RESULTS OF PROPOSED ALGORITHM WITH TZSEARCH ALGORITHM FOR CLASS B SEQUENCES

Sequence Name	QP	*Num of srch points ΔN*	*ME time T*	*Total Enc Time (ΔE %)*	*BD-PSNR*	*BD-bitrate*
Kimono (class B) 1920x1080	**22**	56.56	53.61	23.62	-0.007	0.223
	27	51.76	50.88	23.61		
	32	45.67	47.45	22.84		
	37	38.74	42.26	20.30		
ParkScene (class B) 1920x1080	**22**	67.91	64.62	7.82	-0.017	0.677
	27	64.39	61.96	8.52		
	32	58.91	57.20	7.70		
	37	51.53	54.23	6.33		
Cactus (class B) 1920x1080	**22**	59.54	56.68	10.08	-0.006	0.315
	27	51.13	50.00	11.24		
	32	41.59	44.25	10.85		
	37	30.84	36.62	9.04		
Basketball_Drive (class B) 1920x1080	**22**	78.24	76.16	6.00	-0.007	0.349
	27	75.34	73.83	6.90		
	32	71.03	71.24	6.94		
	37	65.29	67.32	5.23		
BQTerrace (class B) 1920x1080	**22**	56.56	53.61	23.24	-0.007	0.223
	27	51.76	50.88	23.52		
	32	45.67	47.45	21.72		
	37	38.74	42.26	19.20		
Average		**60.08**	**59.93**	**13.73**	**-0.006**	**0.25**

TABLE 4.23: COMPARISON OF RESULTS OF PROPOSED ALGORITHM WITH TZSEARCH ALGORITHM FOR CLASS C SEQUENCES

Sequence Name	**QP**	***Num of srch points*** **ΔN (%)**	***ME time*** **ΔT (%)**	***Total Enc Time*** **ΔE (%)**	**BD-PSNR**	**BD-bitrate**
Race Horses (class C) 832x480	**22**	80.81	79.12	22.94	-0.025	0.594
	27	79.34	78.13	24.80		
	32	76.94	76.30	25.56		
	37	73.68	74.11	25.00		
BQ Mall (class C) 832x480	**22**	62.90	62.00	7.62	-0.046	1.125
	27	59.90	55.56	7.81		
	32	55.36	56.41	7.49		
	37	49.62	50.00	6.00		
Party Scene (class C) 832x480	**22**	69.15	64.71	7.80	-0.034	0.733
	27	66.68	62.90	9.54		
	32	62.47	61.11	10.13		
	37	57.08	57.78	10.00		
Basketball Drill (class C) 832x480	**22**	71.05	68.00	12.92	-0.032	0.808
	27	68.99	67.16	13.78		
	32	65.13	65.52	13.62		
	37	59.82	61.22	12.61		
Average		66.18	65.00	13.60	**-0.034**	**0.815**

TABLE 4.24: COMPARISON OF RESULTS OF PROPOSED ALGORITHM WITH TZSEARCH ALGORITHM FOR CLASS D AND CLASS E SEQUENCES

Sequence Name	QP	*Num of srch points ΔN (%)*	*ME time T (ΔT %)*	*Total Enc Time (ΔE %)*	*BD-PSNR*	*BD-bitrate*
RaceHorses (class D) 416x240	**22**	78.35	84.85	77.78	-0.044	0.872
	27	76.98	83.80	78.79		
	32	74.62	82.17	75.86		
	37	71.34	79.86	70.83		
BQSquare (class D) 416x240	**22**	18.86	46.77	0.00	-0.037	0.941
	27	17.67	42.37	0.00		
	32	17.10	39.12	0.00		
	37	15.55	37.37	0.00		
BlowingBubbles (class D) 416x240	**22**	62.12	60.18	57.14	-0.027	0.707
	27	58.86	59.82	58.33		
	32	53.58	57.63	50.00		
	37	48.00	53.43	55.56		
BasketballPass (class D) 416x240	**22**	53.11	82.38	50.00	-0.044	0.902
	27	48.96	81.75	42.86		
	32	42.48	80.29	57.14		
	37	35.68	78.60	50.00		
Average		**48.33**	**65.65**	**45.27**	**-0.038**	**0.856**
FourPeople (class E) 1280x720	**22**	57.20	54.26	4.87	-0.007	0.08
	27	53.26	52.06	2.83		
	32	48.71	50.86	4.49		
	37	44.06	47.79	3.71		
Johnny (class E) 1280x720	**22**	48.12	47.34	3.18	-0.003	0.27
	27	42.37	42.34	2.70		
	32	37.25	40.66	2.33		
	37	32.55	36.59	2.27		
KristenAndSara (class E) 1280x720	**22**	63.48	64.78	10.69	-0.002	0.02
	27	57.57	58.29	14.81		
	32	51.14	55.93	7.04		
	37	44.87	53.62	7.41		
Average		**48.38**	**50.38**	**5.53**	**-0.004**	**0.12**

4.8 SOFTWARE TOOLS AND TEST CONDITIONS

The reference software HM version 16.0 is used for all the simulations [130]. The test sequences that were used in each class are listed in Table 2.4, Section 2.9.4 of Chapter

TABLE 4.25: SUMMARY OF COMPARISON OF PROPOSED VS FULL SEARCH AND TZSEARCH ALGORITHMS FOR VARIOUS CONFIGURATIONS

Sequence	LDP									
	Total_encoding_time (%)		Total ME Search Points (%)		Total_ME_Time (%)		BD-PSNR		BD-bitrate	
	FS	TZS	FS	TZS	FS	TZS	FS	TZS	FS	TZS
B	91.752	13.735	99.441	60.081	99.187	59.926	-0.013	-0.006	0.550	0.253
C	85.587	13.601	99.047	66.182	98.703	65.001	-0.051	-0.034	1.225	0.815
D	75.895	8.374	98.241	48.330	97.755	45.268	-0.051	-0.038	1.154	0.856
E	76.264	5.530	98.017	48.380	97.247	50.380	-0.014	-0.004	0.445	0.120
Total Average	**82.375**	**10.310**	**98.686**	**55.743**	**98.223**	**55.144**	**-0.033**	**-0.020**	**0.843**	**0.511**
	LDB									
B	89.595	36.657	99.487	46.584	98.586	46.330	-0.011	-0.006	0.473	0.207
C	83.069	33.931	99.141	54.267	97.896	52.986	-0.035	-0.021	0.809	0.493
D	71.424	31.352	98.383	38.585	95.894	35.424	-0.034	-0.020	0.711	0.415
E	70.999	33.965	98.250	36.563	95.990	34.434	-0.013	-0.012	0.481	0.444
Total Average	**78.772**	**33.976**	**98.815**	**44.000**	**97.092**	**42.293**	**-0.023**	**-0.015**	**0.619**	**0.390**
	RA									
B	93.887	23.601	99.526	43.229	98.932	41.255	-0.041	-0.049	0.848	0.365
C	91.353	20.836	99.371	49.587	98.715	48.189	-0.088	-0.079	0.506	0.325
D	91.101	29.887	99.341	31.582	98.666	29.919	-0.066	-0.052	0.701	0.535
E	92.355	29.286	99.371	13.638	98.923	10.549	-0.011	-0.011	0.394	0.323
Total Average	**92.174**	**25.903**	**99.402**	**34.509**	**98.809**	**32.478**	**-0.055**	**-0.048**	**0.534**	**0.394**
Total Average	**84.440**	**23.396**	**98.968**	**44.751**	**98.041**	**43.305**	**-0.037**	**-0.028**	**0.665**	**0.432**

2. The test conditions for various configurations LDP, LDB and RA are shown in Table 4.26.

In LDB (also called as main configuration) and LDP the hierarchical coding structure was disabled and the GOP size limited to 4. In RA configuration, hierarchical coding structure was enabled and the GOP size is taken as 8. In LDP configuration, bi-

TABLE 4.26: CONFIGURATION SETTINGS USED IN HM

CODING OPTIONS	PARAMETER		
	LDP	*LDB*	*RA*
Encoder Version	HM 16.0		
Reference Frames	4		
R/D Optimization	Enabled		
Motion Estimation	TZSearch/Proposed		
Search Range	64		
GOP Size	4		8
Hierarchical Encoding	Disabled		Enabled
Bi-directional Prediction	Disabled	Enabled	
Intra Period	-1 (only first frame)		32
Coding Unit Size	64		
Coding Unit Depth	4		
Min. Transform Unit Size	4		
Max. Transform Unit Size	32		
Rate Control	Disabled		
Internal Bit Depth	8		
Hadamard ME	Enabled		
Asymmetric Motion Partitioning (AMP)	Enabled		

directional prediction was disabled and only P-frames were used. The intra period (frequency of I frames) of RA mode was 32. For LDP and LDB modes only the first frame can be I frame.

The internal bit depth, that is the number of bits used to indicate the color of each pixel, was taken 8 for all modes, which is the default value. The rate control algorithm was disabled. The AMP was enabled so that ME is performed for all block sizes.

4.9 SUMMARY OF PROPOSED ALGORITHM

The proposed fast motion estimation (ME) algorithm is composed of five ME tools, dynamic search range, initial search point prediction, early termination, rotating hexagonal pattern and hexagonal fine refinement. Each of these tools contributes to reduce the ME complexity. The overall reduction in the ME complexity compared to

full search and fast ME algorithm (TZSearch) implemented in HEVC is significant with negligible loss in the PSNR and bitrate, as it was further supported using the Bjontegaard metric results.

5 PROPOSED VLSI ARCHITECTURE

5.1 INTRODUCTION

Motion estimation is usually performed sequentially in software based encoders. The CPU calculates the RD cost for each point in SW sequentially. Furthermore the whole ME operation is conducted for each block size of the current block sequentially, from higher block size to lower block sizes. This is computationally very expensive and particularly inefficient for HEVC, as the number of block modes and block sizes is very large. Some of the complexity can be reduced using the preprocessing steps such as DSR algorithm, motion vector prediction algorithm and early termination algorithm (explained in Chapter 4). But, once the SW is fixed and an optimal MV has to be found in the SW, the complexity will be too high with CPU as the cost calculation is sequential. A good solution to this problem is to accelerate this process by designing a suitable hardware architecture and exploit parallelism and pipelining. The present thesis proposed a ME architecture which was able to comply with all the variable block sizes of HEVC standard.

The architecture is based on a FPGA and uses block RAMs to store SW pixels data. The top level block diagram of the overall system is shown in Fig. 5.1. The control unit controls all the ME process using a state machine. The current PU (Prediction Unit) and the SW pixels data are stored in their corresponding local memory units which are addressed through an external bus from the external memory unit. The address generation unit sends the address locations of the candidate reference block to the SW memory unit depending on the algorithm. Then the reference candidate block data is sent to the SW buffer for SAD calculation. The SAD unit calculates the SAD between the current PU and reference blocks. The cost calculation unit calculates the RD cost by adding the SAD value (calculated by SAD unit) to the bitrate of the MVD (explained in Section 2.11.6 of Chapter 2). The comparator compares the RD cost of the current and the previous minimum stored value. The final MV (along with its RD cost) of the block which has least RD cost is stored in the register memory and sent to the output. The details of each subsystem and their functional description are explained in the following sections.

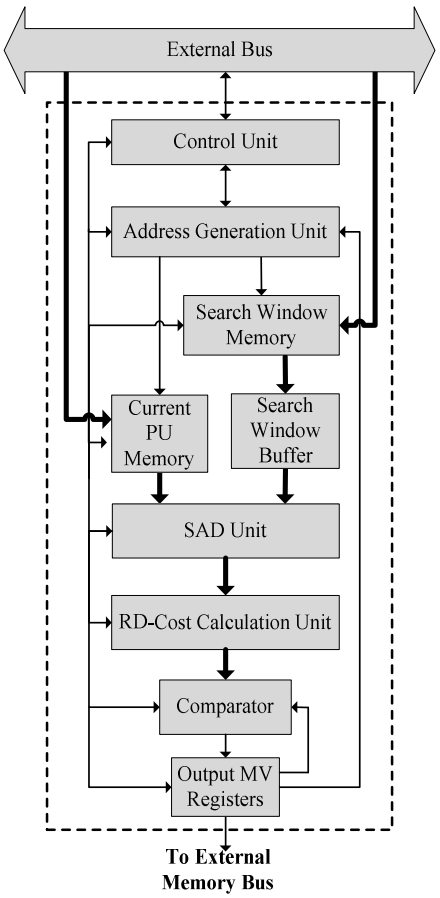


Fig. 5.1 Top Level Block Diagram of Proposed Motion Estimation Architecture

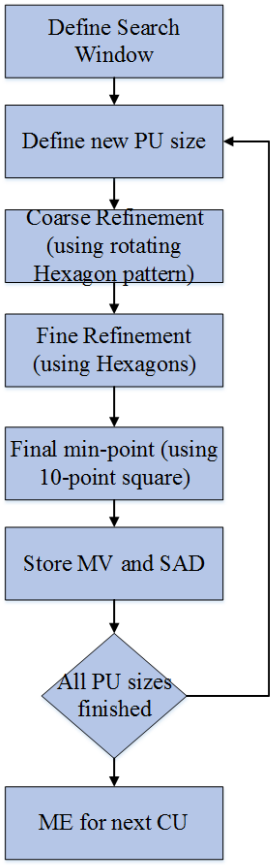


Fig. 5.2 Flowchart of the Proposed Algorithm

5.2 ALGORITHM ADAPTION TO HARDWARE

The proposed algorithm (explained in Chapter 4) is modified to suit the hardware design requirements. The proposed architecture takes one 64x64 current block (prediction unit) and a SW with 192x192 pixels data (search range=64) and outputs a set of variable block size MVs. The coarse refinement step using rotating hexagonal patterns is performed once the SW data is defined. After obtaining the minimum cost MV (from coarse search) the fine refinement is performed using the hexagonal based gradient descent method. In the last stage, the search is performed for the remaining 10 search points in the hexagon (explained in Section 4.5, Fig. 4.3, of Chapter 4). A graphical representation of these steps is depicted in the flowchart of Fig. 5.2.

