TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Jarno Vanne

Design and Implementation of Configurable Motion Estimation Architecture for Video Encoding

Julkaisu 982 • Publication 982

Tampere 2011

Tampereen teknillinen yliopisto. Julkaisu 982
Tampere University of Technology. Publication 982

Jarno Vanne

Design and Implementation of Configurable Motion Estimation Architecture for Video Encoding

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB109, at Tampere University of Technology, on the 23rd of September 2011, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2011

ISBN 978-952-15-2628-2 (printed)
ISBN 978-952-15-2679-4 (PDF)
ISSN 1459-2045

Abstract

A key factor behind the success of video products and services is video compression that makes digital video practical in the current communication networks and storage devices. However, video compression involves complex encoding algorithms whose real-time execution is particularly challenging in portable devices due to strict constraints on cost, size, and power consumption. In addition, modern devices are expected to be flexible enough to support plethora of video coding standards.

This Thesis focuses on motion estimation (ME) that is the main coding tool for removing temporal redundancy of video scenes and it typically accounts for 50 - 90% of the video encoding complexity. Due to these two reasons, hundreds of algorithms and architectures have been proposed for non-standardized ME. However, most of the current ME architectures are limited to a single video coding standard or ME algorithm, whereas existing flexible architectures are realized with large silicon area, limited processing speed, unsustainable power budget, or over restricted ME parameters.

The main contribution of this Thesis is a novel ME architecture that overcomes all the crucial limitations faced by the contemporary approaches. The designed and implemented hardware architecture is compatible with all inter coding modes of H.261/3, MPEG-1/2, MPEG-4 Visual, H.264/AVC, and VC-1 standards. In addition, it can perform rate-constrained integer ME (IME) with various fast ME algorithms such as BBGDS, CDS, DS, HEXBS, and TSS. It also reduces the complexity of the subsequent fractional ME (FME) by conducting mode decision jointly with IME. The flexibility of the architecture is based on a parametrizable search strategy control and associated separable search path generation through which different ME algorithms and inter coding modes are easily implemented and efficiently executed. The architecture is composed of the three accurately optimized and seamlessly coupled components: a control unit, a memory system, and distortion computation unit.

The results illustrate that the architecture can process real-time (30 fps) single reference frame ME up to 1080p format (1920×1080 pixels) with H.261/3, MPEG-1/2, MPEG-4 Visual, and VC-1 standards. When processing CIF, D1, and 1080p formats at 30 fps, the architecture consumes only 22.3 - 25.1 kgates, 20.5 KB of memory with 123×123 pixel search range, and 3 - 184 mW of power with a 0.13-micrometer CMOS standard cell technology. Supporting ME for H.264/AVC 1080p video at 30 fps requires a duplicated architecture whose respective metrics are 49.7 kgates, 41.0 KB of memory, and 364 mW of power. The performance comparison shows that the designed architecture consumes 39 - 89% less logic gates than the existing approaches.

Preface

The research work for this Thesis has been carried out in the Department of Computer Systems at Tampere University of Technology during the years 2003 - 2011.

I would like to express my gratitude to my supervisor Prof. *Timo D. Hämäläinen* for his guidance, motivation, and successful efforts that have made this research work possible. Sincere acknowledgements go also to Prof. *Holger Blume* and Prof. *Janne Heikkilä* for reviewing and providing constructive comments on the manuscript of this Thesis.

I would like to thank all my colleagues for assistance, fruitful discussions, and inspiring atmosphere. I express my special thanks to *Kimmo Kuusilinna,* Dr. Tech., for co-authoring publications and providing thorough guidance especially in the first years of my academic career. In addition, I am deeply grateful to my closest colleague *Eero Aho,* Dr. Tech., who has co-authored publications and given valuable support during these years. Special thanks also to Mr. *Marko Viitanen*, *Olli Lehtoranta*, Dr. Tech., *Antti Rasmus*, M.Sc., *Ari Kulmala*, Dr. Tech, *Juha Pirttimäki*, M.Sc., and *Tero Arpinen*, M.Sc., for their technical and less technical contribution to my research work. In addition, I would like to thank the deceased Mr. *Jussi Uusitalo* whose promising research career was over far too early.

This Thesis was financially supported by Tampere Graduate School in Information Science and Engineering (TISE), Academy of Finland, Nokia Foundation, Emil Aaltonen Foundation, Heikki and Hilma Honkanen Foundation, and HPY foundation. They are all gratefully acknowledged.

Finally, I would like to thank my parents *Riitta* and *Arto* for all their support during my whole life and my parents-in-law *Liisa* and *Erkki* for assistance during busy times. The warmest thanks are dedicated to my wife *Tarja* for her endless love, understanding, and support and my daughter *Nelli* for giving me so much fun and happiness every day. You two are the sunshines of my life.

Kangasala, August 2011

Jarno Vanne

Table of Contents

List of Publications

This Thesis consists of an introductory part and a part containing the following publications. In the text, these publications are referred to as [P1], [P2], and [P3].

[P1] J. Vanne, E. Aho, T. D. Hämäläinen, and K. Kuusilinna, "A high-performance sum of absolute difference implementation for motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 7, Jul. 2006, pp. 876-883.

[P2] J. Vanne, E. Aho, T. D. Hämäläinen, and K. Kuusilinna, "A parallel memory system for variable block-size motion estimation algorithms," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, no. 4, Apr. 2008, pp. 538-543.

[P3] J. Vanne, E. Aho, K. Kuusilinna, and T. D. Hämäläinen, "A configurable motion estimation architecture for block-matching algorithms," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 4, Apr. 2009, pp. 466-476.

List of Figures

List of Tables

Abbreviations

1080p	Video format (1920×1080 pixels, progressive scan)
720p	Video format (1280×720 pixels, progressive scan)
1D, 2D, 3D	One-, Two-, Three-Dimensional
3DTV	Three-Dimensional Television
4SS	Four-Step Search
ABS	Absolute
ADSL	Asymmetric Digital Subscriber Line
ALUT	Adaptive Look-Up Table of Altera
ARM	Advanced RISC Machine
ASIC	Application-Specific Integrated Circuit
ASP	Advanced Simple Profile of MPEG-4 Visual
AVC	Advanced Video Coding
AVS	Audio Video coding Standard of China
B	Bidirectional prediction
BBGDS	Block-Based Gradient Descent Search
BD-PSNR	Bjøntegaard Delta Peak Signal-to-Noise Ratio
BD-rate	Bjøntegaard Delta bit rate
BMA	Block-Matching Algorithm
BP	Baseline Profile of H.264/AVC
CABAC	Context-Adaptive Binary Arithmetic Coding
CAVLC	Context-Adaptive Variable Length Coding
CBEA	Cell Broadband Engine Architecture
CD	Compact Disc
CDS	Cross-Diamond Search
CIF	Common Intermediate Format (352×288 pixels)
CMOS	Complementary Metal Oxide Semiconductor

CPU	Central Processing Unit
CRT	Cathode Ray Tube
CS	Cross Search
CSA	Carry-Save Adder
CSP	Cross-shaped Search Pattern
D1	Video format (720×576 pixels for PAL)
DCT	Discrete Cosine Transform
DE	Disparity Estimation
DF	Deblocking Filter
DFD	Displaced Frame Difference
DFD’	Reconstructed Displaced Frame Difference
DPB	Decoded Picture Buffer
DS	Diamond Search
DSP	Digital Signal Processor
DVB-C, -S, -T	Digital Video Broadcasting, -Cable, -Satellite, -Terrestrial
DVD	Digital Versatile Disc
EC	Entropy Coding
EPZS	Enhanced Predictive Zonal Search
FBSME	Fixed Block-Size Motion Estimation
FFS	Fast Full-Search
FIR	Finite Impulse Response
FME	Fractional Motion Estimation
FPGA	Field Programmable Gate Array
fps	Frames per second
FS	Full-Search
FTV	Free viewpoint Television
GIPS	Giga Instructions Per Second
GFLOPS	Giga Floating Point Operations Per Second
GPP	General-Purpose Processor
GPU	Graphics Processing Unit
H.261, H.262, H.263	Video coding standards issued by ITU-T
H.264/AVC	Video coding standard issued by ITU-T and ISO/IEC

HD	High Definition
HDTV	High-Definition Television
HEVC	High Efficiency Video Coding
HEXBS	Hexagon-Based Search
HIBI	Heterogeneous IP Block Interconnection
HiP	High Profile of H.264/AVC
HVS	Human Visual System
HW	Hardware
I	Intra
ICT	Integer Cosine Transform
IEC	International Electrotechnical Commission
IEEE	The Institute of Electrical and Electronics Engineers
IME	Integer Motion Estimation
INV	Inversion
IP	Intra Prediction, also Internet Protocol
IPP	Integrated Performance Primitives
IPTV	Internet Protocol Television
ISDN	Integrated Services Digital Network
ISO	International Organization for Standardization
ITRS	International Technology Roadmap for Semiconductors
ITU	International Telecommunication Union
ITU-T	Telecommunication Standardization Sector of ITU
JCT-VC	Joint Collaborative Team on Video Coding
JM	Joint Model of H.264/AVC
JVT	Joint Video Team
LCD	Liquid Crystal Display
LDSP	Large Diamond Search Pattern
LSB	Least Significant Bit
LUT	Look-Up Table
MAD	Mean Absolute Difference
MAE	Mean Absolute Error
MB	Macroblock

MBD	Minimum Block Distortion
MC	Motion Compensation
MCP	Motion-Compensated Prediction
ME	Motion Estimation
MPEG	Moving Picture Experts Group
MPEG-1, -2, -4	Video coding standards issued by ISO/IEC
MRFME	Multiple Reference Frame Motion Estimation
MP	Main Profile of MPEG-2
MPSoC	Multiprocessor System-on-Chip
MSB	Most Significant Bit
MSD	Mean Squared Difference
MSE	Mean Squared Error
MV	Motion Vector
MVC	Multiview Video Coding, Annex H of H.264/AVC
MVD	Motion Vector Difference
MVP	Motion Vector Predictor
NoC	Network-on-Chip
NRE	Non-Recurring Engineering
NTSC	National Television System Committee
P	Prediction
P4EE	Pentium 4 Extreme Edition
PAL	Phase Alternate Line
PC	Personal Computer
PE	Processing Element
PMVFAST	Predictive MV Field Adaptive Search Technique
PPE	Power Processor Element of CBEA
PSNR	Peak Signal-to-Noise Ratio
Q	Quantization
Q^{-1}	Inverse Quantization
QCIF	Quarter Common Intermediate Format (176×144 pixels)
QFHD	Quad Full High Definition format (3840×2160 pixels)
QP	Quantization Parameter

QVGA	Quarter Video Graphics Array format (320×240 pixels)
WQHD	Wide Quad High Definition format (2560×1440 pixels)
RC	Rate Control
RD	Rate-Distortion
RDO	Rate-Distortion Optimization
RGB	Red Green Blue color space
RISC	Reduced Instruction Set Computer
RM	Reference Model of H.261
ROS	Region Of Support
SAD	Sum of Absolute Differences
SATD	Sum of Absolute Transformed Differences
SAE	Sum of Absolute Errors
SD	Standard Definition
SDTV	Standard-Definition Television
SEA	Successive Elimination Algorithm
SECAM	Sequential Color with Memory
SIF	Source Input Format (352×240 pixels for NTSC)
SIMD	Single Instruction Multiple Data
SDSP	Small Diamond Search Pattern
SMPTE	Society of Motion Picture and Television Engineers
SoC	System-on-Chip
SP	Simple Profile of MPEG-2 or MPEG-4 Visual
SPE	Synergistic Processor Element of CBEA
SRAM	Static Random Access Memory
SSD	Sum of Squared Differences
SSE	Sum of Squared Errors
Sub-QCIF	Sub-Quarter Common Intermediate Format (128×96 pixels)
SVC	Scalable Video Coding, Annex G of H.264/AVC
SW	Software
T	Transform
T^{-1}	Inverse Transform
TC	Transform Coding

TCOEFF	Transform Coefficient
TI	Texas Instruments
TM	Test Model of MPEG-1/2
TMN	Test Model Near-term of H.263
TSS	Three-Step Search
TV	Television
UHDTV	Ultra High Definition Television (7680×4320 pixels)
UMHexagonS	Unsymmetrical-cross Multi-Hexagon-grid Search
UVLC	Universal Variable Length Coding
VBS	Variable Block-Size
VBSME	Variable Block-Size Motion Estimation
VC-1	Video Codec-1, informal name of the SMPTE 421M video coding standard
VCD	Video Compact Disc
VCEG	Video Coding Experts Group
VGA	Video Graphics Array format (640×480 pixels)
VHDL	VHSIC Hardware Description Language
VHS	Video Home System
VHSIC	Very High Speed Integrated Circuit
VLC	Variable Length Coding
VLSI	Very Large Scale Integration
VM	Verification Model of MPEG-4 Visual
VoD	Video on Demand
WMV-9	Windows Media Video 9 codec
XP	Extended Profile of H.264/AVC
YCbCr	Luminance, Blue chrominance, Red chrominance color space
YUV	Color space

Symbols

$\Delta r_\alpha(\Delta ri_\alpha, \Delta rj_\alpha)$	Initial offset of r_{RR} in the 2D search area memory
$\Delta r_\beta(\Delta ri_\beta, \Delta rj_\beta)$	Prediction-based offset of r_{RR} in the 2D search area memory
$\Delta r_\delta(\Delta ri_\delta, \Delta rj_\delta)$	Checking point offset of r_{RR} in the 2D search area memory
$\Delta r_\varepsilon(\Delta ri_\varepsilon, \Delta rj_\varepsilon)$	Base block offset of r_{RR} in the 2D search area memory
$\Delta r_\varepsilon^{\mathrm{dlvr}}(\Delta ri_\varepsilon^{\mathrm{dlvr}}, \Delta rj_\varepsilon^{\mathrm{dlvr}})$	Base block offset of r_{RR}^{dlvr} in the 2D search area memory
$\Delta r_\chi(\Delta ri_\chi, \Delta rj_\chi)$	BMA movement offset of r_{RR} in the 2D search area memory
η	Index of the partition
η^*	Index of the best matching block
Λ	Sampling grid (orthogonal lattice)
λ	Lagrange multiplier
λ_{MD}	Lagrange multiplier in mode decision
λ_{ME}	Lagrange multiplier in ME
ρ	Number of the reference frames
σ	4×4 pixel block index of the MB
σ_η^ψ	The first 4×4 pixel block index of p_η^ψ
$\sigma_{\eta^*}^{\psi^*}$	The first 4×4 pixel block index of $p_{\eta^*}^{\psi^*}$
σ_x	Bit x of σ
ψ	Index of the inter coding mode
ψ^*	Index of the best ψ
ψx	Index of the inter coding mode in left (x = 1), top-left (x = 2), top (x = 3), and top-right (x = 4) candidate of MVP_η^ψ
D	Distortion between a current p_η^ψ and its reconstruction
$d_{k-1}^k(di_{k-1}^k, dj_{k-1}^k)$	Linear displacement vector in the 2D field (from F_{k-1} to F_k)

$d_k^{k-1}(di_k^{k-1}, dj_k^{k-1})$	Linear displacement vector in the 2D field (from F_k to $F_{k\text{-}1}$)
d_{CI}	Current frame data (current block memory input data)
d_{CO}	Current block data (current block memory output data)
d_{RI}	Reference frame data (search area memory input data)
d_{RO}	Search area data (search area memory output data)
d_{RO}^*	The best matching block (search area memory output data)
E_k	Error of the pixel(s) predicted for F_k
F_C	Current frame
F_k	The *k*th frame of a video sequence
F_R	Reconstructed reference frame(s)
h	Height of the search area (in pixels)
H^d	Diagonally interpolated ½-pixel value
H^h	Horizontally interpolated ½-pixel value
H^v	Vertically interpolated ½-pixel value
I	Integer-pixel value
i	Horizontal pixel coordinate of the frame
i_k	Horizontal pixel coordinate in F_k
J	RD optimized cost function
J^{ψ}	RD cost of m^{ψ}
$J^{\psi *}$	RD cost of $m^{\psi *}$
J_{η}^{ψ}	RD cost of p_{η}^{ψ}
$J_{\eta *}^{\psi}$	RD cost of $p_{\eta *}^{\psi}$
$J_{\eta *}^{\psi *}$	RD cost of $p_{\eta *}^{\psi *}$
$J_{16\times 16}^{\psi}$	RD cost of a MB
$J_{8\times 8}^{\psi}$	RD cost of a sub-MB
J_{tmp}^{ψ}	Unfinished $J_{16\times 16}^{\psi}$ or $J_{8\times 8}^{\psi}$ value
j	Vertical pixel coordinate of the frame
j_k	Vertical pixel coordinate in F_k
K	Number of the frames in a video sequence

k	Index of a frame in a video sequence
L_x	Quality level x of the ME architecture
L_x^y	ME architecture configuration for format y and quality level x
$L0_x^{y/2}$	The first instance of the dual-instance L_x^y
$L1_x^{y/2}$	The second instance of the dual-instance L_x^y
MV(MVi, MVj)	MV in 2D F_k
$\mathrm{MV}_\eta^\psi(\mathrm{MV}i_\eta^\psi, \mathrm{MV}j_\eta^\psi)$	MV of p_η^ψ in the 2D search area (a top-left corner of p_η^ψ)
$\mathrm{MV}_{\eta*}^\psi(\mathrm{MV}i_{\eta*}^\psi, \mathrm{MV}j_{\eta*}^\psi)$	MV of $p_{\eta*}^\psi$ in the 2D search area (a top-left corner of $p_{\eta*}^\psi$)
$\mathrm{MV}_{\eta*}^{\psi*}(\mathrm{MV}i_{\eta*}^{\psi*}, \mathrm{MV}j_{\eta*}^{\psi*})$	MV of $p_{\eta*}^{\psi*}$ in the 2D search area (a top-left corner of $p_{\eta*}^{\psi*}$)
$\mathrm{MV1}_{\eta x}^{\psi x}$	Left (x = 1), top-left (x = 2), top (x = 3), and top-right (x = 4) candidate of MVP_η^ψ
$\mathrm{MVD}_\eta^\psi(\mathrm{MVD}i_\eta^\psi, \mathrm{MVD}j_\eta^\psi)$	MVD for p_η^ψ in the 2D search area
$\mathrm{MVP}_\eta^\psi(\mathrm{MVP}i_\eta^\psi, \mathrm{MVP}j_\eta^\psi)$	MVP for p_η^ψ in the 2D search area
m^ψ	Inter coding mode ψ
$m^{\psi*}$	The best m^ψ
$m^{\psi x}$	Inter coding mode ψx in left (x = 1), top-left (x = 2), top (x = 3), and top-right (x = 4) candidate of MVP_η^ψ
n^ψ	The number of partitions in m^ψ
P_k	Predicted pixel(s) for F_k
p_η^ψ	Partition η of m^ψ
$p_{\eta*}^\psi$	The best matching block for the current p_η^ψ
$p_{\eta*}^{\psi*}$	Partition η of $m^{\psi*}$
p_h	Horizontal search range in the positive/negative direction
p_w	Vertical search range in the positive/negative direction
Q	Height of the frame (in pixels)
q	Horizontal pixel coordinate of p_η^ψ
Q^ψ	Height of p_η^ψ (in pixels)
$Q^{\psi*}$	Height of $p_{\eta*}^{\psi*}$ (in pixels)

Q^d	Diagonally interpolated ¼-pixel value
Q^h	Horizontally interpolated ¼-pixel value
Q^v	Vertically interpolated ¼-pixel value
R	Bit count of the encoded prediction error
R^{ψ}	$R^{\psi}_{16\times16}$ or $R^{\psi}_{8\times8}$
$R^{\psi}_{16\times16}$	Bit count of the MB header
$R^{\psi}_{8\times8}$	Bit count of the sub-MB header
$r_{CR}(ri_{CR}, rj_{CR})$	Scanning point to read d_{CO} from the 2D current block memory
$r_{CW}(ri_{CW}, rj_{CW})$	Scanning point to write d_{CI} into the 2D current block memory
$R_{\mathrm{MV}}\ (R_{\mathrm{MV}i}, R_{\mathrm{MV}j})$	Bit count of the $\mathrm{MVD}^{\psi}_{\eta}$
$r_{RR}(ri_{RR}, rj_{RR})$	Scanning point to read d_{RO} from the 2D search area memory
$r^{\mathrm{dlvr}}_{RR}(ri^{\mathrm{dlvr}}_{RR}, rj^{\mathrm{dlvr}}_{RR})$	Scanning point to read d^{*}_{RO} from the 2D search area memory
$r_{RW}(ri_{RW}, rj_{RW})$	Scanning point to write d_{RI} into the 2D search area memory
s	Index of the sub-MB
t_k	Discrete time instant k
U	Width of the frame (in pixels)
U^{ψ}	Width of p^{ψ}_{η} (in pixels)
$U^{\psi*}$	Width of $p^{\psi*}_{\eta*}$ (in pixels)
u	Vertical pixel coordinate of p^{ψ}_{η}
w	Width of the search area (in pixels)

1. Introduction

The evolution of digital technology has made possible to process, transmit, and store video in a digital form. In the past two decades, *digital video* has penetrated a wide range of industries, especially in the area of entertainment, communications, and broadcasting. The widespread applications of digital video include products in consumer electronics, digital cinema, video on demand (VoD), video conferencing, digital television, and surveillance. These products and services have huge revenue potential since the amount of end users increases continuously. Currently, online video communities have reached over one billion users [22] who, among others, watch two billion Youtube videos daily [139]. In addition, the shortened product life cycle drives consumers and companies to replace their obsolete digital products with newer ones. For example, feature phones are often replaced with smartphones that are forecast to be sold 2.5 billion units during the years 2010 - 2015 with the annual growth rate of 24% [24]. To ensure continuous growth in the long term, digital video industry invests a lot in the research and development of video technology.

In the video applications, the vast amount of video data is the major challenge for efficient video storage and communication. Therefore, *digital video compression* (*video coding*) has been actively developed by researches, companies, and standard bodies since the 1980s [54]. The purpose of video compression is to represent a digital video with reduced amount of data but still with acceptable visual quality.

Video compression involves two processes: a *video encoder* encodes (compresses) the original video material for storage and transmission after which the encoded video is decoded (decompressed) by a *video decoder* back to the displayable video before playback and editing. In *two-way applications* such as video communication, both ends of the system encode and decode videos. Instead, in *one-way applications* such as broadcast and streaming, only one end encodes and the other end decodes videos. International standardization work ensures that these applications can interoperate between different platforms, storage devices, and communication networks. The well-known *video coding standards* include former H.261 [55], MPEG-1 [49], MPEG-2 [50], H.263 [56], and MPEG-4 Visual [51] as well as state-of-the-art H.264/AVC [57] and VC-1 [109].

Video compression makes digital video practical in the current communication networks and storage devices that have to cope with steadily increasing volume of digital material. According to Cisco [22], *global IP (Internet Protocol) traffic* was 14.7 exabytes (14.7×10^{18} bytes) per month (EB/month) in 2009 and it is estimated to reach 63.9 EB/month in 2014 with the annual growth rate of 34%. The majority of the global IP traffic is and will be consumer video data that is produced by households, university populations, and Internet cafes. Figure 1.1 depicts a global consumer traffic forecast and portions of the popular video services. The forecast assumes that 7% annual gain will be attained in video compression.

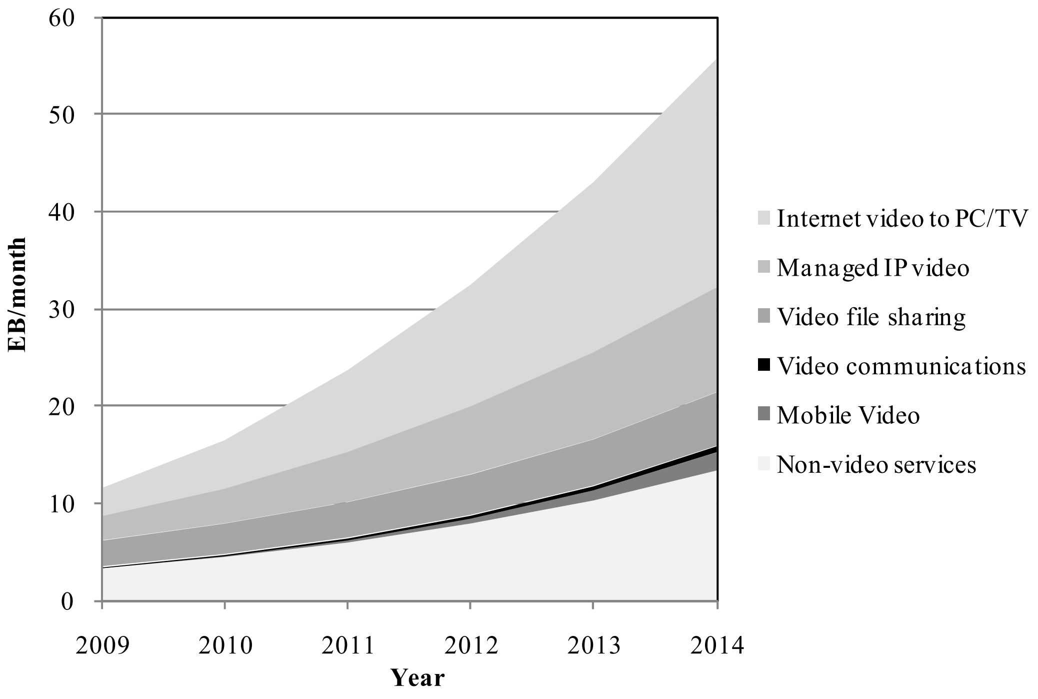


Figure 1.1. Consumer IP traffic forecast [22].

In 2014, the overall *consumer video traffic* will be over five times higher that it was in 2009. Actually, it will be over 3.5 times the whole *consumer IP traffic* in 2009. The largest traffic type, *Internet video to PC/TV*, includes all free TV, pay TV, and VoD data delivered via Internet to PC or TV. It will increase eightfold from 2009 to 2014. *Managed IP video* represents traffic generated under control of TV service providers (IPTV and cable TV). The amount of managed IP video data will quadruple from 2009 to 2014. Traffic caused by the exchange of video files is covered by *video file sharing*, whose amount will almost triple during the examined period. It is estimated that video file sharing accounts for 70 - 80% of the whole file sharing traffic. The smallest fraction is *video communications* which includes services like Internet video calling, video monitoring, and webcams. It will increase sevenfold from 2009 to 2014. Due to popularization of smartphones and other portables, *mobile video* traffic will be 70 times higher and it will account for almost 66% of the overall mobile data traffic by 2014 [23]. In 2014, the portion of *non-video traffic* (web browsing, email, instant messaging, file transfer, Internet gaming, etc.) will be below 20%.

There are several drivers for the growth of digital video material. Firstly, digital video capture and playback are nowadays standard features in numerous personal and professional electronic products. The price erosion has and will accelerate sales of these products. Secondly, the expansion of digital screens together with user demand for better quality drive video resolutions to grow from *standard definition* (*SD*) to *high definition* (*HD*) and beyond. Finally, the proliferation of *3D applications* such as 3DTV will further augment the amount of video data since several video signals are needed per a single 3D video sequence. Compared to 2009, HD and 3D traffic are assumed to be 23-fold in 2014 [22].

Advances of the underlying transmission, storage, and processing technologies ease proliferation of high-resolution video content. For example, the global average broadband speed is assumed to quadruple from 2009 to 2014 [22] and the similar growth rate is expected in the amount of logic and main memory of the portable devices [48]. Since advances in technology cannot completely compensate the explosion of video traffic, efficient *video coding algorithms* have been and will be needed [34], [92]. For example, H.264/AVC and VC-1 have made high quality video services possible. However, the new algorithms typically improve coding efficiency at a cost of higher computational complexity.

Real-time video coding is extremely sensitive to complexity increase, since the processing result has to appear without user-perceivable delay once the input becomes available [6], [64]. The importance of real-time video coding will emphasize in the future [22] and the modern devices are expected to be compatible with plethora of video coding standards. These two requirements are particularly challenging for portable devices which also have to satisfy strict constraints on cost, size, and power consumption.

A standard-compliant video encoder is usually about 5 - 10 times more complex than a respective video decoder and the complexity ratio can grow up to two orders of magnitude if all encoder options are applied [68], [96]. The encoder/decoder complexity ratio has significantly augmented with state-of-the-art standards. For example, H.264/AVC encoder is at least ten times more complex than MPEG-4 Visual encoder, but respective decoder complexity has increased only by a factor of two [96]. The encoder/decoder complexity gap is mainly caused by a single encoder tool: block-based *motion estimation* (*ME*). ME is only included in the encoder and its complexity accounts for 50 - 90% of the total encoder complexity [11], [44], [68]. Hence, ME is the main challenge for implementing real-time encoding with low cost and low power consumption.

The video encoder utilizes ME in *motion compensated prediction* (*MCP*) to reduce temporal redundancy of video sequences. MCP has been included in all international video coding standards. However, ME is excluded from these standards, so it is open for competition. Due to its extensive complexity and importance in video encoding, hundreds of *architectures* and *algorithms* have been proposed for ME since its introduction in 1981 [58].

Majority of the *real-time ME architecture* proposals have been implemented in *hardware* (*HW*) instead of more flexible *software* (*SW*) in order to meet real-time requirements in allowed power budget [6], [11], [68]. Introduced ME algorithms have mainly concentrated on speeding-up the original *exhaustive ME algorithm*, whose estimation strategy produces excellent compression performance but at a cost of intensive computation [42], [68]. With a specific motion content (low, medium, or high) and resolution, single *fast ME algorithms* have proved to be competitive with the original exhaustive ME. However, their compression efficiency decreases more clearly below the exhaustive ME when diverse motion contents and resolutions are processed [41]. In addition, most of the fast ME algorithms have been considered to complicate HW implementation.

1.1 Objective and scope of research

The scope of this Thesis is on MCP in video encoding. The coding tools and options of MCP are examined in the context of the following underlying video coding standards: H.261/3, MPEG-1/2, MPEG-4 Visual, H.264/AVC, and VC-1. The emphasis is on state-of-the-art standards, especially in H.264/AVC. Among the coding tools of MCP, this Thesis primarily focuses on *integer ME* (*IME*) and its practical HW implementation in portable devices. The proposed techniques are particularly designed for progressively scanned natural videos.

The problem addressed in this Thesis is how to overcome the computational complexity of ME. Most of the contemporary ME architectures are tailored to the exhaustive ME algorithm whose inherent complexity requires lots of HW resources. On the other hand, fast ME algorithms are typically implemented with dedicated *algorithm-specific architectures*, whose compression performance is vulnerable to variation of video resolutions and contents.

The objective of this Thesis is to illustrate that the best trade-off between HW complexity and ME compression performance is reached with a *configurable ME architecture* that can adaptively select the most appropriate fast ME algorithm at run time. Another goal is to ensure that the same architecture can be applied to all well-known video coding standards without noticeable overheads. In the literature, a couple of configurable HW approaches have been presented before, but they contain standard-specific limitations. In addition, their flexibility is realized at a cost of large silicon area, limited processing speed, unsustainable power budget, or over restricted ME parameters.

The main claim of this Thesis is that a single configurable HW architecture is able to support multiple video coding standards and various fast ME algorithms at adequate processing speed, low cost, and acceptable power consumption. The claim is proved by the presented configurable ME architecture that overcomes all the critical limitations of the existing flexible ME architectures.

1.2 Main contributions

The main contribution of this Thesis is the design and implementation of configurable ME architecture for video encoding. The architecture can be divided into three main components: a *control unit*, a *memory system*, and a *distortion computation unit*. In summary, the most significant contributions are:

1. The overall ME architecture that performs *rate-constrained IME* with fast ME algorithms such as BBGDS, CDS, DS, HEXBS, and TSS. The architecture is configurable to different ME algorithms and *inter coding modes* at run time. It achieves low area cost by reusing the HW as much as possible. It also reduces the complexity of the subsequent *fractional ME* (*FME*) by conducting *inter mode decision* jointly with IME. To the best of Author's knowledge, the designed HW architecture is the first one that supports fast rate-constrained IME algorithms and

is compatible with all inter coding modes of H.261/3, MPEG-1/2, MPEG-4 Visual, H.264/AVC, and VC-1 standards.

2. The *ME framework* behind the joint rate-constrained IME and inter mode decision. The framework replaces fixed search strategies of the individual ME algorithms with a single *generic search strategy* that is parametrizable to algorithm-specific coding modes, search centers, and search patterns. The parametrizable search strategy is realized by composing the search paths of the algorithms from five mutually independent offsets so that each offset is a function of the associated parameter(s). The *separable search path generation* provides easy access to individual algorithm-specific parameters which can be modified without changing other features of the algorithm. In addition, the control and computation of each offset can be modularly designed and optimized. All the coding modes of the algorithm undergo similarly parameterized search. The mode decision monitors the serially executed coding modes and selects the best one.

3. The control unit of the ME architecture. The unit implements control for the ME framework. A chosen framework configuration determines ME algorithms available at run-time and reparametrization of the framework can be done at design-time. The execution time of each ME algorithm is almost directly proportional to the number of tested search points, so the implementation is very tolerant of different algorithm-specific search strategies.

4. The memory system of the ME architecture. It includes a novel combination of two *parallel memory architectures*, in which the distribution of data among the memory modules is improved over contemporary approaches. In addition, memory address generation and data permutation scheme are optimized for ME. The memory system provides efficient data storage and supports arbitrary memory accesses needed by fast ME algorithms.

5. The distortion computation unit of the ME architecture. The unit implements a HW-oriented *rate-distortion* (*RD*) *cost function*. It contains several new algorithm-level and circuit-level optimizations for RD cost computation and minimum RD cost selection. The optimizations of RD cost function are primarily focused on *sum of absolute difference* (*SAD*) computation.

Besides the designed ME architecture, the contributions of this Thesis include:

6. A survey of existing real-time H.264/AVC video encoders.

7. An overview and feasibility evaluation of the essential coding tools in MCP.

8. A survey of ME algorithms and state-of-the-art ME architectures.

9. Performance comparison between several ME algorithms and architectures.

1.3 Summary of publications

The distortion computation unit, the memory system, and the control unit of the ME architecture have originally been introduced in the included publications [P1], [P2], and [P3], respectively.

The Author acted as the first author and the main contributor in [P1]-[P3]. Eero Aho, Dr. Tech., provided valuable criticism and comments according to which [P1]-[P3] were improved. Prof. Timo D. Hämäläinen and Kimmo Kuusilinna, Dr. Tech, assisted in writing and supervised the research work.

1.4 Outline of Thesis

The introductory part of this Thesis is organized as follows.

Chapter 2 introduces the basic concepts of digital video encoding and analyzes state-of-the-art video encoders. Besides providing the relevant background information for this Thesis, the need for HW-accelerated video encoding and ME is justified.

Chapter 3 considers fundamentals of MCP in which the main focus is on ME. Relying on the experiments in the literature, it is shown that many ME parameters can be simplified or completely excluded without compromising compression performance.

Chapter 4 surveys contemporary ME algorithms and architectures. Related works on distinct components of the ME architecture are presented in [P1]-[P3].