As shown in Fig.4.3 of Section 4.2.5, there will be 6 rotating hexagonal grids resulting in 36 search points. Apart from hexagons, there is a small diamond search pattern near the co-located point, making a total of five additional search points, as shown in (5.1), where 'R' is the search range value. After finding the coarse minimal point, the algorithm refines it using small hexagons (shown in Fig. 4.6 of Chapter 4). In each iteration there will be three new search points. For simplicity in hardware implementation, the maximum number of iterations for fine refinement is limited to 10. The number of total search points which takes fine refinement iterations more than 10 is less than 0.001% (with negligible RD loss) and further the fixed number of maximum iterations value would be easy to calculate the maximum throughput instead of having a variation in number of iterations between various current blocks. Hence the number 10 is chosen. The 10-point square pattern has 10 search points. The total fine refinement points are given by (5.2), where 'i' represents the number of iterations in the fine refinement stage. In the worst case there will be 43 (= 6+3x9+10) search points in the fine refinement stage and hence the total number of search points is 84 (= 36+5+33+10). The total number of search points is given by (5.3).

$$n_{cs} = 5 + (6 \times \log_2 R) \tag{5.1}$$

$$n_{fr} = \big(6 + 3(i-1)\big) + 10 \tag{5.2}$$

$$n_{total} = (5 + (6 \times \log_2 R)) + \big(6 + 3(i-1)\big) + 10 \tag{5.3}$$

5.3 CONTROL UNIT AND ADDRESS GENERATION UNIT

The control unit controls the entire ME process using a finite state machine described in the following sub-section. The control unit also controls the operations of all the other sub systems by transmitting and receiving input and output signals.

5.3.1 State Machine of the System

The state machine is designed using Moore FSM (Finite State Machine) design logic [132], which is safer to use as the outputs change synchronously at every clock edge. A graphical representation of the proposed FSM is shown in Fig. 5.3. The FSM contains 7 states – *IDLE, LOAD_DATA, GET_ADDR, UPDATE_BUFFER, SAD, RD_COST, COMPARE.* Initially after reset the system goes into *IDLE* state. Then after the 'start' signal is given, it goes to the *LOAD_DATA* state. Here, the SW and current block data

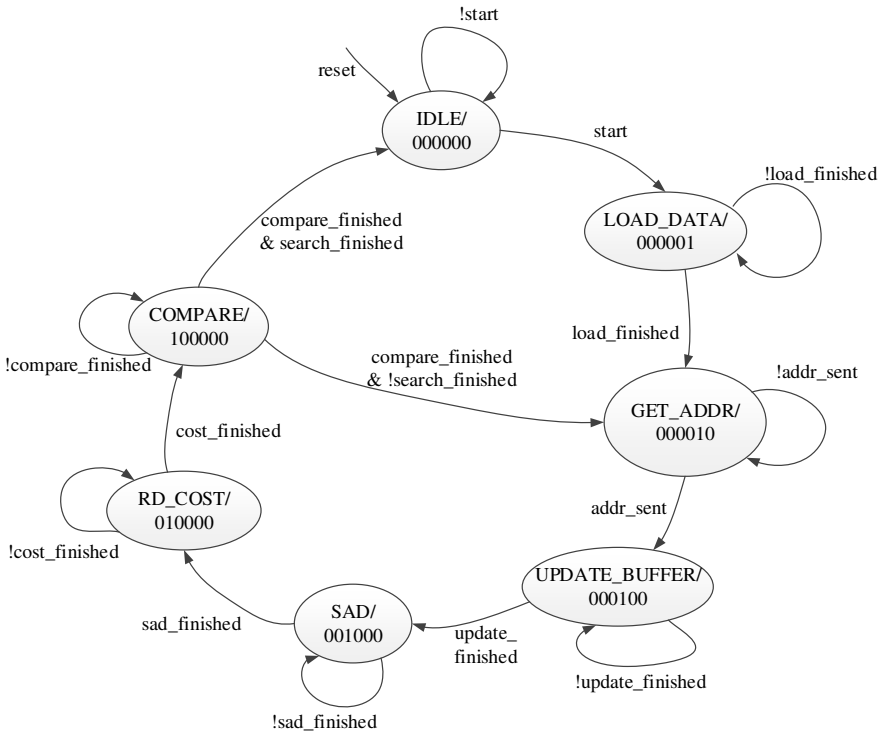


Fig. 5.3 State Machine for the Proposed Architecture

are updated from the external memory. After the data is loaded, the system goes into state *GET_ADDR*. In this state, the AGU (Address Generation Unit) sends the new address to the SW memory for which the cost has to be calculated. Then in the next state (*UPDATE_BUFFER*), the SW buffer is updated with the new reference PU block data. After the buffer is updated, the system calculates the SAD and RD cost in the states *SAD* and *RD_COST* respectively.

In the final state *COMPARE*, the RD cost is compared using a comparator. The MV and SAD registers are updated with the final results. The states *GET_ADDR*, *SAD*, *RD_COST* and *COMPARE* continue to loop until all the total search points are checked, from global search stage to fine refinement. If the search points are finished, the *COMPARE* state goes to *IDLE* state.

5.3.2 Address Generation Unit

The Address Generation Unit (AGU) sends addresses for the search window memory and the current PU memory units so that they can send the corresponding data to the SAD unit for calculation. The address sequences are generated according to the ME

algorithm. As explained, initially there are 41 search points (six hexagons, one small diamond and co-located point) for finding the coarse minimum point. As the addresses of these points are fixed for every current block, these values are stored in a ROM (Read Only Memory) with width 14-bit and depth 41. The width is taken as 14 because each co-ordinate (x and y) requires 7-bit with MV ranging from -64 to 63. After the minimum point is calculated, the offset address is sent back from the comparator to the AGU. Then the AGU generates offset addresses for small hexagon patterns to do fine refinement.

5.4 PROPOSED MEMORY ARCHITECTURE

5.4.1 Search Window Memory Architecture

One of the critical design considerations for ME is the search window memory architecture. Broadly classifying, there are two types of search window memories that can be considered for the proposed fast ME architecture. The first type (Type-I memory architecture) operates by sending only the required reference candidate block (or some blocks) from the external memory to the on-chip search window memory. The second type (Type-II memory architecture) reads the entire search window memory from the external memory to the on-chip search window memory. Type-I memory architecture increases the memory traffic because all pixels of each candidate block have to be transmitted to the local memory even though some parts were transmitted previously. Even with data reuse schemes to send only the part of candidate block which was not sent previously, the pixels in the local memory have to be shifted arbitrarily each time to meet the SAD calculation requirements. The Type-II memory architecture decreases the memory traffic, but increases the on-chip memory size compared to Type-I. Furthermore, the candidate reference block memory in Type-I has to be updated every time (after SAD operation is finished) from the external memory via a limited width external bus. This will introduce a latency in the entire ME operation. Hence, the proposed ME architecture uses Type-II memory architecture to store the entire search window memory.

To store the entire SW memory, the number of hardware resources (registers) will be too high. One of the major advantages in using modern FPGAs is the availability of embedded memory blocks (block RAMs or BRAMs). A comparison between hardware

TABLE 5.1: COMPARISON BETWEEN DISTRIBUTED RAM ARCHITECTURE AND BRAM BASED ARCHITECTURE

	Distributed RAM Architecture	**BRAM based Architecture**
Slice LUTs	19986 (13%)	3441 (2%)
Slice Registers	4608 (1%)	64 (<1%)
BRAMs	NA	24 (5%)

costs for SW memory (with search range 64) between BRAMs and distributed RAM in a Xilinx Virtex-6 FPGA [135] is shown in TABLE 5.1. The distributed RAM uses register memory (flip flops) available from slice LUTs (Look Up Tables) of the FPGA. As seen from the results, the distributed RAM uses 13% of the available FPGA LUTs. On contrary the BRAM based architecture uses only 2% of slice LUTs and use the available BRAMs for memory (5% of total BRAMs available). Hence, the present work uses BRAMs to design the SW memory architecture for the proposed ME engine. The number of BRAMs required depends on the required SW size and maximum memory size of each available BRAM. The BRAM data and address bus width is designed based on the number of pixel bytes needed concurrently for SAD operation.

The proposed memory architecture uses simple dual port BRAM which has two data ports, (one for writing data and other for reading data) and with 36k memory as shown in the diagram of Fig. 5.4. When writing the SW pixels from external memory to the on-chip BRAM, the proposed architecture uses a 64-bit data bus. The read data bus width is also taken 64, so as to read 8 pixels concurrently.

For the proposed ME algorithm, the maximum Search Range (SR) is taken as 64 and the maximum default size of the current PU (Prediction Unit) block in HEVC is 64x64. Hence the SW size will be 192x192 pixels, calculated using (5.4), where W_{SW}, H_{SW} represents width and height of SW, W_{PU}, H_{PU} represent width and height of PU block. Hence the depth of each BRAM is taken as 192 which is equivalent to the height of the SW memory. The number of BRAMs required is calculated using (5.5), where n_{BRAM} represents number of BRAM units, W_{SW} represents width of SW and $W_{readBus}$ represents required width of read data bus.

$$W_{SW} \times H_{SW} = \left((2 \times SR) + W_{PU}\right) \times \left((2 \times SR) + H_{PU}\right) \tag{5.4}$$

$$n_{BRAM} = \frac{W_{SW}}{W_{readBus}} \tag{5.5}$$

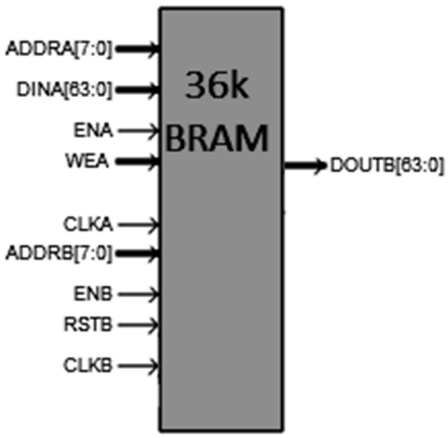


Fig. 5.4 BRAM used for search window memory architecture

The search window width W_{SW} is 1536 bits (= 192 × 8 bits/pixel). The read data bus width should be considered based on the maximum allowable read data bits in the BRAM specification. The proposed memory architecture uses Xilinx Virtex-6 BRAM36 [135] embedded memory, that has maximum allowable concurrent read bits 72 (with 64 data bits and 8 error correction bits). Hence the proposed memory architecture uses 64

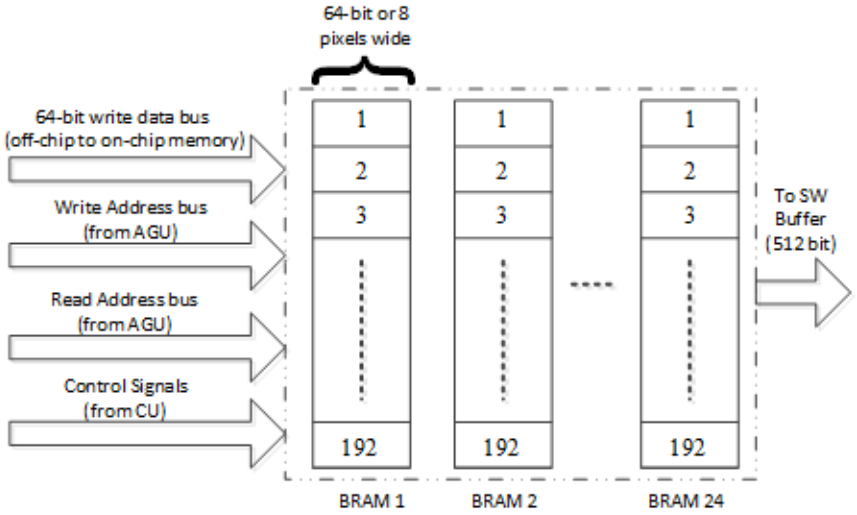


Fig. 5.5 Proposed BRAM based search window memory architecture

TABLE 5.2: SYNTHESIS RESULTS OF CURRENT PU MEMORY AND SW BUFFER BLOCKS

	Slice LUTs	**Slice Registers**	**Max freq. (MHz)**
Cur. PU Mem.	4390	3846	500.95
SWB	4519	3974	500.95

bits wide read data bus (which is equivalent to 8 pixels) for each BRAM. Hence from (5.5), n_{BRAM} is equivalent to 24 (=1536/64).

In a similar manner, for any BRAM specification, the number of BRAMs can be calculated using (5.5). The complete memory architecture is shown in Fig. 5.5, which consists of 24 BRAMs, a write data bus (with width 64) to store SW pixels from the external to the on-chip memory, address and control signals. The read data bus width is 512 bits (or 64 pixels).

5.4.2 Current Block and Search Window Buffer Memory Architecture

For each search operation, the current PU memory is fixed and hence the current PU pixels data is stored into the on-chip memory. The data from the PU memory needs to be accessed according to the input requirements of the SAD unit. The SAD unit accesses each row from four 4x4 pixel blocks in one clock cycle. Accessing these memory locations is easy if implemented in distributed RAM architectures. Furthermore the memory required for 64x64 PU block is less when compared to SW memory. Hence the PU memory is stored using distributed RAM. In order to make the memory access regular, the candidate block from SW is stored in a 64x64 buffer (SWB or SW Buffer).

The synthesis results of the current PU memory block and SW buffer blocks are shown in TABLE 5.2. Both the memory blocks operate at same frequency (500.95 MHz) and occupy almost the same number of slice registers (3.8k and 3.9k) and slice LUTs (4.3k and 4.5k).

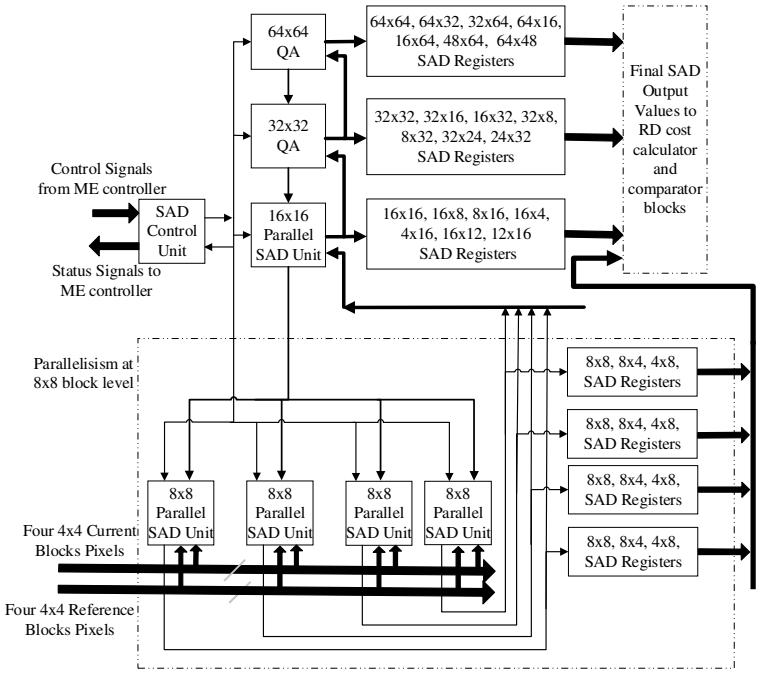


Fig. 5.6 Architecture of Proposed SAD Calculation Unit

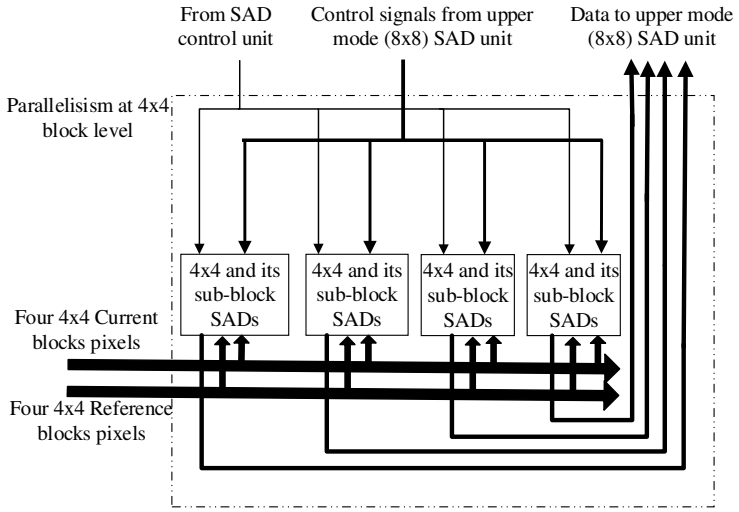


Fig. 5.7 Internal Architecture of 8x8 SAD Unit

5.5 PROPOSED SAD ARCHITECTURE

The proposed SAD architecture is shown in Fig. 5.6. The architecture consists of four 8x8 SAD calculation units, one 16x16 SAD calculation unit and two Quad-tree Adders (QAs) of size 32x32 and 64x64 for calculating variable block size SADs. Each 4x4 SAD block contains 16 PE (Processing Elements) and calculates one 4x4, two 4x2 and two 2x4 partial SADs. In each clock cycle, one set of four 4x4 pixel blocks (or one 8x8 pixel blocks) of current PU and search window PU block is sent to the 4x4 SAD units.

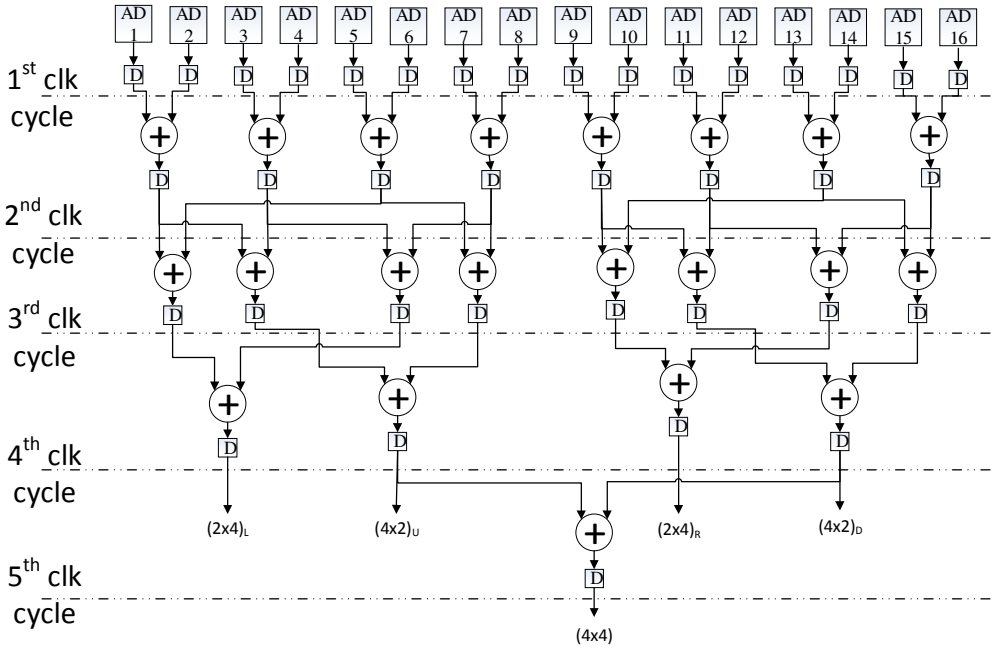


Fig. 5.8 Internal Architecture of 4x4 SAD Unit

All the partial SADs (four 4x4, eight 4x2 and eight 2x4) are stored in temporary registers along with offset addresses and sent to the 8x8 SAD unit.

The internal architecture of the 8x8 SAD unit is shown in Fig. 5.7, which consists of four 4x4 block size SAD units. Each 8x8 SAD then calculates one 8x8, two 8x4 and two 4x8 partial SADs. These values are stored in registers (along with their MV offset addresses) and then sent to upper depth PU (16x16) SAD unit. The process repeats until 64x64 SAD unit. The 32x32 QA and 64x64 QAs add and accumulate their lower mode partial SAD while the 8x8 and 16x16 SAD units calculate the partial SADs in parallel.

5.5.1 Adder Tree Architecture for 4x4 SAD Unit

The internal architecture of the 4x4 SAD blocks is shown in Fig. 5.8. It consists of sixteen absolute difference (AD) circuits and twenty-one 32-bit carry select adders. Each AD (Absolute Difference) block takes one current pixel input and one search window PU block pixel input and calculates absolute difference using the circuit shown in Fig 5.10. After the absolute difference operation, the 4x4 SAD-block adds all the sixteen AD values using an adder tree architecture and produces 4x4 partial SAD along with its variable block size SADs – $(4x2)_0$, $(4x2)_1$, $(2x4)_0$, $(2x4)_1$. Although the 4x4 SAD and its variable size SADs are not used for the calculation of RD costs and finding

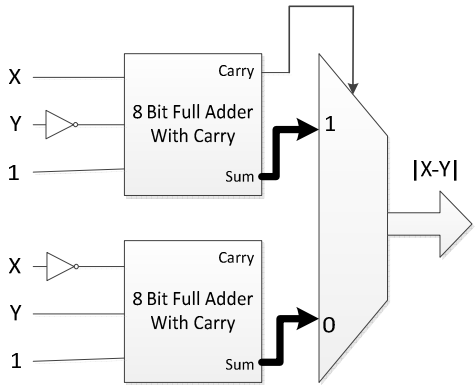


Fig. 5.9 Internal Architecture of Absolute Difference Circuit

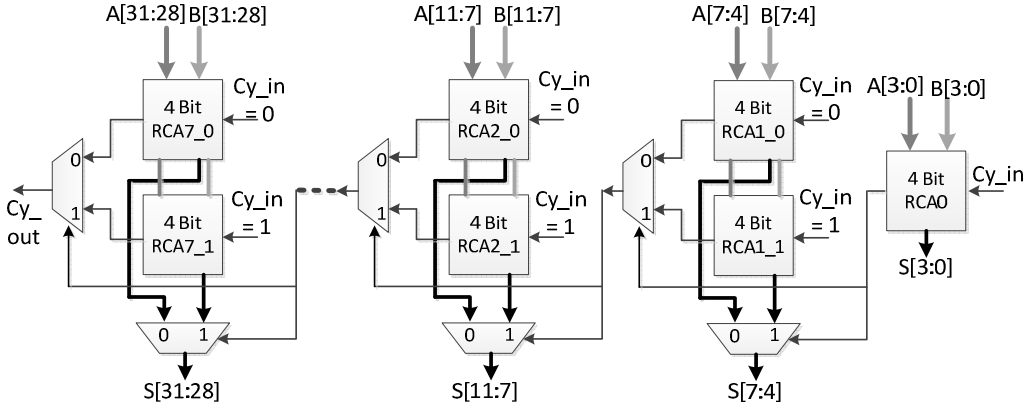


Fig. 5.10 Internal Architecture of Carry Select Adder

MVs, they are used to calculate upper mode 8x8 partial SADs and its variable block size SADs – 8x4, 4x8.

5.5.2 Quad-tree Adders

The Quad-tree Adder (QA) calculates variable block size SADs by adding and accumulating lower size PU SADs. The schematic block diagram is shown in Fig. 5.11. Each QA takes 5 variable block size inputs and outputs 13 upper mode variable block size outputs, which includes Asymmetric Mode Partitions (AMPs). The internal architecture of each output is shown in Fig. 5.12. In each clock cycle, one new input of the quad-tree SAD is added to the previously stored SAD. There will be 13 adder-register pairs corresponding to each variable block size. The proposed architecture has two QAs corresponding to the 32x32 and 64x64 SAD units.

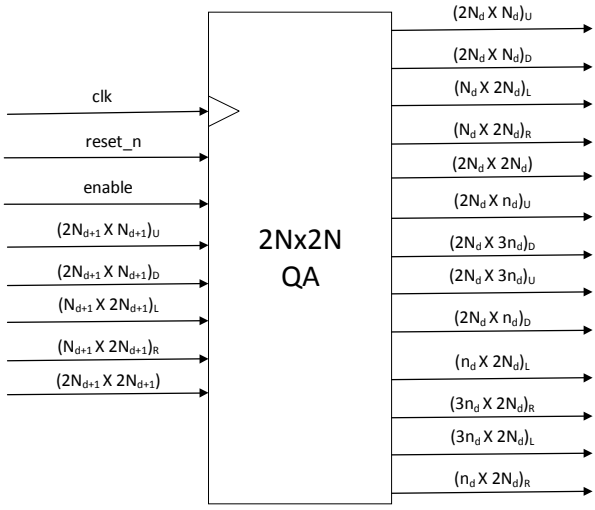


Fig. 5.11 Schematic Diagram of Quad-Tree Adders

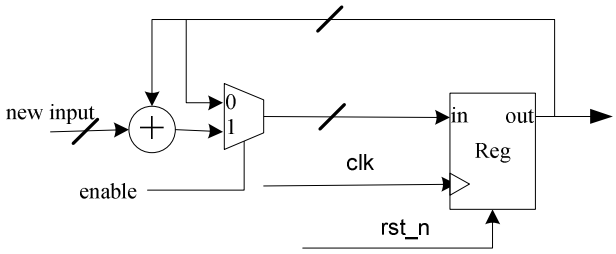


Fig. 5.12 Internal architecture of each Quad Tree Adder

5.5.3 Latency Calculations

The first set of sixteen 4x4 SADs are received at the end of 5 clock cycles (as illustrated in Fig. 5.8). After that, for each clock cycle there will be a new set of sixteen 4x4 SAD units. Similarly for the 16x16 SAD unit there will be two cycles delay (using adder tree architecture) with a total delay of 7 clock cycles. After that, for every clock cycle there will be one 16x16 SAD and its variable block size SADs (16x12, 16x8, 16x4, 12x16, 8x16, 4x16). Hence, the first set of four 16x16 SAD outputs is registered at the end of 10^{th} clock cycle and afterwards each set of four 16x16 SADs take four additional clock cycles. The 32x32 SAD takes each set of four 16x16 SADs and outputs one 32x32 SAD (and its variable block size SADs) to the 64x64 SAD unit. Altogether there will be sixteen 16x16 SAD units that has to be processed for one 64x64 SAD unit, each taking one clock cycle, except the first 16x16 SAD that take 7 cycles. Hence, the total number of clock cycles will be 23 (= 7+16).

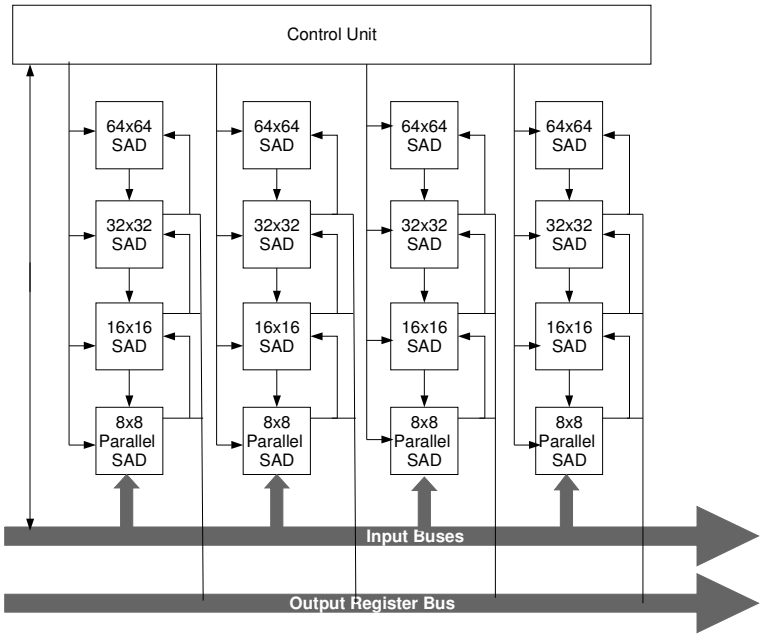


Fig. 5.13 Quad-core 1-stage (Type-I) SAD Architecture

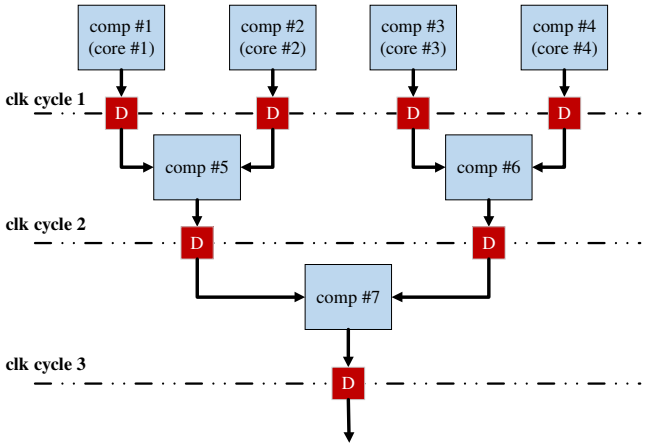


Fig. 5.14 Comparator Tree Architecture for Type-II SAD Unit

5.5.4 Area Calculations

There are four 8x8 SADs and each 8x8 SAD unit contain four 4x4 SAD units. Each 4x4 SAD unit contain 16 Absolute Difference (AD) circuits and 21 Carry Select Adders (CSAs) while each 8x8 SAD unit contain three CSAs to add the four 4x4 partial SADs. After that the 32x32 and 64x64 SAD units are designed using Quad-tree Adders (QAs). Hence, altogether there are 256 ADs, 336 CSAs (21x16) and 2 QAs.