Chapter 5 provides an overview of the designed ME architecture whose components are more thoroughly considered in [P1]-[P3]. In addition, new architecture features introduced after publishing [P1]-[P3] are considered in detail.

Chapter 6 presents performance comparisons between the proposed and contemporary ME architectures. The component-specific performance comparisons are available in [P1]-[P3].

Chapter 7 concludes the introductory part of Thesis.

2. Video Encoding

This chapter presents the basic principles of digital video representation and video encoding. In addition, a brief overview is given about international video coding standards and associated video encoders. Finally, state-of-the-art encoders and their implementations in the current platforms are analyzed.

2.1 Digital video

Digital video represents a visual scene in a binary format in which intensities and colors of the scene are specified with strings of bits (logical 0s and 1s). The *visual scene* describes a real world, an imaginary world, or a hybrid of them. The real world visual information is obtained from *natural video* material whereas the imaginary world represents computer-generated visual scenes or objects (*synthetic video*). This Thesis focuses on natural video.

2.1.1 Digitization of natural video

Current digital video cameras capture natural video scenes via image sensors that convert the arriving light into *spatially* and *temporally* continuous analog video signals. *Digitization* of these signals is accomplished by sampling them in spatial and temporal domains and quantizing the obtained continuous-valued samples to a discrete range of intensities and colors.

The produced spatio-temporal samples are referred to as *picture elements*, *pels*, or *pixels*. The simultaneously sampled pixels reconstruct a two-dimensional (2D) digital *still image*, whose maximum resolution is determined by the sampling grid. Limited *spatial resolution* may cause *spatial aliasing*, i.e., details of the original scene are missing or they are incorrectly represented in the reconstructed image. To avoid spatial aliasing, the well-known Nyquist sampling theorem states that the sampling grid in horizontal and vertical directions has to be two times denser than the upper frequency bound in the respective directions. The current consumer applications typically utilize resolutions from Sub-QCIF (128×96 pixels) up to high-definition television (HDTV) formats such as 1080p (1920×1080 pixels). In the future, the resolutions will evolve towards WQHD (2560×1440 pixels) and QFHD (3840×2160 pixels) formats and even up to UHDTV (7680×4320 pixels) format. This Thesis mainly considers QCIF (176×144 pixels), CIF (352×288 pixels), D1 (720×576 pixels), 720p (1280×720 pixels), and 1080p formats that are the current mainstream.

Digital video can be seen as a sequence of still images (*frames*) which are displayed at a certain *frame rate*. A temporal sampling rate (*temporal resolution*) determines the maximum frame rate. The sampling rate below the Nyquist criterion causes *temporal aliasing*. I.e., motion may seem jerky and unnatural if the frame rate is too low with respect to object motion. Applications of low data rate typically use 10 - 20 *frames per second* (fps), standard-definition TV (SDTV) supports 25 - 30 fps, and high-end applications require up to 50 - 60 fps. Humans perceive continuous motion if screen updates occur at 30 fps [64], so 30 fps is often considered as a minimum frame rate for real-time video. The *refresh rate* at which a display device redraws images in a second is typically higher (e.g., 50 Hz, 60 Hz, 100 Hz, or 200 Hz) than a frame rate since the human eye detects flicker if the refresh rate is below 50 Hz [6]. The frame rate is accommodated to refresh rate by repeating the same frame multiple times.

The displays utilize *progressive scan* or *interlaced scan*. The progressive scan traces all the lines of the frame simultaneously. The interlaced scan halves the data rate by displaying odd- and even-numbered lines of frames (*fields*) alternately. Hence, the field rate can be doubled over frame rate without bandwidth increase, so that flicker effect is removed. However, interlacing introduces artifacts such as interline twitter, i.e., an aliasing effect perceived as a scintillation or rapid up-and-down motion. Traditional cathode ray tube (CRT) TV displays and all three SDTV standards (NTSC, PAL, and SECAM) are interlaced. However, HDTV formats such as 720p and 1080p as well as modern displays, e.g., plasma and LCD displays, are progressive. Currently, the progressive scan is dominating technique since it doubles the resolution with the same frame rate and avoids interlace artifacts. This Thesis concentrates on progressively scanned video sequences whose frame rates are 25 - 30 fps.

2.1.2 Color spaces

The *color space* of digital video determines how the brightness (*luminance*) and color are described. The *RGB color space* is suitable for video capture and display, whereas *YCbCr* (*YUV*) [52] *color space* is better suited for storage and transmission. In the RGB space, colors are created by combining red (*R*), Green (*G*), and Blue (*B*) in varying proportions. The YCbCr space separates the luminance (*Y*) from the color information, where *Cb* and *Cr* represent color difference (*chrominance*) components. If the YCbCr color space has the same resolution for *Y*, *Cb*, and *Cr* components, the associated sampling pattern is referred to as 4:4:4. However, *Cb* and *Cr* components are typically represented with lower resolution than *Y*, since the *human visual system* (*HVS*) is more sensitive to luminance than color. A 4:2:2 sampling pattern halves the horizontal resolution of *Cb* and *Cr*, whereas a 4:2:0 pattern reduces the resolution of *Cb* and *Cr* to one fourth (halves horizontal and vertical resolution) without essentially sacrificing visual quality.

This Thesis considers designs that operate only on brightness information, so they are independent on the selected color sampling pattern. I.e., pixels are assumed to contain only the *Y* component of the *YCbCr* color space. In typical consumer applications, the used bit depth for pixels (*Y* component only) is 8 bits which equals 256 (0 - 255) different intensity levels from black (weakest) to white (strongest).

2.1.3 Motivation for video compression

A *raw* (*uncompressed*) video requires impractical high transfer capacity. For example, a raw bit rate for QCIF format (frame rate: 30 fps, bit depth: 8, sampling pattern: 4:4:4) is $30 \times (176 \times 144) \times 8 \times 3 = 17.4$ megabits per second (Mbit/s). The respective bit rates for CIF, D1, and 1080p formats are 69.6 Mbit/s, 285 Mbit/s, and 1.39 Gbit/s. At the current technology [108], it would be impossible to deliver raw 1080p real time video over any of the normal Internet Protocol TV (IPTV), cable (DVB-C), satellite (DVB-S), or terrestrial (DVB-T) TV networks. For example, ADSL2+ [53] has a maximum downstream bandwidth of 24 Mbit/s, so it could theoretically be able to transmit a single QCIF stream in real time, whereas 1080p would require 60 times more bandwidth. The bandwidth problem is emphasized with wireless networks [61].

Another obstacle is insufficient capacities of storage devices. For example, the storage requirement of one hour raw QCIF format is $3600s \times 17.4\,\text{Mbit/s} = 7.6\,\text{GB}$ and the respective values for CIF, D1, and 1080p are 31 GB, 125 GB, and 626 GB. A typical single-layer Digital Versatile Disc (DVD) having a capacity of 4.7 GB would be sufficient for less than 3 minutes of D1 format instead of a whole movie. On the other hand, a two hour movie in 1080p format would need 25 dual-layered Blu-ray discs.

Although the price per transmitted or stored bit is continually falling, the capacities of communication networks and storage devices are growing much more slowly than the amount of produced video material. Therefore, video compression will also play a significant role in the future [34], [92], [132].

2.1.4 Lossless and lossy compression

Video compression is performed by a *video codec* that contains a complementary pair of systems: a *video encoder* and a *video decoder*. This Thesis focuses on the encoder.

The objective of the encoder is to achieve high *compression ratio* with minimum visual quality degradation. It uses different video coding algorithms to detect and remove *statistical, spatial*, and *temporal* redundancies inherent in video [104]. The statistical redundancy can be efficiently compressed with *lossless* (reversible) compression methods but the compression ratios are only about 3:1 or 4:1 in the best case [104]. *Lossy* (irreversible) compression methods remove perceptually irrelevant parts from the video. In a normal video sequence, removable temporal and spatial redundancies occur between adjacent frames and adjacent pixels, respectively. The stronger the compression, the more original information is lost. Typical lossy compression ratios are from 50:1 to 200:1 and even above [67]. The inverse of compression ratio is called *coding efficiency*, i.e., the degree to which the encoder reduces the bit rate. The encoder is typically evaluated with *rate-distortion* (*RD*) performance that describes a trade-off between the coding efficiency (bit rate) and loss of information (distortion).

2.1.5 PSNR

The distortion is typically measured by computing *Peak Signal to Noise Ratio* (*PSNR*) between original video sequence and its reconstruction after compression. PSNR between original 8-bit luminance frame (F_C) and its reconstruction (F_R) is computed as

$$\text{PSNR}(F_C, F_R) = 10 \times \log_{10}\left(\frac{255^2}{\text{MSE}(F_C, F_R)}\right). \tag{2.1}$$

In (2.1), the square of the maximum available 8-bit brightness value is divided by the *mean square error* (*MSE*) between F_C and F_R. For F_C and F_R of size $Q \times U$ pixels, MSE is defined as

$$\text{MSE}(F_C, F_R) = \frac{1}{(Q-1)\times(U-1)}\sum_{i=0}^{Q-1}\sum_{j=0}^{U-1}\left(F_C(i,j) - F_R(i,j)\right)^2, \tag{2.2}$$

where $F_C(i, j)$ and $F_R(i, j)$ indicate individual pixels of F_C and F_R, respectively. In practice, PSNR between original and reconstructed sequences is measured by computing PSNR separately for each frame and taking the arithmetic mean of the frame-specific PSNRs.

PSNR is an *objective* mathematical measure, which produces results independent of viewers' opinions, viewing conditions (viewing distance, illumination, etc.), and display technologies. However, MSE used in PSNR is sensitive to the energy of errors instead of structural image distortions, so PSNR does not necessarily correlate with *subjective* video quality perceived by HVS. It is roughly approximated in [74], that 40 dB, 33 dB, and 30 dB are minimum PSNR values for high, good, and adequate video qualities, respectively.

The most reliable measure for visual quality would require subjective assessments with large number of human observers, but subjective tests are cumbersome, expensive, and time-consuming to organize. Therefore, developers typically rely on automatic and repeatable objective quality measures of which PSNR is by far the most widely used technique. PSNR is also used in this Thesis. As suggested in [2], RD performance of the encoders is compared either *as Bjøntegaard delta PSNR* (*BD-PSNR*) for the same output bit rate or, alternatively*, Bjøntegaard delta bit rate* (*BD-rate*) for the same PSNR.

Besides high RD performance, the encoder has to produce a decoder-compatible bit stream that is often determined by some video coding standard. This Thesis considers only widespread *international video coding standards* and omits proprietary compression schemes (e.g., Sorenson Video 3), nonstandard open codecs (e.g., VP8), as well as national standards (e.g., AVS China [10]).

2.2 International video coding standards

The international standardization work on video coding is mainly coordinated by *ISO/IEC Moving Picture Experts Group* (*MPEG*), *ITU-T Video Coding Experts Group* (*VCEG*), and *Society of Motion Picture and Television Engineers* (*SMPTE*).

2.2.1 MPEG-1, MPEG-2, and MPEG-4 Visual

The widely used MPEG standards for audio/video coding are *MPEG-1* [49], *MPEG-2* [50], and *MPEG-4 Visual* (*Part 2*) [51]. MPEG-1 is optimized to VHS-quality video at bit rate of 1.5 Mbit/s. Its applications include Video Compact Disc (VCD).

As an extension to MPEG-1, MPEG-2 is targeted to compress higher quality video at bit rates of 2 - 100 Mbit/s. The main feature distinguishing MPEG-2 from MPEG-1 is the efficient coding of interlaced videos. MPEG-2 is a commonly used compression scheme in DVD as well as in DVB-C, DVB-S, and DVB-T broadcasting standards. It is capable of coding SDTV and HDTV at bit rates of 3.5 Mbit/s and 15 Mbit/s, respectively [39]. MPEG-2 classifies the group of application specific features into *profiles* (subsets) which specify a set of coding tools that can be used to generate a bit stream. The most widely used profiles are *simple profile* (*SP*) and *main profile* (*MP*) which are targeted for low delay and high quality applications, respectively.

MPEG-4 Visual achieves 20 - 30% BD-rate savings over MPEG-2 [133]. The target bit rates of MPEG-4 Visual are 5 - 64 kbit/s in mobile applications, but it is able to support bit rates of about 1 Gbit/s in studio editing resolutions ($4\text{K} \times 4\text{K}$ pixels). Its most popular profiles are *SP* and *ASP* (*Advanced Simple Profile*). Nowadays, MPEG-4 SP is widely supported feature in Internet-based videos and mobile applications [6], whereas MPEG-4 ASP is used in surveillance cameras and other higher quality video systems. Besides coding rectangular frames, MPEG-4 Visual also supports *object-based coding* that enables access and manipulation of individual objects in a video scene. At least so far, the object-based coding has been impractical solution due its huge complexity.

2.2.2 H.261, H.262, and H.263

The ITU-T standards *H.261*, *H.262*, and *H.263* are primarily developed for video communication over telecommunication and computer networks. They share many identical features with MPEG standards.

H.261 is targeted for videophone and video conferencing over Integrated Services Digital Network (ISDN). It was designed to support target bit rates of multiples of 64 kbit/s (64 - 1920 kbit/s). H.262 is omitted here, because its functionality is equivalent to MPEG-2. H.263 was primarily developed to support low bandwidth applications on telephony and data networks. Although its target bit rate is below 30 kbit/s, it also offers good performance at higher bit rates (up to 16 Mbit/s).

2.2.3 H.264/AVC

The latest video coding standard of ISO and ITU-T is entitled as *H.264/AVC* (*Advanced Video Coding*) [57] and it is published jointly by MPEG (*MPEG-4 Part 10*) and VCEG (H.264). Contrary to MPEG-4 Visual, H.264/AVC omits the complex object-based coding and focuses only on traditional coding of rectangular frames. It supports bit rates from 5 kbit/s to 960 Mbit/s. The most popular profiles of H.264/AVC are *Baseline Profile* (*BP*), *High Profile* (*HiP*), and *Extended Profile* (*XP*). BP is widely used in videoconferencing and mobile applications, HiP is targeted for broadcast and disc storage applications, and XP is intended for streaming and error prone environments such as wireless networks.

H.264/AVC achieves about 50 - 60% BD-rate reduction over MPEG-2, whereas the BD-rate decrease over MPEG-4 Visual and H.263 is at least 30% [133]. Due to its high compression and error resilience capabilities, H.264/AVC has become a widely deployed coding technique in various products and services such as Blu-ray Disc, YouTube videos, Adobe Flash, and DVB standards. For example, H.264/AVC is able to code SDTV and HDTV at bit rates of 2 - 3.2 Mbit/s and 7.5 - 13 Mbit/s, respectively [39].

H.264/AVC has been recently extended by two amendments that consider *scalable video coding* (*SVC*) and *multiview video coding* (*MVC*) [57]. SVC provides a compressed video with two or more quality options (spatial resolution, temporal resolution, and fidelity) from which the most suitable one is selected according to available transmission, display, or storage capability. MVC, in turn, operates on multiview video that is captured by synchronized cameras from different viewpoints. Multiview video provides three-dimensional (3D) depth impression of the observed scenery. It is utilized by emerging 3D applications such as *3DTV* and *free viewpoint TV* (*FTV*) [116] in which the user can control the viewpoint and generate new views from any 3D position.

2.2.4 VC-1

SMPTE 421M [109] is another state-of-the-art video coding standard. SMPTE 421M is informally referred to as *VC-1* and it was initially developed as a proprietary video format by Microsoft. Depending on the applied compression options and source content, the BD-rate of VC-1 over H.264/AVC is approximately $\pm$ 10% [73]. Therefore, VC-1 is widely characterized as an alternative to H.264/AVC.

VC-1 has target bit rates of 96 kbit/s - 135 Mbit/s and its applications include Blu-ray Discs, Windows Media Video 9 (WMV-9), and Microsoft Silverlight framework. VC-1 contains three profiles: *Simple profile* for low-complexity applications, *Main profile* for high data rate Internet applications, and *Advanced profile* for broadcast applications.

2.2.5 Future international video coding standards

Although H.264/AVC and VC-1 significantly outperform the older video coding standards, their compression efficiencies tend to be inadequate for the wireless and wired transmission of the next generation resolutions (QFHD and beyond) [92]. Therefore, MPEG and VCEG have established a *Joint Collaborative Team on Video Coding* (*JCT-VC*) to develop a successor to H.264/AVC.

This forthcoming international video coding standard is tentatively referred to as *High Efficiency Video Coding* (*HEVC*) [34], [132]. HEVC focuses on coding of progressively scanned rectangular frames whose resolution can vary at least between QVGA (320×240 pixels) and UHDTV formats. JCT-VC is currently integrating and refining the features of the best-performing HEVC proposals [5], [35], [63], [89], [124]. The combination of these most promising proposals roughly halves the bit rate over H.264/AVC with the same subjective perceptual quality, whereas their respective BD-rate savings are measured to be around 30 - 40% [132]. The coding gain tends to increase as a function of resolution. To obtain better trade-off between complexity and compression efficiency, the coding tools of HEVC are separately specified for high efficiency and low complexity applications [92].

The plan of JCT-VC is to publish draft versions of HEVC in 2012 and the first version of the final standard in early 2013. Since HEVC is still a moving target with many uncertain elements and open questions, this Thesis mainly focuses on the existing ratified standards.

2.3 Video encoder

All the considered standards (MPEG-1/2, MPEG-4 Visual, H.261/3, H.264/AVC, and VC-1) only specify the decoding process and output bit stream (syntax and semantics) of the encoder. Provided that the encoder design is kept compatible with the decoder specification, manufacturers can freely optimize encoding efficiency, quality, and speed. This approach allows industrial competition and further evolution of technology in the *non-normative* parts of the standard.

Figure 2.1 depicts an exemplary *hybrid video encoder* for these standards. They are all based on *hybrid video coding scheme* [6], [10], [104], which combines *motion compensated prediction* (*MCP*), *transform coding* (*TC*), and *entropy coding* (*EC*). These three stages are described in detail in Sections 2.3.1, 2.3.2, and 2.3.3, respectively. The blocks and data flows illustrated with dashed lines are only supported by H.264/AVC and/or VC-1. The respective model for a decoder is presented, e.g., in [6], [10], and [104].

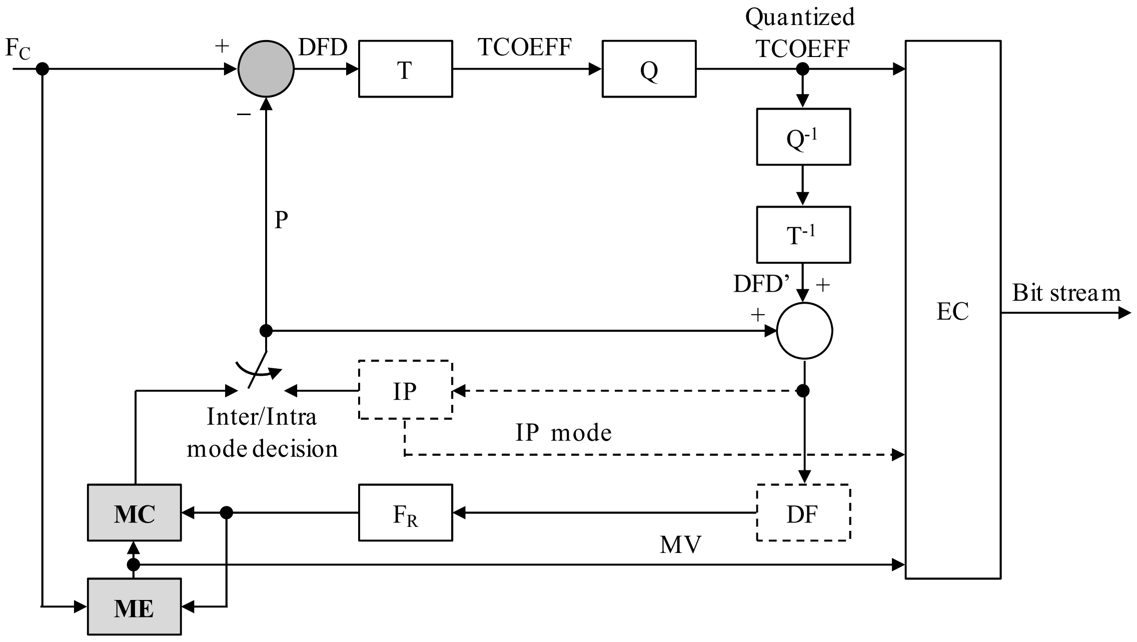


Figure 2.1. Generalized block diagram of the hybrid video encoder.

The encoder receives a *current frame* (F_C) as an input. It processes F_C in the units of a *macroblock* (*MB*). F_C is usually captured in the RGB color space but it is converted into the YUV color space before encoding in order to reduce storage and transmission requirements. In the YUV color space, each of the processed MBs consists of 16×16 luminance samples and associated chrominance samples, whose amount depends on the used sampling pattern (4:4:4, 4:2:2, or 4:2:0). For example, 4:2:0 format includes 8×8 blocks of blue and red chrominance.

2.3.1 MCP stage

MCP stage of the encoder reduces temporal redundancy by exploiting similarities between consecutive video frames. This stage is denoted by the grey components in Figure 2.1.

All the considered standards utilize *block-based MCP*. H.261 and MPEG-1/2 operate on the 16×16 pixel block, whereas H.263, MPEG-4 Visual, and VC-1 support block sizes of 16×16 and 8×8. H.264/AVC implements the most complex MCP which makes use of seven block sizes (16×16, 16×8, 8×16, 8×8, 8×4, 4×8, and 4×4).

The first phase of MCP is *motion estimation* (*ME*) which searches for the best matching counterpart for the processed *current MB* from the previously encoded and *reconstructed reference frame(s)* (F_R). In *forward prediction*, F_R equals a single past frame or a set of several past frames. Similarly, F_R represents one or more future frames in *backward prediction*. In order to apply future frames as reference, they have to be encoded before F_C (out of display order). H.261 supports only forward prediction with a single past frame. MPEG-1/2, MPEG-4 Visual, H.263, and VC-1 allow forward and backward prediction with one past and one future frame, respectively. Instead, H.264/AVC supports up to 16 reference frames in forward and backward prediction. In addition, MVC amendment of H.264/AVC extends ME to *disparity estimation* (*DE*) which performs inter-view prediction between parallel processed views.

The search technique of ME is called *block-matching* and a displacement between the positions of the processed MB and the best match is known as a minimum *motion vector* (*MV*). ME is performed only on luminance blocks and associated chrominance MVs are derived from luminance MVs. The chrominance-based ME is omitted since HVS is relatively insensitive to color information. Hence, it would provide only small gains compared with increase in complexity.

The second phase of MCP is *block-based motion compensation* (*MC*) during which a *MCP block* (*P*) of the processed MB is generated. *P* is a direct copy of the best match whose position in F_R is indicated by MV.

In the third phase of MCP, a *displaced frame difference* (*DFD*), i.e., *residual prediction error*, is computed by subtracting *P* from the processed MB. Hence, MVs are used to compensate motion and only DFD is encoded.

This Thesis focuses on MCP stage that is introduced in detail in Chapter 3. In the following, the remaining parts of the encoder are shortly considered.

2.3.2 TC stage

The first phase of TC stage is *transform* (*T*). It reduces spatial redundancy in DFD by exploiting similarities between neighboring pixels of DFD. DFD is transformed from a spatial domain into a transform domain in which the energy associated to DFD is represented more compactly with a small number of *transform coefficients* (*TCOEFFs*). Typically, perceptually more significant information is concentrated on low-frequency TCOEFFs whereas higher frequency TCOEFFs can be discarded. MPEG-1/2, MPEG-4 Visual, and H.261/3 implement *T* with a block-based 2D *Discrete Cosine Transform* (*DCT*) [102] which successively operates on 8×8 luminance and chrominance blocks. Instead, H.264/AVC and VC-1 employ *Integer Cosine Transform* (*ICT*) which is an integer approximation of the floating point DCT. Exact ICT reduces complexity of inverse transform computations and eliminates mismatches between inverse transforms of the encoder and decoder. H.264/AVC supports either 4×4 or 8×8 ICT (4×4 ICT always for chrominance), whereas VC-1 adaptively supports 4×4, 4×8, 8×4, and 8×8 ICT. DCT-based transforms have high transform coding gain and low computational complexity, but they tend to suffer from artifacts at block edges (*blockiness*). *Discrete Wavelet Transform* as a frame-based transform would provide better energy compaction over DCT-based transforms, but it has not replaced DCT due to its higher computational complexity and memory requirements [104]. DCT is also better compatible with block-based MCP.

The second phase of TC stage is *quantization* (*Q*) that maps the dynamic range of TCOEFFs to the limited range of quantized TCOEFFs. The irreversible *Q* only retains a reduced number of quantized TCOEFFs by converting insignificant TCOEFFs to zero. As a lossy operation, *Q* is the primary source of the compression gain. All the considered standards support *scalar quantization* that quantizes each TCOEFF independently [6]. The *quantization parameter* (*QP*) controls a tradeoff between compression efficiency and quality. QP of H.264/AVC takes integer values from 0 to 51, whereas the other considered standards use QP values from 1 to 31. Larger QP value increases quantization step size which improves compression efficiency, but reduces quality. In H.264/AVC, example QP values for high, medium, and low bit rates are 20, 30, and 40, respectively [44].

2.3.3 EC stage

EC stage removes statistical redundancy from outputs of MCP (MVs) and TC (quantized TCOEFFs) stages. The quantized TCOEFFs are typically converted to a one-dimensional (1D) array by substituting low frequency TCOEFFs before high frequency ones (*zigzag scanning*). EC converts MVs, quantized TCOEFFs, and other *syntax elements* (e.g., MB and slice headers) to binary codewords which are multiplexed together to a bit stream.

The widely used lossless entropy coding techniques are *variable length coding* (*VLC*) and *arithmetic coding*. All the considered standards support VLC, whereas arithmetic coding is additionally supported by H.263 and MPEG-4 Visual. H.264/AVC also contains two entropy coding modes. The first mode has two entropy coding tools: *context-adaptive VLC* (*CAVLC*) and *universal VLC* (*UVLC*). TCOEFFs are encoded with CAVLC and the other syntax elements with UVLC. The other mode is *context-adaptive binary arithmetic coding*

(*CABAC*) that is used for all syntax elements. Compared to CAVLC, CABAC typically provides 5 - 15% reduction in BD-rate, but at a cost of higher complexity [134].

2.3.4 Decoding path

In parallel with the EC stage, the *decoding path* (*feedback loop*) of the encoder reconstructs the processed MB to provide a decoder-compatible reference for future predictions. The quantized TCOEFFs are *inverse quantized* (Q^{-1}) and *inverse transformed* (T^{-1}) in order to reproduce DFD (DFD') after which a *reconstructed MB* is yielded by adding *P* to DFD' (*P* + DFD'). The reconstructed MB data equals MB data that is available for the decoder.

All the reconstructed blocks of the processed frame are stored in F_R which is used by MCP stage when a subsequent frame is processed. The blockiness of reconstructed blocks tends to lead inaccurate MCP. Therefore, H.264/AVC and VC-1 apply an *in-loop deblocking filter* (*DF*) which removes blockiness by smoothing the sharp edges of the reconstructed blocks. In the other considered standards, the deblocking filter is optional.

2.3.5 MB coding modes

All the considered standards support at least *intra*, *inter*, and *skip* MB coding modes. The best mode is chosen according to applied *mode decision* function.

The intra mode operates without MCP. The intra mode is chosen for a MB if the inter mode cannot meet its coding cost threshold or is not available. H.264/AVC performs *intra prediction* (*IP*) for an intra MB in the spatial domain and only the spatial prediction error is forwarded to the TC stage. IP predicts *P* from previously encoded and reconstructed neighboring blocks of the current frame. IP supports luminance block sizes of 16×16, 8×8, and 4×4 so that one type can be used per MB. H264/AVC provides four spatial prediction modes for block size of 16×16, and nine prediction modes for block sizes of 8×8 and 4×4. In addition, prediction is separately performed for 8×8 chrominance blocks. The other considered standards conduct an intra MB directly to the TC stage without prediction ($P = 0$) and perform intra prediction in transform domain.

The inter mode uses MCP and it is further classified as *unidirectional prediction mode* (*P mode*) and *bidirectional prediction mode* (*B mode*). In MPEG-1/2, MPEG-4 Visual, H.263, and VC-1, the *P* mode is restricted to forward prediction whereas the *B* mode allows bidirectional (forward and backward) prediction. In addition, the *B* mode can select an average of forward and backward predictions as a predictor candidate. Through averaging, noise can be reduced. H.261 supports only the *P* mode. The *P* mode of H.264/AVC enables also backward prediction, so the only difference between the *P* and *B* modes of H.264/AVC is the support for the average predictor in the *B* mode.

In the skip mode, the encoder does not transmit prediction error or MVs for a MB, but a MB can be simply replicated from the previously decoded frame. To improve compression ratio, the encoder can increase the occurring rate of skip MBs by skipping also MBs that have only a diminutive effect on distortion.

H.261/3, MPEG-1/2, MPEG-4 Visual, and VC-1 determine available coding modes for a MB in a frame level. They all support *intra-coded frames* (*I*-frames) and *unidirectionally predicted frames* (*P-frames*). *I*-frame can contain only intra MBs. *P*-frames are predicted from the closest preceding *I*/*P*-frame and they provide more compression than the *I*-frames. In addition, H.263, MPEG-1/2, MPEG-4 Visual, and VC-1 improve compression efficiency further with *bidirectionally predicted frames* (*B-frames*) which can reference to the closest past and/or to the closest future *I*/*P*-frames. The usage of intra and skipped MBs in *P*/*B*-frames is specified in detail by the standard [49]-[51], [55], [56], [109].

H.264/AVC defines available prediction modes for a MB in a *slice* level. A slice is a group of MBs. The sizes and shapes of slices are highly flexible in H.264/AVC. *I-slices* can only contain intra MBs. MBs in *P-slices* can be coded in the intra, inter, or skip mode, whereas MBs in *B-slices* can be coded in the intra, inter, skip, or *direct* modes. The direct mode resembles the skip mode, except that the prediction error is sent. Each slice is coded separately and *I*, *P*, or *B*-slices can be mixed within a single frame.

2.3.6 Encoder control

With constant coding parameters, the output bit rate of the encoder changes depending on the frame content. This variation tends to be a problem for practical applications which typically necessitate a constant or at least a constrained bit rate output. Encoder *rate control* (*RC*) ensures that a video sequence is encoded at the target bit rate without violating the channel bandwidth, timing constraints, or encoder/decoder buffer sizes. An *RC algorithm* is needed to allocate bits among all coding units (e.g., frame, slice, and MB) and to determine an optimum QP that can properly encode each coding unit with allocated bits. The sophisticated RC algorithms use *RD optimization* (*RDO*) techniques to jointly consider the quality degradation and number of bits used. In order to find the best possible RD trade-off, RDO techniques are also adopted to other encoding functions such as mode decision and ME.

The standards recommend their own non-normative RC schemes [73]. H.264/AVC adopts the most complex schemes, because it supports various MB coding modes and RDO-based mode decision. In the literature, several optimized RC schemes have been presented especially for H.264/AVC [88], [123], but they are out of the scope of this Thesis.

2.4 State-of-the-art encoders

The state-of-the-art video codecs are primarily designed for H.264/AVC and VC-1. Although VC-1 is simpler to implement, the lack of public reference implementation has limited the activity on open VC-1 encoder development [73]. Therefore, this section only examines encoders that are compatible with more popular H.264/AVC.

2.4.1 H.264/AVC encoders

The platform-independent reference SW implementation of H.264/AVC is referred to as *Joint Model* (*JM*) whose current version is JM 17.0 [59]. JM has low performance since it contains all features of H.264/AVC. According to [43], *Baseline Profile* (*BP*) of JM 7.3 requires 3600 giga instructions per second (GIPS) computation and 5570 gigabytes per second (GBytes/s) memory access when encoding 720p format at 30 fps. Hence, it is mainly used in conformance testing and research rather than practical *real-time encoders*.

The other publicly available SW encoder is x264 [136], which has been adopted by popular web video services such as Youtube and applications including VLC media player. Due to various algorithmic-level optimizations, x264 is approximately 50 times faster than JM at a maximum BD-rate increase of 5% [90].

Commercial H.264/AVC encoders have been released, e.g., by Apple Computer (Quicktime), Ateme (Nero Digital AVC), Intel Corporation (Intel IPP), MainConcept, and Sony (Blu-code). However, the comparisons such as [125] between practical SW encoders have shown that the publicly available x264 encoder achieves the lowest average BD-rate.

2.4.2 Practical implementation constraints

Selecting an appropriate profile and level of H.264/AVC encoder requires a trade-off between RD performance, encoding delay, and computational complexity. The best available *RD* performance is usually applied only in *nonreal-time applications*. They allow *multi-pass encoding* in which the first encoding pass analyzes the video and the subsequent pass(es) use the results of the first pass to adjust the best possible video quality within the bit rate limits. Since multi-pass encoding is normally performed only once, the complexity and delay of the encoding are less critical issues.

The importance of encoding delay is emphasized whenever live events are being broadcast or streamed. The *live applications* require that transmitting and/or receiving end(s) are equipped with real-time encoders. The complexity of real-time encoding becomes very critical issue in portable devices due to their limited size, weight, and cost. In these devices, the growing convergence of video, audio, and graphics functionalities further limits the computation power and memory space available for encoding.

Power consumption is particularly essential in portable devices, since their usage and standby times are often dependent on battery life. The growing gap between battery capacity and the needed power accelerates battery depletion in these devices. In addition, typical handheld devices such as mobile phones restrict heat dissipation to be below 3 W of which less than 1 W is reserved for the application processing and memories [94], [107]. Intelligent mobile devices request power-aware encoders that can adjust the set of the utilized coding tools according to existing conditions such as processed video content, battery capacity, environmental condition, and user preferences [16], [79]. Non-battery powered embedded systems are also limited by power, since power consumption influences the system cost and lifetime of the chip. The power consumption of below 2 W enables the usage of a cheap plastic chip package and external cooling is avoided if power dissipation is below 10 W [95]. In addition, the density of parts on the board is minimized when the power dissipation of each individual chip is restricted to around 10 W [62].

Companies and academic institutions have introduced various approaches for H.264/AVC video encoding [71]. They are here classified as *SW*, *HW/SW*, and *HW implementations*.

2.4.3 Software implementations

A single-core *general-purpose processor* (*GPP*) provides a traditional platform for a SW implementation of H.264/AVC encoder [17]. The modern GPPs are extended with *single instruction multiple data* (*SIMD*) instructions and *multithreading* capabilities which can accelerate video encoding through data and task level parallelism, respectively. However, despite these extensions, GPP performance tends to be insufficient for real-time H.264/AVC encoding. For example, Intel Pentium 4 Extreme Edition (P4EE) [47] is one of the fastest single-core GPP. With two simultaneous threads, it achieves theoretical peak performance of approximately 10 GIPS, or alternatively, 6 giga floating point operations per second (GFLOPS) in single precision when operating at 3.2 GHz. According to x264 benchmark [117], P4EE is ideally able to encode 720×480 resolution x264 video at 18 fps when operating at 2.4 GHz. In addition, its power consumption of 92 W at 3.2 GHz (130 nm process technology) is far beyond the requirements of handheld devices.

In the recent years, the single GPPs have evolved into *homogeneous multiprocessor system-on-chip* (*MPSoC*) platforms [62], [135] that replicate the processor cores on a single die. For example, Intel Core i7 Extreme Edition 980X [46] contains six cores and up to 12 simultaneous threads with multithreading. It is able to achieve theoretical peak performance of 109 single-precision GFLOPS at 3.3 GHz. With x264 HD benchmark [117], it encodes 720p format at 87 fps. Although this performance corresponds encoding of 1080p format at 38 fps in an ideal case, the power consumption of 130 W (32 nm process technology) is very high.

The computation load of GPPs can be alleviated by accelerating H.264/AVC encoding with *graphics processing units* (*GPUs*) that have become an integral part of current mainstream GPPs [20]. The GPUs have traditionally been tailored to 3D graphics acceleration, but recently they have evolved into *programmable special-purpose MPSoCs* [97]. For example, the high-end Nvidia Tesla S2050 GPU computing card has 4 GPUs, each of which contains 448 processor cores. Due to huge number of fine-grained parallel processors, GPUs deliver order-of-magnitude performance gains over GPPs. For example, Tesla S2050 GPU has theoretical peak performance of over 4000 single-precision GFLOPS at a cost of 900 W (40 nm process technology). The GPU can assist GPP by executing the most *computation-intensive* SIMD-oriented tasks of H.264/AVC encoding, such as ME, MC, and IP (Figure 2.1), whereas *control-intensive* tasks, such as EC, are typically allocated to GPP [20]. Unfortunately, performance figures of GPU-accelerated H.264/AVC encoder are not publicly available.

The Cell Broadband Engine Architecture (CBEA) [98] is an example of *general-purpose heterogeneous MPSoC*, in which computation can be divided between different types of cores in a single die. CBEA has one dual-threaded power processor element (PPE) and eight synergistic processor elements (SPEs). The PPE runs the operating system and controls SPEs which are designed for SIMD execution. The peak performance of CBEA is about 200 single-precision GFLOPs at 3.2 GHz and the power consumption is estimated to be close to 20 W (45 nm process technology) [3], [114]. CBEA is able to encode x264 video in 1080p format at 30 fps [38].

Although general-purpose multicore processors can execute DSP algorithms successfully, they are not suitable for use in embedded systems because of power supply and space constraints. In embedded systems, video is typically encoded with *digital signal processors* (*DSPs*), whose architecture is particularly designed for signal processing tasks. However, the performance of single-core DSP platforms tends to be insufficient for real-time H.264/AVC encoding. For example, Texas Instruments (TI) DaVinci family contains video-oriented processors such as a single-core TMS320DM642 DSP [118]. At maximum operating frequency of 720 MHz, it achieves almost 6 GIPS performance and its typical power consumption is diminished close to 2 W (90 nm process technology). However, its peak encoding speed of x264 VGA (640x480) resolution video is only 22 fps [80].

Table 2.1 summarizes publicly available characteristics and x264 encoding performance of the examined processors. Although Core i7-980x and CBEA can provide enough performance for 1080p encoding, their power consumption is far beyond the requirements of handheld devices. Therefore, HW acceleration is used either for the encoding assistance or for the whole encoding process.

2.4.4 Hardware/software implementations

Let us first consider HW acceleration in heterogeneous MPSoC platforms that are targeted for embedded applications. These MPSoCs include a specific set of programmable GPPs, DSPs, GPUs, and nonprogrammable *HW accelerators* for standardized signal processing tasks such as video encoding. The GPP is typically a *reduced instruction set computer* (*RISC*) processor that runs the operating system and executes control-intensive tasks. For example, the high-end ARM Cortex A9 processor can contain 1 - 4 cores and a power-optimized dual-core configuration of it achieves 4 GIPS performance with power consumption of 0.5 W (40 nm process technology).

State-of-the-art *video-oriented MPSoCs* such as Renesas SH-MobileHD1 [103], Qualcomm Snapdragon QSD8672 [101], TI TMS320DM368 [119], and Liu's encoder [86], [87] are able to encode H.264/AVC 1080p video at 30 fps. SH-MobileHD1 contains 500 MHz RISC processor, two DSPs for audio, as well as a HW accelerated unit for video processing. QSD8672 has two 1.5 GHz ARM-based cores, a GPU, and HW accelerators for video encoding. TMS320DM368 includes 432 MHz ARM core and HW-accelerated video/imaging coprocessor. Liu's encoder contains one configurable HW-accelerated RISC processor and dedicated HW accelerators.

Table 2.1. Encoding performance of the well-known processors.

Device	P4EE	Core i7-980X	CBEA	TMS320DM642
Manufacturer	Intel [47]	Intel [46]	Sony/Toshiba/IBM [98]	Texas Instruments [118]
Year	2003	2010	2008	2004
Encoder	x264	x264	x264	x264
Process	130 nm	32 nm	45 nm	90 nm
Transistors	178 M	1170 M	241 M	n.a.
Clock Frequency	2400 MHz	3300 MHz	3200 MHz	720 MHz
Max. Resolution	720×480@18fps	720p@87fps	1080p@30fps	640×480@22fps
Power	90 W (@3200 MHz)	130 W	≈ 20 W	2 W

n.a. = not available

To obtain a good balance between programmability and efficiency, SH-MobileHD1, QSD8672, TMS320DM368, and Liu's encoder realize real-time H.264/AVC encoding of 1080p format by partitioning the encoding functions between HW and SW. They off-load most of the encoding functions to associated HW accelerators, whereas programmable control-intensive parts are managed in SW. In addition, programmability is typically needed to adjust the supported feature set, profiles, and standards.

TMS320DM368 realizes all *multi-standard kernel functions* of ME, MC, *T*, *Q*, Q^{-1}, T^{-1}, IP, EC, and DF in HW (Figure 2.1) [127]. It only uses programmable DSP to execute *standard-specific functions* such as ME algorithm control, rate control, and mode decision. When encoding 720p at 30 fps, TMS320DM368 consumes power close to 1 W. In Liu's encoder, the RISC implements overall control and memory management, whereas DF and EC are realized with HW extensions of the RISC. The rest of the functions are mapped to the associated HW accelerators. When operating at 200 MHz, the architecture is able to encode 1080p format at 30 fps. The encoder core is realized with 1140 Kgates and 108.3 KB of static random access memory (SRAM). Its power consumption is about 1.2 W.

The available characteristics of these four MPSoCs are summarized in Table 2.2. The transistor count of Liu's design is computed by supposing four transistors per a single 2-input NAND-gate and six transistors per an on-chip SRAM bit. As a conclusion, the current MPSOCs enable HW accelerated H.264/AVC encoding up to 1080p format with reasonable power budget without losing programmability.

2.4.5 Academic hardware implementations

Encoding performance and/or power economy can be further improved by implementing encoders completely in HW as an *application-specific integrated circuit* (*ASIC*).

Huang *et al.* [12], [43], introduced the first single-chip *HW encoder* (H.264/AVC BP) in 2005. It runs at 108 MHz and supports 720p format at 30 fps without quality degradation compared with JM reference encoder. The architecture consumes 923 Kgates, 35 KB of SRAM, and 785 mW of power (180 nm process technology). Huang's encoder adopts 4-stage MB-level pipeline architecture, since the MB-level pipeline reduces data buffer sizes needed between modules when compared with processing in frame units. Later on, several power and performance optimized HW encoders have been announced. They all utilize MB-level pipelining.

Table 2.2. Encoding performance of the state-of-the-art video-oriented MPSoCs.

Device	SH-MobileHD1	Snapdragon QSD8672	TMS320DM368	MPSoC
Manufacturer	Renesas [103]	Qualcomm [101]	Texas Instruments [119]	Liu et al. [86] [87]
Year	2009	2009	2010	2007
Encoder	H.264	H.264 MP	H.264 HiP	H.264 BP
Process	65 nm	45 nm	65 nm	180 nm
Transistors	n.a.	n.a.	n.a.	10 M
Max. Resolution	1080p@30fps	1080p@30fps	1080p@30fps	1080p@30fps
Clock Frequency	500 MHz (1080p@30fps)	1500 MHz (1080p@30fps)	432 MHz (1080p@30fps)	200 MHz (1080p@30fps)
Power	n.a.	1400 mW (1080p@30fps)	1027 mW (720p@30fps)	1219 mW (1080p@30fps)

n.a. = not available

Chen *et al.* [13], [16], Chang *et al.* [8], [9], and Mochizuki *et al.* [91] presented power-efficient H.264/AVC BP encoders for power-limited portable devices. The power dissipation of these architectures were lowered through HW-oriented algorithmic optimizations (particularly for ME, IP, and mode decision), data reuse at the algorithm and architecture levels (to save memory power), and circuit-level optimization (clock gating). Chen's and Chang's approaches support power-adaptive encoding via power modes that fulfill different power constraints. They lower power consumption at a cost of reduced encoding quality. Chen's architecture realizes multiple power modes by adjusting the complexities of ME, IP, and DF modules. The 3-stage encoder consumes 453 Kgates and 17 KB of SRAM, its maximum operating frequency is 54 MHz, and it can encode 720×480 pixel resolution at 30 fps in which case power consumption is 44 - 67 mW (180 nm process technology) in different power modes. Chang's encoder contains quality-adjustable ME and IP modules in order to configure power consumption with four power modes. The 3-stage implementation is able to encode 720p format at 30 fps with a power mode-specific operating frequency of 72/108 MHz and power consumption of 122 - 183 mW. Mochizuki's encoder has only one power mode. However, the 3-stage architecture is still very power-efficient consuming only 64 mW for encoding of 720p format at 30 fps.

Lin *et al.* [81] proposed the first H.264/AVC HiP real-time HW encoder for 1080p format. Despite high performance and HiP compatibility, the 3-stage architecture has relative low resource and power consumption. The reported optimization methods include complexity reduction of ME, IP, and mode decision. In addition, the architecture utilizes increased parallelism, cross-stage resource sharing, and data reuse. When running at 145 MHz, it encodes 1080p at 30 fps with quality loss of 0.1 dB and consumes power 242 mW (176 mW for H.264/AVC PB). The required resources are 593 Kgates and 22 KB of SRAM.

Among the existing implementations, the peak encoding performance is provided by Ding *et al.* [25], [26]. Ding's 8-stage architecture supports MVC amendment of H.264/AVC (Multiview HiP) and HiP for 3D and 2D applications, respectively. It supports view scalability for encoding single-view 4096×2160 pixel resolution at 24 fps, three-view 1080p videos at 30 fps, and seven-view 720p format at 30 fps. These properties are realized through extending ME to support DE, optimizing IP, doubling EC modules, increasing pipelining, and rescheduling the encoder functions to support MVC. Compared to previous approaches, the gate count of Ding's architecture is more than doubled, but memory and power consumptions are comparable to other approaches.

Table 2.3 gathers the characteristic of most competitive implementations that support at least 720p format. Compared to HW/SW approaches, HW encoders achieve relative performance with much less resources and power dissipation. For example, Lin's encoder (Table 2.3) consumes about one third of Liu's encoder (Table 2.2) resources. In addition, Lin's encoder consumes one fifth of power, if an assumed 30% decrease of power per CMOS process generation [4] (from 180 nm to 130 nm) is taken into account (constant voltage scaling).

Among all the evaluated encoders, Chang's encoder is the most cost-effective solution for CIF-sized videos and below due to its smallest amount of resources and low power consumption. For 720p and 1080p formats, Lin's encoder provides the smallest implementation. On the other hand, Ding's approach has the smallest power consumption, when 30% decrement of power from 130 nm to 90 nm is assumed.

Table 2.3. Encoding performance of the state-of-the-art academic HW encoders.

Device	HW encoder	HW encoder	HW encoder	HW encoder
Manufacturer	Chang et al. [8][9]	Mochizuki et al. [91]	Lin et al. [81]	Ding et al. [25][26]
Year	2007	2008	2008	2009
Encoder	H.264 BP	H.264 BP	H.264 HiP	H.264 Multiview HiP/HiP
Process	130 nm	90 nm	130 nm	90 nm
Gate count	470 K	n.a.	593 K	1732 K
On-chip SRAM	13.3 KB	56.0 KB	22.0 KB	20.1 KB
Transistors	2.6 M	n.a.	3.5 M	8.1 M
Max. Resolution	720p@30fps	720p@30fps	1080p@30fps	4096 × 2160@24fps
Quality loss	< 0.6 dB	< 0.5 dB	0.1 dB	< 0.1 dB
Clock Frequency	72/108MHz (720p@30fps)	144MHz (720p@30fps)	145 MHz (1080p@30fps)	280MHz (4096 × 2160@24fps)
	10-28MHz (CIF@30fps)		63 MHz (720p@30fps)	81MHz (1080p@30fps)
			7 MHz (CIF@30fps)	36MHz (720p@30fps)
Power	122-183mW (720p@30fps)	64mW (720p@30fps)	242 mW (1080p@30fps)	522 mW (4096 × 2160@24fps)
	7-25mW (CIF@30fps)		116 mW (720p@30fps)	148 mW (1080p@30fps)
			7 mW (BP CIF@30fps)	58 mW (720p@30fps)

n.a. = not available

2.4.6 Commercial hardware implementations

H.264/AVC HW encoders have also been unveiled by several companies such as Imagination Technologies (POWERVR VXE382 encoder [45]), CAST (H264-MP-E encoder [7]), EyeLytics [29], and Jointwave (E760 encoder [60]). Their publicly available performance figures on ASIC are tabulated in Table 2.4. In summary, E760 encoder provides twice the encoding speed with equal resources than the other industrial approaches. E760 is also competitive with Lin's and Ding's encoders (Table 2.3). However, its search quality is not reported.

Among the considered ASIC-based encoder approaches, H264-MP-E, EyeLytics, and E760 encoders are additionally designed for *field programmable gate array* (*FPGA*). Although current FPGAs are not suitable for power-sensitive designs, the latest FPGAs are still sufficient for real-time H.264/AVC encoders. H264-MP-E encoder has been implemented on Altera Stratix IV logic device. It encodes 1080p at 30 fps in which case the required resources are 79 K adaptive look-up tables (ALUTs), 67 embedded DSPs blocks, and 109 embedded memory blocks of size 9 Kb (M9K). Eyelytics achieves the similar performance on Altera Stratix III at a cost of 20 K ALUTs, 5 DSPs, and 154 M9Ks. E760 is reported to perform 1080p encoding at 60 fps with 58 K ALUTs.

2.4.7 Discussion

In conclusion, high-end multicore processors are able to encode 1080p H.264/AVC video at 30 fps. However, their power consumption is one to two orders of magnitude higher than the power budget of handheld devices. Power efficiency of direct-mapped HW can be up to four orders of magnitude greater than those of GPPs [48]. With HW acceleration, real-time encoding of 1080p H.264/AVC video can be performed below 1 W power budget. HW acceleration also makes real-time encoding feasible with higher resolutions.

Table 2.4. Encoding performance of the state-of-the-art industrial HW encoders.

Device	POWERVR VXE382	H264-MP-E	EyeLytics	E760
Manufacturer	Imagination Technologies [45]	CAST [7]	EyeLytics [29]	Jointwave [60]
Year	2010	2011	2011	2009
Encoder	H.264 HiP	H.264 MP	H.264 MP	H.264 HiP
Process	n.a.	90 nm	130 nm	90 nm
Gate count	n.a.	600 K	300 K	730 K
On-chip SRAM	n.a.	88.5 KB	166.4 KB	26.9 KB
Transistors	n.a.	6.8 M	9.4 M	4.3 M
Max. Resolution	1080p@30fps	1080p@30fps	1080p@30fps	1080p@60fps
Clock Frequency	200 MHz (1080p@30fps)	333 MHz (1080p@30fps)	300 MHz (1080p@30fps)	304 MHz (1080p@60fps)
Power	n.a.	n.a.	n.a.	260 mW (1080p@60fps)

n.a. = not available

The current and next generation approaches use special-purpose HW either for the whole encoding process or for the encoding assistance in a processor-controlled environment. Hardwired, ASIC-based encoders outperform other approaches in terms of performance, silicon area, and power consumption. Due to their limited flexibility and high non-recurring engineering (NRE) costs [69], ASICs are the most optimal solution for low-cost and high-volume products that support a single standard. The ever-increasing computational complexity of encoding and low-power limitations ensure that hardwired encoders will be one of the practical solutions also in the future.

When there is a room for a slight overhead in performance and power consumption, more flexible solutions are preferred, since the rising trend is to support plethora of standard, nonstandard, and proprietary video codecs. Multi-standard support can be accomplished with modern reconfigurable FPGAs and video-oriented MPSoCs that are viable alternatives for ASICs in less power-sensitive encoders. Compared to ASICs, the current FPGAs have higher per unit costs, but as off-the-shelf products they offer faster time to market [69]. MPSoCs are built by the combination of validated processing elements, so they have short design and validation times as well as low design and manufacturing costs. In addition, derivative MPSoC designs can be created with rapid time-to-market and even the same platform can be reusable for a different set of applications. The reuse rate of a typical MPSoC design is estimated to linearly grow from the current 46% (in 2009) to 96% by 2024 [48]. MPSoCs will probably be the most popular approach for the next generation video encoding.

Although the allocation of encoding tasks between HW and SW varies in the considered encoder implementations, they all share the common feature of using HW acceleration for motion estimation (ME). Since ME is the most compute-intensive part of video encoding, the performance of the whole video encoding is dependent on ME implementation. Hereafter, this Thesis will focus on design and implementation of ME.

3. Motion Estimation

This chapter presents the basic concepts of motion and its modeling. The emphasis is on the motion model of motion compensated prediction (MCP). Among the coding tools of MCP, the main focus is on block-based ME whose characteristics and practical limitations are considered in the context of H.261/3, MPEG-1/2, MPEG-4 Visual, H.264/AVC, and VC-1. In addition, the basic operating principle of block-based ME and its realization through *rate-constrained block matching* is explained. Finally, *rate-constrained mode decision* is considered.

3.1 Concept of motion

Motion is an integral part of our visual experience. The capture of motion is the main reason for the popularization of video. A still image provides only a snapshot of a scene, but video also records dynamics of a scene by relating spatial image features to temporal changes.

In a 3D scene, the differences between frames may be induced by *rigid object motion* (e.g., a moving vehicle) and/or *deformable object motion* (e.g., a moving arm). A moving camera can also cause several types of desired 3D scene motions through rotation, pan, tilt, and zoom functionalities or undesired motions due to camera shake. The object-induced motion is referred to as *local motion*, whereas *camera motion* (also called *ego-motion*) is considered as *global motion*, since it occurs across the entire image.

3.1.1 True and apparent motion

A video sequence is captured by a camera that projects a 3D scene onto 2D image plane. Similarly, the relative true 3D motion between a camera and a scene is projected onto 2D image plane in which a 2D motion field is perceived through intensity variations. The perceivable motion of intensity patterns is referred to as an *optical flow* [40] or *apparent motion* [111].

The apparent motion typically differs from the true 2D motion field due to several well-known reasons. Firstly, *occlusions* between moving objects such as appearance and disappearance of object parts introduce uncovered and covered regions whose intensities are not visible all the time. Secondly, transparent objects induce multiple independent motions in a single image point. Thirdly, a uniform intensity region may be interpreted as a stationary even if it is moving because of an *aperture problem*. I.e., only the velocity component that is perpendicular to the moving edge of region can be measured. Fourthly, the *correspondence problem* prevents apparent motion to unambiguously reveal the original relation of corresponding features in successive frames. For example, the rotation direction of the wheel is ambiguous due to temporal aliasing that results from the limited frame rate. Finally, camera noise, quantization noise, illumination variations, rain/snow, and shadows also induce intensity changes that are incorrectly interpret as true motion.

Although apparent motion is only an approximation of the true 2D motion field, it is widely used by motion-related tasks, since it is the only accessible motion parameter from 2D motion sequences. Apparent motion is beneficial for two reasons. Firstly, it contains information about spatio-temporal relationships between objects. Secondly, the temporal correlation of intensities (and color) is high in the direction of motion. For example, the intensity of the moving object tends to be close to constant over time.

3.1.2 Practical usage of apparent motion

The apparent motion information is typically used to recover object-induced motion in the scene or to remove temporal redundancy in video compression. These motion-related tasks can be further classified as *motion segmentation*, *motion detection*, and ME. The purpose of motion segmentation is to identify independently moving objects from the background motion, whereas motion detection attempts to separate moving image points from stationary ones. Motion detection is actually a special case of motion segmentation with only two regions: changed/unchanged regions with a static camera and global/local motion regions in the case of a moving camera.

The idea of ME is to identify the movement of image points. It can be used to search for true motion (e.g., to counteract camera motion in motion segmentation or detection) or to reduce the bit rate in video compression. The inherent differences between the real 3D motion field and apparent motion complicate the recovery of true motion. The task is extremely challenging if surfaces of objects have little texture (uniform luminance). Instead, the *compression-optimized ME* need not resemble the true motion of image points as long as the best RD performance is achieved. Hence, it can be efficiently implemented although the actual motion field is inaccessible.

This Thesis considers only compression-optimized ME that is a central part of MCP in all the considered video coding standards (Section 2.3). In order to estimate and compensate object-induced local or camera-induced global motion, underlying motion model has to be determined for MCP.

3.2 Motion model for motion-compensated prediction

The choice of a motion model depends on a target application. Therefore, various application-dependent models have been developed for the local and global motions [111]. This Thesis focuses on *spatially translational and temporally linear motion model* [6]. All the considered standards use this compression-oriented model to estimate and compensate local motion during MCP. The global motion models are omitted here since global motion can be separately compensated with low effort [70].

3.2.1 Spatially translational and temporally linear motion model

A 2D model for apparent motion can be derived from the 3D models that describe camera projection geometry, motion of 3D object, and surface of 3D object. The spatially translational and temporally linear motion model (abbreviated as *translational/linear model*) is applied under the *orthographic projection* in which all the projection lines are perpendicular to the projection plane. Orthographic projection discards depth information totally. It projects 3D scenes onto 2D image planes with pixel coordinates $(i, j) \in \Lambda$, where Λ is a sampling grid (orthogonal lattice). Upon the orthographic projection, each frame (F_0, F_1, …, $F_{K\text{-}1}$) of a K-frame video sequence can be interpreted as a 2D image that is projected from a 3D scene at a discrete time instant (t_0, t_1,…, $t_{K\text{-}1}$). The interval between $t_{k\text{-}1}$ and t_k depends on the temporal sampling rate.

In practice, individual image points between any contiguous frames $F_{k\text{-}1}$ and F_k can move along arbitrary trajectories following object motion. However, the translational/linear model estimates apparent motion of visible objects through rigid 3D translation model. It supposes that motion of 3D objects (arbitrary surface) is only translational without rotation, scaling, or any 3D deformation. In addition, the translational/linear model assumes that object trajectories between any $F_{k\text{-}1}$ and F_k are temporally linear and all the points on a rigid object move with the same velocity. In discrete frames, the object velocity is presented by displacements. As a result, the model has only two parameters: the horizontal and vertical components of the displacement.

Since objects have more degrees of freedom than just the translational one, more complex motion models with a higher number of motion parameters would provide more precise description of a motion field. For example, a six-parameter *affine motion model* yields smaller prediction error than translational/linear model when non-translational motion such as rotation is approximated [111]. However, compression-optimized ME is not so vulnerable to inaccuracies in a motion model since ME is an ill-posed problem, i.e., local motion approximations are inherently ambiguous. A higher number of motion parameters can even increase overall coding cost, since the translational/linear model provides a close enough approximation for most natural images with two parameters. Therefore, the translational/linear model serves as the basis for ME and motion compensation (MC) in all the considered standards.

3.2.2 Motion modeling, estimation, and compensation

For the sake of simplicity, let us derive the translational/linear motion parameters for a single image point that moves from $F_{k\text{-}1}$ to F_k. The image point is assumed to be located at pixel coordinates $(i_{k\text{-}1}, j_{k\text{-}1}) \in \Lambda$ in $F_{k\text{-}1}$ and $(i_k, j_k) \in \Lambda$ in F_k.

Figure 3.1 a) presents an example motion trajectory of the image point (a dashed arrow from $F_{k\text{-}1}$ to F_k) and a modeled translational/linear motion for it. The translational/linear model describes the inter-frame movement from ($i_{k\text{-}1}$, $j_{k\text{-}1}$) to (i_k, j_k) with a *linear displacement vector*

$$d_{k-1}^{k}(di_{k-1}^{k}, dj_{k-1}^{k}) = (i_k - i_{k-1}, j_k - j_{k-1}), \tag{3.1}$$

which is measured in the positive direction of time (from $t_{k\text{-}1}$ to t_k). To better illustrate d_{k-1}^{k} in Figure 3.1 a), the tail $(i_{k\text{-}1}, j_{k\text{-}1})$ of d_{k-1}^{k} is projected along a dotted line from $F_{k\text{-}1}$ to co-located position in F_k.

The respective motion parameters can be utilized in temporal prediction to estimate and compensate the inter-frame displacement of the image point. Figure 3.1 b) shows how the linear displacement vector is used to predict the pixel intensity value $F_k(i_k, j_k)$ from $F_{k\text{-}1}$. In practice, a linear displacement vector is determined by ME that searches from $F_{k\text{-}1}$ an image point that closely matches $F_k(i_k, j_k)$. In $F_{k\text{-}1}$, the search center is assigned by (i_k, j_k), so an yielded linear displacement vector is set to point from (i_k, j_k) to a candidate position. For example, when $(i_{k\text{-}1}, j_{k\text{-}1})$ is tested,

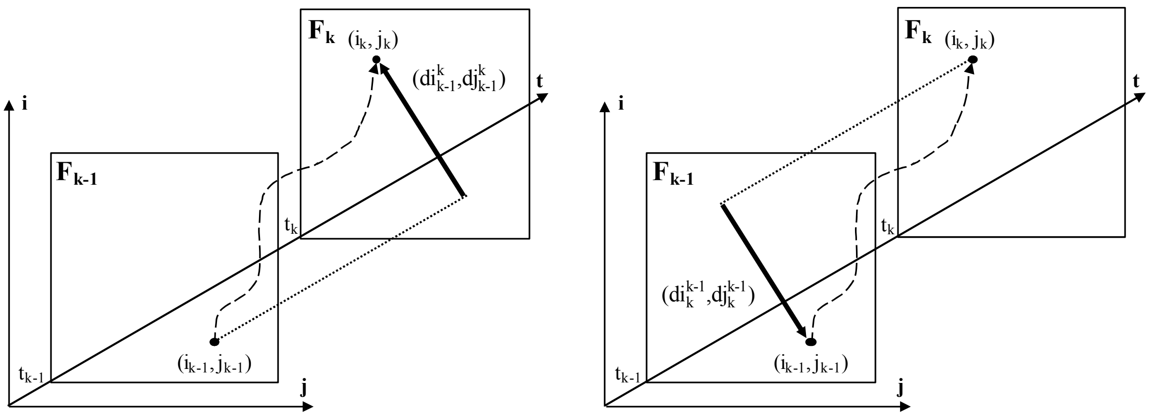
$$d_k^{k-1}(di_k^{k-1}, dj_k^{k-1}) = (i_{k-1} - i_k, j_{k-1} - j_k)\,. \tag{3.2}$$

Since the prediction occurs in the negative direction of time (from t_k to $t_{k\text{-}1}$), d_k^{k-1} in (3.2) is in opposite direction to d_{k-1}^{k} in (3.1).

If the intensity of the image point remains close to constant along its motion trajectory, locations $(i_{k\text{-}1}, j_{k\text{-}1})$ and (i_k, j_k) have approximately equal pixel intensity values, i.e.,

$$F_k(i_k, j_k) \approx F_{k-1}(i_{k-1}, j_{k-1})\,. \tag{3.3}$$

Under this assumption, $F_{k-1}(i_{k-1}, j_{k-1})$ can be regarded as the *best match* for $F_k(i_k, j_k)$.



a) Motion model of image point b) Temporal prediction of pixel value

Figure 3.1. A trajectory of an image point and associated linear displacement vector.

In order to compensate the movement of the image point, MC selects the best match $F_{k-1}(i_{k-1}, j_{k-1})$ as a prediction $P_k(i_k, j_k)$ for $F_k(i_k, j_k)$. $F_{k-1}(i_{k-1}, j_{k-1})$ can be addressed from the search center (i_k, j_k) with (3.2), so

$$P_k(i_k, j_k) = F_{k-1}\left(i_k + di_k^{k-1}, j_k - dj_k^{k-1}\right). \tag{3.4}$$

However, according to (3.3), $F_k(i_k, j_k) \neq P_k(i_k, j_k)$ due to a small *prediction error*. The prediction error $E_k(i_k, j_k)$ is computed by subtracting (3.4) from $F_k(i_k, j_k)$ as

$$E_k(i_k, j_k) = F_k(i_k, j_k) - P_k(i_k, j_k). \tag{3.5}$$

As a result, $F_k(i_k, j_k)$ equals the sum of (3.4) and (3.5). Let us assume that a system contains the transmitter and receiver ends. The receiver end can accurately resolve $F_k(i_k, j_k)$ from $F_{k\text{-}1}$ without having access to F_k, if the transmitter end computes and transmits (3.2) and (3.5) to the receiver end.

This simplified prediction example can be generalized to MCP technique that is used in the existing video coders. The encoder performs the prediction whereas the decoder uses the prediction result in reconstruction of original data. Using prediction result instead of original data is reasonable, since prediction result can be compressed more efficiently for data storage and transmission. In the following, MCP in the general video encoding scheme is considered.

3.2.3 Motion compensated prediction in video encoders

MCP is introduced in the 1960s [37]. As described in Section 2.3.1, the current video encoders implement MCP in three phases: ME, MC, and DFD computation.

MCP predicts pixel values from previously encoded and reconstructed reference frame(s) F_R instead of original frames. By that way, the effects of lossy encoding are avoided and the encoder and decoder access identical reference data during MC. F_R can contain a single frame or a set of multiple frames. In addition, the coding order of the current frame F_C and F_R may deviate from the original (display) order of frames. Hence, the time interval and chronological order of F_C and F_R can vary arbitrarily in the range of the underlying standard. Due to aforementioned reasons, F_C and F_R symbols are not mapped to any specific time instants, i.e., F_C, F_R, and $(i, j) \in \Lambda$ are indentified without previously used time indexes *k* and *k*-1, respectively.

The first phase of MCP is ME. It searches for the best match and addresses search locations with displacement vector. ME replaces the location-specific notation of (3.2) with a generalized displacement vector that is called a *motion vector*

$$\text{MV} = (\text{MV}i, \text{MV}j). \tag{3.6}$$

MV specifies the displacement between a search center (i, j) and an arbitrary search location in F_R. The output of ME is a MV that represents the best match for $F_C(i, j)$.

After ME, MC uses the best match as a prediction $P(i, j)$ for $F_C(i, j)$. It addresses the best match from F_R with the *minimum MV*. For MCP, (3.4) can be rewritten as

$$P(i, j) = F_R(i + \mathrm{MV}i, j + \mathrm{MV}j), \tag{3.7}$$

MCP uses $P(i, j)$ in DFD computation. In MCP, $E_k(i_k, j_k)$ is referred to as DFD(i, j), so (3.5) can be rewritten as

$$\mathrm{DFD}(i, j) = F_C(i, j) - P(i, j). \tag{3.8}$$

So far, MCP has only concentrated on a single pixel. However, the same principles can be extended to other *regions of support* as well.

3.2.4 Region of support

Region of support (ROS) is the set of image points to which motion models apply. The common sizes of ROS are: *a single pixel* $((i, j))$, *the whole image* (Λ), *irregular-shaped region of pixels*, and *rectangular region of pixels* $(Q \times U)$ [111].

The least constrained approach is obtained when ROS = (i, j). In this case, each pixel has an individual MV (as in Figure 3.1). This approach provides the smallest prediction error [111]. However, the computational complexity of this approach is intolerable, since the movement of each image point is described with at least two parameters. In addition, transmitting a MV for each pixel would outweigh the gain of small residuals in practice. Hence, the pixel-based MV field is too dense for *global* or *local ME/MC*.

The opposite extreme is reached when $\mathrm{ROS} = \Lambda$ in which case a single motion model refers to all pixels. In this approach, the motion of the whole image is approximated with a small set of motion (*warping*) parameters. Since $\mathrm{ROS} = \Lambda$, the frame is not partitioned at all, so artifacts at the borders of adjacent partitions are avoided. In general, the representable motion field of this approach is too constrained and sparse for local ME/MC. Instead, it can be used to predict global motion, according to which all image points are considered as *inliers* by default, i.e., points that follow global motion. However, a typical video sequence also contains image points (*outliers*) that move independently of the global motion. Outliers may have a crucial influence on the accuracy of global ME/MC, so they should be eliminated. Global MC has been adopted in MPEG-4 ASP. Although the share of warping parameters is negligible in total transmission bit rate, global ME/MC induces substantial computational overhead during encoding. H.264/AVC replaces global ME/MC by allowing a non-zero MV for the skip mode [70] whose MV has conventionally been set to zero. This modification can be implemented without additional encoding complexity and it achieves even better performance than global ME/MC. In addition, part of the global motion is typically counteracted before compression by physically measuring it at the camera. For example, many professional and consumer camcorders incorporate image stabilization systems that automatically compensate camera shake. Due to these reasons, the importance of global ME/MC is not high in video encoding nowadays, so further examination of it is omitted in this Thesis.

Irregular-shaped ROS applies to irregular region of pixels. Its motivation is to find a trade-off between prediction accuracy and the number of motion parameters. In a typical video scene, irregular-shaped moving objects are rarely aligned along the boundaries of a regular ROS. Therefore, the irregular-shaped ROS is needed to address these arbitrarily located objects. *Region-based ME/MC* with irregular-shaped ROS is used in object-based video coding. In region-based ME/MC, a square block is divided into arbitrarily shaped regions and each of them is allowed to undergo independent translational motion. In this case, motion representation includes a set of motion parameters and description of region boundary. Object-based coding has been included in MPEG-4 Visual, but it has not been popularized, since it is a complex task to segment a video scene into meaningful objects and encode them. The benefit of smaller prediction residual is also easily lost due to more complex motion representation. These practical difficulties prevent the efficient usage of irregular-shaped ROS which remains primarily interest of research purposes today.

Hereafter, this Thesis concentrates only on the rectangular ROS that refers to rectangular region of pixels. Although it is more rigid than irregular-shaped ROS, it has still proved to be the most powerful approach for local ME/MC in practice.

3.3 Block-based motion estimation and compensation

All the practical standard-compliant encoders implement MCP with *block-based ME/MC* that uses translational/linear model over the rectangular ROS. Block-based ME/MC has proved to predict a variety of different motions accurately enough, if the rectangular ROS is applied to small region of pixels. Block-based ME/MC is well suited for video encoding, since it is compatible with rectangular video frames and with block-based transforms such as DCT and ICT (Section 2.3.2). In addition, it can be relatively easily implemented in HW due to its regularity. In the existing video encoders, block-based ME/MC is capable of the largest reduction of the output bit rate. It outperforms pure spatial encodings at least by a factor of three [6].