5.5.5 Intra-parallelism

The aforementioned SAD architecture in Fig. 5.6, has parallelism at the CU depth three and four (8x8 and 4x4 PU). This is two-stage (2-stage) intra-parallelism. The total

number of clock cycles can be formulated using (5.6), where 'p' represents the number of parallel stages ($p \leq 4$). In 2-stage parallel architecture, putting 'p' equal to 2 gives a total of 23 clock cycles. Similarly the number of adder circuits and other logic can also be estimated using (5.7), (5.8) and (5.9) where n_{AD}, n_{CSA}, n_{QA} represents the number of ADs, CSAs and QAs, respectively. Substituting p=2 gives 256 ADs, 336 CSAs and 2 QAs.

The intra-parallelism can be extended to the next CU depth level (d=2 or 16x16 PU unit and p = 3) to further reduce the delay. Substituting p=3 in (5.6) reduces the latency to 11 clock cycles but at huge expense of ADs and CSAs - 1024 ADs and 1344 CSAs. Although the QA is reduced to 1 QA, the adder circuits increase the total hardware resources. On the other hand, if we reduce the total number of parallel stages to 1 (p=1), the hardware resources will be reduced but the delay will be increased to 69 clock cycles. Hence, 2-stage parallelism was selected to balance the trade-off between latency and hardware resources.

$$n_{clk} = (3 + 2p) + 2^{8-2p} \tag{5.6}$$

$$n_{AD} = 4^{p+2} \tag{5.7}$$

$$n_{CSA} = 21 \times 4^{p} \tag{5.8}$$

$$n_{QA} = 4 - p \tag{5.9}$$

5.5.6 Inter-parallelism

Apart from intra-parallelism, the SAD processing unit can also exploit parallelism at inter-level. In an n-level inter-parallel SAD architecture, the SAD unit can compute SADs for 'n' ME search points in parallel. This is similar to multicore architectures in general purpose CPUs. The SAD architecture with 4-level inter parallelism (quad-core SAD unit with 1-stage intra-parallelism) is shown in Fig. 5.13. Since the intra-parallelism is only up to 8x8 SAD stage (1-stage), each set of four search points takes 69 clock cycles (calculated using (5.6) without considering the comparator delays). But the hardware cost is quadrupled compared to single-core 1-stage intra-parallel SAD. The quad-core 1-stage (Type-II) SAD architecture is compared with single-core 2-stage intra-parallel (Type-I) SAD architecture.

TABLE 5.3: COMPARISON BETWEEN TYPE-I SAD AND TYPE-II SAD ARCHITECTURES

	Type-I (Single-core 2-stage intra-parallel SAD)	**Type-II (Quad-core 1-stage intra-parallel SAD)**
n_{clk}	23 (for 1 SAD) or 92 (for 4 SADs)	69 (for 4 64x64 SADs)
n_{AD}	256	1024
n_{CSA}	336	1344
n_{QA}	2	12
n_{Comp}	24 (1x)	168 (4x+3x)
max freq.	250.7 MHz	251.3 MHz
Slice LUTs	20416	36388
Slice Registers	20361	28284

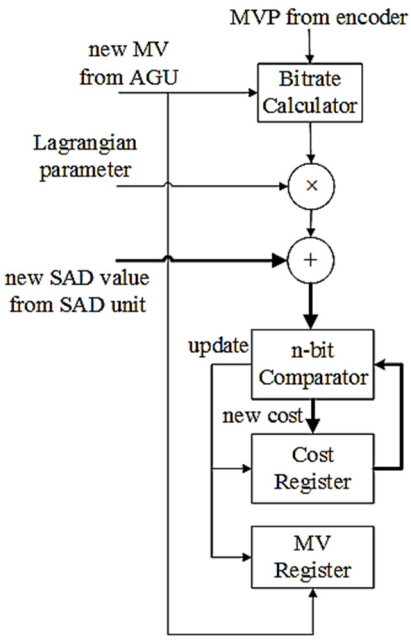


Fig. 5.15 RD Cost Calculation and Comparator Unit Architecture

TABLE 5.4: TRUTH TABLE OF 4-BIT PRIORITY ENCODER

Input = n	**Decimal eqvlnt.**	**Output = $\lfloor \log_2 n \rfloor$**
0000_0000_0000_0001	1	0
0000_0000_0000_001x	[2-3]	1
0000_0000_0000_01xx	[4-7]	2
0000_0000_0000_1xxx	[8-15]	3
0000_0000_0001_xxxx	[16-31]	4
0000_0000_001x_xxxx	[32-63]	5
0000_0000_01xx_xxxx	[64-127]	6
0000_0000_1xxx_xxxx	[128-255]	7
0000_0001_xxxx_xxxx	[256-511]	8
0000_001x_xxxx_xxxx	[512-1023]	9
0000_01xx_xxxx_xxxx	[1024-2047]	10
0000_1xxx_xxxx_xxxx	[2048-4095]	11
0001_xxxx_xxxx_xxxx	[4096-8191]	12
001x_xxxx_xxxx_xxxx	[8192-16383]	13
01xx_xxxx_xxxx_xxxx	[16384-32767]	14
1xxx_xxxx_xxxx_xxxx	[32768-65535]	15

TABLE 5.3 lists the comparison results. The results show that the maximum clock frequency is almost the same for both the architectures. The number of clock cycles in Type-II architecture is relatively less compared to Type-I architecture, as the Type-II architecture computes four SADs in parallel. But the number of CSAs, QAs, ADs and comparators is higher in Type-II architecture compared to that of Type-I. Each core uses 24 comparators corresponding to 24 modes of the SAD – 7 modes for 64x64

(64x64, 64x32, 32x64, 64x16, 64x48, 16x64, 48x64), 7 modes for 32x32, 7 modes for 16x16 and 3 modes for 8x8 (8x8, 8x4, 4x8). The comparators have to be used for each core in Type-II architecture and hence they are quadrupled and for each mode the comparator tree has to be formed which adds an extra three comparators for each mode (3x) as shown in Fig. 5.14. Due to this, the hardware resources including slice LUTs and slice registers are higher in Type-II SAD architecture. Furthermore, Type-II SAD architecture requires parallel accessing of multiple candidate reference blocks and due to this, the SW buffer and current block memories have to be replicated, which increases the FPGA slices. Hence, the proposed architecture uses Type-I SAD architecture.

5.6 COMPARATOR AND RD COST CALCULATION UNIT

There are 24 modes corresponding to each block size in HEVC. For each PU block size, the corresponding RD cost calculation and its comparator is designed. The architecture is shown in Fig. 5.15, where the new SAD values come from the SAD unit and the lagrangian cost is calculated using (1.2).

For each current block (and for its sub-blocks), the predicted MV and its lagrangian parameter are sent from the encoder. The AGU sends the MV of the reference block. Then the MVD (Motion Vector Difference) is calculated using (5.10), where MV is the reference block MV and PMV represent the predicted MV. The total number of bits occupied by MVD is calculated by the bitrate calculator using (5.11) and (5.12) (deducted from reference software HM [7]). The $\log_2$ operation is implemented using a priority encoder. The truth table of 4-bit priority encoder which is used to calculate logarithmic operation is shown in TABLE 5.4. The second column in the table represents the equivalent decimal value range. The output column represents the floor function of $\log_2 n$.

$$MVD(x,y) = MV(x,y) - PMV(x,y) \tag{5.10}$$

$$bits(n) = \begin{cases} 2 \times floor\left(log_2(1-2n)\right) + 1, & n \leq 0 \\ 2 \times floor\left(log_2(2n)\right) + 1, & n > 0 \end{cases} \tag{5.11}$$

$$bits = bits(x) + bits(y) \tag{5.12}$$

After the bitrate calculation, the obtained number of bits is multiplied by lagrangian cost and then added to the SAD value. If the new cost value is less than previously stored cost (obtained from the cost register), then the value is updated to the same cost register in the next clock cycle and the corresponding MV value is also updated to the MV register. This is done using a control signal 'update' coming from the comparator (as shown in the figure).

5.7 RESULTS AND ANALYSIS

5.7.1 Synthesis Results

The proposed architecture was designed using Verilog HDL and implemented in Xilinx Virtex-6 XC6VLX240T FPGA [135]. The architecture can process full HD frames at the rate of 60fps with a maximum operating frequency of 241.6 MHz for HEVC based 64x64 blocks, with search range ±64. The FPGA has 150720 slice LUTs, 301440 slice registers and 416 BRAMs (of each 36Kb size). The total FPGA slice LUTs occupancy of the design is 27998, the slice registers occupancy is 29571 and the total BRAMs occupancy is 24. In percentages, the total slice LUTs occupancy is 18%, the slice registers occupancy is 9% and the total BRAMs occupancy is 5% approximately. The total memory size is calculated assuming 100% BRAM occupancy. Each BRAM size is 36 Kbits and there are 24 BRAMs used. Hence the total memory usage is 108 Kbytes.

5.7.2 Data Schedule, Total Delay and Throughput

The data schedule diagram for the proposed architecture is shown in Fig. 5.16. Initially, just after the SW and current block data are loaded, the AGU generates and sends the MV row and column addresses to the search window memory (SWM) in the first clock cycle. Then the data is loaded to the SWB, row by row in total of 64 clock cycles. After the first 16 rows of data are loaded in SWB, they are processed by SAD unit in order to reduce the pipeline delay. In each clock cycle, 16 4x4 blocks of reference block and current block are accessed. Each 4x4 SAD takes 5 clock cycles to compute the 4x4 partial SAD (as explained using Fig. 5.6). During each of the subsequent clock cycle, the calculated 4x4 partial SADs are added in pipeline to get variable block size SADs. As explained earlier, altogether it takes 23 clock cycles to process one 64x64 reference block in the SAD unit. The RD cost calculator takes 3 clock cycles (1 for bitrate

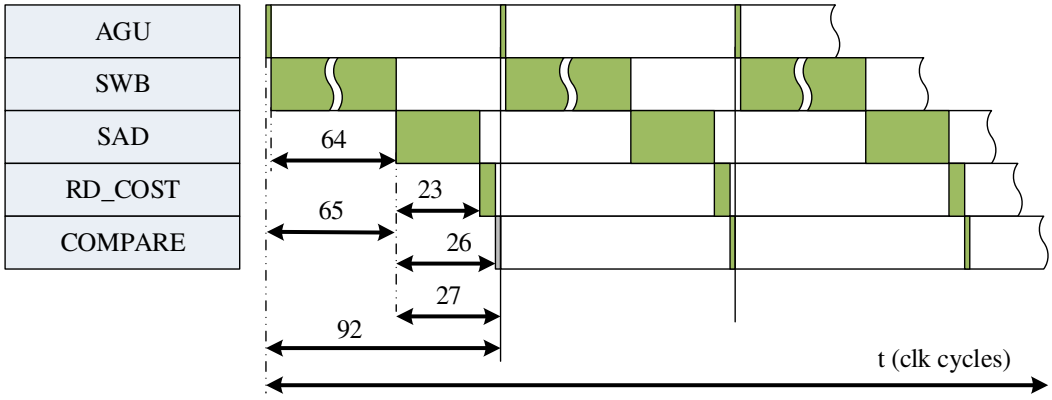


Fig. 5.16 Data Schedule for the Proposed ME Architecture

calculation, 1 for multiplication and 1 for addition). Finally, the comparator takes 1 clock cycles to compare the RD costs. Altogether it takes 92 (1 for AGU, 64 for SWB, 23 for SAD, 3 for RD cost and 1 for comparator) clock cycles to calculate costs of one reference block (and its variable size blocks).

Altogether there are 84 search points (as explained in Section 5.2) and it takes 7728 (=84x92) clock cycles to process one current block. For the full search architecture using any systolic array, it takes 4096 clock cycles to process the first 64x64 block. After that, each search point takes one clock cycle as the memory accessing is regular (in the best case scenario, without considering line refresh delay). But the total number of search points are equal to 16384 (=128x128) for a search range of 64. Hence the total number of clock cycles is 20479 (=4096+16384-1) which is higher compared to that of the proposed architecture. Further, the hardware resources (processing elements) required in FS algorithm based architecture will be more as the partial SADs have to be calculated and pipelined for each clock cycle.

The throughput (Trp) of the architecture can be calculated using (5.13), where coding tree block (CTB) represents coded tree blocks or current blocks. For the current architecture, full HD (1920x1080) resolution is used and hence the number of 64x64 CTBs is found to be 506 (1920x1080/64x64). For our proposed architecture, each CTB takes 7728 clock cycles to process one CTB and we consider one reference frame. The maximum clock frequency achieved was 241.6 MHz. Substituting all these in (5.13), we

achieve a throughput of 61 frames/sec. Thus our proposed architecture supports ME for full HD frames at 60 fps frame rate.

$$Trp = \frac{max\_freq}{\left(\frac{num\_cycles}{CTB}\right) \times \left(\frac{num\_CTBs}{frame}\right) \times num\_ref\_frms} \quad (5.13)$$

5.7.3 **Comparison with other FPGA based Design**

TABLE 5.5 shows the synthesis results of the proposed architecture along with comparison of some recent works. Compared to the architecture in [136], the proposed architecture supports more block modes including AMP modes. Due to this, the RD performance in the proposed architecture is higher, with a BD-rate decrease of only 0.84% compared to that of [136] which has BD-rate decrease of 12%. The architecture in [127] supports all block sizes but it is based on full-search algorithm with more circuit area requirements. When converted to effective gate count, the proposed architecture occupies 0.87M gates (with a maximum of 24 gates per slice LUT and 7 gates per slice register [135]) whereas the gate count in [127] is 3.56M gates. The architectures in [112] and [128] support block sizes only up to 16x16. The architecture in [126] has a higher operating frequency (270 MHz), but does not support AMP modes.

The architectures [129] and [137] are FPGA based architecture supporting HEVC block modes. Both the architectures operates with less frequency (110 MHz and 125 MHz) compared to the proposed architecture. Compared to [129], the proposed architecture use less number of slice LUTs (27.9k vs 55.3k) and BRAMs (24 vs 33) and slightly higher number of slice registers (29.5k vs 19.7k). But the architecture in [129] used block sizes only until 32x32 and with search range 24. The architecture in [137] supports all block sizes until 64x64 (including AMP modes) and with search range 64. But it occupies more FPGA resources – 85.01k slice LUTs, 141k slice registers and 298 BRAMs.

TABLE 5.5: SYNTHESIS RESULTS AND COMPARISON OF PROPOSED ARCHITECTURE WITH OTHER ARCHITECTURES

	Sinangil et. al. [136]	**Byun et. al** [127]	**Sanchez. et. al.** [128]	**Jou et. al** [126]	**Ndili et. al.** [112]	**Xu. et. al.** [129]	**Thomas et. al.** [137]	**Proposed**
Process	ASIC 65 nm CMOS	ASIC 65 nm CMOS	ASIC TSMC 90 nm	ASIC TSMC 90nm	Xilinx Virtex-2 Pro	Xilinx Virtex-6 XC6VLX-550T (40nm)	Xilinx Virtex-5 LX 330T FPGA (65 nm)	Xilinx Virtex-6 LX 220T FPGA (40 nm)
Slice LUTs	1830 K gates	3.56 M gates	50 K gates	787.7 K gates	10.8 K	55346 (16%)	85017 (41%)	27998 (18%)
Slice Registers					11.3 K	19744 (2.9%)	141004 (68%)	29571 (9%)
BRAMs					10	33 (5.2%)	298 (92%)	24 (5%)
Memory Size	208 KB	20.23 KB (SRAM)	82 Kb	17.4 KB	2.5 KB	148 KB	1.34 MB	108 KB
Max Operating frequency	200 MHz	250 MHz	41.3 MHz	270 MHz	246.5 MHz	110 MHz	125 MHz	241.6 MHz
Search Range	±64	±64	±44	±64	±16	±24	±64	±64
Max. Block Sizes	64x64	64x64	16x16	64x64	16x16	32x32	64x64	64x64
Supported Block Sizes	16×16, 32×32, 64×64	All	All until 16x16	8×8, 8×4, 4×8, 16×16, 16×8, 8×16, 32×32, 64×64	All until 16x16	All up to 32x32	All	All
Max. Resolution	3840×2160 (4kx2k) @30fps	3840 × 2160P (4k UHD) @30fps	1080p @30fps	4096×2048 (4kx2k QFHD @60fps	CIF @30fps	1080p @30fps	1080p @27fps	1080p @60fps
Algorithm	8-pixel SW subsampling + Three Step Search	Full Search	Multi Point Diamond Search (MPDS)	TZSearch (used in HM)	HW. Modified Diamond Srch. (HMDS)	Full Search	Full Search	Rotating Hexagon based Fast Search + Hexagon Refinement
Supported Tools	IME, FME	IME	IME	IME, FME	IME	IME	IME	IME
BD-rate decrease	12 %	-	1.7% bitrate increase compared to EPZS algorithm	5.14 %	0.16% to 1.85% bitrate increase compare to FS	-	-	0.84 %
Num. Ref. Frames	1	±1	1	1	5	1	1	1
AMP Support	No	Yes	No	No	No	Yes	Yes	Yes

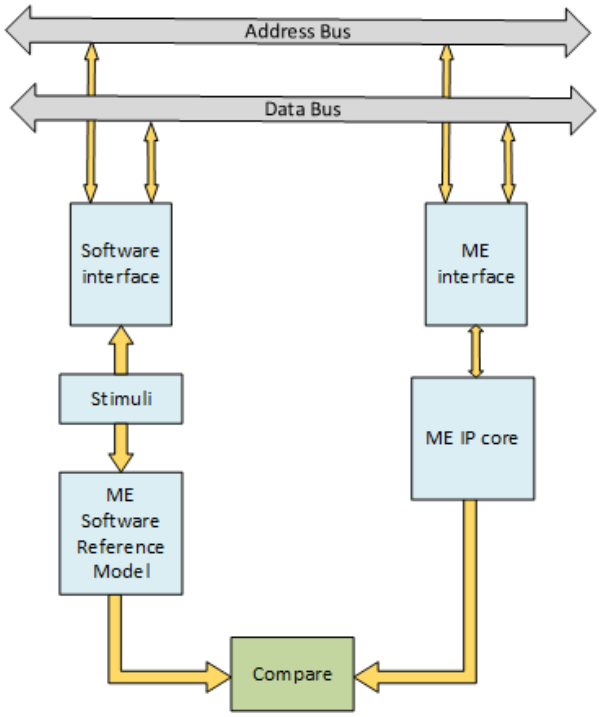


Fig. 5.17 Verification Setup Used to Validate the Proposed Design

5.7.4 **Verification Setup and Results**

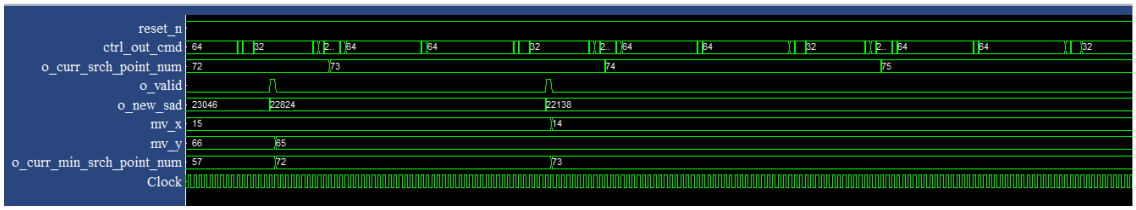
The proposed design was verified using the testbench shown in Fig. 5.17. The test stimuli (search window pixels and current block pixels) was applied to the software model for the proposed algorithm. The same test inputs are converted to 64 bit data values and sent to ME IP core through data bus and address bus interfaces. The output MVs from software are compared against the output values of ME IP core.

A sample values of inputs and outputs are shown below. The inputs from the test stimuli are the current block pixels and the search window block pixels which are shown as pictures in Fig.5.18 (a) and 5.18 (b) respectively. The output MV for these inputs (for 64x64 current block size) is (14,-1) which was obtained with ME algorithm software model. The same value is matched with that of the result obtained hardware architecture model, shown in Fig. 5.19 (a) and its zoomed version in fig 5.19 (b). The MV output in Fig 5.19 (b) is 7-bit unsigned decimal number. Hence MV_x is seen as 14, while MV_y is seen as 65 which is equal '1 000001' in binary. The first bit represents sign (1 for -ve and 0 for +ve) and the rest 6 digits represents magnitude. Hence value 65

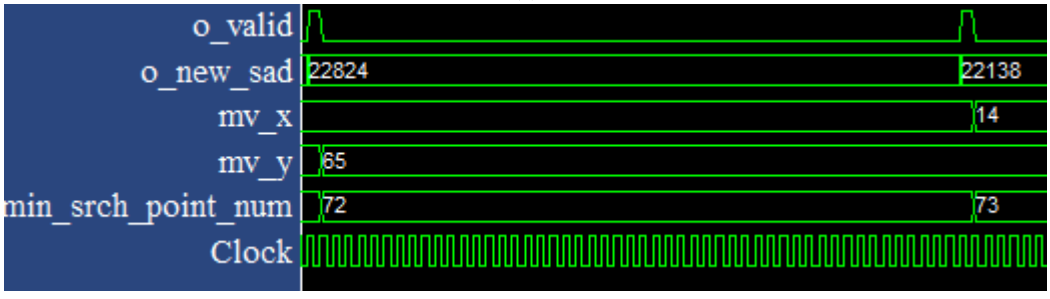
is equivalent to '-1'. For the rest of block sizes, the MVs and their hardware verification results are shown in Table 5.6. Each CTU block is divided into four partition blocks, from 64x64 until 8x8, which results in many portioned CTUs. For simplicity, only the first CTU size and their MVs are shown for CTUs 32x32, 16x16, 8x8. For each PU (prediction unit) its corresponding MV is obtained from software and compared with

(a) (b)

Fig. 5.18 Current Block and Search window Pixels used for Verification



(a)



(b)

Fig. 5.19 Hardware Simulation Output for the Current Block and SW Pixels

hardware MV result. For all the block sizes, the MV results from software and hardware matches, as shown in the table.

TABLE 5.6: HARDWARE VERIFICATION RESULTS FOR ALL THE BLOCK SIZES

CTU size	CTU index	PU size	PU index	S/W MV	Test Pass/Fail
64x64	0	64x64	0	(14,-1)	pass
		64x32	0	(6,0)	pass
			1	(14,29)	pass
		64x48	0	(6,0)	pass
			1	(13,47)	pass
		64x16	0	(5,0)	pass
			1	(14,15)	pass
		32x64	0	(14,-3)	pass
			1	(38,1)	pass
		48x64	0	(13,-2)	pass
			1	(25,6)	pass
		16x64	0	(3,0)	pass
			1	(32,-1)	pass
32x32	(0,0)	32x32	0	(-64,-32)	pass
		32x16	0	(-64,-32)	pass
			1	(-63,32)	pass
		32x24	0	(-64,-32)	pass
			1	(-63,32)	pass
		32x8	0	(-64,-32)	pass
			1	(-63,21)	pass
		16x32	0	(-64,-32)	pass
			1	(-62,20)	pass
		24x32	0	(-64,-32)	pass
			1	(-64,24)	pass
		8x32	0	(-64,-32)	pass
			1	(-61,16)	pass
16x16	(0,0,0)	16x16	0	(-64,33)	pass
		16x8	0	(-19,-7)	pass
			1	(-62,31)	pass
		16x12	0	(-17,-8)	pass
			1	(-64,29)	pass
		16x4	0	(-63,-33)	pass
			1	(-64,27)	pass
		8x16	0	(-56,40)	pass
			1	(-61,31)	pass
		12x16	0	(-56,38)	pass
			1	(-16,6)	pass
		4x16	0	(-16,-8)	Pass
			1	(-61,28)	Pass
8x8	(0,0,0,0)	8x8	0	(-10,5)	Pass
		8x4	0	(-1,-16)	Pass
			1	(-10,4)	Pass
		4x8	0	(-9,3)	Pass
			1	(-9,3)	Pass

5.8 SUMMARY OF OVERALL DESIGN

A fast search ME algorithm for HEVC and its FPGA hardware architecture was proposed and implemented. The architecture can perform ME for all the block sizes until 64x64 including asymmetric mode partitions and with search range of ±64. The synthesis results show that the proposed architecture outperforms in terms of area and operating frequency compared to recent works in the literature. Further research is being carried out to implement and embed a fractional ME architecture.

6 CONCLUSIONS AND FUTURE RESEARCH

6.1 SUMMARY OF RESEARCH

Motion estimation is one of the most complex tasks in block based video encoders. Especially in the encoder of the latest video coding standard HEVC, the complexity of ME is further increased due to increase in the block size to 64x64 pixels. The ME has to perform the operation in all variable block sizes including AMP modes. At the search window level, the search range also increased to 64 in HEVC and hence the number of search points also increases, where for each point the rate distortion cost has to be calculated for all block sizes. Hence to reduce the ME complexity, reducing the number of search points effectively without much effect in RD performance is one of the main approach used in the present thesis. To do this, the present thesis uses a fast ME algorithm which employs effective search patterns to get the optimal search point faster. Compared to full search and state-of-the-art fast ME algorithm used in HEVC reference software encoder, the overall reduction achieved in the ME complexity is significant with negligible loss in the PSNR and bitrate.

The ME complexity can be further decreased by using an appropriate hardware architecture. By introducing parallel stages for computing the SAD cost function, the computational cost can be further reduced. The present thesis also focused on effective implementation of an architecture of the fast ME algorithm which is able to perform in real time. The simulation and synthesis results show that the proposed architecture outperforms in terms of area and operating frequency compared with recent works.

The summary of achieved results are as follows.

- The proposed algorithm uses rotating hexagonal patterns and an efficient adaptive early termination strategy to reduce the ME time. On an average the total gain in ME time is 98.04% and 43.305% compared to full search and TZSearch algorithm respectively.
- The proposed algorithm also uses hexagonal fine refinement strategy to further reduce ME time and total encoding complexity costs. On an average it achieves overall gain of 84.44% and 23.39% in encoding time compared to full search and TZSearch algorithm respectively.

- The total reduction in ME search points complexity for the proposed algorithm compared to full search and TZSearch algorithm is 98.96% and 44.75% respectively.
- The overall BD-PSNR loss is 0.037 and 0.028 compared to full search and TZSearch algorithm respectively.
- The overall BD-bitrate increase is 0.665 and 0.432 compared to full search and TZSearch algorithm respectively.
- The proposed hardware architecture uses high speed SAD architecture which effectively uses sixteen 4x4 parallel SAD calculating cores.
- The architecture can perform ME with a throughput of 61 frames/sec and with a maximum clock frequency of 241.6 MHz for all the block sizes until 64x64 including asymmetric mode partitions, at a search range of ±64.