In block-based ME/MC, the current frame is divided into non-overlapping luminance blocks of 16×16 pixels, i.e., MBs whose union tiles the whole current frame. Block-based ME/MC processes current MBs individually. For each MB, block-based ME searches for the best match in the reference frame(s). After the best match has been found, block-based MC applies it as a prediction for the current MB.

Block-based ME belongs to the non-normative part of the considered standards. However, only limited freedom is allowed to its realization, since the standardized block-based MC sets several normative rules and constrains that also impact on ME. These standard-specific rules and constraints include definitions for the reference block sizes, prediction accuracy (IME/FME), MV prediction/coding schemes, and the amount of reference frames. All these aspects are considered in this section.

3.3.1 Fixed and variable block-size motion estimation

The standards specify block sizes of ME through *inter coding modes*, so that each mode is tailored to a single block size. In progressively-coded frames, let us denote inter coding modes of the encoder with m^{ψ} and associated block sizes as

$$Q^{\psi} \times U^{\psi} \in \{16\times16, 8\times16, 16\times8, 8\times8, 8\times4, 4\times8, 4\times4\}, \tag{3.9}$$

where $\psi \in [1,7]$ is the index of the inter coding mode.

Figure 3.2 depicts m^1,..., m^7. The pixel blocks called *partitions* (p_{η}^{ψ}) of each m^{ψ} are presented in 2D coordinates (q, u) and identified with $\eta \in [0, n^{\psi}-1]$, where $n^{\psi} = 256/(Q^{\psi} \times U^{\psi})$. For example, m^3 is composed of $n^3 = 256/(8\times16) = 2$ partitions denoted as p_0^3 and p_1^3. These two 8×16 pixel blocks of m^3 are indicated with numbers 0 and 1 in Figure 3.2. In this Thesis, MV of p_{η}^{ψ} is denoted as $\mathrm{MV}_{\eta}^{\psi}(\mathrm{MV}i_{\eta}^{\psi}, \mathrm{MV}j_{\eta}^{\psi})$ and it is assumed to point to the top-left corner of p_{η}^{ψ}. For example, MVs of p_0^3 and p_1^3 are MV_0^3 and MV_1^3, respectively. Only a single MV is defined per p_{η}^{ψ}, since the pixels of the same p_{η}^{ψ} are expected to undergo uniform motion.

The considered standards implement block-based ME with different subsets of (3.9). H.261 and MPEG-1/2 are compatible with *fixed block-size ME* (*FBSME*). FBSME operates only on MBs that are partitioned according to m^1 ($\psi = 1$). For each current MB, FBSME searches for the best matching MB (p_0^1) and yields a single MV (MV_0^1) for it.

Since moving objects rarely follow boundaries of MBs, smaller partition sizes may improve prediction accuracy of ME. H.263, MPEG-4 Visual, and VC-1 address this possibility by specifying two inter coding modes: *1MV mode* and *4MV mode*. 1MV mode equals m^1, whereas 4MV mode divides a MB into four 8×8 pixel blocks according to m^4. Particularly, m^4 tends to obtain smaller prediction error than m^1 when the covered area contains complex or non-uniform motion such as object boundaries. However, transmitting four MVs ($\mathrm{MV}_0^4, \mathrm{MV}_1^4, \mathrm{MV}_2^4, \mathrm{MV}_3^4$) instead of one can easily outweigh the benefit of the reduced prediction error. Therefore, H.263, MPEG-4 Visual, and VC-1 adapt the block size to the content of video by supporting either m^1 or m^4 on a MB by MB basis $(\psi \in \{1,4\})$.

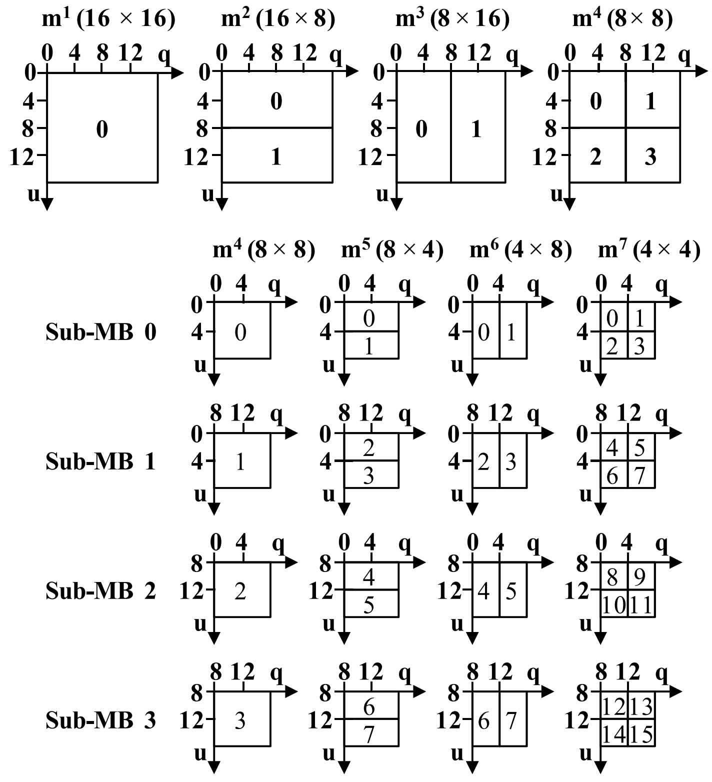


Figure 3.2. Seven inter coding modes and associated MB partitions.

H.264/AVC increases the amount of inter coding modes further by introducing *variable block-size ME* (*VBSME*) that covers the entire set of (3.9). At MB level, a 16×16 pixel block can be partitioned into 16×16, 8×16, 16×8, or 8×8 blocks according to $m^1,\ldots, m^4$. In addition, 8×8 blocks (sub-MBs 0 - 3) obtained with m^4 can be further sub-divided into disjoint $8\times4, 4\times8$, or 4×4 blocks according to $m^5,\ldots, m^7$. A single MB or sub-MB can only contain partitions of the same size. For example, p_0^2 cannot be combined with p_2^4 and p_3^4. Instead, each of the four sub-MBs can be partitioned according to m^4, m^5, m^6, or m^7.

VBSME improves the prediction accuracy over the previous approaches, since adaptively selected partition sizes accommodate better of different object movements. A large partition size is often suitable for homogeneous areas whereas a small partition size tends to be beneficial for detailed areas. However, m^1 is typically the dominating mode in VBSME, since the coding cost of MVs and other side information restrains the selection of smaller partitions. For example, ten widely used CIF sequences have been tested with H.264/AVC encoder in [44]. When QP = 20, average shares of m^1, m^2/m^3, m^4, and intra MBs are 60%, 10%, 28%, and 2%, respectively. The respective shares are 76%, 11%, 11%, 2% with QP = 30 and 89%, 6%, 2%, 3% with QP = 40. The proportion of m^1 increases with QP, since the coding cost of MVs and other side information is emphasized at low bit rates.

As the large share of m^1 suggests, restricting the inter coding modes to smaller subset of (3.9) does not necessarily degrade the coding quality. In [75], six well-known CIF sequences were executed with H.264/AVC encoder using eight different QPs (QP = 12, 16, …, 40) and three different sets of modes: m^1, m^1,…, m^4, and m^1,…, m^7. Constraining modes of complete VBSME (m^1,…, m^7) to m^1,…, m^4 and to m^1 increases average BD-rate by 2.8% and 11.8%, respectively. Hence, removing m^5,…, m^7 increases BD-rate only slightly. Furthermore, m^5,…, m^7 become less significant with higher resolutions. For example, with 1080p format, VBSME seldom selects m^5,…, m^7 and they typically do not improve coding result at all (especially when RDO is off) [87].

In the progressive video considered above, a *coded picture* always represents a complete progressively-coded frame. In turn, a coded picture in an interlaced video represents either an entire frame (*frame-picture*) or a single field (*field-picture*). In the frame-picture, odd- and even-numbered fields are interleaved and coded together as a single picture, whereas a single field-picture only contains either odd- or even-numbered fields.

MPEG-2, MPEG-4 Visual, VC-1, and H.264/AVC define two MCP techniques for interlaced video formats: *frame-based MCP* and *field-based MCP*. Frame-based MCP obtains prediction based on the frame-pictures. It uses the same predictive method as MCP for the progressive frames. For example, a MB is considered as a 16×16 pixel block. In field-based MCP, prediction is obtained based on reference fields. Field-based MCP can be applied either for the frame-pictures or field-pictures. It processes the frame-picture as two separate fields and performs independent predictions for them. Respectively, field-based MCP obtains prediction from two field-pictures by performing separate predictions for them. In both cases, a MB is considered as two 16×8 pixel blocks (one block per field). The standard-specific details [50], [51], [57], [109] of frame-based MCP and field-based MCP are omitted here, since this Thesis only focuses on MCP for the progressive frames.

3.3.2 Integer and fractional motion estimation

Integer ME (*IME*) assumes that inter-frame displacement of objects always equals integer number of pixels. However, objects often move horizontally and vertically to a position that is not on an integer-pixel grid but between the integer pixels in a fractional-pixel position. Therefore, most of the considered standards are compatible with *fractional ME* (*FME*) that refines integer-pixel accurate MVs to fractional-pixel (sub-pixel) accuracy. MPEG-1/2 and H.263 allow *half-pixel* (*½-pixel*) accurate MVs, whereas MPEG-4 Visual, VC-1, and H.264/AVC extend MVs to a *quarter pixel* (*¼-pixel*) accuracy.

The original reference frame F_R contains only integer-pixel values. Hence, reference pixel values on fractional-pixel grids have to be artificially generated before FME. MPEG-1/2 and H.263 obtain pixel values at ½-pixel positions with simple *bilinear interpolation* from the pixel values at the nearest integer-pixel locations. In MPEG-4 Visual, ½-pixel values are interpolated from the surrounding integer-pixel values using *eight-tap Finite Impulse Response (FIR) filter* after which ¼-pixel values are obtained with bilinear interpolation between the nearest ½ or integer-pixel values. Interpolation scheme in H.264/AVC resembles that of MPEG-4 Visual, except *six-tap FIR filter* is used to generate ½-pixel locations. VC-1 supports two interpolation methods: bilinear and *bicubic* [73], [109]. They can be adaptively selected. Bilinear interpolation is primarily targeted for resource constrained environments, whereas more complex bicubic interpolation is recommended for high quality video applications. For chrominance samples, all the considered standards use bilinear interpolation.

Figure 3.3 shows luminance interpolation scheme of H.264/AVC as an example. All ½-pixel values (H^h, H^v, and H^d) are interpolated from integer-pixel values (I) using a 1D six-tap FIR filter with weights of 1/32, -5/32, 5/8, 5/8, -5/32, and 1/32. For example, H^h value at $(i.5, j)$ is filtered from the six horizontally adjacent integer-pixel values and rounded as

$$H^h_{(i.5,j)} = \left(\left(I_{(i-2,j)} - 5\times I_{(i-1,j)} + 20\times I_{(i,j)} + 20\times I_{(i+1,j)} - 5\times I_{(i+2,j)} + I_{(i+3,j)}\right) + 16\right)/32, \quad (3.10)$$

Respectively, H^v value at $(i, j.5)$ is filtered from vertically adjacent integer-pixel values and rounded as

$$H^v_{(i,j.5)} = \left(\left(I_{(i,j-2)} - 5\times I_{(i,j-1)} + 20\times I_{(i,j)} + 20\times I_{(i,j+1)} - 5\times I_{(i,j+2)} + I_{(i,j+3)}\right) + 16\right)/32, \quad (3.11)$$

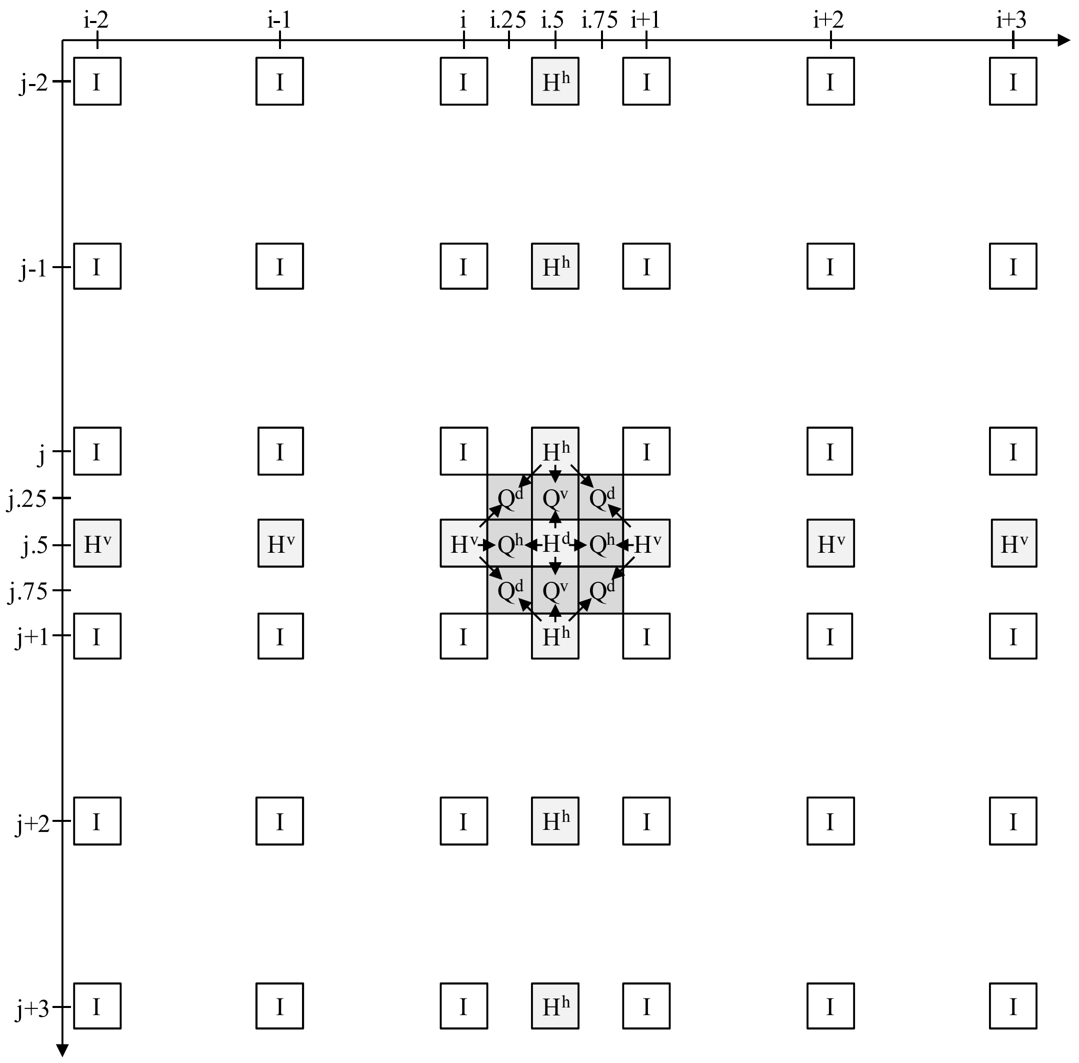


Figure 3.3. Interpolation of luminance samples in H.264/AVC.

After all H^h and H^v values have been computed according to (3.10) and (3.11), the remaining ½-pixel values (H^d) can be filtered either from the six horizontally adjacent (unrounded) H^v values or from the six vertically adjacent (unrounded) H^h values.

Once all H^h, H^v, and H^d values are available, the ¼-pixel values (Q^h, Q^v, and Q^d) are obtained through bilinear interpolation. I.e., two immediately adjacent integer or ½-pixel values are averaged and rounded. In Figure 3.3, the arrows indicate averaged values. For example, Q^d value at (i.25, j.25) is produced by interpolating between a diagonally opposite H^h and H^v values as

$$Q^d_{(i.25,j.25)} = \left(H^h_{(i.5,j)} + H^v_{(i,j.5)} + 1\right)/2. \tag{3.12}$$

In the considered standards, the primary purpose of the interpolation filter is to compensate prediction error caused by *spatial aliasing* [85], [129]. Spatial aliasing occurs between high frequency signals that represent sharp edges in the observable image. Its impact vanishes at integer-pixel displacements, but grows to maximum at ½-pixel displacements. Therefore, accurate interpolation of pixel values at fractional-pixel positions has a crucial role in reducing the impact of aliasing. MPEG-4 Visual, VC-1, and H.264/AVC have addressed the importance of interpolation by adopting more accurate interpolation filters at a cost of increased complexity.

FME lowers the overall bit rate significantly compared with plain IME, although fractional MVs consume more bits than integer MVs. In [75], the effect of FME has been evaluated with six well-known CIF sequences and with eight different QPs. Compared to plain IME, FME with ½ and ¼-pixel accuracies decreases average BD-rate by 24.6% and 37.3%, respectively. Interpolating luminance samples to 1/8-pixel accuracy would reduce prediction error further [129]. However, none of the considered standards supports 1/8-pixel accuracy, since it increases the complexity of interpolation. In addition, coding overheads of 1/8-pixel accurate MVs would easily outweigh the bit savings in prediction error especially with higher QP values.

This Thesis focuses only on IME stage in MCP. However, the subsequent FME stage is also considered when design decisions at IME stage are made.

3.3.3 Motion vector coding and prediction

The considered standards limit the allowed *MV range* in the reference frame by defining the maximum number of bits per MV. For example, H.263 limits the ranges of MV components (at fractional pixel accuracy) to $\mathrm{MV}i, \mathrm{MV}j \in [-16, 15.5]$ by default. Although the MV range of H.263 can be optionally extended to $\mathrm{MV}i, \mathrm{MV}j \in [-31.5, 31.5]$, the other standards excluding H.261 allow significantly larger MV ranges. For example, $\mathrm{MV}i \in [-2048, 2047.75]$ in H.264/AVC, whereas the level-specific range of MVj varies between $\mathrm{MV}j \in [-64, 63.75]$ and $\mathrm{MV}j \in [-512, 511.75]$. The specified MV range is unsymmetrical, since typical video scenes have stronger horizontal movement than vertical movement.

In a typical video scene, object motion easily extends across large regions of a frame. Therefore, MVs of neighboring blocks are often highly correlated. The correlation is particularly strong for small blocks and large moving objects. Hence, the compression of the MV field can be improved by predicting each MV from spatially adjacent, previously-processed MVs.

In the considered standards, an actual $\mathrm{MV}_{\eta}^{\psi}$ of each partition p_{η}^{ψ} is differentially encoded as

$$\mathrm{MVD}_{\eta}^{\psi} = \mathrm{MV}_{\eta}^{\psi} - \mathrm{MVP}_{\eta}^{\psi}, \tag{3.13}$$

where $\mathrm{MVP}_{\eta}^{\psi}(\mathrm{MVP}i_{\eta}^{\psi}, \mathrm{MVP}j_{\eta}^{\psi})$ is a *MV predictor* (*MVP*) for p_{η}^{ψ} and $\mathrm{MVD}_{\eta}^{\psi}(\mathrm{MVD}i_{\eta}^{\psi}, \mathrm{MVD}j_{\eta}^{\psi})$ indicates *MV difference* (*MVD*). $\mathrm{MVD}_{\eta}^{\psi}$ is encoded and transmitted as a pair of variable length codewords, one for $\mathrm{MVD}i_{\eta}^{\psi}$ and the other for $\mathrm{MVD}j_{\eta}^{\psi}$. Each MVD value is assigned a unique codeword, so that the shorter the MVD value, the shorter the codeword. In addition, none of the codewords is a prefix for the other. Encoding MVDs instead of MVs typically reduce output bit rate of the encoder, since MVDs tend to require less bits for their VLC than the associated MVs. Each video coding standard specifies VLC syntax for MVD in detail [49]-[51], [55]-[57], [109]. These standard-specific syntax specifications are omitted in this Thesis.

Video coding standards also describe the computation of MVP which depends on the current block size and the availability of *MVP candidates* (MVs of adjacent partitions). Figure 3.4 illustrates the MVP and associated MVP candidates when the current block as well as all the adjacent blocks equal MBs, i.e., they are partitioned according to m^1 (Figure 3.2). In this case, MVP for the current p_0^1 is denoted as MVP_0^1. The possible MVP candidates are located immediately to the *left* ($\mathrm{MV1}_0^1$), *top-left* ($\mathrm{MV2}_0^1$), *top* ($\mathrm{MV3}_0^1$), and *top-right* ($\mathrm{MV4}_0^1$).

H.261 and MPEG-1/2 use a simple prediction by assigning $\mathrm{MVP}_0^1 = \mathrm{MV1}_0^1$. If $\mathrm{MV1}_0^1$ is unavailable, $\mathrm{MVP}_0^1 = 0$. Common reasons for unavailability of the MVP candidate are that the neighboring partition is outside the frame boundaries or it is coded as an intra mode. The newer video coding standards predict MVP_0^1 through *median prediction* of three MVP candidates as

$$\mathrm{MVP}_0^1 = \mathrm{median}(\mathrm{MV1}_0^1, \mathrm{MV3}_0^1, \mathrm{MV4}_0^1), \tag{3.14}$$

where the median value is independently computed for $\mathrm{MVP}i_0^1$ and $\mathrm{MVP}j_0^1$. If all MVP candidates are not available, (3.14) is modified according to underlying standard. Usually, missing $\mathrm{MV1}_0^1$ is set to zero and missing $\mathrm{MV4}_0^1$ is replaced by $\mathrm{MV2}_0^1$. If only $\mathrm{MV1}_0^1$ is available, $\mathrm{MVP}_0^1 = \mathrm{MV1}_0^1$. When none of the MVP candidates is available, $\mathrm{MVP}_0^1 = 0$.

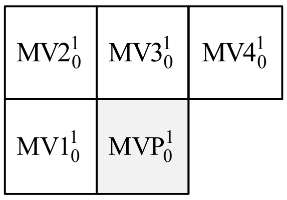


Figure 3.4. Basic principle of the motion vector prediction.

H.263, MPEG-4 Visual, and VC-1 specify MVP_0^1 in 1MV mode and separate MVPs (MVP_0^4, …, MVP_3^4) for p_0^4, …, p_3^4 in 4MV mode. Correspondingly, H.264/AVC supports individual MVP_η^ψ for each p_η^ψ.

EC stage of the encoder (Figure 2.1) computes standard-specific MVPs for the best matching partitions and applies them to yield MVDs according to (3.13). MVs for (3.13) are received from ME. The accurate computation of each MVP_η^ψ is omitted here but specified in detail by the underlying video coding standard [49]-[51], [55]-[57], [109].

3.3.4 Reference frames in motion estimation

As discussed in Section 2.3.1, H.261/3, MPEG-1/2, MPEG-4 Visual, and VC-1 limit ME to reference to the closest past and/or to the closest future frame. Instead, H.264/AVC extends the set of available reference frames by introducing *multiple reference frame ME* (*MRFME*). For MRFME, H.264/AVC specifies a *decoded picture buffer* (*DPB*) that can contain up to 16 reference frames. MCP of each MB can be derived from one or more of the reference frames in the DPB. MBs in *P*-slices can have four reference frames at maximum, i.e., one reference frame per sub-MB (Figure 3.2). Correspondingly, MBs in *B*-slices can reference to eight frames due to double references in each sub-MB.

The H.264/AVC encoder marks an encoded frame as *short-term*, *long-term*, or *unused* reference frame. Classification of reference frames depends on the encoder. By default, an encoded frame is marked as a short-term frame, i.e., a recently-coded frame available for prediction. Long-term frames are selected based on temporal correlation impact over a relatively long range in time. The encoder can assign a short-term frame to a long-term frame or any frame to an unused frame that can be replaced from the DPB. If the DPB is full, the oldest short-term picture is removed from the buffer, whereas long-term frames remain in the DPB until explicitly removed or replaced. The encoder transmits DPB control commands as side information in the slice header.

The DPB is organized into two lists of reference frames denoted as *list 0* and *list 1*. These possibly overlapping lists can be arbitrarily constructed from the available past or future reference frames in the DPB. *P*-slices reference only list 0, whereas *B*-slices can use both lists. However, the usage of arbitrarily constructed lists decouples the prediction direction and the slice type. Hence, the only difference between *P* and *B* slices is that the blocks of *B*-slice can additionally support average of two distinct predictions. Figure 3.5 illustrates an example of MRFME in which list 0 and list 1 contain four and five reference frames, respectively. Reference frames in the lists are identified by a reference index (*refidx*).

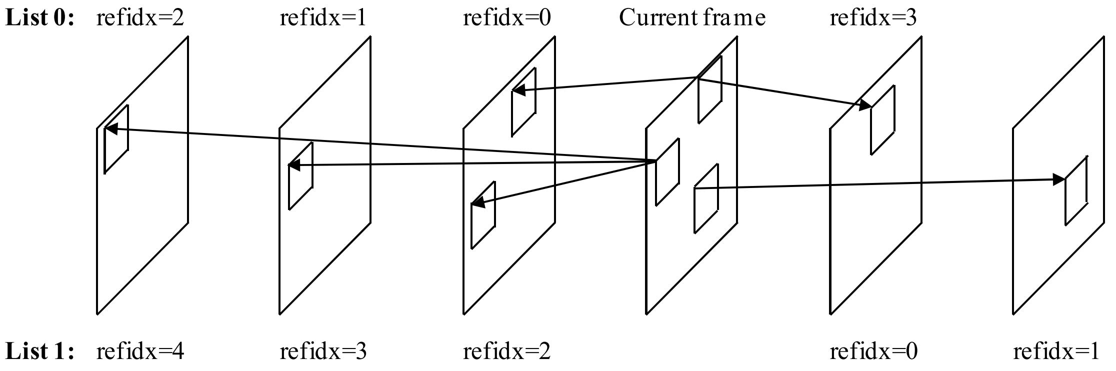


Figure 3.5. Multiple reference frame motion estimation in H.264/AVC.

By default, H.264/AVC uses ordinary reference frame data in MCP. In addition, H.264/AVC supports *weighted prediction*, in which the pixel values of the candidate blocks are scaled by weighting factors prior to MCP. Weighted prediction allows controlling relative contributions of reference frames in *P*/*B*-slice encoding. An encoder can use customized weighting factors that are explicitly transmitted in the slice header. When encoding *B*-slices, weighting factors can also be implicitly derived for each reference frame (list 0 and list 1) as an inverse proportion to its temporal distance from the current frame. Weighted prediction provides coding gains in scenes that contain illumination change and fade transitions, i.e., when one scene fades into other.

The potential benefits of MRFME over single reference frame ME are listed, e.g., in [75] and [112]. MRFME may find a better match for image regions that are invisible in the immediately previous frame but visible two or more frames ago. The discontinuous appearance of objects and backgrounds may be caused by repetitive object motion and occlusions. Secondly, camera-induced periodic global motion such as camera shake or alternating camera angles may be better predicted from the set of previous frames. Thirdly, illumination and shadow conditions as well as noise and environmental characteristics may be more reliable predicted from the multiple references. In all these cases, the common reason behind possible improvement of MRFME is that the larger number of tested candidates increases the probability for a better match. However, the drawback of MRFME is huge computational complexity and memory consumption.

MRFME with ρ reference frames requires ρ times the memory capacity over ME with a single reference frame. Similarly, the computational complexity increases linearly with ρ, if the same search scheme is used for each reference frame. Although MRFME may be useful in the cases listed above, it still typical that the closer reference frame has higher correlation with the current frame. For example, ten widely used CIF sequences have been tested with five reference frames in [44] and averagely 68%, 81%, and 92% of the optimal MVs belong to the nearest reference frame with QP values of 20, 30, and 40, respectively. The probability increases with QP, since the coding cost of reference indices is emphasized at lower bit rates.

In [75], MRFME has been tested with six well-known CIF sequences and with eight different QPs. Compared to single reference frame ME, MRFME with 2, 4, 8, and 16 reference frames reduced the average BD-rate by 4.0%, 6.9%, 7.7%, and 7.6%, respectively. Hence, the highest BD-rate gain per additional frame is achieved when incrementing the amount of reference frames from one to two after which the impact of an additional reference frame degraded exponentially. Utilizing more than eight frames had negligible or even slightly negative effect on BD-rate.

The bit rate gain of MRFME is mainly due to removal of spatial aliasing [85], so it compensates the same source of prediction error than interpolation filter (Section 3.3.2). MRFME is the most powerful with low-motion video sequences, since motion reduces spatial aliasing by smoothing sharp edges of the image [75], [85]. Spatial aliasing is also inherently diminished through denser sampling grid, so the benefit of MRFME is significantly degraded at higher resolutions [87]. Three popular 1080p test sequences have been tested in [87] according to which MRFME with five reference frames only provided 0.1 dB PSNR gain over single reference frame ME at maximum. Therefore, spatial aliasing can be adequately removed at higher resolutions with interpolation filter without increasing the amount of reference frames from one.

Since considerable RD performance gains of MRFME are limited to low-resolution and low-motion sequences with small QP, only single reference frame ME is considered hereafter. However, the proposed principles can be extended to MRFME on demand.

3.3.5 Summary of standard-specific techniques and emerging trends

Table 3.1 gathers the essential standard-specific ME/MC characteristics considered in this section. MPEG-4 Visual, H.264/AVC, and VC-1 specify several MV ranges of which the minimum and maximum are reported.

The best-performing proposals for the future HEVC standard [5], [35], [63], [89], [124] are all based on the conventional hybrid video coding scheme applied already in H.261. These proposals also adopt the basic principles of MCP from the prior standards without fundamental changes. I.e., they rely on block-based ME/MC with variable block sizes and pixel accuracies. In general, ME/MC characteristics of these HEVC proposals are close to that of H.264/AVC (Table 3.1). However, compared with H.264/AVC, the following essential upgrades are introduced to achieve additional coding gain.

Firstly, the majority of the HEVC proposals extend the available inter coding modes to also cover 32×32 and 64×64 pixel blocks [63], [89], [124] or even 128×128 pixel blocks [35]. Larger partitions exploit spatial correlation more efficiently, so they are well suited to represent large homogeneous areas that are common in high resolution videos. The HEVC proposals divide these enlarged partitions into smaller ones according to *flexible* [35], [63], [89] or *simplified* [124] *multi-depth quadtree structure*. Typically, they also support *symmetric rectangular partitions* [35], [63], [124], *asymmetric rectangular partitions* [35], and/or *non-rectangular geometric partitions* [63] in MCP. Instead, none of the proposals suggests special partitions for the obsolete interlaced video format.

Table 3.1. Standard-specific key characteristics of motion estimation and compensation.

Standard	H.261	MPEG-1	MPEG-2	H.263	MPEG-4 Visual	H.264/AVC	VC-1
Year of standardization	1990	1993	1996	1996	1999	2003	2006
MVi range	[-15, 15]	[-512, 511.5]	[-2048, 2047.5]	[-16, 15.5], [-31.5, 31.5]	[-16, 15.5] to [-1024, 1023.5]	[-2048, 2047.75]	[-64, 63.75] to [-1024, 1023.75]
MVj range	[-15, 15]	[-512, 511.5]	[-2048, 2047.5]	[-16, 15.5], [-31.5, 31.5]	[-16, 15.5] to [-1024, 1023.5]	[-64, 63.75] to [-512, 511.75]	[-32, 31.75] to [-256, 255.75]
MVP	non-median	non-median	non-median	median	median	median	median
Supported modes	1	1	1	1 and 4	1 and 4	1-7	1 and 4
Global ME/MC	no	no	no	no	yes	no	no
Region-based ME/MC	no	no	no	no	yes	no	no
Support for interlace	no	no	yes	no	yes	yes	yes
ME/MC accuracy	integer pixel	½-pixel	½-pixel	½-pixel	¼-pixel	¼-pixel	¼-pixel
Interpolation	-	Bilinear	Bilinear	Bilinear	8-tap FIR/ Bilinear	6-tap FIR/ Bilinear	Bicubic, Bilinear
No. of reference frames	0-1	0-2	0-2	0-2	0-2	0-16	0-2
Bidirectional prediction	no	yes	yes	yes	yes	yes	yes
Weighted prediction	no	no	no	no	no	yes	no

Secondly, HEVC proposals improve the generation of ¼-pixel luminance values by using, e.g., more accurate interpolation filters [35], multiple interpolation filters [63], [124], and/or adaptive interpolation filters [5]. Some of the proposals also consider adaptive selection of MV resolution, so that MVs at ¼-pixel accuracy can be optionally refined to 1/8-pixel [5], [63] or 1/12-pixel [35] accuracies.

Thirdly, conventional median MV prediction is enhanced through *MV competition* [5], [35] which selects the best one of the several MVP candidates. The candidates can be obtained from spatially-located neighboring partitions or temporally co-located partitions.

Generally speaking, the common trend of these HEVC proposals is to increase the amount of the adaptive inter coding tools whose parameters are adjustable at run time [36]. The adaptive tools achieve bit rate savings over the contemporary fixed-parameter tools especially in high resolution video scenes that tend to contain strong signal variations in spatial and temporal domains. Since HEVC is still in an unfinished state, the final tool set for MCP can deviate more or less from the current draft versions. Therefore, further considerations on MCP in HEVC are omitted in this Thesis.

3.4 Block matching

In block-based ME, the search for the best match is realized with *block matching* [58]. This search technique compares the current block (current p_η^ψ) to a set of candidate blocks (candidate p_η^ψ) in the reference frame(s) and selects one of the candidates as the best matching block for the current p_η^ψ. In this Thesis, the best matching block is denoted as $p_{\eta*}^\psi$ and its MV as $\mathrm{MV}_{\eta*}^\psi$.

3.4.1 Basic operating principle

The actual *search range* of $p^{\psi}_{\eta*}$ is normally much smaller than the standard-specific maximum MV range (Table 3.1), since the computational complexity of the block matching tends to grow drastically when the search range is enlarged. However, the limited search area can still be sufficient, if it is properly selected as a function of the motion content, resolution, and time between F_C and F_R. Common test conditions for JM suggest that MVP-centered search range shall be $\mathrm{MV}i, \mathrm{MV}j \in [-32, 32]$ for QCIF/CIF/D1 formats (30 fps) and $\mathrm{MV}i, \mathrm{MV}j \in [-64, 64]$ for 720p (60 fps) and 1080p (24 fps) formats [115]. Especially in the real-time encoders, the search range may be further limited [6].

Figure 3.6 depicts a basic operating principle of the block matching. The current p^{ψ}_{η} is assumed to be a $Q^{\psi} \times U^{\psi}$ pixel block, whose top-left pixel is located at coordinates (i, j) in F_C. In F_R, $p^{\psi}_{\eta*}$ is sought from the $w \times h$ pixel rectangular region that is commonly referred to as a *search area* or a *search window*. In Figure 3.6, the corner pixels of the search area are separately identified for clarity.

There is a high probability that the best match is located close to the current p^{ψ}_{η} position, so the search area is typically centered according to (i, j) and the block matching is started either from MVP or from (i, j). The search range of $Q^{\psi} \times U^{\psi}$ pixel block is determined as $[-p_w, p_w]$ horizontally and $[-p_h, p_h]$ vertically, where $p_w = (w - Q^{\psi})/2$ and $p_h = (h - U^{\psi})/2$. A more compact representation of the search range is yielded by combining horizontal and vertical ranges as

$$(2p_w + 1) \times (2p_h + 1) = (w - Q^{\psi} + 1) \times (h - U^{\psi} + 1)\,. \tag{3.15}$$

In Figure 3.6, the current p^{ψ}_{η} (in F_C) contains three black objects: a square, a circle, and a triangle. In F_R, $p^{\psi}_{\eta*}$ found for the current p^{ψ}_{η} is pointed by $\mathrm{MV}^{\psi}_{\eta*}$, i.e., it is located at $(i + \mathrm{MV}i^{\psi}_{\eta*}, j + \mathrm{MV}j^{\psi}_{\eta*})$. The missing triangle describes the prediction error associated to $p^{\psi}_{\eta*}$.

The block matching is typically conducted in 1 - 3 steps. The number of available steps depends on the standard. At the first step, the block-matching is performed at IME stage and the search range equals (3.15). At the second step, the block-matching is conducted at FME stage with ½-pixel accuracy. Since interpolating ½-pixel values for the entire search range would be too complex in practice, the search range typically covers the interpolated positions that are immediately adjacent to the best integer-pixel match. At the third step, the block matching is continued at FME stage with ¼-pixel accuracy. In this case, the search range often covers the interpolated positions that are located around the best ½-pixel match.

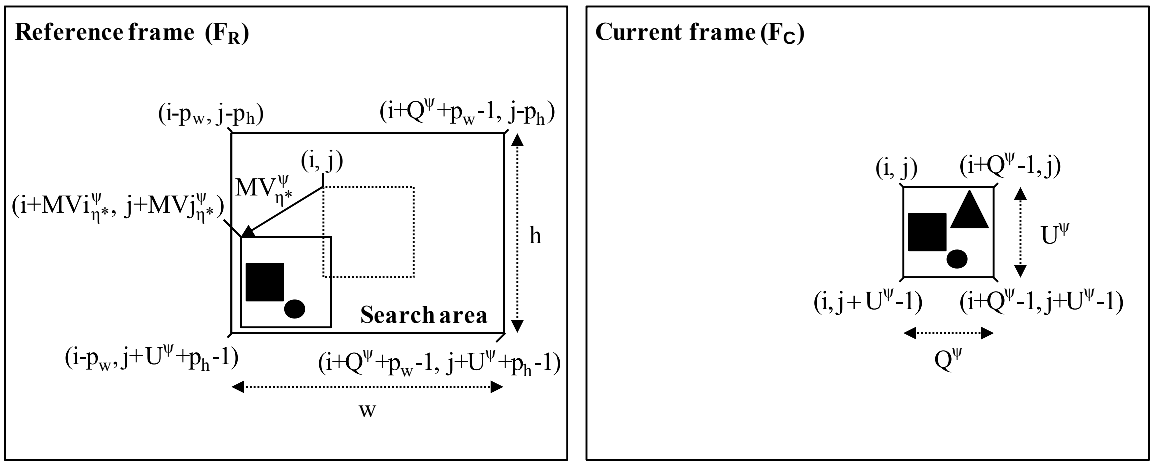

Reference frame (F_R)
(i-p_w, j-p_h)
(i+Q^ψ+p_w-1, j-p_h)
(i, j)
$MV^\psi_{\eta*}$
(i+$MVi^\psi_{\eta*}$, j+$MVj^\psi_{\eta*}$)
h
Search area
(i-p_w,j+U^ψ+p_h-1)
(i+Q^ψ+p_w-1, j+U^ψ+p_h-1)
w
Current frame (F_C)
(i, j)
(i+Q^ψ-1,j)
U^ψ
(i,j+U^ψ-1)
(i+Q^ψ-1,j+U^ψ-1)
Q^ψ

Figure 3.6. Block matching process.

3.4.2 Motion vector prediction for block matching

Starting the block matching from MVP instead of (*i*, *j*) usually improves search result, speeds up the search, and increases the correlation between adjacent $\mathrm{MV}^{\psi}_{\eta*}$ [122]. However, computing individual $\mathrm{MVP}^{\psi}_{\eta}$ for each p^{ψ}_{η} of a MB increases ME complexity and complicates its parallelization. Therefore, typical HW implementations of ME such as [11] and [27] replace exact MVP computation by a HW-oriented MV prediction, which only uses a single MVP per MB.

The most obvious choice for common MVP is MVP^{1}_{0}, since it predicts MV of the whole MB. The correlation between MVPs of different p^{ψ}_{η} is usually strong, so MVP^{1}_{0} is a good estimate for other $\mathrm{MVP}^{\psi}_{\eta}$. A common MVP is used to predict a search center, so the search may result in different $p^{\psi}_{\eta*}$ than with real standard-compliant MVP. In the worst case, the suboptimal $p^{\psi}_{\eta*}$ increases coding cost, but the compatibility with the underlying standard is still maintained.

This Thesis relies on HW-oriented MV prediction, which uses MVP^{1}_{0} as a common MVP for each p^{ψ}_{η} of a MB. The computation of MVP^{1}_{0} is derived from (3.14) and realized as

$$\mathrm{MVP}^{1}_{0} = \mathrm{median}(\mathrm{MV1}^{\psi 1}_{\eta 1}, \mathrm{MV3}^{\psi 3}_{\eta 3}, \mathrm{MV4}^{\psi 4}_{\eta 4}), \tag{3.16}$$

where neighboring MBs are partitioned according to $m^{\psi 1}$, $m^{\psi 3}$, and $m^{\psi 4}$. The ranges of $\psi 1$, $\psi 3$, and $\psi 4$ comply with standard-specific ranges of ψ. In (3.16), integer MVP candidates are used instead of fractional MVP candidates. By that way, MVP candidates are available immediately after IME. The utilized MVP candidates are unambiguously defined as a function of $\psi 1$, $\psi 3$, and $\psi 4$ as

$$\mathrm{MV1}_{\eta 1}^{\psi 1} \in \left\{\mathrm{MV1}_0^1, \mathrm{MV1}_0^2, \mathrm{MV1}_1^3, \mathrm{MV1}_1^4, \mathrm{MV1}_2^5, \mathrm{MV1}_3^6, \mathrm{MV1}_5^7\right\},$$

$$\mathrm{MV3}_{\eta 3}^{\psi 3} \in \left\{\mathrm{MV3}_0^1, \mathrm{MV3}_1^2, \mathrm{MV3}_0^3, \mathrm{MV3}_2^4, \mathrm{MV3}_5^5, \mathrm{MV3}_4^6, \mathrm{MV3}_{10}^7\right\},$$

$$\mathrm{MV4}_{\eta 4}^{\psi 4} \in \left\{\mathrm{MV4}_0^1, \mathrm{MV4}_1^2, \mathrm{MV4}_0^3, \mathrm{MV4}_2^4, \mathrm{MV4}_5^5, \mathrm{MV4}_4^6, \mathrm{MV4}_{10}^7\right\}.$$

The selection of these MVP candidates for MVP_0^1 is adopted from H.264/AVC. An example prediction of MVP_0^1 is depicted in Figure 3.7 in which the neighboring MBs (Figure 3.2) are partitioned according to m^2, m^6, and m^5 ($\psi 1 = 2$, $\psi 3 = 6$, and $\psi 4 = 5$). In (3.16), missing $\mathrm{MV1}_{\eta 1}^{\psi 1}$ and $\mathrm{MV3}_{\eta 3}^{\psi 3}$ are set to zero. If $\mathrm{MV4}_{\eta 4}^{\psi 4}$ is unavailable, it is replaced by $\mathrm{MV2}_{\eta 2}^{\psi 2} \in \left\{\mathrm{MV2}_0^1, \mathrm{MV2}_1^2, \mathrm{MV2}_1^3, \mathrm{MV2}_3^4, \mathrm{MV2}_7^5, \mathrm{MV2}_7^6, \mathrm{MV2}_{15}^7\right\}$ (Figure 3.4).

3.4.3 Matching criteria in block matching

In block matching, the selection of $p_{\eta *}^{\psi}$ is done by minimizing a cost function which is typically based on similarity criterion. Various similarity criteria have been proposed for the block distortion computation in the literature [68].

In IME, the most widely used distortion criteria are the *sum of squared differences* (*SSD*) and the *sum of absolute differences* (*SAD*). At the current block location (*i*, *j*), SSD and SAD are defined as

$$\mathrm{SSD}\left(F_C, F_R(\mathrm{MV}_\eta^\psi)\right) = \sum_{q=i}^{i+Q^\psi-1} \sum_{u=j}^{j+U^\psi-1} \left(F_C(q,u) - F_R(q+\mathrm{MV}i_\eta^\psi, u+\mathrm{MV}j_\eta^\psi)\right)^2 \quad (3.17)$$

$$\mathrm{SAD}\left(F_C, F_R(\mathrm{MV}_\eta^\psi)\right) = \sum_{q=i}^{i+Q^\psi-1} \sum_{u=j}^{j+U^\psi-1} |F_C(q,u) - F_R(q+\mathrm{MV}i_\eta^\psi, u+\mathrm{MV}j_\eta^\psi)| \quad (3.18)$$

where $F_C(q, u)$ and $F_R(q+\mathrm{MV}i_\eta^\psi, u+\mathrm{MV}j_\eta^\psi)$ indicate pixels of the current frame and reference frame, respectively. A tested location of F_R is addressed by a candidate motion vector MV_η^ψ which equals a displacement of a candidate p_η^ψ from a location (*i*, *j*) of F_R. The size of the current and candidate p_η^ψ is $Q^\psi \times U^\psi$.

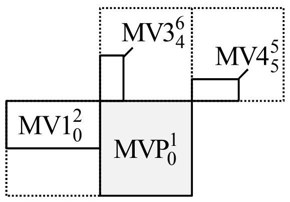


Figure 3.7. Example of the hardware-oriented motion vector prediction.

SAD and SSD are also referred to as the *sum of absolute errors* (*SAE*) and the *sum of squared errors* (*SSE*), respectively. Furthermore, dividing SAD by $Q^{\psi} \times U^{\psi}$ results in the well-known *mean absolute difference* or *error* (*MAD* or *MAE*). Respectively, the *mean squared difference* or *error* (*MSD* or *MSE*) is obtained by dividing SSD by $Q^{\psi} \times U^{\psi}$. The division by $Q^{\psi} \times U^{\psi}$ is often omitted in practice, i.e., SSD and SAD are used rather than their derivatives.

Sometimes, SAD and SSD criteria yield unequal result. For example, $1+1+6<3+3+4$, but $1^2+1^2+6^2>3^2+3^2+4^2$. In general, SSD tends to end up with better PSNR than SAD [32]. As shown in (2.1), PSNR is based on MSE, so there is a strong correlation between PSNR and SSD. In SSD, possible outliers (e.g., value 6 in the example above) are heavily weighted when squared. Hence, a block with homogeneous values often yields smaller SSD value than a block with outliers. In SAD, the quadratic function of SSD is replaced by an absolute value function that is less sensitive to outliers. Hence, SAD selects heterogeneous block as the best match more easily.

Although SSD yields better PSNR, it requires a multiplication per difference, whereas SAD can be implemented without multiplication. The range of SSD values is also larger than that of SAD values. For example, SSD value between two 8-bit pixels consumes 16 bits, whereas respective SAD value uses 8 bits. SAD processes each pixel pair separately, so it is also easily parallelizable. Due to these reasons, SAD is the most widely used criterion in HW encoders.

In FME, a common alternative to SAD is to use the *sum of absolute transformed differences* (*SATD*) as the distortion criterion. SATD implementations are typically based on 4×4 pixel blocks. In SATD computation, a residual block between corresponding pixels of F_C and F_R is first computed through subtraction as in (3.17) or (3.18) except that interpolated pixels of F_R are used in FME. The residual block is then *Hadamard transformed* through multiplication of Hadamard matrices. The matrix multiplications can be implemented with additions and shifts [128]. Finally, SATD cost is yielded by accumulating the absolute values of the Hadamard transformed residuals and dividing the sum by two.

Hadamard transform is a simplified approximation of the more complex ICT transform that H.264/AVC and VC-1 utilize in the subsequent TC stage of the encoder. Therefore, SATD emulates frequency characteristics of the prediction error better and provides more accurate estimation of the RD cost than SAD [99]. Higher estimation accuracy is beneficial in FME since it is the final refinement stage of ME. As a drawback, the computational complexity of SATD is increased over SAD.

JM supports alternatively SAD, SSD, or SATD in ME and inter mode decision. By default, it selects SAD for IME, SATD for FME, and SATD for inter mode decision. In order to balance the prediction result between distortion and consumed bits, modern encoders such as JM consider distortion measures jointly with bit rate [113], [133]. I.e., they implement rate-constrained block matching and inter mode decision by utilizing Lagrangian optimization techniques [28].

3.4.4 Rate-constrained block matching

Lagrangian RD optimization (*RDO*) techniques [113], [133] provide a systematic way to minimize *RD cost function* (*J*) that is generally expressed as

$$J = D + \lambda \times R, \tag{3.19}$$

where *D* is the distortion between a current p_η^ψ and its reconstructed prediction. *D* is introduced by quantization and it can be measured with distortion criteria discussed above. *R* represents the total bit consumption (bit rate) of an encoded prediction error. The error is computed between the current p_η^ψ and its non-reconstructed prediction. The *Lagrange multiplier* ($\lambda \geq 0$) is used to control a trade-off between *D* and *R*. Decreasing λ increases bit rates (and PSNR) and vice versa.

In practice, the computation of (3.19) necessitates execution of the entire encoding loop. I.e., DFD has to be processed through *T*, *Q*, and EC to derive *R* from the encoded bit stream (Figure 2.1). *R* accounts for the encoded bits consumed by the quantized TCOEFFs, MVs, and headers (reference frame index, inter coding mode, etc.). The reconstructed prediction of the current p_η^ψ, in turn, is obtained by processing quantized TCOEFFs through Q^{-1} and T^{-1} and adding *P* to the result (*P* + DFD').

In block matching, computing an actual RDO cost for each prediction candidate as in (3.19) would be impractical due to huge amount of candidates. Therefore, all practical encoders implement rate-constrained ME without complete RDO. I.e., RDO is disabled and block matching is conducted between the current p_η^ψ and its non-reconstructed prediction candidates. For efficiency reasons, *D* between the current p_η^ψ and each candidate p_η^ψ is typically measured with SAD in IME and SATD in FME. Since SAD and especially SATD value indirectly approximates the bit rate of TCOEFFs also [106], *R* is often reduced to rate term R_{MV} that represents bits of MVD_η^ψ only (Section 3.3.3). R_{MV} can be determined without encoding the DFD, so entire encoding loop is avoided. Inclusion of R_{MV} in RD cost computation is essential since its portion can be 20 - 30% of the overall bit rate [141].

A popular method is to estimate R_{MV} with a *look-up table* (*LUT*) that is constructed assuming that UVLC coding scheme is applied in MVD_η^ψ encoding [59], [78]. UVLC scheme (Section 2.3.3) obtains codewords from a simple and regular *Exp-Golomb code table* [33]. Hence, R_{MV} can be computed as a function of the length of Exp-Golomb codeword as

$$R_{\text{MV}}(\text{MVD}_\eta^\psi) = R_{\text{MV}}(\text{MV}_\eta^\psi - \text{MVP}_\eta^\psi) = \begin{cases} 1, & |\text{MVD}_\eta^\psi| = 0 \\ 2 \times \lfloor \log_2 |\text{MVD}_\eta^\psi| \rfloor + 3, & |\text{MVD}_\eta^\psi| > 0 \end{cases}. \tag{3.20}$$

This method favors smaller $\mathrm{MVD}_{\eta}^{\psi}$ lengths, so more regular MV fields are obtained.

The respective method is also introduced for CABAC coding scheme [78]. In UVLC and CABAC schemes, the bit consumption of reference frame index can be added to R_{MV} if MRFME is supported.

When D is computed with SAD, optimized RD cost computation can be derived for rate-constrained ME by substituting (3.18) and (3.20) to (3.19) as

$$J_{\eta}^{\psi}\left(\mathrm{MV}_{\eta}^{\psi},\lambda_{\mathrm{ME}}\right)=\mathrm{SAD}\left(F_C,F_R(\mathrm{MV}_{\eta}^{\psi})\right)+\lambda_{\mathrm{ME}}\times R_{\mathrm{MV}}(\mathrm{MV}_{\eta}^{\psi}-\mathrm{MVP}_{\eta}^{\psi}), \tag{3.21}$$

where λ_{ME} changes as a function of QP. Experiments in [113] and [133] have determined efficient standard-specific relationships between QP and λ_{ME} as

$$\lambda_{\mathrm{ME}}=\begin{cases}\sqrt{0.85\times 2^{(\mathrm{QP}\text{-}12)/3}}, & \text{with H.264/AVC}\\ \sqrt{0.85\times \mathrm{QP}^2}, & \text{with H.263 and MPEG-4 Visual}\end{cases} \tag{3.22}$$

when SAD or SATD is applied.

The rate-constrained ME selects a candidate p_{η}^{ψ} whose $\mathrm{MV}_{\eta}^{\psi}$ minimizes (3.21) as the best matching candidate ($p_{\eta*}^{\psi}$) for the current p_{η}^{ψ}. The minimum cost yielded by $p_{\eta*}^{\psi}$ is denoted as $J_{\eta*}^{\psi}\left(\mathrm{MV}_{\eta*}^{\psi},\lambda_{\mathrm{ME}}\right)$. The minimization of (3.21) is first conducted at IME stage after which the obtained IME result is refined at FME stage. FME results approximate actual RD costs more accurately, if SAD is replaced with SATD in (3.21).

The rate-constrained IME and FME are likewise conducted for each p_{η}^{ψ} belonging to the same candidate m^{ψ} (Figure 3.2). After all valid candidate modes are entirely processed, the inter mode decision selects the best one (m^{ψ^*}) of them. The decision is based on the computed mode costs (J^{ψ}) of which the smallest one (J^{ψ^*}) represents m^{ψ^*}.

3.4.5 Rate-constrained inter mode decision

In the RDO mode decision scheme [113], [133], the RDO costs of each candidate m^{ψ} are computed according to (3.19). At *MB-level* ($1\leq\psi\leq 4$), RDO costs are computed for entire MBs to obtain MB mode costs. For example, RDO cost of m^2 is composed of RDO costs of p_0^2 and p_1^2. At *sub-MB level* ($4\leq\psi\leq 7$), RDO costs are separately computed for each 8×8 pixel sub-MB. Hence, the lowest mode cost can be obtained with a single MB mode cost ($1\leq\psi\leq 4$) or the combination of four sub-MB mode costs ($4\leq\psi\leq 7$). RDO mode decision suggests SSD in computation of D due to which $\lambda_{\mathrm{MD}}=\lambda_{\mathrm{ME}}^2$.

Since ME has already selected the best matching blocks for each m^{ψ}, only a single RDO cost per m^{ψ} has to be computed for each MB ($1 \le \psi \le 4$) or sub-MB ($4 \le \psi \le 7$). However, the complexity of RDO mode decision tends to be still too high for real-time video encoders. Therefore, RDO is typically disabled and encoders implement a *low-complexity mode decision* [59], in which FME results are directly utilized without conducting the encoding loop at all.

In the low-complexity mode decision, MB mode cost ($J^{\psi}_{16\times16}$) of an entire m^{ψ} can be composed of the associated $J^{\psi}_{\eta*}$ values. When $1 \le \psi \le 4$, $J^{\psi}_{16\times16}$ is derived for m^{ψ} as

$$J^{\psi}_{16\times16} = \lambda_{\mathrm{ME}} \times R^{\psi}_{16\times16} + \sum_{\eta=0}^{n^{\psi}-1} J^{\psi}_{\eta*}\left(\mathrm{MV}^{\psi}_{\eta*}, \lambda_{\mathrm{ME}}\right), \tag{3.23}$$

where $J^{\psi}_{\eta*}\left(\mathrm{MV}^{\psi}_{\eta*}, \lambda_{\mathrm{ME}}\right)$ values are resolved by minimizing (3.21) at FME stage. $R^{\psi}_{16\times16}$ represents m^{ψ}-specific header bit consumption that can be optionally added in (3.23).

When $4 \le \psi \le 7$, sub-MB mode costs ($J^{\psi}_{8\times8}$) are separately determined for each sub-MB s as

$$J^{\psi}_{8\times8}(s) = \lambda_{\mathrm{ME}} \times R^{\psi}_{8\times8} + \sum_{\eta=s\times(n^{\psi}/4)}^{(s+1)\times(n^{\psi}/4)-1} J^{\psi}_{\eta*}\left(\mathrm{MV}^{\psi}_{\eta*}, \lambda_{\mathrm{ME}}\right), \tag{3.24}$$

where $s \in [0,3]$. As in the RDO mode decision scheme, the best inter mode can be one of the MB modes ($1 \le \psi \le 4$) or the combination of four sub-MB modes ($4 \le \psi \le 7$). As discussed in Section 3.3.1, MB modes are dominating ones in typical encoded video sequences and their impact increase further at higher resolutions.

4. Contemporary Motion Estimation Algorithms and Architectures

This chapter surveys the related work on ME algorithms and architectures. Among the hundreds of algorithm and architecture proposals in the literature, the emphasis here is on the most well-known ones. The considered ME algorithms are classified based on their speed-up techniques. The main focus is on lossy fast ME algorithms that complete block-matching without testing all available search positions. ME architectures are examined according to their target ME algorithm(s). Previous work on FBSME architectures is considered in [P3], so this chapter concentrates only on existing VBSME architectures for H.264/AVC.

4.1 Contemporary block matching algorithms

The well-known *full-search* (*FS*) is the simplest, but the most computation-intensive *block-matching algorithm* (*BMA*), which exhaustively tests all the *checking points* (*search positions*) in the search area (Figure 3.6). When searching for the best match for a $Q^{\psi} \times U^{\psi}$ pixel block with FS, the amount of checking points equals (3.15). For example, with the recommended search ranges for QCIF/CIF/D1 formats ($\mathrm{MV}i, \mathrm{MV}j \in [-32, 32]$) and 720p/1080p formats ($\mathrm{MV}i, \mathrm{MV}j \in [-64, 64]$) [115], FS consumes 4225 and 16641 checking points per search, respectively.

FS was originally introduced to FBSME, but it can also be extended to VBSME by repeating FS for each p_{η}^{ψ} (Figure 3.2). Implementing FS in VBSME requires approximately sevenfold computational complexity over FBSME, since the number of inter coding modes is increased from one (m^1) to seven (m^1, …, m^7). However, the VBSME overhead involved in traditional FS can be almost totally avoided with *fast FS* (*FFS*) that is specially tailored for VBSME. FFS computes SADs of all 4×4 blocks for m^7 and then derives SADs of the larger p_{η}^{ψ} by reusing and merging SADs of the 4×4 blocks. This *SAD reusing scheme* can be realized without search quality degradation over traditional FS. FFS is implemented, e.g., in JM reference encoder.

Although the complexity of FFS has been reduced close to traditional FS in FBSME, the relative complexity of FFS is still huge in H.264/AVC encoder. In [44], JM (BP) was profiled and MCP (IME, FME, and interpolation) was reported to consume 97% of the total encoding time. IME was executed with FFS having the search range of $\mathrm{MV}i, \mathrm{MV}j \in [-16.75, 16.75]$ and five reference frames. Despite the usage of FFS, the runtime percentage of IME was still 52%. The search range (the amount of tested checking points) applied in the profiling can be roughly assumed to equal $\text{MV}i, \text{MV}j \in [-36, 36]$ in the case of one reference frame. Hence, the complexity of IME would be about three times higher with the search range of $\text{MV}i, \text{MV}j \in [-64, 64]$.

To reduce the computation load of FS and FFS, numerous fast BMAs have been developed in the literature [42], [68]. The speed-up techniques utilized by these BMAs can be roughly classified into lossy and lossless ones. This Thesis provides only a concise overview of these techniques. More comprehensive surveys are presented, e.g., in [42], [68], and [121].

4.1.1 Lossless speed-up techniques

Lossless speed-up techniques enhance FS/FFS by eliminating unnecessary checking points as early as possible without search quality compromises. In general, a simple pre-criterion is used to test whether the checking point can be the optimum one and only the potential checking points are further processed through detailed distortion computation. The popular lossless speed-up techniques include *successive elimination* [30], [77] and *partial distortion elimination* (PDE).

Successive elimination algorithm (*SEA*) [77] is a two-level recursive BMA. At the first level, it computes the difference between the sum norms of the current and the candidate block pixels. I.e., the matching criterion is subsampled to compare a single sum norm pair instead of multiple individual pixel pairs. For example, the subsampling factor is 1:256 for a MB. The yielded (subsampled) result is compared with the current (non-subsampled) *minimum block distortion* (*MBD*). If the result is smaller than the current MBD, the candidate block cannot be skipped and a complete distortion is computed between it and the current block at the second level. SEA is reported to reduce computation over FS by 75 - 85% [42], [77]. At a cost of additional computational complexity, the elimination percentage of the first level can be improved by lowering the subsampling factor, e.g., to 1:64 or 1:16 for a MB [30]. MV prediction and a spiral scanning order of the checking points also increase elimination percentage, since they often find the final MBD earlier [42].

Multilevel SEA (*MSEA*) [30] extends SEA to multiple hierarchy levels. MSEA uses several subsampling factors in a coarse-to-fine manner so that the computation load increases as a function of the level. The subsampling factors associated to three successive levels can, e.g., be 1:256, 1:64, and 1:16, respectively. In MSEA, a complete distortion is computed only if a candidate block passes all prior levels. MSEA is reported to reduce computation over FS by 95% [42].

PDE is a halfway-stop technique developed to speed up distortion computation of a tested checking point. It can be used if a complete distortion between two blocks is composed of sequentially computed partial distortions. PDE monitors the accumulation of partial distortions and skips the tested checking point immediately, if the intermediate result exceeds the current MBD. As in SEA, MV prediction and the spiral scanning order of the checking points increase the effect of PDE.

4.1.2 Lossy speed-up techniques

Lossy speed-up techniques include *simplification of matching criterion* [66], [83], *reduction of the bit width* [1], *MV prediction* [140], [143], *hierarchical search* [93], and *reduction* (*decimation*) *of the checking points* [58].

Simplification of matching criterion is a speed-up technique that reduces the number of pixels involved in distortion computation. For example, the subsampling factor of the pixels can be 1:2 [66] or 1:4 [83] and the subsampling pattern can be regular or irregular. Typically, neighboring pixels have high correlation in homogeneous areas, in which subsampling can be done without search quality degradation. However, the spatial aliasing makes subsampling less accurate technique in highly textured areas. The subsampling can be also done hierarchically in coarse-to-fine manner.

Reducing the bit width of the pixels from the original eight bits simplifies pixel comparison and accumulation of partial distortions. The simplest way to reduce the bit width is truncation. According to [1], 8-bit pixels can be averagely truncated to four MSBs in FBSME without significantly reducing RD performance of H.261. In [11], FFS was evaluated with truncated 4-bit and 5-bit pixels. 4-bit pixels caused PSNR degradation of 0.1 - 0.2 dB and the PSNR gap was almost converged to zero with 5-bit pixels.

MV prediction can be divided into *spatial* [140] and *temporal* [143] prediction. Spatial prediction was already considered in Sections 3.3.3 and 3.4.2 in which robust median predictor was recommended for block matching. Existing BMAs also exploit other spatial MV predictors of which the most popular ones are a search center (i, j) and individual MVs composing the widely used median predictor [122]. Temporal MV prediction utilizes MVs of the previous frame(s). It is often limited to use the MV of the collocated block in the previous frame [122]. A single BMA may test all these spatial and temporal predictor candidates out of which it typically selects the best one as a search center. However, MV prediction can also return several MV candidates if multiple search centers are allowed [14], [122].

Hierarchical search uses a multiresolution (pyramid) structure in which the first level operates on the lowered resolution and the resolution is gradually increased to the original one at the subsequent levels. The multiresolution structure is constructed either with simple subsampling or filtering [93]. A hierarchical BMA proceeds through the levels in a coarse-to-fine manner so that the initial estimate of the MBD point is searched at the first level and the estimate is refined at the subsequent levels [93]. Typically, a lossy hierarchical BMA contains two or three hierarchy levels [42], [93] and each level defines the search center(s) for the next level. Entering the next level diminishes the search range, e.g., by half, in order to compensate the computational overhead of the finer search. A multiresolution BMA in [82] deviates from the traditional hierarchical approach since it parallelizes the hierarchy levels by making the levels independent on each other.

Reduction of the checking points is based on the assumption that the distortion monotonically decreases towards the global minimum [58]. Under this *unimodal error surface assumption*, the best match can be found without testing all the checking points in the search area. This speed-up technique is the most popular one. Therefore, it is considered next in more detail.

4.1.3 Reduction of checking points

Majority of fast BMAs are optimized to test only a fraction of the checking points in the search area. Since 1981 [58], hundreds of this kind of fast BMAs have been introduced. The well-known fast BMAs include *2D logarithmic search* [58], *three-step search* (*TSS*) [66], *cross search* (*CS*) [31], *four-step search* (4SS) [100], *block-based gradient descent search* (*BBGDS*) [84], *diamond search* (*DS*) [120], [145] *hexagon-based search* (*HEXBS*) [144], *cross-diamond search* (*CDS*) [19], *predictive MV field adaptive search technique* (*PMVFAST*) [122], *enhanced predictive zonal search* (*EPZS*) [122], *unsymmetrical-cross multi-hexagongrid search* (*UMHexagonS*) [18], and *Simplified UMHexagonS* [138]. Some of these fast BMAs are also adopted by JM (EPZS, UMHexagonS, and simplified UMHexagonS) and x264 (DS, HEXBS, and UMHexagonS).

In these kinds of fast BMAs, the location of the search center, the applied search patterns, and available early termination mechanisms are an essential part of the search strategy. Firstly, the conventional BMAs [19], [31], [58], [66], [84], [100], [120], [144], [145] start the search from the search area center (i, j), but more sophisticated BMAs [18], [122], [138] use MVP or the best one of the several MVP candidates as a search center. Secondly, the sequence of the applied search patterns determines the accessible checking points during the search. Each search pattern has a specific shape (rectangle, diamond, hexagonal, cross, etc.), size (4, 6, 8 points etc.), and type (single-pass or recursive). Thirdly, more sophisticated BMAs terminate the search if certain threshold value(s), MBD point position(s), or the maximum number of checking points are met.

Although the search strategies vary between these BMAs, they still mainly follow the generalized BMA flow below:

Step 1 The initial search pattern is positioned at the search center. The search center is tested and selected as the initial MBD point. If early termination occurs, the search proceeds to Step 3. Otherwise, the search proceeds to Step 2.

Step 2 All the valid checking points surrounding the pattern center are tested. A checking point is invalid, if it is out of the search area boundary or if it has already been tested by the previous pattern. The search proceeds to Step 3, if at least one of the following conditions holds: no new MBD point is found, a single-pass pattern is used, or early termination occurs. Otherwise, the recursive search pattern is re-positioned so that the new MBD point is at the center of the pattern and Step 2 is recursively repeated.

Step 3 If all the available search patterns have already been executed or early termination occurs, the current MBD point is selected as the final MBD point and the search is stopped. Otherwise, the search pattern is switched to a next search pattern which is positioned so that the MBD point found in Step 2 is in the center of it and the search proceeds to Step 2.

Most of the fast BMAs are originally introduced for FBSME, but they can be extended to VBSME by repeating the similar search for each p_{η}^{ψ}. Although this method was considered impractical for traditional FS, the sevenfold increase in computation is not such crucial issue with fast BMAs, since the total amount of checking points still remains moderately low. In addition, conducting a BMA individually for each p_η^ψ enables that motion is properly traced when motion directions of adjacent or nested p_η^ψ diverge. More sophisticated VBSME-compatible BMAs also use already processed p_η^ψ to adjust search centers and patterns of the unprocessed p_η^ψ [21]. Some of them reduce overall computation further by processing overlapped search paths of different p_η^ψ in parallel [110], even if 100% reuse of search paths cannot be reached as in FFS.

Most of these fast BMAs [19], [31], [58], [84], [120], [144], [145] are based on the assumption that the final MBD point is close to the search area center (i, j). In many cases, this assumption holds, since real-world sequences tend to have centrally biased MV distribution. Six popular CIF/SIF (352×240 pixels) sequences with various motion contents have been tested in [19]. According to those tests, approximately 45% of the final MBD points are at the search area center (stationary) and about 81% of the final MBD points are enclosed in a window of size ± 2 pixels around the search area center. However, in the cases when the final MBD point is far from the search area center and the unimodal error surface assumption is violated, these heuristic fast BMAs may drop into the local minimum point. Selecting a local minimum often reduces the search quality of the BMA. The probability to find a local minimum increases when the motion of the objects is high or irregular [18] and the sizes of the search patterns are small.

Since low, medium, and high motion contents are often mixed together in real video sequences, the search accuracy of the conventional fast BMAs can be improved by using MVP as their search center. In addition, the advanced RD cost criterion (3.21), early termination mechanisms, and other speed-up techniques considered in this chapter can be combined with these conventional BMAs. More sophisticated BMAs refine the search result further with multiple fixed search patterns [18], [138] or adaptively selected search patterns [21], [41], [122]. For example, UMHexagonS uses various search patterns, MVP candidates, and early termination mechanisms. It requires 90% less computation than FFS with average PSNR loss of 0.056 dB [18]. Simplified UMHexagonS enhances UMHexagonS further by reducing its computation by 55% (94% less than FFS) without compromising search quality [138].

The features of these sophisticated BMAs are best utilized in SW implementations, but they are not easily mapped to HW. Particularly, their highly irregular execution flows would require complex control circuits. Instead, the enhanced conventional BMAs can be better mapped to HW.

Hereafter, this Thesis uses TSS, BBGDS, DS, HEXBS, and CDS as examples of the fast BMAs, so they are here considered in detail. Figure 4.1 depicts example paths of these BMAs. Their search centers are set to (0, 0) and search ranges are limited to $\mathrm{MV}i, \mathrm{MV}j \in [-7, 7]$ for simplicity. All these examples conduct only FBSME but they can be respectively extended to VBSME. The initial search patterns are denoted by black circles and the final search patterns by white circles. The final MBD point ($\mathrm{MV}_{\eta*}^\psi$) is marked by a square. For the sake of uniformity, it is located at (5, -2) in each case.