6.2 FUTURE RESEARCH DIRECTIONS

To make the present research more complete, the present thesis work is planned to extend in several directions, listed below.

- *Effective mode decision algorithm*: The ME complexity can be further reduced by using effective mode decision algorithms, which focus on eliminating some of the unnecessary modes of the current block. Hence for each selected mode, the proposed fast ME algorithm can be used and thus the overall performance of the video encoder can be increased. In the hardware architecture, the present work can be extended in order to incorporate mode decision and produce MVs for the blocks that are only necessary.
- *Fractional ME architecture*: As explained in chapter 2, the ME is also performed at sub-pixel level. After performing the integer ME in hardware, the MVs can be used to get the half-pixel and quarter-pixel accurate MVs. A future line of work may focus on the implementation of the fractional ME hardware architecture.
- *Motion Compensation unit*: The ME generates the MVs for each block in every frame. The motion compensation unit utilizes these MVs and generate motion compensated frames. These motion compensated frames are subtracted from the

current frames to get the residual frames. To verify the effectiveness of the ME architecture, the motion compensation unit hardware architecture can be designed and integrated with ME architecture.

References

[1] "The Zettabyte Era: Trends and Analysis," Cisco Visual Networking Index (VNI) White Paper, June 2014.

[2] G. J. Sullivan, J. R. Ohm, W.-J. Han and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 22, no. 12, pp. 1649-1668, Dec 2012.

[3] G. Delagi, "Harnessing technology to advance the next-generation," in *IEEE International Solid-State Circuits Conference (ISSCC 2010)*, San Francisco, CA, Feb. 2010.

[4] B. Bross , W.-J. Han , G. J. Sullivan , J.-R. Ohm and T. Wiegand, "High Efficiency Video Coding (HEVC) Text Specification Draft 9," ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC), 2012.

[5] "Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T Rec. H.264/ISO/IEC 14496-10 AVC," Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, JVT-G050, 2003.

[6] T. Wiegand, G. J. Sullivan, G. Bjøntegaard and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY,* vol. 13, no. 7, pp. 560-576, July 2003.

[7] "HEVC Reference Software HM 16.2," [Online]. Available: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/.

[8] B. Frank, F. David, S. Karl and K. Suhring, "HEVC reference software HM 16.2 reference manual," ITU-T/ISO/IEC JCTVC HEVC reference software manual, 2014.

[9] "IEEE standard Verilog hardware description language," IEEE Standards (IEEE

Std 1364-2001), Sept. 2001.

[10] "Xilinx Virtex-6 FPGA Configuration User Guide," Aug. 2014. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug360.pdf.

[11] T. Wiegand and H. Schwarz, Source Coding: Part I of Fundamentals of Source and Video Coding, Hanover MA USA: Now Publishers Inc, 2010.

[12] T. Wiegand and B. Girod, Multi-Frame Motion-Compensated Prediction for Video Transmission, Kluwer Academic Publishers, 2001.

[13] B. Furht, Encyclopedia of Multimedia, Springer Science and Business Media, 2006.

[14] M. Ghanbari, Standard Codecs: Image Compression to Advanced Video Coding, London UK: Institution of Electrical Engineers, 2003.

[15] J. L. Mitchell, W. B. Pennebaker, C. E. Fogg and D. J. LeGall, MPEG Video Compression Standard, London UK: Chapman & Hall Ltd, 1996.

[16] I. E. Richardson, The H.264 Advanced Video Compression Standard, West Sussex, England: John Wiley & Sons, Ltd, 2010.

[17] "Foreman video test sequence QCIF format [online] Available: https://media.xiph.org/video/derf/".

[18] I. E. Richardson, Video Codec Design: Developing Image and Video Compression Systems, West Sussex, England: John Wiley & Sons Ltd., 2002.

[19] I. E. Richardson, H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia, West Sussex, England: John Wiley & Sons Ltd, 2003.

[20] C. Poynton, Digital Video and HDTV: Algorithms and Interfaces, San Francisco CA: Morgan Kaufmann Publishers, 2003.

[21] K. Iguchi, A. Ichigaya, Y. Sugito, S. Sakaida, Y. Shishikui, N. Hiwasa, H. Sakate and N. Motoyama, "HEVC encoder for Super Hi-Vision," in *2014 IEEE International Conference on Consumer Electronics 2014 (ICCE 2014)*, Las Vegas, NV, Jan 2014.

[22] Q. Huynh-Thu and M. Ghanbari, "Scope of validity of PSNR in image/video quality assessment," *Institution of Engineering and Technology (IET) Electronics Letters,* vol. 44, no. 13, pp. 800-801, June 2008.

[23] K. Seshadrinathan, R. Soundararajan, A. Bovik and L. Cormack, "Study of Subjective and Objective Quality Assessment of Video," *Image Processing, IEEE Transactions on ,* vol. 19, no. 6, pp. 1427-1441, 2010.

[24] J. Ohm, G. Sullivan, H. Schwarz, T. K. Tan and T. Wiegand, "Comparison of the Coding Efficiency of Video Coding Standards—Including High Efficiency Video Coding (HEVC)," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 22, no. 12, pp. 1669 - 1684, Dec. 2012.

[25] G. Bjontegaard, "Calcuation of average PSNR differences between RD-curves," ITU-T VCEG Q6/16 Report, Austin, TX, USA, April 2001.

[26] G. Bjontegaard, "Improvements of the BD-PSNR model," ITU-T VCEG Report SG16 Q 6, 2008.

[27] H.-C. Chang, L. Chien-Chang and G. Jiun-In, "A novel low-cost high-performance VLSI architecture for MPEG-4 AVC/H. 264 CAVLC decoding," in *IEEE International Symposium on Circuits and Systems (ISCAS 2005)*, 2005.

[28] X. Tian, T. M. Le and Y. Lian, Entropy Coders of the H.264/AVC Standard: Algorithms and VLSI Architectures, Springer-Verlag, 2011.

[29] K. Rao, N. K. Do and J. H. Jae, Video coding standards: AVS China, H.264/MPEG-4 PART 10, HEVC, VP6, DIRAC and VC-1, Springer, 2014.

[30] "H.120: Codecs for videoconferencing using primary digital group transmission,"

ITU-T Rocommendation, 1993.

[31] "H.261 : Video codec for audiovisual services at p x 384 kbit/s - Recommendation H.261 (11/88)," ITU-T Recommendation, 1988.

[32] "H.262 : Information technology - Generic coding of moving pictures and associated audio information: Video," ITU-T Recommendation, 2012.

[33] "Information technology -- Generic coding of moving pictures and associated audio information -- Part 2: Video," ISO/IEC 13818-2:2013, 2013.

[34] "SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS: Infrastructure of audiovisual services – Coding of moving video, Video coding for low bit rate communication," ITU-T Recommendation H.263.

[35] G. Sullivan and T. Wiegand, "Video Compression—From Concepts to the H.264/AVC Standard," *Proceedings of the IEEE ,* vol. 93, no. 1, pp. 18-31, Jan 2005.

[36] "Information technology -- Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s -- Part 2: Video," ISO/IEC 11172-2:1993, 1993.

[37] G. H. Barry, P. Atul and N. N. Arun, Digital Video: An Introduction to MPEG-2, Massachusetts USA: Kluwer Academic Publishers, 1997.

[38] "Information technology -- Coding of audio-visual objects -- Part 2: Visual," ISO/IEC 14496-2:1999 , 1999.

[39] Phillipe Salembier, Thomas Sikora and B.S. Manjunath, Introduction to MPEG-7: Multimedia Content Description Interface, New York USA: John Wiley & Sons, Inc, 2002 .

[40] "VC-1 Compressed Video Bitstream Format and Decoding Process," SMPTE 421M.

[41] J. Loomis and M. Wasson, "VC-1 Technical Overview," Windows Media. Microsoft, Oct 2007.

[42] J. Bankoski, R. S. Bultje, et al, "Towards a next generation open-source video codec," *SPIE Visual Information Processing and Communication IV,* vol. 8666, Feb 2013.

[43] M. T. Alexis, L. Athanasios, S. Karsten and S. Gary, "H.264/14496-10 AVC Reference Software Manual (revised for JM 18.0) JVT-AE010," ITU-T/ISO/IEC JVT reference software (JM 18.0) manual, July 2009.

[44] "H.264/AVC Reference Software JM 18.0," [Online]. Available: http://iphome.hhi.de/suehring/tml/download/.

[45] B. Frank, "Common HM test conditions and software reference configurations," Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-L1100, Geneva, Jan. 2013.

[46] K. Il-Koo, M. Ken, S. Kazuo, B. Benjamin, H. Woo-Jin and S. Gary, "High Efficiency Video Coding (HEVC) Test Model 15 (HM15) Encoder Description," ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-Q1002, Valencia, Apr. 2014.

[47] "Recommended Test Sequences for HEVC HM Encoder," [Online]. Available: ftp://hevc@ftp.tnt.uni-hannover.de/testsequences/.

[48] S. Vivienne, B. Madhukar and J. S. Gary, High Efficiency Video Coding (HEVC): Algorithms and Architectures, 2014: Springer.

[49] M. T. Pourazad, C. Doutre, M. Azimi, and P. Nasiopoulos, "HEVC: the new gold standard for video compression," *IEEE Consumer Electronics Magazine,* vol. 1, no. 3, pp. 36-46, July 2012.

[50] J. Chen and T. Lee, "Planar intra prediction improvement," ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-F483, July

2011.

[51] H. Li, B. Li, L. Li, J. Zhang, H. Yang, and H. Yu, "Non-CE6: Simplification of intra chroma mode coding," ITUT/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document, JCTVC-H0326, Feb. 2012.

[52] J. Chen, V. Seregin, W.-J. Han, J. Kim, and J. Moon, "CE6.a.4: Chroma intra prediction by reconstructed luma samples," ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-E266, March 2011.

[53] E. Alshina, A. Alshin, J. Park, J. Lou and K. Minoo, "CE3: 7 taps interpolation filters for quarter pel position MC from Samsung and Motorola Mobility," ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document, Geneva, Nov. 2011.

[54] F. Bossen, B. Bross, K. Suhring and D. Flynn, "HEVC Complexity and Implementation Analysis," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 22, no. 12, pp. 1685-1696, Oct. 2012.

[55] A. Fuldseth, G. Bjøntegaard, and M. Budagavi, "CE10: Core transform design for HEVC," ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-G495, Nov. 2011.

[56] H. Jingning, A. Saxena and K. Rose, "Towards jointly optimal spatial prediction and adaptive transform in video/image coding," in *IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP 2010)*, Dallas TX, March 2010.

[57] A. C. Bovik, The essential guide to video processing, Academic Press - Elsevier, 2009.

[58] F. Chih-Ming, E. Alshina, A. Alshin, Y.-W. Huang, C.-Y. Chen, C.-Y. Tsai, C.-W. Hsu, S.-M. Lei, J.-H. Park and W.-J. Han, "Sample Adaptive Offset in the HEVC Standard," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 22, no. 12, pp. 1755 - 1764, Oct. 2012.

[59] T. Chia-Yang, C.-Y. Chen, C.-M. Fu, Y.-W. Huang and S. Lei, "One-pass encoding algorithm for adaptive loop filter in high-efficiency video coding," in *IEEE Visual Communications and Image Processing (VCIP 2011)*, Tainan, Nov. 2011.

[60] V. Sze and M. Budagavi, "High Throughput CABAC Entropy Coding in HEVC," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 22, no. 12, pp. 1778 - 1791, Oct. 2012.

[61] J. Lainema, K. Ugur and A. Hallapuro, "Single entropy coder for HEVC with a high throughput binarization mode," ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-G569, Geneva, Nov. 2011.

[62] J. Vanne, M. Viitanen and T. Hamalainen, "Efficient Mode Decision Schemes for HEVC Inter Prediction," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 24, no. 9, pp. 1579 - 1593, 2014.

[63] Z. Guo-Yun, H. Xiao-Hai, Q. Lin-Bo and L. Yuan, "Fast inter-mode decision algorithm for high-efficiency video coding based on similarity of coding unit segmentation and partition mode between two temporally adjacent frames," *SPIE Journal of Electronic Imaging,* vol. 22, no. 2, June 2013.

[64] H. Bing-Yu, H. Yu-Wen, W. Tu-Chih, C. Shao-Yi and C. Liang-Gee, "Fast motion estimation algorithm for H.264/MPEG-4 AVC by using multiple reference frame skipping criteria," *Proc. SPIE 5150, Visual Communications and Image Processing,* vol. 5150, pp. 1551-1560, 2003.

[65] I. Michal and P. Anandan, "About Direct Methods," in *Proceedings of the International Workshop on Vision Algorithms (ICCV-1999)*, London, UK, Springer-Verlag, 2000, p. 267–277.

[66] H. S. T. Philip and Z. Andrew, "Feature Based Methods for Structure and Motion Estimation," in *Proceedings of the International Workshop on Vision Algorithms (ICCV 1999)*, London, UK, Springer-Verlag, 2000, pp. 278-294.

[67] M. K. Peter, Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation, Kluwer Academic Publishers, 1999.

[68] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *IEEE Transactions on Image Processing,* vol. 4, no. 1, pp. 105-107, 1995.

[69] X. Q. Gao, J. D. C. and R. Z. C., "A multilevel successive elimination algorithm for block matching motion estimation," *IEEE Transactions Image Processing,* vol. 9, no. 3, pp. 501-504, 2000.

[70] B. Montrucchio and D. Quaglia, "New sorting-based lossless motion estimation algorithms and a partial distortion elimination performance analysis," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 15, no. 2, pp. 210 - 220, 2005.

[71] C. Changryoul and J. Jechang, "New sorting-based partial distortion elimination algorithm for fast optimal motion estimation," *IEEE Transactions on Consumer Electronics,* vol. 55, no. 4, pp. 2335 - 2340, 2009.