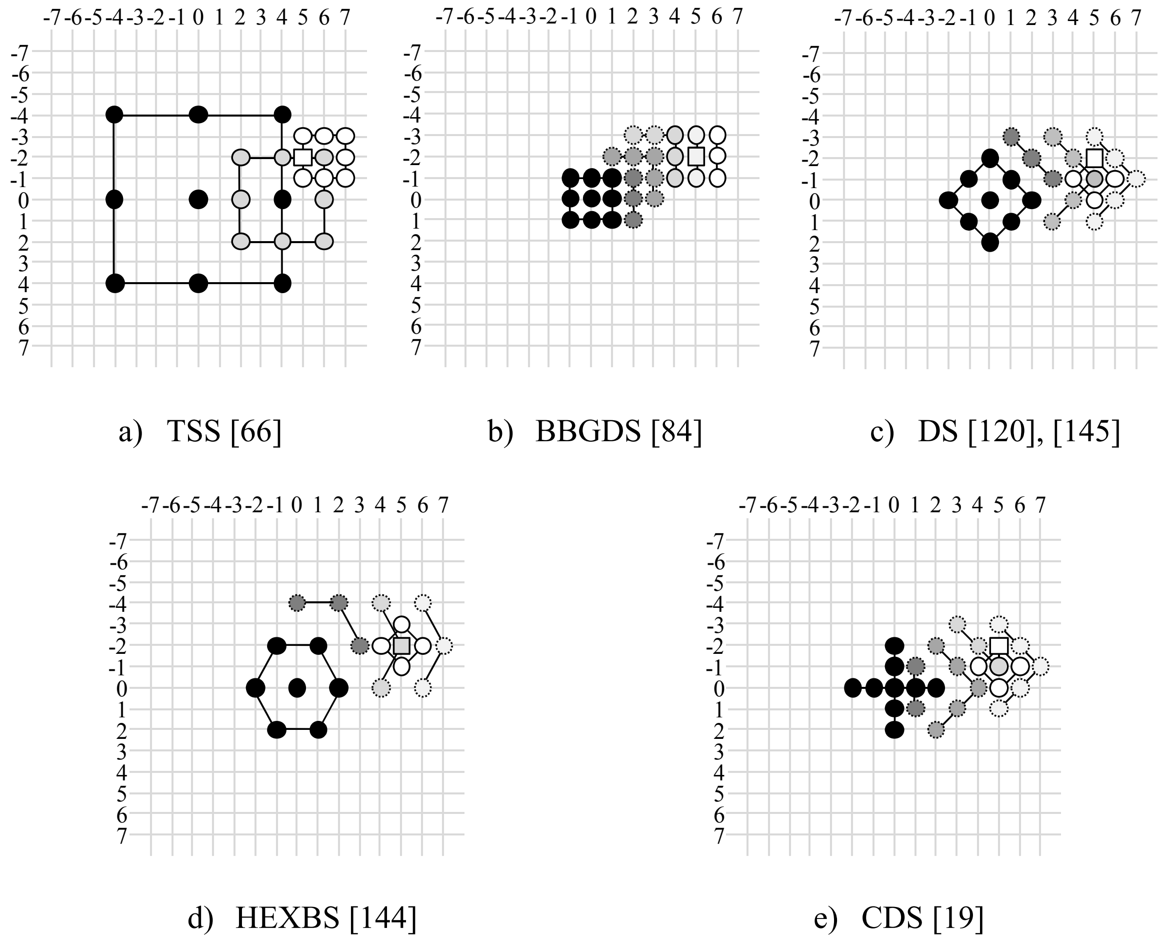


Figure 4.1. Example search paths of the well-known block-matching algorithms.

Figure 4.1 a) presents TSS [66] which uses single-pass search patterns and a coarse-to-fine search strategy with several search steps. The step size of the initial search pattern is approximately half of the search area (the closest power of two). Each search step halves the step size of the pattern and the search is continued until the step size converges to zero. The size of the search area determines the amount of the repeated steps, so TSS can contain more than three steps. In each search step, the resized pattern is positioned so that the MBD point found by the previous search step is in the center of it. The search covers the search center and eight additional checking points per search step.

Figure 4.1 b) visualizes BBGDS [84] which utilizes a square pattern of 3×3 checking points. The pattern moves recursively until MBD point is located at the center of it. MBD at the corner of the pattern causes the pattern to move diagonally and five new checking points are tested. If MBD occurs at the horizontal/vertical edge of the pattern (corners excluded), the pattern moves horizontally/vertically involving three new checking points. The search is limited by the search window boundaries. In Figure 4.1 b), $\mathrm{MV}_{\eta^*}^{\psi}$ is found after the pattern has moved right once, up-right twice, and right twice, respectively.

Figure 4.1 c) depicts DS [120], [145] which applies two diamond-shaped patterns: a *large diamond search pattern* (*LDSP*) with eight checking points and a *small diamond search* *pattern* (*SDSP*) with four checking points. The LDSP moves similarly than the pattern of BBGDS. MBDs at the corner and edge points of the LDSP involve five and three new checking points, respectively. The LDSP is switched to the SDSP when the MBD point is located at the center of the LDSP. The SDSP is a single-pass pattern that yields $\mathrm{MV}_{\eta*}^{\psi}$.

Figure 4.1 d) shows HEXBS [144] which replaces the LDSP of DS with the hexagon-shaped search pattern of six checking points. Each time the pattern is moved three new checking points are tested. HEXBS adopts the SDSP and the search strategy of DS.

Figure 4.1 e) illustrates CDS [19] which begins with a *cross-shaped search pattern* (*CSP*). If the MBD point occurs at the center of the CSP, the search stops. Otherwise, CDS tests two additional checking points among $(\pm1,\pm1)$ of which the selected ones are located around the CSP wing that contains the MBD point. For example, $(1,\pm1)$ are tested in Figure 4.1 e), since the MBD point is located at (2, 0). If the MBD point is found at $(0,\pm1)$, $(\pm1,0)$, or in the additional points, the search stops. Otherwise, five new checking points are tested as in DS when MBD occurs at the corner of the LDSP. Next, the search continues as DS.

4.2 Existing variable block-size motion estimation architectures

In the literature, numerous academic HW architectures have been proposed for VBSME, but the publicly available information of the commercial VBSME architectures is very limited. Therefore, this Thesis surveys only academic HW architectures.

The academic HW architectures can be classified as *BMA-specific* and *flexible architectures*. The former can be further divided into *FFS-based architectures* and *fast BMA -based architectures*.

4.2.1 FFS -based architectures

The majority of contemporary BMA-specific VBSME architectures implement FFS due to its regular data flow and low control overhead. FFS is typically implemented with a 1D or 2D systolic mesh-connected array which provides high throughput through parallel processing, pipelining, and data reuse. Since FFS enables parallel SAD computation for $m^1,\ldots, m^7$ by reusing and merging SADs of the 4×4 blocks, the only impact of extending traditional FBSME architectures to support VBSME is an additional SAD merging logic. All the considered FFS-based architectures [11], [65], [76], [130], [137] output 41 MVs and cost values per MB without mode decision. Only [11] implements rate-constrained IME, i.e., it searches for $\mathrm{MV}_{\eta*}^{\psi}$ according to RD cost criterion, whereas the other architectures rely on simpler SAD criterion.

Yap and McCanny [137] proposed a 1D systolic FFS-based architecture for VBSME. The architecture reuses SADs of 4×4 blocks by incorporating shuffling and combining mechanisms within each processing element (PE).

Chen *et al*. [11] introduced a FFS-based architecture that is composed of eight parallel SAD trees. Each SAD tree consists of a 2D PE array and a 2D adder tree. The HW cost and memory bandwidth of the architecture are reduced by simplifying the matching criterion with pixel truncation and pixel subsampling [66] techniques. The architecture can support up to four reference frames. The encoder presented by Huang *et al.* (Section 2.4.5) contains this IME architecture.

Li *et al*. [76] and Wei *et al*. [130] presented FFS-based architectures that utilize flexible 2D PE arrays to enable efficient data reuse. The architectures support one and two reference frames, respectively. They both achieve 100% PE utilization with low memory bandwidth requirement.

Kim and Park [65] developed a FFS-based 1D architecture that is scalable to different search area sizes and PE configurations. The HW complexity of the implementation is reduced by accessing the 4×4 blocks with a new scan order that enhances SAD reusing during computation.

Despite many optimizations introduced in the considered FFS-based architectures, inherent complexity of FFS either increases HW cost [11], [76], [130] or limits performance [65], [137]. Therefore, architectures with fast BMAs have been presented.

4.2.2 Fast BMA -based architectures

The fast BMA -based architectures [14], [82], [87] are tailored to execute a single fast BMA. They all utilize SAD reusing when computing SADs of the larger p_η^ψ. However, none of them performs rate-constrained IME.

Liu *et al*. [87] designed a VBSME architecture that supports only $m^1,\ldots, m^4$ and one reference frame, since experiments in [87] show that $m^5,\ldots, m^7$ and multiple reference frames do not provide any significant coding gain in HDTV resolution. Computational complexity of the architecture is also reduced with pixel subsampling and coarse-to-fine BMA that first predicts a coarse MV and then refines it with a fine search.

Lin *et al*. [82] presented an architecture for a multiresolution BMA. The first level of BMA performs MVP-centered local FS for each p_η^ψ to cover the most likely locations of the best matching blocks. The second and third BMA levels perform coarser searches with wider search ranges to trace high motion. However, search area subsampling restricts their usage to $m^1,\ldots, m^4$ and m^1, respectively. The BMA is mapped to the implementation that is able to process all these three independent resolution levels in parallel. The architecture also performs initial mode decision by sending only two best modes (one among $m^1,\ldots, m^3$ and the other among $m^1,\ldots, m^7$) to the subsequent FME stage.

Chen *et al*. [14] developed a HW-oriented fast BMA called content-adaptive parallel-VBS 4SS. It realizes a modified 4SS that is executed multiple times with different search centers. The amount of the search centers is adjusted according to the motion activity of neighboring blocks. Search path decisions of 4SS are made according to costs of m^1. An architecture for the BMA is composed of a systolic array and a 2D adder tree. The implementation can operate in three modes. In high-quality and low-power modes, BMA with multiple search centers is performed with two and one reference frames, respectively. In an ultralow-power mode, the BMA has only one search center and one reference frame.

These three IME architectures have been integrated in the encoders presented by Liu *et al.* (Table 2.2), Lin *et al.* (Table 2.3), and Chen *et al.* (Section 2.4.5), respectively. Although these implementations meet the requirements of the specific application with smaller cost [82], [87] or power consumption [14] than the FFS architectures, they are too rigid for a broad range of applications [72], [126], [131], [142]. To increase flexibility, architectures supporting multiple BMAs have been introduced.

4.2.3 Flexible architectures

The flexible architectures [72], [126], [131], [142] have a wide application range. They have been designed to accommodate different restrictions on image quality, timing constraints, and power consumption. None of them supports mode decision and only [142] implements rate-constrained IME.

Lee *et al.* [72] proposed a HW-oriented BMA with a related architecture. The BMA utilizes several search centers in which local searches are iteratively started. The local searches can be performed with FFS or fast BMAs. In addition, the number and positions of the search centers are adjustable. The architecture is composed of five parallel computation units. Each unit processes a separate checking point of the BMA. The architecture supports two reference frames.

Verma and Akoglu [126] applied a network-on-chip (NoC) -based approach for VBSME. Their reconfigurable architecture utilizes NoC routers that manage communication between application-specific PEs. The architecture can be configured for FFS and several fast BMAs such as DS and HEXBS.

Wei *et al.* [131] developed a reconfigurable architecture that can operate at three complexity levels (high, medium, and low) to trade off between image quality and power consumption. Besides FFS, the architecture is compatible with many fast BMAs. FFS is processed at the high level, pixel subsampling algorithms are executed in the medium level, and BMAs utilizing reduction of checking points (e.g., TSS and DS) are available in the low complexity level. At the high level, all the PEs participate in computation, whereas only half and quarter of the PEs can be exploited in the medium and low levels, respectively.

Zhang and Gao [142] introduced a combined architecture for IME and FME. It supports DS and CS during IME. The architecture is able to allocate execution cycles between IME and FME in order to find the best trade-off between coding efficiency and operating cycles. The implementation achieves the relatively best search result by executing DS with $m^1,\ldots, m^4$ at first, then refining the search result of m^1 with CS, after which FME is performed for $m^1,\ldots, m^4$. The architecture is able to process nine checking points of m^1 in parallel. However, there are unused PEs if under nine checking points are simultaneously tested or $m^2,\ldots, m^7$ are computed. The architecture does not utilize SAD reusing as the others [72], [126], [131], but each J_{η}^{ψ} is computed individually.

4.2.4 Summary of architectures

Table 4.1 gathers the essential characteristics of the considered ME architectures. The search ranges of the architectures are reported according to (3.15).

Table 4.1. Characteristics of contemporary motion estimation architectures.

Architecture	Year	Supported BMAs	# of PEs	Supported Modes	# of Ref. Frames	Search Range	SAD Reuse	Distortion Function	Mode Decision
Yap [137]	2004	FFS	16	1-7	1	16×16	Yes	SAD	No
Chen [11]	2006	FFS	2048	1-7	4	$129\times65/65\times33$	Yes	RD cost	No
Li [76]	2007	FFS	256	1-7	2	65×33	Yes	SAD	No
Wei [130]	2008	FFS	256	1-7	1	33×33	Yes	SAD	No
Kim [65]	2009	FFS	16	1-7	1	16×16	Yes	SAD	No
Liu [86] [87]	2007	Coarse-to-fine	2048	1-4	1	192×128	Yes	SAD	No
Lin [82]	2008	Multiresolution	192	1-7	1	$256\times256,...,16\times16$	Yes	SAD	2 best modes
Chen [14]	2007	Parallel-VBS 4SS	256	1-7	2	64×32	Yes	SAD	No
Lee [72]	2008	FFS/Fast BMAs	1280	1-7	2	4096×128	Yes	SAD	No
Verma [126]	2008	FFS/Fast BMAs	21	1-7	1	16×16	Yes	SAD	No
Wei [131]	2007	FFS/Fast BMAs	64	1-7	1	32×32	Yes	SAD	No
Zhang [142]	2007	DS + CS	144	1-4	1	$75\times39,...,13\times13$	No	RD cost	No

5. Designed Configurable Motion Estimation Architecture

This chapter presents the proposed ME framework and an overview of the designed configurable ME architecture. A part of the *ME framework* is originally introduced in [P3]. In addition, the *distortion computation unit*, the *memory system*, and the *control unit* of the architecture are initially proposed and thoroughly considered in [P1], [P2], and [P3], respectively. The content available in [P1]-[P3] is only summarized here and the emphasis is on the novel features added to the architecture after publishing [P1]-[P3].

5.1 Proposed motion estimation framework

The proposed ME framework contains four alternative *quality levels* ($L_0,\dots, L_3$). The chosen level can be changed on MB basis. The levels are specified as:

- At L_0, an executed BMA only processes m^1, so the operation equals FBSME. The utilized RD cost criterion can be replaced with SAD criterion by setting $\lambda_{\mathrm{ME}} = 0$ in (3.21). The framework in [P3] supports only this level with $\lambda_{\mathrm{ME}} = 0$.
- At L_1, m^1 and m^4 are serially tested and the best one ($m^{\psi*}$) is selected. Mode costs ($J^1_{16\times16}$ and $J^4_{16\times16}$) are computed from associated $J^1_{\eta*}$ and $J^4_{\eta*}$ values that are resolved one by one for m^1 and m^4.
- At L_2, $m^1,\dots, m^4$ are serially tested after which $m^{\psi*}$ is selected. $J^{\psi}_{16\times16}$ values are computed from associated $J^{\psi}_{\eta*}$ values as at L_1.
- At L_3, an executed BMA first tests $m^1,\dots, m^4$ like at L_2. If $m^{\psi*}$ is found among $m^1,\dots, m^3$, the operation is terminated. Otherwise, the BMA continues at sub-MB level with four sequentially executed branches, each of which processing one sub-MB. Each branch serially tests $m^5,\dots, m^7$ and selects $m^{\psi*}$ among $m^4,\dots, m^7$.

At $L_0,\dots, L_3$, the level-specific set of modes undergo three phases: *rate-constrained ME*, *inter mode decision*, and *the best inter mode delivery*.

5.1.1 Rate-constrained motion estimation

In the proposed ME framework, rate-constrained ME and associated search path generation are based on (3.21). ME is individually conducted for each p^{ψ}_{η} (Figure 3.2) in order to trace motion better when motion directions of adjacent or nested p^{ψ}_{η} diverge. The framework determines the set of available BMAs and provides a common search center for all p^{ψ}_{η} of the same MB. Arbitrary search center can be used, but choosing the search center according to (3.16) is recommended. The search center and the BMA (among the available BMAs) can be selected on MB basis.

The proposed approach operates on blocks of 4×4 pixels, since a 4×4 block is the greatest common divisor of supported p_η^ψ sizes. Let a size of a search area be $w \times h$ pixels, where $w, h \in \{48, 80, 112, 144, ...\}$ represent *i*- and *j*-directions, respectively. A 4×4 block located in (*i*, *j*) is addressed by a scanning point $\left(r_{RR}(ri_{RR}, rj_{RR})\right)$, where $ri_{RR} \in [0, w-1]$ and $rj_{RR} \in [0, h-1]$.

As presented in [P3], a search path of a BMA can be individually generated for each p_η^ψ by composing r_{RR} from five mutually independent offsets. Figure 5.1 presents the search path generation when $w = h = 48$.

An initial offset $\left(\Delta r_\alpha(\Delta ri_\alpha, \Delta rj_\alpha)\right)$ points to the center of the search area and *a prediction-based offset* $\left(\Delta r_\beta(\Delta ri_\beta, \Delta rj_\beta)\right)$ equals MVP that increments Δr_α. The displacement between $\Delta r_\alpha + \Delta r_\beta$ and a center of the moving search pattern is indicated by *a BMA movement offset* $\left(\Delta r_\chi(\Delta ri_\chi, \Delta rj_\chi)\right)$. An adjustable *checking point offset* $\left(\Delta r_\delta(\Delta ri_\delta, \Delta rj_\delta)\right)$ determines a displacement of a tested checking point from the pattern center. BMA-specific address composition is presented in detail in [P3].

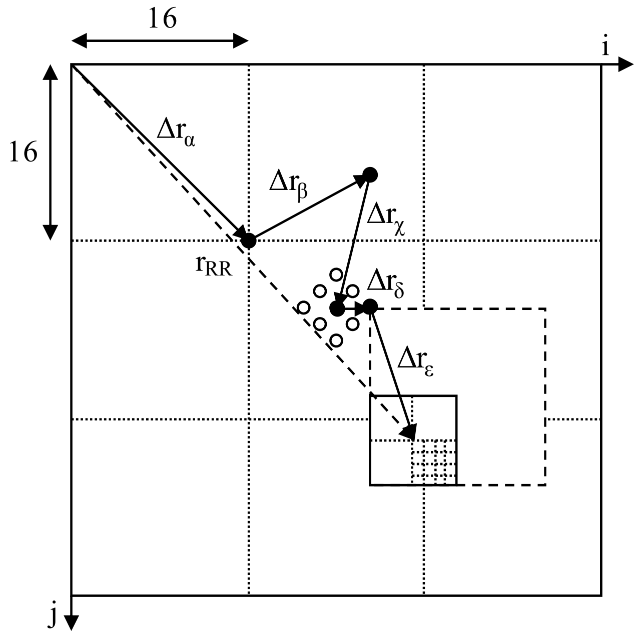


Figure 5.1. Proposed separable composition of a block address in the search area.

A base block offset $\left(\Delta r_{\varepsilon}(\Delta ri_{\varepsilon},\Delta rj_{\varepsilon})\right)$ is responsible for p_{η}^{ψ}-specific offsets. Δr_{ε} represents a displacement of a 4×4 block from Δr_{δ} that addresses a top left corner of the MB to which a tested candidate p_{η}^{ψ} belongs. Each candidate p_{η}^{ψ} is accessible through $(Q^{\psi}\times U^{\psi})/16$ accesses when $\Delta ri_{\varepsilon}\in\{0,4,8,12\}$ and $\Delta rj_{\varepsilon}\in\{0,4,8,12\}$. In Figure 5.1, the smallest grid illustrates the accessed 4×4 block that is one of the four 4×4 blocks of candidate p_{2}^{4} (marked with a solid line).

To minimize temporal data storages, 4×4 blocks belonging to the same p_{η}^{ψ} have to be sequentially accessible. Therefore, special scan sequences of 4×4 blocks are introduced for m^1,…, m^7. Since every p_{η}^{ψ} of each m^{ψ} is processed in the numerical order shown in Figure 3.2, the available scan sequences for m^7 are restricted to one. This scan sequence is depicted in Figure 5.2 a), where Δr_{ε} of each 4×4 block is logically mapped to the 4-bit index $\sigma\,(\sigma_3\sigma_2\sigma_1\sigma_0)$.

The similar sequence is also utilized by [65], except that it accesses pixels of 4×4 blocks row by row. Besides m^7, the same order can be adapted to m^1, m^2, m^4, and m^5. Instead, m^3 and m^6 require special scan sequences described in Figure 5.2 b) and Figure 5.2 c), respectively. However, these two sequences can be converted to that of Figure 5.2 a) by swapping two MSB bits of $\sigma\,(\sigma_2\sigma_3\sigma_1\sigma_0)$ in Figure 5.2 b) and two LSB bits of $\sigma\,(\sigma_3\sigma_2\sigma_0\sigma_1)$ in Figure 5.2 c). Hence, only one LUT with index swapping logic is sufficient for Δr_{ε} generation. The same LUT can also be used to address current MB data.

σ	**Δriε**	**Δrjε**
0	0	0
1	4	0
2	0	4
3	4	4
4	8	0
5	12	0
6	8	4
7	12	4
8	0	8
9	4	8
10	0	12
11	4	12
12	8	8
13	12	8
14	8	12
15	12	12

a) Modes 1, 2, 4, 5, and 7

σ	**Δriε**	**Δrjε**
0	0	0
1	4	0
2	0	4
3	4	4
4	0	8
5	4	8
6	0	12
7	4	12
8	8	0
9	12	0
10	8	4
11	12	4
12	8	8
13	12	8
14	8	12
15	12	12

b) Mode 3

σ	**Δriε**	**Δrjε**
0	0	0
1	0	4
2	4	0
3	4	4
4	8	0
5	8	4
6	12	0
7	12	4
8	0	8
9	0	12
10	4	8
11	4	12
12	8	8
13	8	12
14	12	8
15	12	12

c) Mode 6

Figure 5.2. Scan sequences for the candidate and current MBs.

When the search for p_η^ψ is started, σ is initialized to $\sigma_\eta^\psi = (\eta \times Q^\psi \times U^\psi)/16$ that indexes the address (Δr_ε) of the top-left 4×4 block of p_η^ψ. All 4×4 blocks of p_η^ψ are indexed by incrementing σ by one until $(Q^\psi \times U^\psi)/16$ blocks have been accessed. Then, Δr_δ is updated and σ reinitialized to σ_η^ψ. For example, $\sigma_2^4 = (2 \times 8 \times 8)/16 = 8$ and σ iteratively has the values from 8 to 11 when searching the best match for p_2^4 as in Figure 5.1.

In a horizontally circular search area, $r_{RR}(ri_{RR}, rj_{RR})$ is yielded as

$$ri_{RR} = \left(\Delta ri_\alpha + \Delta ri_\beta + \Delta ri_\chi + \Delta ri_\delta + \Delta ri_\varepsilon\right) \bmod w \tag{5.1}$$

$$rj_{RR} = \Delta rj_\alpha + \Delta rj_\beta + \Delta rj_\chi + \Delta rj_\delta + \Delta rj_\varepsilon. \tag{5.2}$$

The circularity is realized by mod w in (5.1). A horizontal search area reuse can be supported by restricting Δri_α to be a multiple of 16 ($\Delta ri_\alpha \in [0, w-1]$) and incrementing it by 16 after each search. Vertically, the center of the search area is fixed, i.e., $\Delta rj_\alpha = (h-16)/2$ is constant in (5.2).

As a modification to [P3], MV_η^ψ related to r_{RR} is computed without Δr_α and Δr_ε as

$$\mathrm{MV}i_\eta^\psi = \Delta ri_\beta + \Delta ri_\chi + \Delta ri_\delta \tag{5.3}$$

$$\mathrm{MV}j_\eta^\psi = \Delta rj_\beta + \Delta rj_\chi + \Delta rj_\delta. \tag{5.4}$$

Excluding Δr_ε from (5.3) and (5.4) makes MV_η^ψ determination independent of p_η^ψ size.

For simplicity, the search range of each MV_η^ψ is restricted to that of MV_0^1. In addition, the search ranges are reduced by three pixels in each direction. The excluded outermost pixels are reserved for interpolation before FME (Section 3.3.2). Hence, Δr_β, Δr_χ, Δr_δ, and MV_η^ψ are limited in (5.1)-(5.4) as

$$\Delta ri_\beta, \Delta ri_\chi, \Delta ri_\delta, \mathrm{MV}i_\eta^\psi \in \left[-p_w + 3, p_w - 3\right] = \left[-(w-16)/2 + 3, (w-16)/2 - 3\right] \tag{5.5}$$

$$\Delta rj_\beta, \Delta rj_\chi, \Delta rj_\delta, \mathrm{MV}j_\eta^\psi \in \left[-p_h + 3, p_h - 3\right] = \left[-(h-16)/2 + 3, (h-16)/2 - 3\right]. \tag{5.6}$$

Modifying (3.15) according to (5.5) and (5.6) yields

$$\left(2(p_w - 3) + 1\right) \times \left(2(p_h - 3) + 1\right) = (w - 21) \times (h - 21). \tag{5.7}$$

I.e., the (i, j)-centered search range of MV_η^ψ is $(w-21) \times (h-21)$ pixels. Otherwise, MV_η^ψ exceeds the boundaries of the search range.

5.1.2 Inter mode decision

In the proposed framework, inter mode decision needed at $L_1,\ldots, L_3$ is jointly performed with rate-constrained ME. Although a selection of the best mode after IME stage may somewhat reduce the search quality [15], it simplifies the FME stage considerably especially at L_3.

At MB-level, mode decision is based on MB mode costs ($J^{\psi}_{16\times16}$) which are used to select $m^{\psi*}$ among m^1 and m^4 (at L_1) or among $m^1,\ldots, m^4$ (at L_2 and L_3). When $1 \le \psi \le 4$, $J^{\psi}_{16\times16}$ is derived for m^{ψ} as in (3.23). The header bits of each m^{ψ} are assumed to be encoded with UVLC scheme. Hence, the amount of header bits ($R^{\psi}_{16\times16}$) included in (3.23) is obtained from the LUT similarly than R_{MV} values. As in [123], the lengths of Exp-Golomb coded $R^{\psi}_{16\times16}$ values assigned for m^1, m^2, m^3, and m^4 are 1, 3, 3, and 9, respectively.

With L_3, the mode decision is continued at sub-MB level, if m^4 is selected as $m^{\psi*}$ at MB level. In that case, sub-MB mode costs ($J^{\psi}_{8\times8}$) are computed according to (3.24) and $J^{\psi}_{8\times8}$ values are used to select $m^{\psi*}$ among $m^4,\ldots, m^7$. The respective header bit counts ($R^{\psi}_{8\times8}$) for m^4, m^5, m^6, and m^7 are 1, 3, 3, and 5.

Since $R^4_{8\times8} = 1$ is added to each $J^4_{8\times8}$ in (3.24), the respective cost (4×1) is already recognized in (3.23) by assigning $R^4_{16x16} = 5 + (4\times1) = 9$ for m^4. By that way, m^4 is not favored in mode decision at MB level. The incremented R^4_{16x16} does not affect on (3.24), so $R^4_{8\times8} = 1$ can be added to each $J^4_{8\times8}$.

5.1.3 Inter mode delivery

After mode decision, partitions ($p^{\psi*}_{\eta*}$) of $m^{\psi*}$ are returned one by one. Retrieval of each $p^{\psi*}_{\eta*}$ is composed of associated MV ($\mathrm{MV}^{\psi*}_{\eta*}$), RD cost ($J^{\psi*}_{\eta*}$), and pixels of $p^{\psi*}_{\eta*}$. In the modern video encoders, the subsequent stage after the IME stage is FME (Section 3.3.2), so the interface of IME is accommodated to that of FME.

It is assumed that the FME stage applies a six-tap FIR filter first horizontally as in (3.10) and then vertically as in (3.11) to obtain reference pixels at ½-pixel accuracy. Typically, FME tests eight ½ -pixel points around $\mathrm{MV}^{\psi*}_{\eta*}$, so a search area has to be [-½, ½], both horizontally and vertically. Interpolating search area pixels for $Q^{\psi*} \times U^{\psi*}$ block with the six-tap filter requires that $(3 + Q^{\psi*} + 3)\times(3 + U^{\psi*} + 3)$ pixels per $p^{\psi*}_{\eta*}$ are fetched to the FME stage (Figure 3.3).

The 4×4 block access format would require several data accesses per row before the requested pixels are retrieved for horizontal interpolation. Therefore, a row format of 16×1 pixels is used instead of it. For $p_{\eta*}^{\psi*}$ of size $Q^{\psi*}\times U^{\psi*}$, the 16×1 row format needs

$$\left\lceil(3+Q^{\psi*}+3)/16\right\rceil\times(3+U^{\psi*}+3)=\left\lceil(Q^{\psi*}+6)/16\right\rceil\times(U^{\psi*}+6) \qquad (5.8)$$

accesses to retrieve pixels to FME. In (5.8), $Q^{\psi*}\in\{4,8,16\}$ limits that $\left\lceil(Q^{\psi*}+6)/16\right\rceil\in[1,2]$. I.e., the pixels can always be accessed with one $(Q^{\psi*}=4,8)$ or two $(Q^{\psi*}=16)$ 16-pixel wide data strips of height $(U^{\psi*}+6)$ pixels.

The row format requires vertically adjacent scanning points from $(\mathrm{MV}i_{\eta*}^{\psi*}-3,\mathrm{MV}j_{\eta*}^{\psi*}-3)$ to $(\mathrm{MV}i_{\eta*}^{\psi*}-3,\mathrm{MV}j_{\eta*}^{\psi*}+U^{\psi*}+3-1)$ in order to access the first strip. With the second strip, the scanning points are located between $(\mathrm{MV}i_{\eta*}^{\psi*}+8-3,\mathrm{MV}j_{\eta*}^{\psi*}-3)$ and $(\mathrm{MV}i_{\eta*}^{\psi*}+8-3,\mathrm{MV}j_{\eta*}^{\psi*}+U^{\psi*}+3-1)$. I.e., a second data strip has a horizontal offset of eight pixels compared with the first strip.

Accessing the first and the second strips of a 16×16 block is illustrated in Figure 5.3 a) and Figure 5.3 b), respectively. In both cases, the pixels inside a dashed rectangle are needed in interpolation and the grey pixels represent 8×16 partitions of the original block.

When pixels of $p_{\eta*}^{\psi*}$ are delivered, a scanning point in the search area (r_{RR}^{dlvr}) is composed as

$$ri_{RR}^{\mathrm{dlvr}}=\left(\Delta ri_{\alpha}+\mathrm{MV}i_{\eta*}^{\psi*}+\Delta ri_{\varepsilon}^{\mathrm{dlvr}}\right)\mathrm{mod}\,w \qquad (5.9)$$

$$rj_{RR}^{\mathrm{dlvr}}=\Delta rj_{\alpha}+\mathrm{MV}j_{\eta*}^{\psi*}+\Delta rj_{\varepsilon}^{\mathrm{dlvr}}, \qquad (5.10)$$

where $\Delta r_{\varepsilon}^{\mathrm{dlvr}}(\Delta ri_{\varepsilon}^{\mathrm{dlvr}},\Delta rj_{\varepsilon}^{\mathrm{dlvr}})$ represents a displacement of the 16×1 row format from $(\mathrm{MV}i_{\eta*}^{\psi*},\mathrm{MV}j_{\eta*}^{\psi*})$ which is derived according to (5.3) and (5.4). When σ is set to $\sigma_{\eta*}^{\psi*}=(\eta*\times Q^{\psi*}\times U^{\psi*})/16$ (Figure 5.2), $\Delta ri_{\varepsilon}^{\mathrm{dlvr}}$ is yielded as a function of Δri_{ε} as $\Delta ri_{\varepsilon}^{\mathrm{dlvr}}=\Delta ri_{\varepsilon}-3$ for the first strip and $\Delta ri_{\varepsilon}^{\mathrm{dlvr}}=\Delta ri_{\varepsilon}+8-3$ for the second strip. During data delivery, $\Delta rj_{\varepsilon}^{\mathrm{dlvr}}\in\left[-3,U^{\psi*}+3-1\right]$ is incremented one by one from -3 to $U^{\psi*}+3-1$.

The proposed result retrieval has a small overhead on the whole IME execution time. However, it enables that the FME stage does not need to operate on the whole search area data but it can store and process only a single $p_{\eta*}^{\psi*}$ at a time.

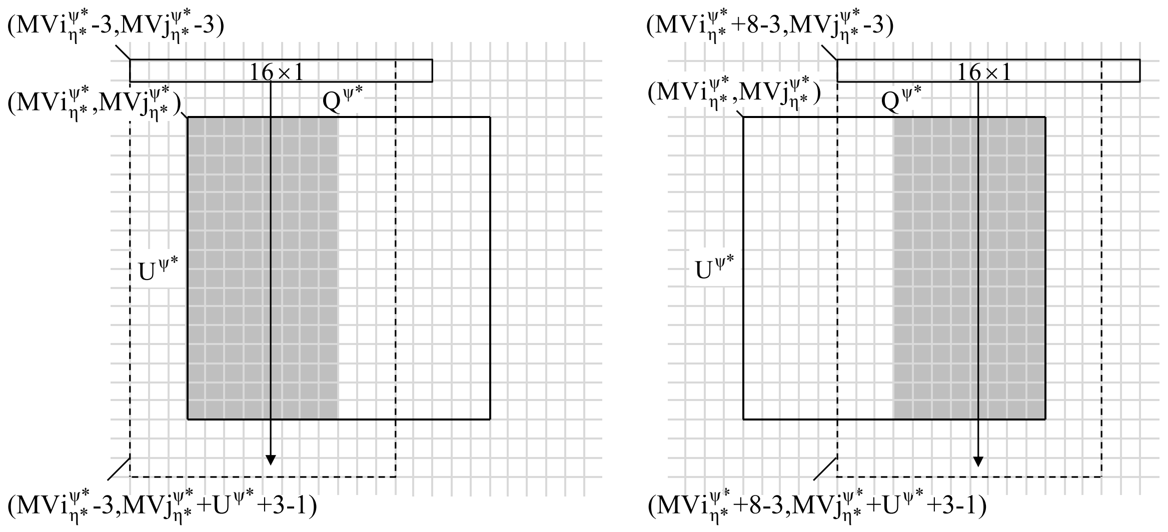

(MVi$_{\eta*}^{\psi*}$-3,MVj$_{\eta*}^{\psi*}$-3)
16×1
(MVi$_{\eta*}^{\psi*}$,MVj$_{\eta*}^{\psi*}$)
Q$^{\psi*}$
U$^{\psi*}$
(MVi$_{\eta*}^{\psi*}$-3,MVj$_{\eta*}^{\psi*}$+U$^{\psi*}$+3-1)
(MVi$_{\eta*}^{\psi*}$+8-3,MVj$_{\eta*}^{\psi*}$-3)
16×1
(MVi$_{\eta*}^{\psi*}$,MVj$_{\eta*}^{\psi*}$)
Q$^{\psi*}$
U$^{\psi*}$
(MVi$_{\eta*}^{\psi*}$+8-3,MVj$_{\eta*}^{\psi*}$+U$^{\psi*}$+3-1)
a) The first strip
b) The second strip

Figure 5.3. Retrieving pixels of a 16 × 16 block to FME.