[72] A. Heinrich, C. Bartels, R. J. van der Vleuten, C. N. Cordes and G. de Haan, "Optimization of Hierarchical 3DRS Motion Estimators for Picture Rate Conversion," *IEEE Journal of Selected Topics in Signal Processing,* vol. 5, no. 2, pp. 262 - 274 , Aug. 2010.

[73] S. Tedmori and N. Al-Najdawi, "Hierarchical stochastic fast search motion estimation algorithm," *IET computer vision,* vol. 6, no. 1, pp. 21-28, 2012.

[74] M. Bierling, "Displacement estimation by hierarchical blockmatching," *Proc. SPIE 1001, Visual Communications and Image Processing '88: Third in a Series,* vol. 942, pp. 942-953, Oct. 1988.

[75] C.-M. Kuo, C. Shu-Chiang and S. Po-Yi, "Kalman filtering based rate-constrained motion estimation for very low bit rate video coding," *IEEE Transactions on Circuits and Systems for Video Technology, ,* vol. 16, no. 1, pp. 3-18, Jan. 2006.

[76] L. Hyungjun and G. A. Tae, "An hierarchical motion estimation method using adaptive image down-sizing," in *IEEE International Conference on Consumer Electronics (ICCE 2014)*, Las Vegas, NV, Jan. 2014.

[77] T. Koga, K. Iinuma, A. Hirano, Y. Iijima and T. Ishiguro, "Motion compensated interframe coding for video conferencing," in *Proc. of National Telecommunication Conference*, New Orleans, USA, 1981.

[78] W. Chung-Neng, Taiwan, Y. Shin-Wei, L. Chi-Min and C. Tihao, "A hierarchical decimation lattice based on N-queen with an application for motion estimation," *IEEE Signal Processing Letters,* vol. 10, no. 8, pp. 228 - 231, Aug. 2003.

[79] A. Saha, M. Jayanta and S. Shamik, "New pixel-decimation patterns for block matching in motion estimation," *Image Communication Signal Processing,* vol. 23, no. 10, pp. 725-738, Aug. 2008.

[80] L. Liu and E. Feig, "A block-based gradient descent search algorithm for block motion estimation in video coding," *IEEE Transactions on circuits and systems for Video Technology,* vol. 6, no. 4, pp. 419-422, Aug. 1996.

[81] Z. Shan and M. Kai-Kuang, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Transactions on Image Processing,* vol. 9, no. 2, pp. 287 - 290, Feb. 2000.

[82] J. R. Jain and A. K. Jain, "Displacement measurements and its application in interframe image coding," *IEEE Transaction on Communication,* vol. 29, no. 12, pp. 1799-1808., Dec. 1981.

[83] Z. Ce, L. Xiao and L.-P. Chau, "Hexagon-based search pattern for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 12, no. 5, pp. 349 - 355, May 2002.

[84] H. Ko. Y., H. S. Kang and S. W. Lee, "Adaptive search range motion estimation using neighboring motion vector differences," *IEEE Transactions on Consumer Electronics,* vol. 57, no. 2, pp. 726-730, May 2011.

[85] D. Wei, O. C. Au, L. Sijin, S. Lin and Z. Ruobing, "Adaptive search range algorithm based on Cauchy distribution," in *IEEE Visual Communications and Image Processing (VCIP-2012)*, San Diego, CA, Nov. 2012.

[86] L. Xiaobing and X. Chuangbai, "A new strategy to predict the search range in H.264/AVC," in *IEEE International Conference on Multimedia and Expo, (ICME 2009).*, New York, NY, June 2009.

[87] C. Zhibo, Z. Peng and H. Yun, "Fast motion estimation for JVT," ISO/IEC MPEG & ITU-T VCEG Joint Video Team (JVT) document JVTG016, Mar. 2003.

[88] C. Zhenxing, L. Qin, T. Ikenaga and S. Goto, "A motion vector difference based self-incremental adaptive search range algorithm for variable block size motion estimation," IEEE International Conference on Image Processing, (ICIP 2008), San Diego, CA, Oct. 2008.

[89] S. Zhiru, W. Fernando and A. Kondoz, "Adaptive Direction Search Algorithms based on Motion Correlation for Block Motion Estimation," *IEEE Transactions on Consumer Electronics,* vol. 57, no. 3, pp. 1354-1361, Aug. 2011.

[90] S. Zhiru, W. Fernando and A. Kondoz, "An Efficient Fast Motion Estimation in H.264/AVC by Exploiting Motion Correlation Character," in *IEEE International Conference on Computer Science and Automation Engineering (CSAE 2012)*, May 2012.

[91] K. Chung-Ming, K. Yu-Hsin, H. Chaur-Heh and L. Yi-Hui, "A Novel Prediction-Based Directional Asymmetric Search Algorithm for Fast Block-Matching Motion Estimation," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 19, no. 6, p. 893–899, June 2009.

[92] X. Xu and Y. He, "Comments on Motion Estimation Algorithms in Current JM Software," ITU-T/ISO/IEC Joint Video Team (JVT) document JVT-Q089, Nice, France, Oct. 2005.

[93] B. Xuena, Z. Dajiang, L. Peilin and S. Goto, "An Advanced Hierarchical Motion

Estimation Scheme With Lossless Frame Recompression and Early-Level Termination for Beyond High-Definition Video Coding," *IEEE Transactions on Multimedia,* vol. 14, no. 2, pp. 237-249, Oct. 2011.

[94] M. Sarwer and Q. Wu, "Adaptive Variable Block-Size Early Motion Estimation Termination Algorithm for H.264/AVC Video Coding Standard," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 19, no. 8, pp. 1196-1201, Apr. 2009.

[95] A. M. Tourapis, "Enhanced predictive zonal search for single and multiple frame motion estimation," in *Visual Communications and Image Processing (VCIP 2002)*, Jan. 2002.

[96] A. M. Tourapis, O. C. Au and M. L. Liou, "Predictive motion vector field adaptive search technique (PMVFAST): enhancing block-based motion estimation," *Photonics West 2001 - Electronic Imaging,* pp. 883-892, 2000.

[97] H. Schwarz, T. Hinz and K. Suehring, "H.264/AVC Multiview Refence Software JMVC Version 8.2 Reference Manual," ITU-T/ISO/IEC Joint Video Team JVT, May 2010.

[98] C. J. Duanmu, Z. Yu, C. Xing and Z. Shuihong, "Multi-octagon-grid search algorithm for fast motion estimation," in *International Conference on Information and Automation (ICIA 2008)*, June 2008.

[99] W. Zhu, X. Chen and X. Li, "A New Search Algorithm Based on Muti-Octagon-Grid," in *International Congress on Image and Signal Processing (CISP 2009)* , Oct. 2009.

[100] N. Ka-Ho, P. Lai-Man, W. Ka-Man, T. Chi-Wang and C. Kwok-Wai, "A Search Patterns Switching Algorithm for Block Motion Estimation," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 19, no. 5, pp. 753-759, Mar. 2009.

[101] N. Ka-Ho, P. Lai-Man and W. Ka-Man, "Search Patterns Switching for Motion

Estimation using Rate of Error Descent," in *IEEE International Conference on Multimedia and Expo (ICME 2007)*, July 2007.

[102] N. Ragasudha, D. Singh and S. Meher, "Block based Motion Estimation using hybrid hexagon kite cross diamond search algorithm," in *International Conference on Communications and Signal Processing (ICCSP 2014)*, Apr. 2014.

[103] H.-Y. C. Tourapis and A. M. Tourapis, "Fast motion estimation within the H.264 codec," in *International Conference on Multimedia and Expo. (ICME 2003)*, July 2003.

[104] Y. Xiaoquan, Z. Jun, L. Nam and S. Weijia, "Improved and simplified fast motion estimation for JM," ITU-T/ISO/IEC Joint Video Team (JVT) document JVT-P021, Poznan, Poland, July 2005.

[105] H. Yu-Wen, W. Tu-Chih, H. Bing-Yu and C. Liang-Gee, "Hardware architecture design for variable block size motion estimation in MPEG-4 AVC/JVT/ITU-T H.264," in *IEEE International symposium on circuits and systems (ISCAS 2003)*, Bangkok, Thailand, May 2003.

[106] T. Jen-Chieh, T. Hsinchu, C. Tian-Sheuan and J. Chein-Wei, "On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 12, no. 1, pp. 61 - 72, Jan. 2002.

[107] C. Tung-Chien, C. Shao-Yi, H. Yu-Wen, T. Chen-Han, C. Ching-Yeh, C. To-Wei and C. Liang-Gee, "Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 16, no. 6, pp. 673 - 688, June 2006.

[108] K. Chao-Yang, T. Hsinchu and L. Youn-Long, "A Memory-Efficient and Highly Parallel Architecture for Variable Block Size Integer Motion Estimation in H.264/AVC," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 18, no. 6, pp. 866 - 874, June 2009.

[109] O. Ndili and T. Ogunfunmi, "Efficient fast algorithm and FPSoC for integer and fractional motion estimation in H.264/AVC," in *IEEE International Conference on Consumer Electronics (ICCE 2011)*, Las Vegas, NV, Jan 2011.

[110] O. Ndili and T. Ogunfunmi, "Hardware-oriented Modified Diamond Search for motion estimation in H.246/AVC," in *IEEE International Conference on Image Processing (ICIP 2010)*, Hong Kong , Sept. 2010.

[111] M. Porto, L. Agostini, S. Bampi and A. Susin, "A high throughput and low cost diamond search architecture for HDTV motion estimation," in *IEEE International Conference on Multimedia and Expo (ICME 2008)* , 2008.

[112] O. Ndili and T. Ogunfunmi, "Algorithm and Architecture Co-Design of Hardware-Oriented, Modified Diamond Search for Fast Motion Estimation in H.264/AVC," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 21, no. 9, pp. 1214 - 1227, Mar. 2011.

[113] O. Ndili and T. Ogunfunmi, "FPSoC-based architecture for a fast motion estimation algorithm in H.264/AVC," *EURASIP journal on embedded systems,* vol. 2009, Oct. 2009.

[114] C. A. Rahman and B. Wael, "UMHexagonS Algorithm Based Motion Estimation Architecture For H.264/AVC," in *IEEE International Workshop on System-on-Chip for Real-Time Applications*, 2005.

[115] A. Juri and A. Jambek, "UMHexagonS based motion estimation architecture comparison," in *International Conference on Intelligent and Advanced Systems (ICIAS 2012)*, June 2012.

[116] B. Myung-Suk, S. Yil-Mi and C. Yong-Beom, "Hardware Architecture for Fast Motion Estimation in H.264/AVC Video Coding," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences,* Vols. E89-A, no. 6, pp. 1744-1745, Jun. 2006.

[117] J. Lee and K. Yoo, "Multi-algorithm targeted low memory bandwidth architecture

for H.264/AVC integer-pel motion estimation," *Proceedings of IEEE International conference on multimedia expo,* pp. 701-704, 2008.

[118] X. Xiong, S. Yang and A. Akoglu, "Architecture design of variable block size motion estimation for full and fast search algorithms in H. 264/AVC.," *Computers & Electrical Engineering,* vol. 37, no. 3, pp. 285-299, 2011.

[119] R. Verma and A. Ali, "A coarse grained and hybrid reconfigurable architecture with flexible NoC router for variable block size motion estimation," in *IEEE International Symposium on Parallel and Distributed Processing*, April 2008.

[120] V. Jarno, A. Eero, K. Kimmo and D. H. Timo, "A Configurable Motion Estimation Architecture for Block-Matching Algorithms," *IEEE Transactions On Circuits And Systems For Video Technology,* vol. 19, no. 4, pp. 466-477, April 2009.

[121] D. Anargyros, L. George and R. Dionysios, "Programmable Motion Estimation Architecture," in *IEEE International Conference on Electronics, Circuits, and Systems*, December 2009.

[122] J. L. Nunez-Yanez, A. Nabina, E. Hung and G. Vafiadis, "Cogeneration of Fast Motion Estimation Processors and Algorithms for Advanced Video Coding," *IEEE Transactions on Very Large Scale Integration Systems,* vol. 20, no. 3, pp. 437-448, March 2012.

[123] S. K. Raman, P. Vladimir and K. Jagannath, "Implementing streaming SIMD extensions on the Pentium III processor," *IEEE micro,* vol. 20, no. 4, pp. 47-57, 2000.

[124] S. D. Kim and M. H. Sunwoo, "MESIP: A Configurable and Data Reusable Motion Estimation Specific Instruction-Set Processor," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 23, no. 10, pp. 1767-1780, 2013.

[125] M. E. Sinangil, V. Sze, M. Zhou and A. P. Chandrakasan, "Cost and coding efficient motion estimation design considerations for high efficiency video coding

(HEVC) standard," *IEEE Journal of Selected Topics in Signal Processing,* vol. 7, no. 6, pp. 1017 - 1028, July 2013.

[126] S. Jou, S. Chang and T. Chang, "Fast Motion Estimation Algorithm And Design For Real Time QFHD High Efficiency Video Coding," *IEEE Transactions on Circuits and Systems for Video Technology,* no. 99, Jan. 2015.

[127] J. Byun, Y. Jung and J. Kim, "Design of integer motion estimator of HEVC for asymmetric motion-partitioning mode and 4K-UHD," *Electronics Letters,* vol. 49, no. 18, pp. 1142-1143, Aug. 2013.

[128] G. Sanchez, M. Porto and L. Agostini, "A hardware friedly motion estimation algorithm for the emergent HEVC standard and its low power hardware design," in *IEEE International Conference on Image Processing (ICIP 2013)*, Melbourne, Sept. 2013.

[129] Y. Xu, J. Liu, L. Gong, Z. Zhang and R. Teng, "A high performance VLSI architecture for integer motion estimation in HEVC," in *IEEE 10th International Conference on ASIC (ASICON 2013)*, Oct. 2013.

[130] "HM Reference Software 9.0," [Online]. Available: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/branches/HM-9.0-dev/.

[131] Y. T. Jo, R. Surendra, M. Ranganath and A. A. Kassim, "A novel unrestricted center-biased diamond search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 8, no. 4, pp. 369 - 377, Aug. 1998.