The control of the proposed ME framework is mapped to the control unit of the ME architecture. The control unit is seamlessly coupled to a parallel memory system and RD cost unit which are customized for p_{η}^{ψ}-specific data storage and RD cost computation, respectively. The overall ME architecture is considered in the following.

5.2 Proposed motion estimation implementation

The proposed ME architecture can be integrated into the H.261/3, MPEG-1/2, MPEG-4 Visual, H.264/AVC, and VC-1 encoders. The most flexible approach is to realize the ME architecture as an intellectual property block that is connected to other encoder components via an on-chip communication network.

Figure 5.4 presents an exemplary system architecture in which the on-chip communication network is used to connect the proposed ME architecture to other system components (CPU, frame memory, and other HW accelerators). The other function-specific accelerations may include HW modules for FME, IP, DCT, Q, or EC, etc. The ME architecture is best suited for a communication network that is composed of 128-bit wide data and 9-bit wide command buses. Depending on the system requirements, a network topology can vary from a shared bus to highly scalable network such as HIBI [105].

The network-specific wrappers are used to integrate the system components to the network. The ME architecture requires a wrapper that reconciles its unidirectional data buses ($data_{IN}$, $data_{OUT}$) and control/status signals ($ctrl_{IN}$, $ctrl_{OUT}$) to the bidirectional data and command buses of the system, respectively. Besides integration, the wrapper of the ME architecture implements protocols to retrieve reference and current frame data directly from the frame memory. It also forwards the ME result to the CPU and other components.

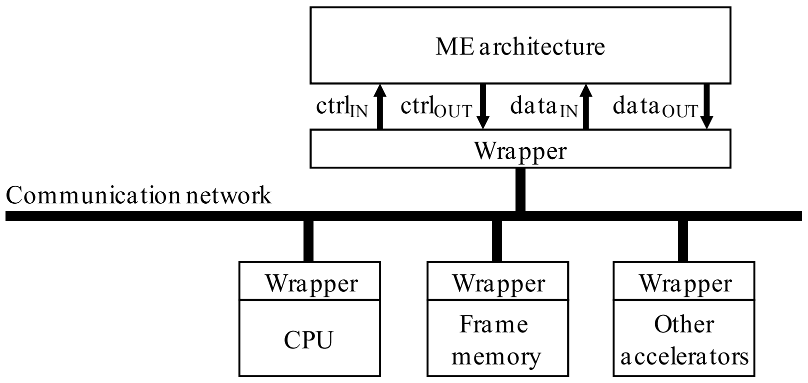


Figure 5.4. An exemplary system architecture for a video encoder.

Figure 5.5 depicts the designed ME architecture. The main components of it are a *control unit*, a *memory system*, and an *RD cost unit*. The quality level (L_0 - L_3) and the BMA can be selected at run time with 2-bit *level* and *BMA_id* inputs, respectively. The *data$_{IN}$* bus is time multiplexed between reference frame data (d_{RI}), current frame data (d_{CI}), and two input parameters: λ_{ME} and MVP_0^1, which are recommended to be computed as in (3.22) and (3.16), respectively. The control unit asserts the one-hot coded 3-bit *new* signal to request new d_{RI}, d_{CI}, and $\lambda_{\mathrm{ME}}\,\&\,\mathrm{MVP}_0^1$. It detects valid input data (d_{RI}, d_{CI}, or $\lambda_{\mathrm{ME}}\,\&\,\mathrm{MVP}_0^1$) by monitoring the 3-bit *vld* signal. The *reuse* signal determines whether to reuse part of the previous search area. For valid d_{RI} or d_{CI}, the control unit generates scanning points (r_{RW} or r_{CW}) and respective storage control signals ($ctrl_M$) for the memory system. The architecture has a separate two-stage pipeline for data storage. After d_{RI}, d_{CI}, and $\lambda_{\mathrm{ME}}\,\&\,\mathrm{MVP}_0^1$ have been stored, the control unit asserts all bits in the 3-bit *stored* signal.

For an executed BMA, the control unit composes BMA-specific scanning points (r_{RR} and r_{CR}) to access search area data (d_{RO}) and current block data (d_{CO}) from the memory system. The control unit controls data retrieval ($ctrl_M$) and RD cost computation ($ctrl_J$). It also delivers $\lambda_{\mathrm{ME}}\,\&\,\mathrm{MVP}_0^1$ and MV_η^ψ to the RD cost unit which computes J_η^ψ, $J_{16\times16}^\psi$, and $J_{8\times8}^\psi$ according to (3.21), (3.23), and (3.24), respectively. The RD cost unit informs the control unit of J_η^ψ and J^ψ ($J_{16\times16}^\psi$ and $J_{8\times8}^\psi$) completions with J_rdy_0 and J_rdy_1 signals, respectively. In addition, it uses J_min_0 and J_min_1 signals to identify $J_{\eta*}^{\psi*}$ and $J^{\psi*}$. The control unit and the memory system contain three pipeline stages. The RD cost unit applies three stages to compute a single J_η^ψ and one additional stage for J^ψ computation.

After $m^{\psi*}$ has been found, the control unit requests permission for data delivery with the *rdy* signal. The permission is acknowledged with the *ack* signal after which the RD cost unit outputs the *info* data. It also sends $\mathrm{MV}_{\eta*}^{\psi*}$ to the control unit that uses $\mathrm{MV}_{\eta*}^{\psi*}$ to access the respective best matching block (d_{RO}^*) from the memory system. The *info* and d_{RO}^* are identified with the one-hot coded 2-bit *id* signal. The *sMB* signal is active when sub-MBs are delivered. Since sub-MBs are processed one at a time, four separate sub-MB deliveries are needed per MB.

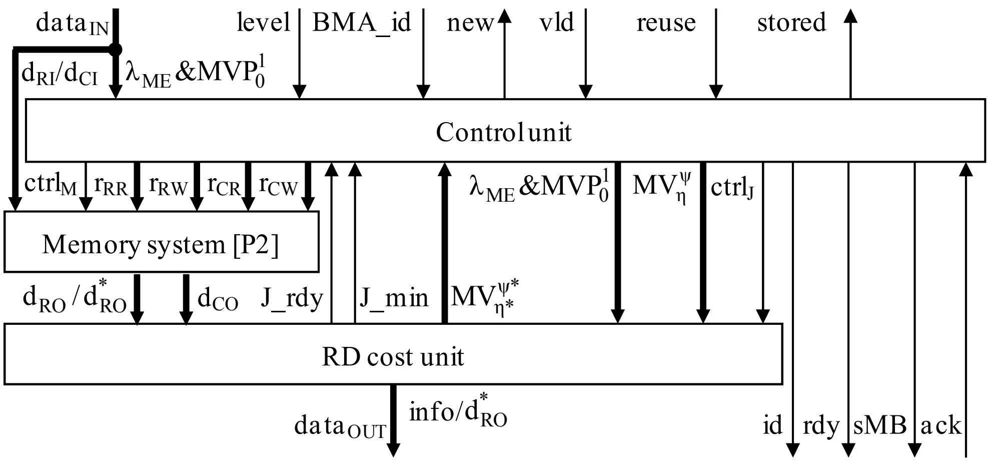


Figure 5.5. The implemented motion estimation architecture.

5.2.1 Control unit

The main components and essential signals of the control unit are presented in Figure 5.6. A *VBSME controller* comprises a core of the ME control together with a *BMA control table* and a *VBSME control table*. These tightly coupled components control an r_{RR} *computation unit* that computes $r_{RR} / r_{RR}^{\mathrm{dlvr}}$ and $\mathrm{MV}_{\eta}^{\psi}$. The BMA control table is presented in detail in [P3]. The rest of the components are refined after publishing [P3], so they are considered here more thoroughly.

Figure 5.7 presents a flowchart of the VBSME controller that is extended from the original controller introduced in [P3]. The grey blocks represent operations analogous to the original controller, whereas the white blocks depict extensions explained here.

The VBSME controller stays in the data store mode until the ME architecture has received all the input data (d_{RI}, d_{CI}, and λ_{ME} & MVP_0^1). After data storage, the VBSME controller is initialized to test p_0^1 with the BMA assigned by the *BMA_id* input.

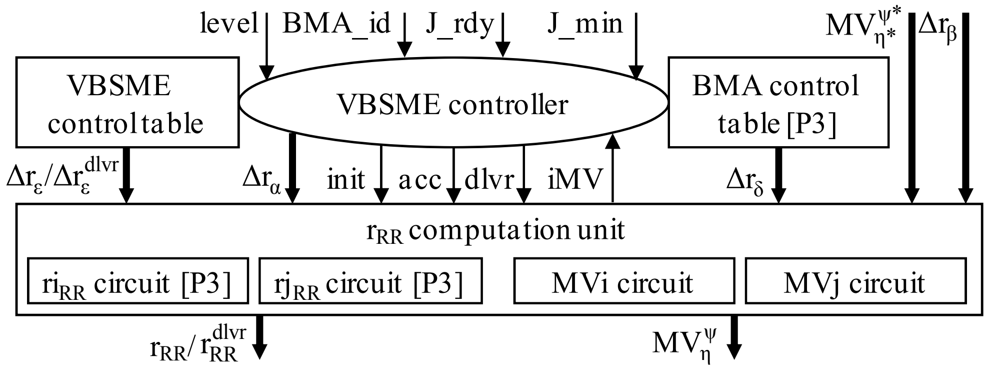


Figure 5.6. Control unit.

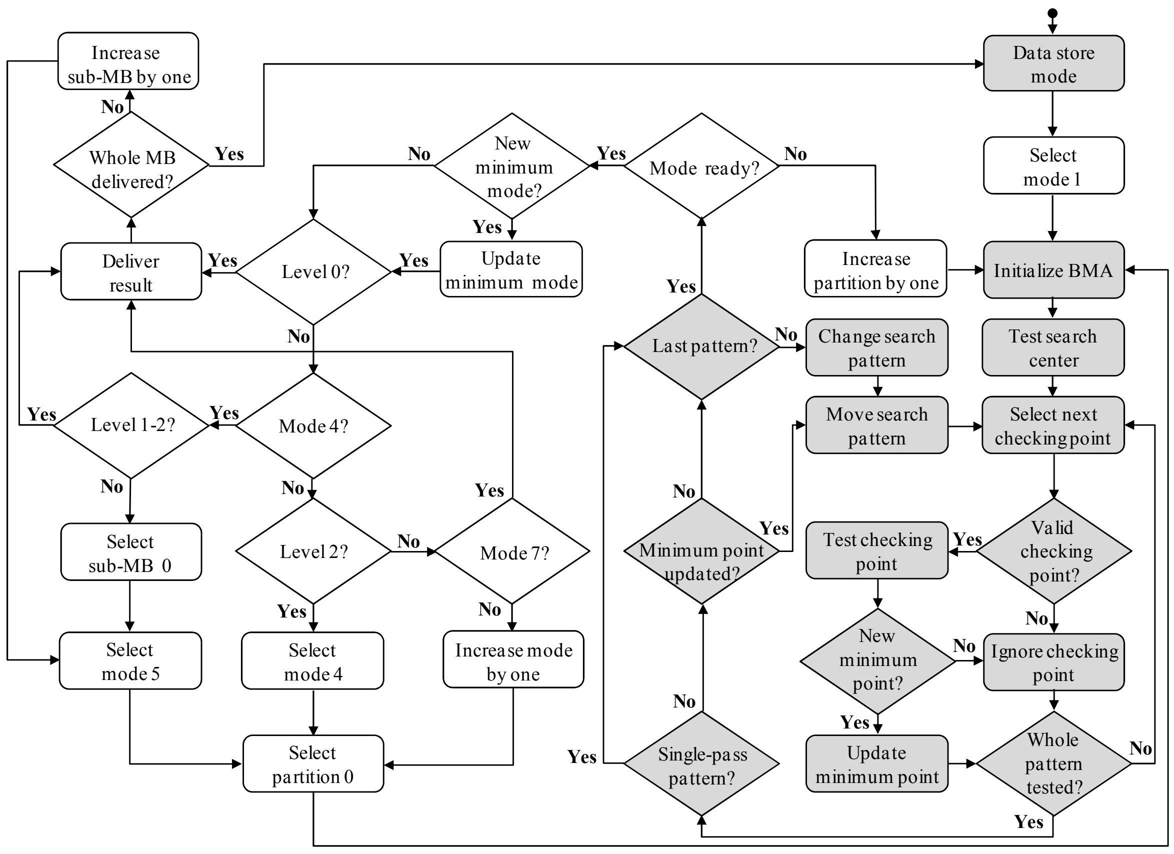


Figure 5.7. Flowchart of the VBSME controller.

The processed m^{ψ} is ready when each p_{η}^{ψ} of it has been tested and J^{ψ} has been computed for it (*J_rdy*$_1$ = '1'). As the first mode, m^1 is always selected as a new minimum after p_0^1 has been tested. At L_0, the search is completed and result data related to m^1 is delivered. During the result delivery, the VBSME controller asserts the *dlvr* signal and the VBSME control table produces an associated $\Delta r_{\varepsilon}^{\text{dlvr}}$ for p_0^1.

At L_2 and L_3, p_0^2 and p_1^2 are tested one by one like p_0^1. After p_1^2 is ready, m^2 is selected as a new minimum, if *J_min*$_1$ signal is asserted together with *J_rdy*$_1$ signal. Next, m^3 and m^4 are tested similarly. At L_1, m^1 and m^4 are tested as in L_2 and L_3, but m^2 and m^3 are skipped. At L_1 and L_2, result data related to $m^{\psi*}$ is delivered after m^4 has been completed. Otherwise (L_3), sub-MB 0 is tested with m^5,…, m^7 and the result data related to $m^{\psi*}$ is sent. The operation continues likewise with the other sub-MBs.

Figure 5.8 depicts the VBSME control table that operates in a search mode (*dlvr* = '0') or in a result delivery mode (*dlvr* = '1'). It receives all data from the VBSME controller.

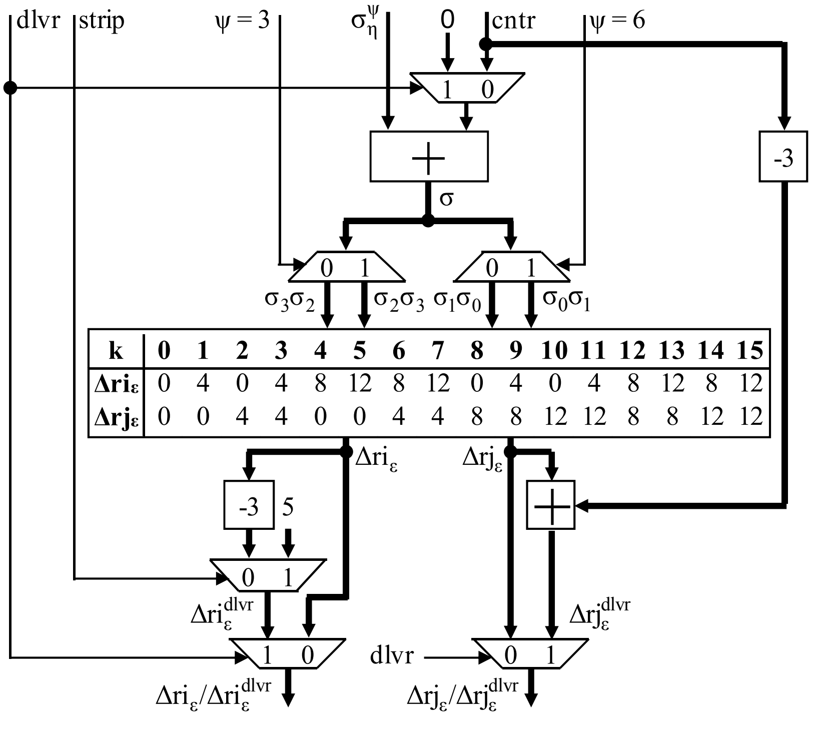
When *dlvr* = '0', an input σ_{η}^{ψ} is set to $(\eta \times Q^{\psi} \times U^{\psi})/16$ according to a processed p_{η}^{ψ}. The content of a LUT equals Figure 5.2 a). It is indexed by σ which is a sum of σ_{η}^{ψ} and a counter (*cntr*). The VBSME controller initializes the *cntr* to 0 and increments it by one per

4×4 block access until $(Q^{\psi} \times U^{\psi})/16$ blocks have been accessed. The unit swaps two MSBs of σ with m^3 as in Figure 5.2 b) and two LSBs of σ with m^6 as in Figure 5.2 c). The output of the unit $(\Delta ri_{\varepsilon}, \Delta rj_{\varepsilon})$ is directly obtained from the LUT.

When *dlvr* = '1', the index of the LUT is merely derived from $\sigma_{\eta^*}^{\psi^*}$ $(\sigma = \sigma_{\eta^*}^{\psi^*} + 0)$. The unit outputs $\Delta ri_{\varepsilon}^{\text{dlvr}} = \Delta ri_{\varepsilon} - 3$ for the first strip (*strip* = '0'). Since $\Delta ri_{\varepsilon} = 0$ for each $p_{\eta^*}^{\psi^*}$ accessed with two strips, $\Delta ri_{\varepsilon}^{\text{dlvr}} = \Delta ri_{\varepsilon} - 3 = (0+8) - 3 = 5$ is constant for the second strip (*strip* = '1'). For both strips, the unit outputs $\Delta rj_{\varepsilon}^{\text{dlvr}} = \Delta rj_{\varepsilon} + cntr - 3$. In this mode (*dlvr* = '1'), the *cntr* equals the amount of accessed 16-pixel wide rows. The VBSME controller increments the *cntr* from 0 to $U^{\psi^*} + 5$ in $U^{\psi^*} + 6$ cycles.

The r_{RR} computation unit (Figure 5.6) includes ri_{RR} and rj_{RR} *circuits* which compute (5.1) and (5.2), respectively. These circuits have been earlier introduced in [P3]. As a new feature, the circuits also compute r_{RR}^{dlvr} according to (5.9) and (5.10) during data delivery. This is realized by additional input multiplexers that select either operands of (5.1)-(5.2) or (5.9)-(5.10) for the circuits.

In parallel with r_{RR} determination, the r_{RR} computation unit also computes (5.3) and (5.4) with mutually identical MV*i* and MV*j* circuits, respectively. Figure 5.9 depicts the MV*i* circuit. It computes $\text{MV}i_{\eta}^{\psi}$ for the search center (*init* = '1') as $\text{MV}i_{\eta}^{\psi} = \Delta ri_{\delta} + \Delta ri_{\beta}$ and the result is stored in the register (Reg). Since $\Delta ri_{\delta} = 0$ for the search center, $\text{Reg} = \Delta ri_{\beta}$.



k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Δriε	0	4	0	4	8	12	8	12	0	4	0	4	8	12	8	12
Δrjε	0	0	4	4	0	0	4	4	8	8	12	12	8	8	12	12

Figure 5.8. VBSME control table.

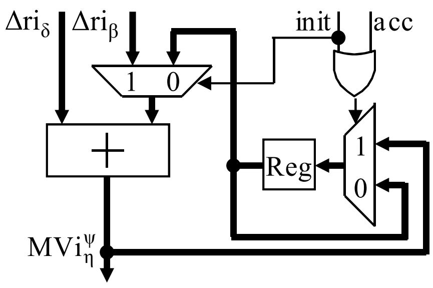


Figure 5.9. MV*i* circuit.

When *init* = '0', the circuit computes $\mathrm{MV}i_\eta^\psi$ for a tested checking point around the pattern center as $\mathrm{MV}i_\eta^\psi = \Delta ri_\delta + \mathrm{Reg}$. After the first search pattern has been completed, Δri_δ corresponding to temporary $\mathrm{MV}i_{\eta*}^\psi$ is redelivered to the circuit and Reg is accumulated (*acc* = '1') with it as $\mathrm{Reg} = \Delta ri_\delta + \mathrm{Reg}$. Now, the updated Reg points to the center of the next search pattern, i.e., $\mathrm{Reg} = \Delta ri_\beta + \Delta ri_\chi$ (Figure 5.1).

The r_{RR} computation unit includes an additional monitoring logic that asserts an invalidation signal (*iMV*), if tested $\mathrm{MV}i_\eta^\psi$ violates (5.5) or (5.6). In that case, the VBSME controller selects a next valid checking point.

5.2.2 Memory system

The memory system (Figure 5.5) is introduced in [P2]. It contains two separate local on-chip memories: a *search area memory* for d_{RI} and a *current block memory* for d_{CI} that include 16 1-pixel (8-bit) wide and four 4-pixel (32-bit) wide single-port parallel memory modules, respectively. The search area memory is configurable for the search areas of $w, h \in \{48, 80, 112, 144, ...\}$ at design time. Besides the memory modules, the memory system includes two *data rotators* (one for d_{RI} /d_{CI} and the other for d_{RO} / d^*_{RO}) and an *address computation unit* which composes memory addresses from r_{RR}, r_{RW}, r_{CR}, and r_{CW}.

The pixel addressable search area memory supports an arbitrary placement of the 4×4 block access format during data retrieval. Hence, it is able to provide data for any p_η^ψ through the desired scan sequence (Figure 5.2) in $(Q^\psi \times U^\psi)/16$ cycles. As a new feature, the refined result delivery scheme also requires an unrestricted placement set for the 16-pixel row format. Therefore, the addressing of the search area is upgraded from [P2] in which the placement set of the row format is restricted. Further description of this modification is omitted here, since it causes only slight changes in the original computation structures without practical effect on the memory system operation.

5.2.3 RD cost unit

Figure 5.10 shows a high-level structure and essential signals of the RD cost unit. An *ABS unit*, a *compression array*, and a *min RD cost unit* are initially introduced in [P1], whereas a *rate computation unit* and a *min mode cost unit* are presented here for the first time.

The ABS unit computes absolute values (ABS) between current (d_{CO}) and reference (d_{RO}) pixels. It completes 16 ABS values per cycle, so a 4×4 block can be processed in parallel. The compression array adds subsequent groups of 16 ABS values together. Hence, it is able to compute J_η^ψ for p_η^ψ in $(Q^\psi \times U^\psi)/16$ passes. The array is initialized (*init* = '1') during the first pass. The initialization does not set the previously accumulated result to zero as in [P1], but it replaces the previous result by $\lambda_{\mathrm{ME}} \times R_{\mathrm{MV}}$. This modification enables the accumulation array to compute (3.21) instead of (3.18).

The completed J_η^ψ is delivered to the min RD cost unit which compares it to the stored minimum RD cost ($J_{\eta*}^\psi$) when *cmpr* = '1'. If $J_\eta^\psi < J_{\eta*}^\psi$, the unit activates *J_rdy*₀ and *J_min*₀ signals and replaces the stored $J_{\eta*}^\psi$ with J_η^ψ. Otherwise, only *J_rdy*₀ signal is asserted. In the case of the search center (*sc* = '1'), J_η^ψ is always stored as a new $J_{\eta*}^\psi$.

Figure 5.11 presents the rate computation unit that produces $\lambda_{\mathrm{ME}} \times R_{\mathrm{MV}}$ for the compression array. For each p_η^ψ, the unit computes absolute MV difference $|\mathrm{MVD}_\eta^\psi| \left(|\mathrm{MVD}i_\eta^\psi|, |\mathrm{MVD}j_\eta^\psi|\right)$ between $\mathrm{MV}_\eta^\psi(\mathrm{MV}i_\eta^\psi, \mathrm{MV}j_\eta^\psi)$ and $\mathrm{MVP}_0^1(\mathrm{MVP}i_0^1, \mathrm{MVP}j_0^1)$. $|\mathrm{MVD}i_\eta^\psi|$ and $|\mathrm{MVD}j_\eta^\psi|$ address a LUT1 that outputs lengths of respective Exp-Golomb codewords ($R_{\mathrm{MV}i}$ and $R_{\mathrm{MV}j}$) for them. $R_{\mathrm{MV}i}$ and $R_{\mathrm{MV}j}$ are added together to yield R_{MV} that is multiplied by λ_{ME}.

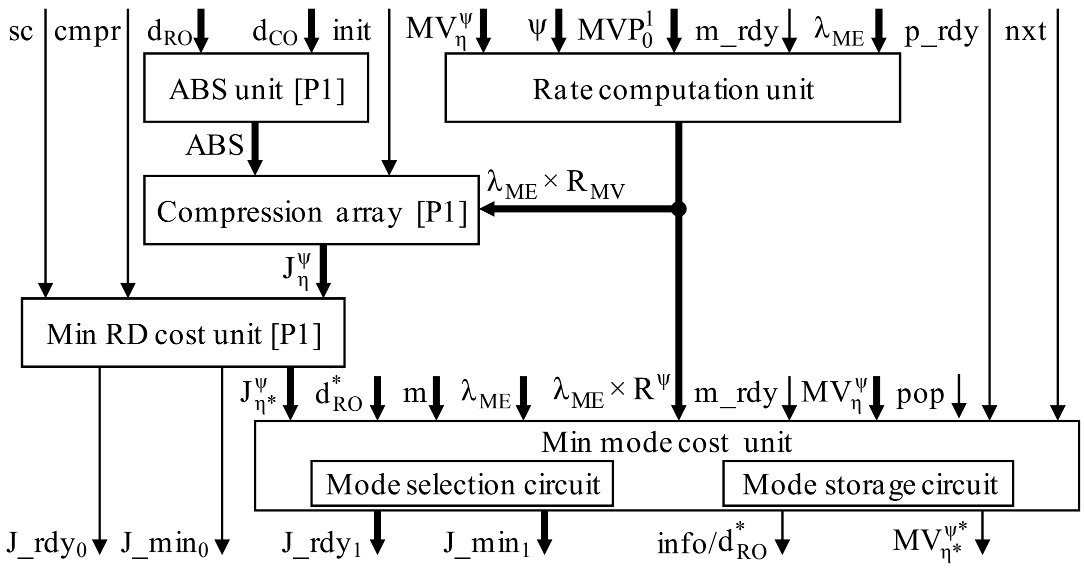


Figure 5.10. RD cost unit.

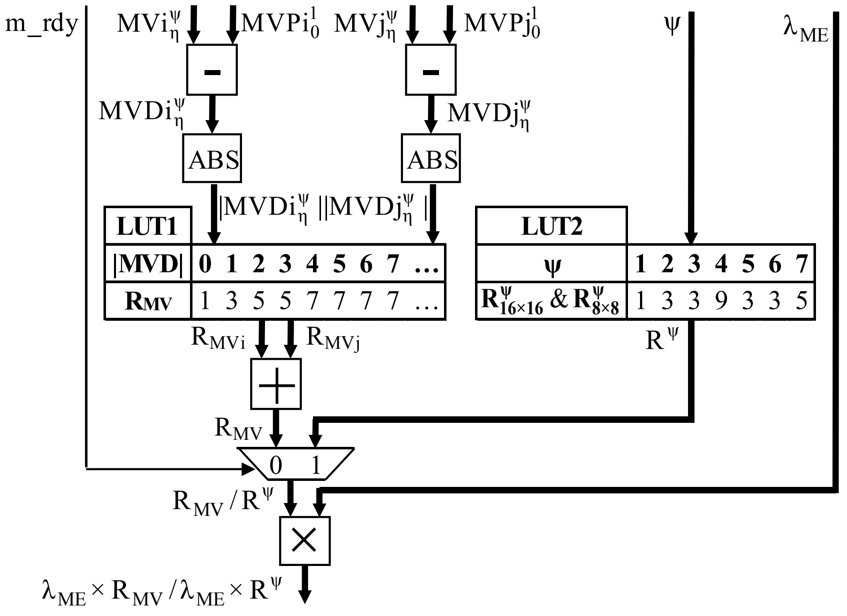

m_rdy
MVi$^\psi_\eta$
MVPi$^1_0$
MVj$^\psi_\eta$
MVPj$^1_0$
ψ
λ$_{ME}$
-
-
MVDi$^\psi_\eta$
MVDj$^\psi_\eta$
ABS
ABS
|MVDi$^\psi_\eta$||MVDj$^\psi_\eta$|
LUT1
|MVD| 0 1 2 3 4 5 6 7 ...
R$_{MV}$ 1 3 5 5 7 7 7 7 ...
LUT2
ψ 1 2 3 4 5 6 7
R$^\psi_{16\times16}$ & R$^\psi_{8\times8}$ 1 3 3 9 3 3 5
R$_{MVi}$
R$_{MVj}$
R$^\psi$
+
R$_{MV}$
0 1
R$_{MV}$ / R$^\psi$
×
λ$_{ME}$ × R$_{MV}$ / λ$_{ME}$ × R$^\psi$

Figure 5.11. Rate computation unit.

When m^ψ is completed (*m_rdy* = '1'), the unit outputs $\lambda_{\text{ME}} \times R^\psi$ instead of $\lambda_{\text{ME}} \times R_{\text{MV}}$. R^ψ is produced by accessing a LUT2 with input ψ. R^ψ represents $R^\psi_{16\times16}$ in (3.23) when $1 \le \psi \le 4$ and $R^\psi_{8\times8}$ in (3.24) when $5 \le \psi \le 7$. The missing $R^4_{8\times8} = 1$ is assigned for $J^4_{8\times8}$ in the min mode cost unit (Figure 5.10).

The min mode cost unit selects $m^{\psi*}$ after which it sequentially outputs *info* data ($\psi*$, $\eta*$, $J^{\psi*}_{\eta*}$, and $\text{MV}^{\psi*}_{\eta*}$) and d^*_{RO} related to each $p^{\psi*}_{\eta*}$ of $m^{\psi*}$. The unit contains a *mode selection circuit* for $m^{\psi*}$ selection and a *mode storage circuit* for $J^{\psi*}_{\eta*}$ and $\text{MV}^{\psi*}_{\eta*}$ storage. It also has a separate output for $\text{MV}^{\psi*}_{\eta*}$ according to which an associated d^*_{RO} is sent to its input from the memory system (Figure 5.5).

The mode selection circuit is depicted in Figure 5.12. It computes (3.23) for m^1,..., m^4 in $n^\psi + 1$ passes and (3.24) for m^5,..., m^7 in $(n^\psi / 4) + 1$ passes. For these operations, the unit receives $J^\psi_{\eta*}$ values from the min RD cost unit and $\lambda_{\text{ME}} \times R^\psi$ from the rate computation unit (Figure 5.10). Each time $p^\psi_{\eta*}$ is ready (*p_rdy* = '1' and *m_rdy* = '0'), a temporary mode cost value (J^ψ_{tmp}) is incremented by the input $J^\psi_{\eta*}$ and the sum is stored in a register Reg$_2$. The accumulation of J^ψ_{tmp} takes n^ψ passes with m^1,..., m^4 and $n^\psi / 4$ passes with m^5,..., m^7. For example, two passes are needed to compute $J^2_{\text{tmp}} = J^2_0 + J^2_1$ for m^2.

During the last pass (*m_rdy* = '1' and *p_rdy* = '0'), $\lambda_{\text{ME}} \times R^\psi$ input is added to J^ψ_{tmp}. The produced sum (J^ψ) equals $J^\psi_{16\times16}$ at MB level and $J^\psi_{8\times8}(s)$ of the sub-MB s at sub-MB level. In $J^4_{8\times8}(s)$ computation, the unit reuses $J^4_{\eta*}$ values that have already been computed for m^4 at MB-level, i.e., $J^4_s = J^4_{\eta*}$ when $s = \eta$. J^4_s values are stored in the mode storage circuit from which the mode selection circuits obtains them and computes $J^4_{8\times8}(s) = J^4_s + \lambda_{\mathrm{ME}}$ ($R^4_{8\times8} = 1$).

The last pass also triggers comparison between J^{ψ} and $J^{\psi*}$ that is stored in Reg$_3$. However, $J^0_{16\times16}$ ($\psi = 0$) and $J^4_{8\times8}(s)$ (*nxt* = '1') are the first values at MB and sub-MB levels, respectively, so they are stored in Reg$_3$ without comparison due to the absence of valid $J^{\psi*}$. In other cases, the comparison is performed. For the comparison, J^{ψ}_{tmp}, $\lambda_{\mathrm{ME}} \times R^{\psi}$, and the bit-inverted (INV) $J^{\psi*}$ are added together with carry-save adder (CSA) that yields two difference vectors (*d_s* and *d_c*). They are added together. To convert the bit-inverted $J^{\psi*}$ to two's complement representation, the carry-in of the adder is '1'. The adder outputs only MSB of the sum. If *MSB* = '0' after the comparison, $J^{\psi} < J^{\psi*}$ and $J^{\psi*}$ is replaced by J^{ψ} in Reg$_3$. In this case, both outputs (*J_rdy*$_1$ and *J_min*$_1$) are asserted. Otherwise, $J^{\psi*}$ is maintained in Reg$_3$ and *J_rdy*$_1$ is only activated. In both cases, Reg$_2$ is set to zero for the next J^{ψ}_{tmp} accumulation.

Figure 5.13 shows a part of the mode storage circuit: a storage logic for $J^{\psi}_{\eta*}$ values. $\mathrm{MV}^{\psi}_{\eta*}$ values are stored respectively. The storage for $J^{\psi}_{\eta*}$ values contains three buffers (*Buf*$_1$,..., *Buf*$_3$) of depth four (indexes 0 - 3). When *p_rdy* = '1', the associated $J^{\psi}_{\eta*}$ is pushed into Buf$_1$(0) and the other values in Buf$_1$ are shifted forward by one position. When *J_min*$_1$ = '1', the content of Buf$_1$ is copied to Buf$_2$.

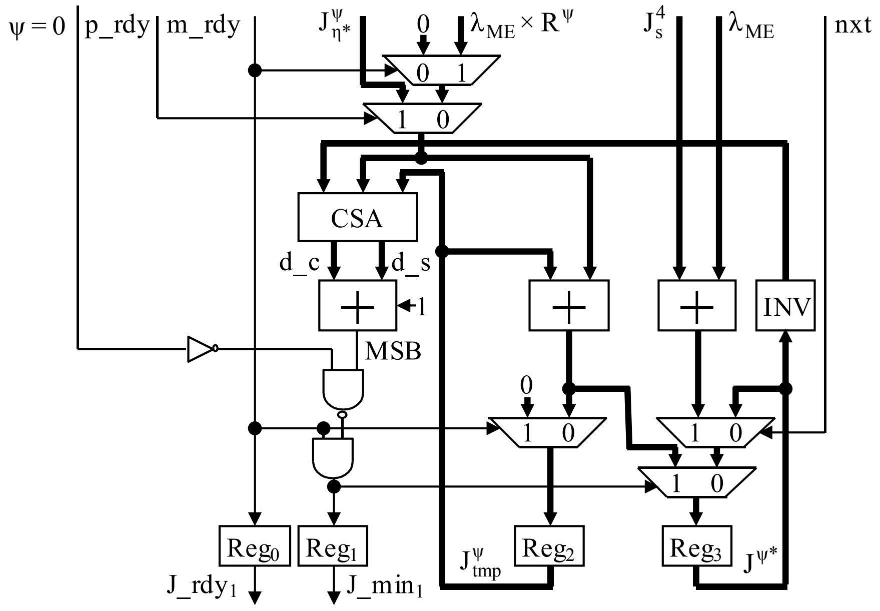


Figure 5.12. Mode selection circuit.