[132] S.-H. Yang, J.-Z. Jiang and H.-J. Yang, "Fast motion estimation for HEVC with directional search," *IET Electronic Letters,* vol. 50, no. 9, p. 673–675, April 2014.

[133] N. Parmar and M. H. Sunwoo, "Enhanced test zone search motion estimation algorithm for HEVC," in *International SoC Design Conference (ISOCC)*, Jeju, Nov. 2014.

[134] D. E. Thomas and P. R. Moorby, The Verilog® Hardware Description Language, Springer Science & Business Media, 2002.

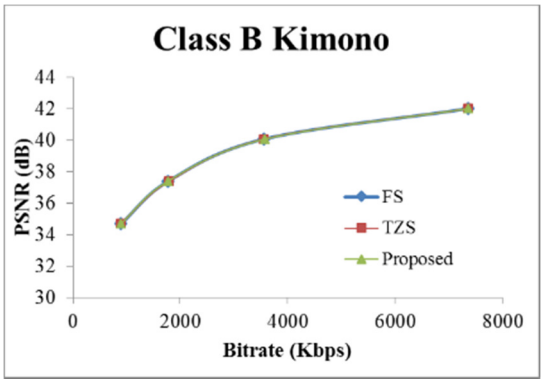
[135] "Xilinx Virtex-6 FPGA Configuration User Guide," Aug. 2014. [Online]. Available: http://www.xilinx.com/support /documentation/user_guides/ug360.pdf.

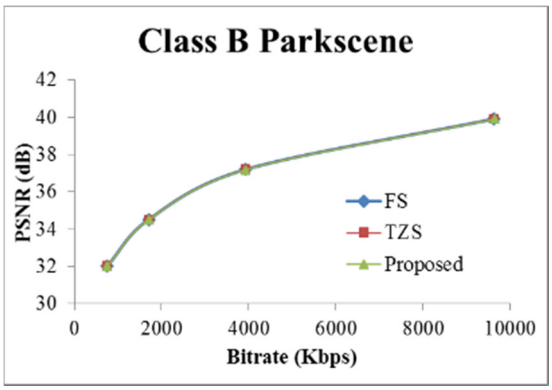
[136] M. E. Sinangil, A. P. Chandrakasan, V. Sze and M. Zhou, "Hardware-aware motion estimation search algorithm development for high-efficiency video coding (HEVC) standard," in *IEEE International Conference on Image Processing (ICIP 2012)*, Orlando, FL, Sept. 2012.

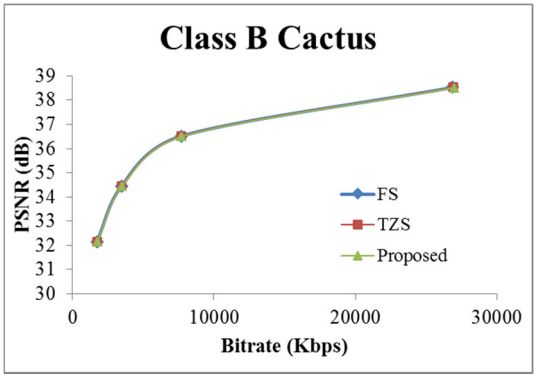
[137] D. Thomas, S. Momcilovic, F. Pratas and L. Sousa, "Reconfigurable data flow engine for HEVC motion estimation," in *IEEE International Conference on Image Processing (ICIP 2014)*, Paris, France, Oct. 2014.

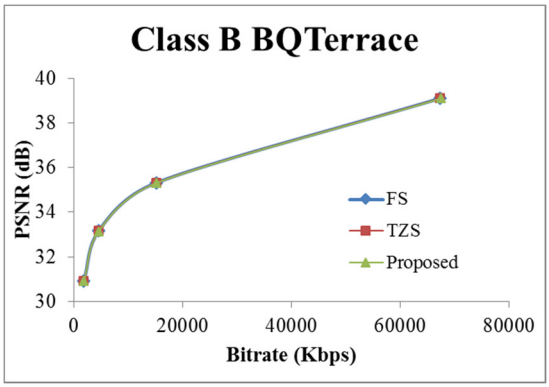
Annex A

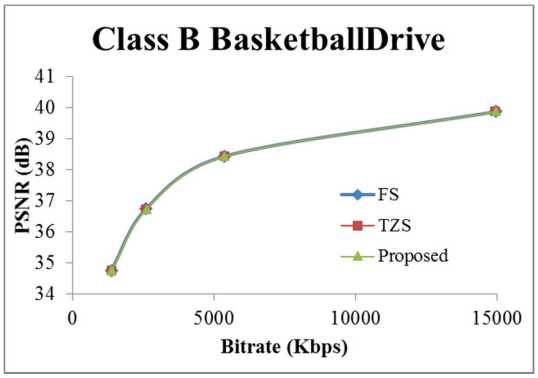
RD curves of the all the test sequences for Full Search, TZSearch and the proposed algorithm.

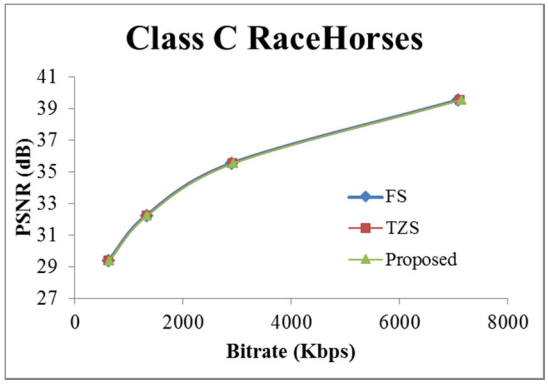


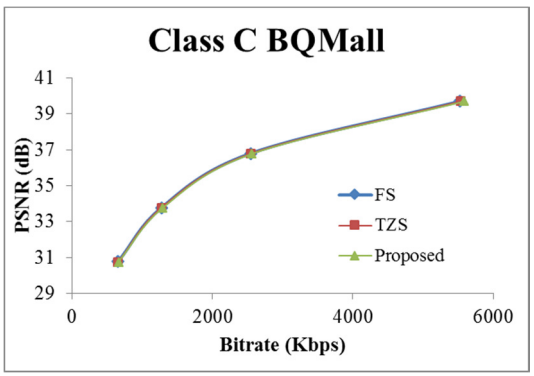


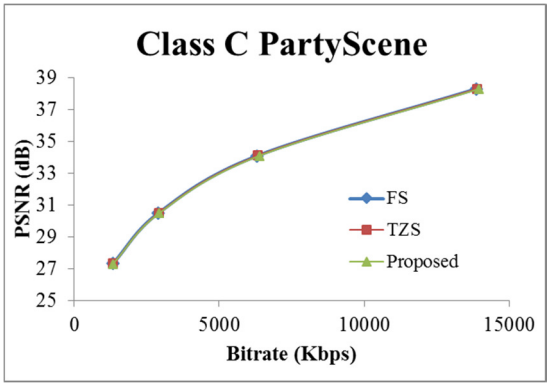


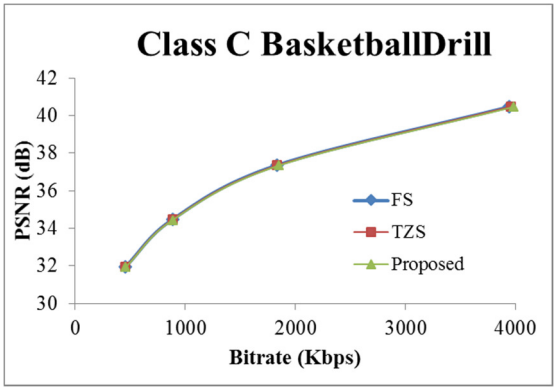


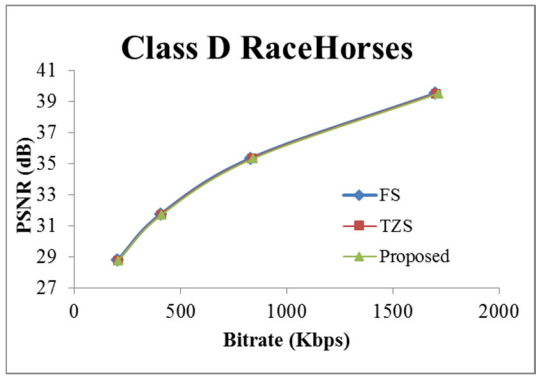


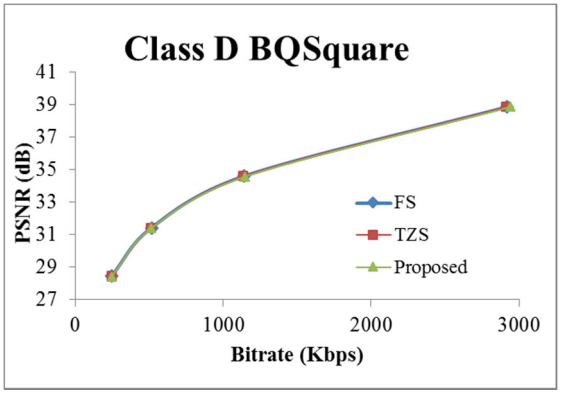


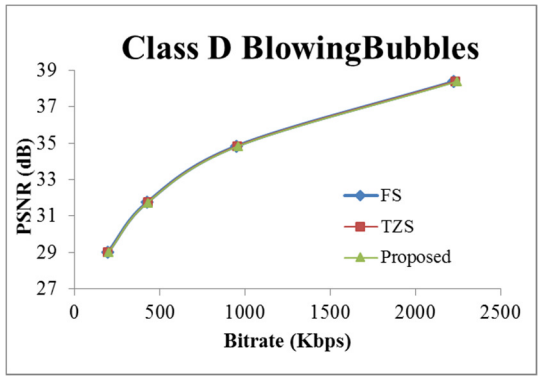


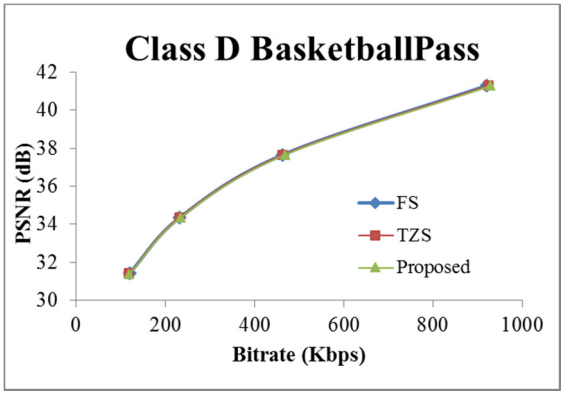


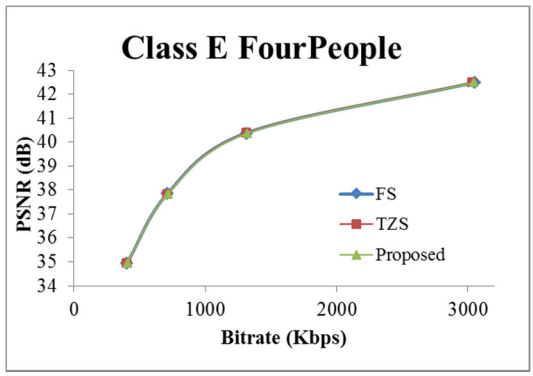

Class C BasketballDrill
PSNR (dB)
42
40
38
36
34
32
30
0
1000
2000
3000
4000
Bitrate (Kbps)
FS
TZS
Proposed

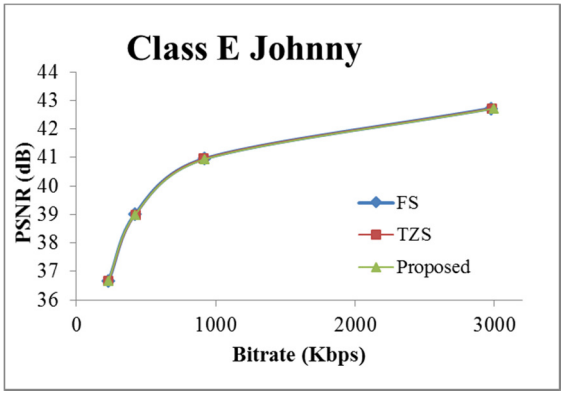

Class D RaceHorses
PSNR (dB)
41
39
37
35
33
31
29
27
0
500
1000
1500
2000
Bitrate (Kbps)
FS
TZS
Proposed

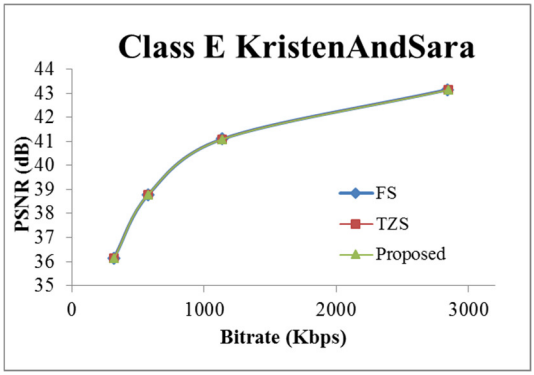

Class D BQSquare
PSNR (dB)
41
39
37
35
33
31
29
27
0
1000
2000
3000
Bitrate (Kbps)
FS
TZS
Proposed


Class D BlowingBubbles
PSNR (dB)
39
37
35
33
31
29
27
0
500
1000
1500
2000
2500
Bitrate (Kbps)
FS
TZS
Proposed


Class D BasketballPass
PSNR (dB)
42
40
38
36
34
32
30
0
200
400
600
800
1000
Bitrate (Kbps)
FS
TZS
Proposed


Class E FourPeople
PSNR (dB)
43
42
41
40
39
38
37
36
35
34
0
1000
2000
3000
Bitrate (Kbps)
FS
TZS
Proposed


Class E Johnny
PSNR (dB)
44
43
42
41
40
39
38
37
36
0
1000
2000
3000
Bitrate (Kbps)
FS
TZS
Proposed


Class E KristenAndSara
PSNR (dB)
44
43
42
41
40
39
38
37
36
35
0
1000
2000
3000
Bitrate (Kbps)
FS
TZS
Proposed