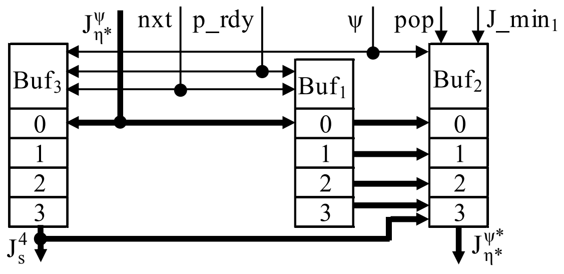


Figure 5.13. Storage logic for partition costs.

In Buf$_2$, the copied values are shifted forward (one position per cycle) until Buf$_2$(3) contains valid data. The amount of shifting is defined by ψ. The values are shifted by $4-n^{\psi}$ positions when $1 \le \psi \le 4$ and $4-n^{\psi}/4$ positions when $5 \le \psi \le 7$. For example, J_0^3 and J_1^3 are copied to Buf$_2$(1) and Buf$_2$(0) from which they are shifted by 4 - 2 = 2 positions to Buf$_2$(3) and Buf$_2$(2), respectively. During data delivery (*pop* = '1'), $J_{\eta*}^{\psi*}$ values are popped one by one from Buf$_2$.

To enable testing of each sub-MB *s* at L_3, $J_{\eta*}^{4}$ values are also stored in Buf$_3$ when *p_rdy* = '1'. When a testing of a new sub-MB is started (*nxt* = '1'), J_s^4 for it is popped from Buf$_3$ and sent to the mode selection circuit (Figure 5.12). J_s^4 is also stored to Buf$_2$(3) which is replaced by new $J_{\eta*}^{\psi}$ values if smaller $J_{8\times8}^{\psi}$ is found among $m^5,\ldots, m^7$.

5.3 Architecture summary

In summary, the designed ME architecture has the following three main features:

1. It supports H.261/3, MPEG-1/2, MPEG-4 Visual, H.264/AVC, and VC-1 standards
2. It can perform rate-constrained IME with various fast BMAs
3. It conducts mode decision/delivery jointly with IME

The architecture realizes multi-standard and multi-BMA support by executing all BMAs with a single *generic search strategy* that is parametrizable to specific coding modes, search centers, and search patterns. The generic search strategy is implemented in the control unit by composing the search paths of the BMAs from five mutually independent offset. The *separable search path generation* provides easy access to individual BMA parameters which can be modified without changing other BMA features. Each coding mode of a BMA undergoes similarly parameterized search and the mode decision selects the best one of the serially executed modes.

Integrating the flexible control with the efficient memory system and RD cost unit enables that the architecture can operate at adequate processing speed, low cost, and acceptable power consumption. The performance of the architecture is analyzed in the next chapter.

6. Performance Analysis

This chapter presents experimental results for the proposed ME framework and for the implemented ME architecture. The framework is evaluated in terms of its impact on overall RD performance and its search speed is benchmarked with well-known fast BMAs. The best framework configurations are selected after which the ME architecture is configured according to their requirements. The implementation results of the architecture configurations are presented and compared with the state-of-the-art ME architectures.

6.1 Framework evaluation

The performance of the ME framework was evaluated with five well-known fast BMAs: TSS, BBGDS, DS, HEXBS, and CDS, whose search strategies and patterns are described in Section 4.1.3. The experiments were accomplished by integrating the framework functionality in JM 17.0 reference encoder [59] and each BMA was individually tested as a part of JM. RD performances and search speeds of the BMAs were measured and compared in order to find the best ones for different resolutions and motion contents.

JM was limited to use BP setting (level 4.0), the IPPPP coding structure, and a single reference frame in MCP. The prediction residuals of intra/inter frames and MVs were entropy coded using CAVLC/UVLC. IME and FME applied SAD and SATD criteria for distortion computation, respectively. RDO and rate control were disabled.

The experiments covered nine popular test sequences: three CIF sequences ("Salesman" "Foreman", and "Football"), three D1 sequences ("Barcelona", "Mobile & Calendar" ("MobCal") and "F1 Car"), and three 1080p sequences ("Station", "Pedestrian", and "Speed Bag"). For each resolution, the sequences represent scenes with low, medium, and high motion contents, respectively. Test sequences were encoded with architecture quality levels L_0 - L_3 and $QP \in \{20, 28, 36\}$. CIF and D1 sequences were encoded entirely, but only 50 frames were encoded with 1080p sequences to speed-up measurements. The "Station" and "Pedestrian" sequences cover the first 50 frames, whereas the last 50 frames of the "Speed Bag" sequence were encoded since its motion content is higher in the end.

The search areas were centered around (i, j) as in Figure 3.6. The BMAs were started from MVP that was computed as in Section 3.4.2. The search ranges were 59×59 pixels $(\mathrm{MV}x, \mathrm{MV}y \in [-29, 29])$ with CIF/D1 formats and 123×123 pixels $(\mathrm{MV}x, \mathrm{MV}y \in [-61, 61])$ with 1080p format, so they are close to guidelines of [115].

6.1.1 Rate-Distortion performance analysis

BMAs were compared in terms of JM output BD-rate. According to the measurements, BBGDS yields the lowest output BD-rate for all low-motion sequences independent of the format. It also suits best for the medium-motion sequences up to D1 resolution. DS outperforms BBGDS in high motion sequences and it is also better for medium-motion sequences at 1080p resolution. DS is reasonable to replace with TSS if the motion content is high and complex. Among the test sequences, TSS performs best only in the “F1 Car” sequence.

Table 6.1 tabulates the sequence-specific encoding results for the best BMAs. Each BMA is also compared with the default FS algorithm in JM 17.0 and the BD-rates between them are reported for each QP value. In addition, average BD-rates per level are computed from QP-specific bit rates. Fast BMAs perform mode decision at IME stage (Section 5.1.2), whereas a complete mode decision is conducted at FME stage after FS. Hence, the effect of the early mode decision at IME stage is also included in BD-rates.

Let us first examine compression ratios of these sequences. For each sequence, average compression ratios per QP can be derived from QP-specific output bit rates at L_0 - L_3. I.e., an average of them is compared with the data rate of the uncompressed sequence. The highest average compression ratios among the sequences are 43:1 at QP = 20, 286:1 at QP = 28, and 678:1 at QP = 36. They are attained with the high-motion “Speed Bag” (QP = 20), low-motion “Salesman” (QP = 28), and low-motion “Station” (QP = 36) sequences. On the other hand, the low-motion “MobCal” sequence has the lowest average compression ratios (4:1 at QP = 20, 8:1 at QP = 28, and 35:1 at QP = 36) due to its various sharp details. These examples illustrate that the compression performance cannot be merely deduced from the motion content, but texture has also essential impact on it.

Secondly, the BD-rate differences are considered between the quality levels. For each format, the average inter-level differences can be derived from the nine QP-specific BD-rate differences (three per sequence). With CIF sequences, the compression ratio increases as a function of the quality level. Compared to L_0, the average decrement of the BD-rate is 3.5%, 5.4%, and 5.5% at L_1, L_2, and L_3, respectively. However, the respective BD-rate variations with D1 sequence are 0.7%, 1.3%, and 0.7%. I.e., L_3 has negative effect (-0.6%) on the BD-rate compared with L_2. The benefits of L_1, L_2, and L_3 are further degraded with 1080p sequences in which the average BD-rate differences are -0.1%, -0.7%, and -0.7%, respectively. These measurements imply that merely L_0 would be adequate for 1080p. However, more thorough experiments in [87] recommend that only L_3 is excluded with 1080p resolutions and above.

Finally, the average BD-rates are computed between the selected fast BMAs and FS. With CIF format, the BD-rate overhead of BBGDS/DS is -0.3 - 4.9%. The highest individual gap (7.8%) exists with the “Salesman” sequence at L_3 when QP = 20. With D1 sequences, the average gain of FS is converged to -4.1 - 3.4%. Although the BD-rate gap is slightly widened with 1080p sequences to 0.8 - 4.5%, the upper bound of the range is still lower than with CIF format. Hence, these experiments state that the fast BMAs are also competitive with high resolution sequences. Among all considered test sequences, the average BD-rate between fast BMAs and FS is only 1.9%.

Table 6.1. RD performance comparison of fast BMAs and FS in JM 17.0

Format	Search Range	Sequence	Motion Content	Selected BMA	Quality Level	QP = 20			QP = 28			QP = 36			Average
						PSNR (dB)	Bit Rate (Mbit/s)	BD-rate (%)	PSNR (dB)	Bit Rate (Mbit/s)	BD-rate (%)	PSNR (dB)	Bit Rate (Mbit/s)	BD-rate (%)	BD-rate (%)
CIF	59×59	Salesman (449 frames)	Low	BBGDS	L_0	42.07	2.96	0.12	36.21	0.24	-1.01	30.97	0.06	-0.09	-0.33
					L_1	42.04	2.83	3.47	36.19	0.23	2.86	30.96	0.05	-0.09	2.08
					L_2	42.04	2.78	3.31	36.22	0.22	3.51	31.03	0.05	1.08	2.63
					L_3	42.04	2.78	7.82	36.22	0.22	6.67	31.03	0.05	-0.30	4.73
		Foreman (300 frames)	Medium	BBGDS	L_0	42.86	2.56	0.41	36.93	0.68	1.04	31.57	0.18	2.26	1.24
					L_1	42.84	2.49	2.86	36.92	0.65	3.69	31.56	0.18	3.21	3.26
					L_2	42.84	2.47	3.79	36.94	0.64	1.06	31.62	0.17	5.14	3.33
					L_3	42.84	2.46	4.72	36.94	0.64	5.42	31.62	0.17	4.40	4.85
		Football (260 frames)	High	DS	L_0	43.28	4.25	1.04	37.13	1.83	2.06	31.43	0.70	4.85	2.65
					L_1	43.27	4.13	2.03	37.13	1.77	3.38	31.41	0.68	5.60	3.67
					L_2	43.27	4.11	2.19	37.13	1.75	3.68	31.41	0.67	6.60	4.16
					L_3	43.26	4.13	-3.25	37.12	1.75	-0.43	31.42	0.67	4.62	0.31
D1	59×59	Barcelona (220 frames)	Low	BBGDS	L_0	42.51	29.22	-0.01	35.60	10.39	0.01	29.34	2.52	-0.51	-0.17
					L_1	42.50	29.01	0.40	35.60	10.25	0.97	29.34	2.47	0.51	0.63
					L_2	42.50	28.94	0.76	35.61	10.18	1.53	29.39	2.41	0.59	0.96
					L_3	42.50	28.99	-1.52	35.61	10.18	0.32	29.39	2.41	1.00	-0.07
		MobCal (220 frames)	Medium	BBGDS	L_0	42.65	40.34	0.02	35.12	18.14	0.12	27.90	4.31	0.04	0.06
					L_1	42.64	40.38	-0.29	35.12	18.07	0.79	27.89	4.24	2.01	0.84
					L_2	42.64	40.30	-0.05	35.12	18.00	1.36	27.91	4.19	3.42	1.58
					L_3	42.64	40.62	-5.54	35.11	18.04	-1.64	27.91	4.19	3.10	-1.36
		F1 Car (220 frames)	High	TSS	L_0	42.84	33.65	0.76	35.77	14.19	2.17	29.90	4.05	7.20	3.38
					L_1	42.84	33.65	-0.36	35.77	14.19	1.19	29.90	4.05	6.45	2.43
					L_2	42.84	33.67	-0.07	35.77	14.18	1.75	29.90	4.02	7.76	3.14
					L_3	42.84	34.79	-9.56	35.77	14.34	-7.66	29.90	4.03	4.80	-4.14
1080p	123×123	Station (50 frames)	Low	BBGDS	L_0	42.78	24.10	0.23	39.32	2.59	0.91	34.98	1.29	3.76	1.63
					L_1	42.77	24.12	0.99	39.32	2.60	-0.21	34.98	1.29	3.28	1.35
					L_2	42.78	24.01	2.57	39.37	2.67	3.26	35.04	1.31	4.21	3.35
					L_3	42.78	24.00	2.53	39.36	2.67	2.69	35.05	1.31	3.70	2.97
		Pedestrian (50 frames)	Medium	DS	L_0	43.63	28.99	1.03	40.52	7.25	3.06	36.78	3.11	5.77	3.29
					L_1	43.62	29.05	1.16	40.52	7.26	2.55	36.78	3.10	5.67	3.13
					L_2	43.63	29.00	2.14	40.53	7.24	4.32	36.85	3.13	7.14	4.53
					L_3	43.63	29.03	0.86	40.53	7.25	3.09	36.84	3.12	6.95	3.63
		Speed Bag (50 frames)	High	DS	L_0	45.42	17.42	0.93	43.14	4.82	1.82	39.83	2.31	0.42	1.05
					L_1	45.42	17.42	0.90	43.14	4.83	1.37	39.82	2.30	0.23	0.83
					L_2	45.42	17.36	1.29	43.17	4.85	2.52	39.92	2.33	1.20	1.67
					L_3	45.42	17.36	0.70	43.17	4.84	1.72	39.92	2.33	1.26	1.23

6.1.2 Search speed analysis

In the analyzed test sequences, the computational complexity of ME increases as a function of resolution and motion content. Hence, the high-motion sequences are the most computation-intensive ones with each resolution. Table 6.2 gathers these worst-case sequences from Table 6.1 and reports the search speeds of the selected BMAs with them.

The architecture-independent search speeds of the selected BMAs are tabulated as an average number of checking points per current MB (points/MB) at QP values of 20, 28, and 36. A checking point tested with m^1 increments points/MB value by one, a checking point tested with m^2 increments points/MB value by ½, etc. TSS applies the same fixed search pattern for each m^ψ, so its points/MB value is directly proportional to the amount of modes at L_0 - L_2. I.e., compared with L_0, the points/MB value of TSS is doubled at L_1

and quadrupled at L_2. The respective ratios are lower with DS (1.7 - 1.8 and 3.3 - 3.6) whose variable-length search paths happen to converge earlier with $m^2,\ldots, m^4$ than with m^1. The conditional execution of $m^5,\ldots, m^7$ significantly restrains the increase of points/MB at L_3. For example, the points/MB ratio of TSS is only 5.0 between L_3 and L_0 although seven times more modes are available. With DS, the respective ratio is 3.4 - 4.2.

The average speed-up ratios tabulated for DS and TSS are computed over FFS which computes coding modes in parallel by reusing J_η^ψ values. Although TSS and DS execute coding modes serially, they are one to two orders of magnitude faster than FFS in all the examined cases. The serial execution decreases the average speed-up ratios of DS and TSS when the quality level is incremented, but the worst-case speed-up ratio is still almost 18.

A total clock cycle count per MB (*t*/MB) reports the search speeds of the BMAs on the designed ME architecture. In the search speed simulations, the ME architecture is configured to use QP = 20 since it is the worst case of the tested QP values. The *t*/MB value is composed of cycles in data storage, distortion computation, and data delivery. The cycle count of distortion computation increases almost linearly with points/MB value if the block size remains constant. Averagely, the architecture computes distortion between two 16×16 pixel blocks in 18 cycles. The BMA-specific variation is approximately ±1 cycles among TSS, DS, BBGDS, CDS, and HEXBS. Computing distortion between two 16×8 or 8×16 pixel blocks elapses 10 cycles on average, so their relative cycle count is about 1.1 times higher than that of 16×16 pixel blocks. The respective ratios for 8×8, $8\times4/4\times8$, and 4×4 pixel blocks are 1.3, 1.7, and 2.5.

The increased cycle counts of the smaller blocks imply that the cycle count of distortion computation grows faster than the points/MB value if the quality level is incremented. However, the increment of the quality level causes only a small overhead in data delivery and no overhead in data storage, so the relative increase in the overall *t*/MB value is still quite moderate. The correlation between *t*/MB and points/MB values is illustrated by a cycle count per checking point (*t*/point) value which is derived by dividing *t*/MB value by points/MB value. Compared with L_0, the average change in *t*/point value is 4% at L_1, -5% at L_2, and 18% at L_3. Hence, the overall *t*/MB value follows points/MB value quite closely also at the higher quality levels.

Table 6.2. Search speeds of BMAs and minimum operating frequencies for real-time IME.

Format	Search range	Sequence	Selected BMA	Quality Level	Points/ MB	Average Speed-up	t/MB (QP=20)	t/point (QP=20)	MHz @ 30 fps (QP=20)
CIF	59×59	Football (260 frames)	DS	L_0	19.3	180.6	570	29.6	7
				L_1	34.9	99.6	1060	30.3	13
				L_2	69.1	50.4	1875	27.1	23
				L_3	81.0	43.0	2809	34.7	34
D1	59×59	F1 Car (220 frames)	TSS	L_0	38.8	89.7	861	22.2	42
				L_1	77.8	44.8	1761	22.6	86
				L_2	155.7	22.4	3278	21.0	160
				L_3	194.9	17.9	5519	28.3	269
1080p	123×123	Speed Bag (50 frames)	DS	L_0	36.9	409.7	1048	28.4	257
				L_1	61.0	248.0	1846	30.3	452
				L_2	121.3	124.7	3403	28.1	834
				L_3	126.3	119.8	3893	30.8	954

Table 6.2 also tabulates minimum operating frequencies for real-time (30 fps) IME. The frequencies are derived for the designed ME architecture as a function of *t*/MB value and resolution. With each format, L_0 specifies minimum operating frequency of real-time IME for H.261 or MPEG-1/2. Respectively, L_1 represents H.263, MPEG-4 Visual, and VC-1. H.264/AVC -compatible real-time IME adopts the frequencies of L_3 with CIF and D1 resolutions. However, it was concluded in Section 6.1.1 that L_2 is adequate with 1080p resolution. Hence, the operating frequency needed by H.264/AVC -compatible real-time IME is reduced from 954 MHz to 834 MHz.

6.2 Implementation results

The designed architecture was described in VHDL at the register transfer level and synthesized to the gate level using Synopsys Design Compiler. Table 6.3 tabulates the implementation results for the selected nine example architecture configurations. They are grouped as $L^{\mathrm{CIF}}_{\{0,1,3\}}$, $L^{\mathrm{D1}}_{\{0,1,3\}}$, and $L^{\mathrm{1080p}}_{\{0,1,2\}}$ according to their capability of process CIF, D1, and 1080p formats at 30 fps, respectively. The supported BMAs, operating frequencies, and search ranges of these configurations are obtained from the experimental results in Table 6.1 and Table 6.2.

In each group, the standards-specific properties of these configurations are realized through the quality level adjustment. Since the quality level can always be lowered without violating the real-time constraints, the configurations also cover the quality levels that are below the specified one. For example, L^{CIF}_3, L^{D1}_3, and L^{1080p}_2 can be reconfigured at run time to other addressed standards without performance compromises. The search range of the architecture is adjustable at design time through VHDL generics.

In Table 6.3, the reported frequency, area, and power consumption values are yielded from logic synthesis on a 0.13-micrometer HCMOS9 standard cell library by STMicroelectronics. The synthesis results are reported under the nominal operating conditions (1.2 V, $25\,^{\circ}\mathrm{C}$). The area values (gate counts) are based on equivalent 2-input NAND gates. The gate count metric includes the pipeline registers and all the units except the SRAM modules. The clock frequencies represent the critical path delay in the pipelined architectures. The frequency values are derived from Table 6.2 and the frequencies are adjusted with Synopsys Design Compiler by using different delay constraints. The power consumption of the configurations was estimated with Synopsys Power Compiler that uses both synthesized gate level net lists and simulated switching activities of the cells to produce power estimates. The power consumption values are separately reported for logic and SRAMs.

$L^{\mathrm{CIF}}_{\{0,1,3\}}$ and $L^{\mathrm{D1}}_{\{0,1,3\}}$ configurations contain one architecture instance which consumes 22.3 - 22.5 kgates and 6.5 KB of SRAM with 59×59 pixel search range. The needed operating frequency is between 7 - 275 MHz depending on the target resolution and standard. The power consumption varies as a function of operating frequency from 3 mW to 68 mW, from which approximately 65% is consumed by the memories.

Table 6.3. Implementation results of the architecture configurations.

Config.	Format	Addressed Standard	Supported BMAs	Search Range	Quality Level	# of instances	Freq. (MHZ)	SRAM (KB)	Area (kgates)	Power (mW) Logic	Power (mW) SRAM
L_0^{CIF}		H.261, MPEG-1/2			L_0	1	7	6.5	22.3	1	2
L_1^{CIF}	CIF	H.263, MPEG-4 Visual, VC-1	BBGDS/DS	59×59	L_1	1	13	6.5	22.3	2	3
L_3^{CIF}		H.264/AVC			L_3	1	34	6.5	22.3	4	7
L_0^{D1}		H.261, MPEG-1/2			L_0	1	43	6.5	22.3	5	9
L_1^{D1}	D1	H.263, MPEG-4 Visual, VC-1	BBGDS/DS/TSS	59×59	L_1	1	91	6.5	22.3	10	18
L_3^{D1}		H.264/AVC			L_3	1	275	6.5	22.5	25	43
L_0^{1080p}		H.261, MPEG-1/2			L_0	1	268	20.5	23.7	25	67
L_1^{1080p}	1080p	H.263, MPEG-4 Visual, VC-1	BBGDS/DS	123×123	L_1	1	455	20.5	25.1	49	135
L_2^{1080p}		H.264/AVC			L_2	2	434	41.0	49.7	97	267

L_0^{1080p} and L_1^{1080p} configurations also contain a single architecture instance. They are implementable with 23.7 - 25.1 kgates and 20.5 KB of SRAM memory when the search range is 123×123 pixels. Their respective power consumptions are 92 mW and 184 mW, from which about 73% is consumed by the memories.

With a single architecture instance, L_2^{1080p} configuration would require 834 MHz operating frequency which is not reached with the applied implementation technology. Therefore, L_2^{1080p} is composed of two identical architecture instances ($L0_2^{1080p/2}$ and $L1_2^{1080p/2}$) that divide the computation burden in a data parallel manner. L_2^{1080p} uses the search range of 123×123 pixels and has a total gate count of 49.7 kgates. The instances operate at 434 MHz, since allocating and balancing the computation between them increases operating frequency about 4%. Altogether, L_2^{1080p} dissipates power 364 mW, from which 267 mW is consumed by the memories.

L_2^{1080p} is able to support search area reuse only if the horizontally consecutive MBs of a reference frame are processed by the same architecture instance. Therefore, a reference frame has to be sub-divided into horizontal slices which are evenly assigned to the instances. The MB dependencies of MVP computation can be resolved by allocating odd rows to $L0_2^{1080p/2}$ and even rows to $L1_2^{1080p/2}$. When $L0_2^{1080p/2}$ starts to process the first row, $L1_2^{1080p/2}$ waits until $L0_2^{1080p/2}$ has completed at least left-most three MBs of the first row. After that delay, the MVP for the left-most MB of the second row is computable. The respective delay is needed through the whole frame.

Besides these example configurations, the designed ME architecture could also be configured to other sets of framework-compatible BMAs. Comparing points/MB values of new potential BMAs with the presented results (Table 6.2) provide a good approximation of the required configuration, i.e., the number of instances and operating frequency. For example, supporting TSS with L_2^{1080p} would need about 500 MHz operating frequency with two architecture instances.

As a proof of concept, the designed ME architecture has also been prototyped on an FPGA using Altera Stratix II EP2S180 DSP development board. On Stratix EP2S180F1020C3 logic device, the architecture configured for TSS, DS, and BBGDS can operate at 192 MHz. It is realized with 2323 ALUTs. Embedded DSP blocks on the FPGA are not used. When the search range is 59×59 pixels, the architecture consumes twenty embedded SRAM blocks of size 4 Kb. This configuration is able to execute D1 format at L_2 in real-time.

6.3 Performance comparison

The characteristics of the designed ME architecture are compared with the contemporary ME architectures (Table 4.1) that are dedicated to the state-of-the-art H.264/AVC. The proposed configurations chosen for the comparison are H.264/AVC -compatible L_3^{CIF}, L_3^{D1}, and L_2^{1080p} (Table 6.3).

Table 6.4 tabulates the performance metrics (frequency, memory, area, power, and throughput) and associated ME parameters (a search range, a set of modes, and a number of the reference frames) of the evaluated ME architectures. The architecture comparison is primarily based on three criteria: logic gate count, throughput, and normalized memory consumption. Throughputs of the architectures have been derived from their maximum resolution and frame rate (Max Res.) values, so throughputs represent normalized architecture performances. Normalized memory consumption equals a ratio of the reported search range and overall memory consumption, i.e., it specifies the amount of memory bits required per a search range pixel. The comparison of normalized power consumption would require that an assumed 30% decrease of power per CMOS process generation [4] is included in the reported power values. However, a detailed comparison of power consumption is excluded here due to absence of most of the power values.

Let us first consider FFS-based architectures. Chen's [11] and Li's [76] architectures achieve the same throughput as L_3^{D1}, but they consume 7.5 and 15 times the gate count of L_3^{D1} and their normalized memory consumptions are 1.6 and 5.9 times that of L_3^{D1}, respectively. Wei's [130] implementation has over 30% lower throughput and its gate count is tripled over L_2^{1080p}. In addition, its search range is far below the recommendations [115] for 720p resolution. Yap's [137] and Kim's [65] architectures are more competitive in terms of gate count, but their search ranges are too limited even for QCIF format.

Chen's [14], Liu's [87], and Lin's [82] fast BMA-specific architectures achieve the target processing speed with lower cost or power than FFS-based architectures. However, for the same throughput, normalized memory consumption of Chen's approach is more than doubled and its gate count is almost 6 times that of L_3^{CIF}. Correspondingly, Liu's architecture uses almost 10 times as much gates as L_2^{1080p}. Lin's implementation supports very large search range for m^1, whereas reduced search ranges are used for the other modes. For example, the search range for $m^5,\ldots, m^7$ is only 16×16 pixels. Compared with L_2^{1080p}, Lin's architecture doubles the throughput, consumes less memory, but its gate count is quadrupled.

Table 6.4. Comparison of the proposed and existing state-of-the-art ME architectures.

Architecture	Supported BMAs	Freq (MHz)	Memory (KB)	Area (Kgates)	Power (mW)	Process (nm)	Search Range	Supported Modes	# of Ref.	Max Res.	Throughput (Mpixels/s)
L_3^{CIF}	BBGDS/DS	34	6.5	22.3	11	130	59×59	1-7	1	CIF@30fps	3
L_3^{D1}	BBGDS/DS/TSS	275	6.5	22.5	68	130	59×59	1-7	1	D1@30fps	12
L_2^{1080p}	BBGDS/DS	434	41.0	49.7	364	130	123×123	1-4	1	1080p@30fps	62
Yap [137]	FFS	294	n.a.	61.0	570	130	16×16	1-7	1	4CIF@45fps	18
Chen [11]	FFS	81	25.4	330.2	n.a.	180	$129\times65/65\times33$	1-7	4	D1@30fps	12
Li [76]	FFS	216	23.8	168.0	n.a.	180	65×33	1-7	2	D1@30fps	12
Wei [130]	FFS	200	3.3	160.0	423	180	33×33	1-7	1	720p@45fps	41
Kim [65]	FFS	416	n.a.	39.0	n.a.	180	16×16	1-7	1	CIF@256fps	26
Liu [86] [87]	Coarse-to-fine	200	n.a.	486.0	n.a.	180	192×128	1-4	1	1080p@30fps	62
Lin [82]	Multiresolution	124	6.0	213.7	n.a.	130	$256\times256,...,16\times16$	1-7	1	1080p@60fps	124
Chen [14]	Parallel-VBS 4SS	27	8.0	131.2	17	180	64×32	1-7	2	CIF@30fps	3
Lee [72]	FFS/Fast BMAs	266	540.0	450.0	n.a.	n.a.	4096×128	1-7	2	1080p@60fps	124
Verma [126]	FFS/Fast BMAs	124	n.a.	82.0	n.a.	90	16×16	1-7	1	1080p@60fps	124
Wei [131]	FFS/Fast BMAs	157	2.0	81.0	47-247	180	32×32	1-7	1	VGA@30fps	9
Zhang [142]	DS + CS	117	n.a.	238.0	n.a.	180	$75\times39,...,13\times13$	1-4	1	720p@30fps	28

n.a. = not available

The rest of the reference architectures support multiple BMAs. Zhang's [142] flexible approach combines IME and FME. It supports adequate search range for m^1, but the search ranges of the other supported modes are significantly smaller. For example, m^4 has a search range of 13×13 pixels. In addition, its logic gate count (IME+FME) is almost 5 times that of L_2^{1080p}. Verma's [126] flexible solution doubles the throughput with 65% gate count overhead compared with L_2^{1080p}, but its search range is too restricted even for QCIF sequences. Lee's [72] architecture outperforms the other architectures in terms of performance, but its HW cost is enormous. Wei's [131] flexible architecture achieves 25% lower throughput than L_3^{D1}, consumes over 3.6 times the gate count of L_3^{D1}, and supports impractical small search range.

L_3^{CIF}, L_3^{D1}, and L_2^{1080p} offer two additional features that are not available in the reference ME architectures: rate-constrained ME with fast BMAs and multi-standard support at run time. Despite these features, L_3^{CIF}, L_3^{D1}, L_2^{1080p} consume 43%, 42%, and 39% less gates than any reference ME architecture targeted for CIF, D1, and 1080p resolution, respectively. Exclusion of the architectures with impractical search ranges increases the gate count savings of L_3^{CIF}, L_3^{D1}, and L_2^{1080p} to 83%, 86%, and 76%, respectively. The only comparable architecture [72] that supports multiple BMAs, adequate search range, and 1080p resolution consumes 9 times the gate count of L_2^{1080p}.

7. Conclusions

Modern video encoders involve complex algorithms whose real-time execution necessitates efficient encoder implementations. The emerging trend is that the same encoder is compatible with several video coding standards, so real-time encoding performance has to be met without compromising flexibility. In mobile handheld devices, real-time constraints and flexibility expectations of the encoders are further combined with strict limitations on cost, size, and power consumption.

Implementing an encoder completely in SW is flexible, but typically too inefficient approach. Although high-end multicore processors would be able to reach acceptable encoding speed (e.g., H.264/AVC 1080p video at 30fps), their power consumption tend to be one to two orders of magnitude higher than the assumed 1W power budget of handheld devices. Therefore, low-power encoders need special-purpose HW either for the whole encoding process or for the encoding assistance in a processor-controlled environment. Complete HW encoders are often implemented in custom ASICs in order to meet the real-time performance with the smallest possible silicon area and power consumption. However, the limited flexibility of ASICs suits best for high-volume encoders that are dedicated to a single standard. FPGAs and video-oriented MPSoCs are viable alternatives for ASICs when encoders need multi-standard support. Particularly, state-of-the-art MPSoCs are flexible, power economical, and efficient enough for real-time encoding.

The allocation of encoding tasks between HW and SW varies in HW-accelerated encoder implementations. However, they all share the common feature of using HW acceleration for ME since it typically accounts for 50 - 90% of the total encoder complexity. Most of the contemporary HW architectures for ME are tailored to a single standard and/or a single BMA. Existing configurable HW architectures support several BMAs, but they also contain standard-specific limitations and are realized with large silicon area, limited processing speed, unsustainable power budget, or over restricted ME parameters.

7.1 Main results

This Thesis introduced a configurable HW architecture that overcomes all the crucial limitations faced by the existing ME architectures. The designed ME architecture supports all inter coding modes of H.261/3, MPEG-1/2, MPEG-4 Visual, H.264/AVC, and VC-1. In each inter coding mode, the architecture is able to perform rate-constrained IME with various fast BMAs such as BBGDS, CDS, DS, HEXBS, and TSS. It can change the target standard, an executed BMA, and a search center of the BMA at run time, whereas the supported set of BMAs and the search range can be adjusted at design time. The architecture also conducts mode decision jointly with IME and returns only the best inter coding mode, so it reduces the complexity of the subsequent FME stage significantly. Its high flexibility and low cost facilitate its usage as a reusable intellectual property block.

The flexibility of the architecture was demonstrated with example architecture configurations that can process real-time single reference frame ME with different standards, resolutions, and search ranges. Configurations for CIF format are implementable with 22.3 kgates and 6.5 KB of SRAM when the search range is 59×59 pixels. Depending on the target standard, they consume 3 - 11 mW of power with a 0.13-micrometer CMOS standard cell technology. The respective performance metrics with D1 format are 22.3 - 22.5 kgates, 6.5 KB of memory, and 14 - 68 mW of power. The configurations targeted for 1080p format use 123×123 pixel search range. They consume 23.7 - 25.1 kgates, 20.5 KB of SRAM, and 92 mW - 184 mW of power when the underlying standard is H.261/3, MPEG-1/2, MPEG-4 Visual, or VC-1. Supporting H.264/AVC with 1080p format requires a duplicated architecture, whose respective metrics are 49.7 kgates, 41 KB of SRAM, and 364 mW of power.

This Thesis verified the search quality of the ME architecture by integrating the introduced IME functionality in JM reference encoder. The experiments with different resolutions (CIF, D1, and 1080p), motion contents (low, medium, and high), and QP values $(\mathrm{QP} \in \{20, 28, 36\})$ show that average output bit rate of JM increases only by 1.9% if the set of available fast BMAs include BBGDS, DS, and TSS.

The main conclusion of this Thesis is that an accurately designed and optimized configurable HW architecture can support multiple standards and process various resolutions and motion contents at adequate processing speed, low cost, acceptable power consumption, and competitive RD performance.

7.2 Future work

In the future, the feasibility of the designed ME architecture will be analyzed with the forthcoming HEVC standard. According to initial evaluations, the ME framework is well suited for the HEVC proposals that use square [5], [89] or symmetric rectangular [124] partitions. It also supports asymmetric rectangular partitions [35] if they can be composed of 4×4 blocks. Instead, the inclusion of non-rectangular partitions [63] in HEVC would require the redesign of the framework.

The future research will also examine the framework compatibility with MVC amendment of H.264/AVC. MVC has been recently popularized due to the emerging 3D and FTV applications. In MVC, the computation load of ME is increased since each view has to undergo temporal or inter-view prediction [26]. The low area cost of the designed ME architecture would enable its duplication in performance-critical MVC architectures. A viable alternative could be to process each view with a separate ME architecture.

To enhance performance of the ME architecture with future standards and resolutions, the research will also consider SAD reuse techniques between overlapped inter coding modes and partitions. Inclusion of these techniques would cause a moderate increase in the complexities of the control and distortion computation units, but they would also augment the throughput of the architecture without increasing the operating frequency. Through SAD reuse, power consumption would be restrained and the designed ME architecture would better meet the needs of MVC and HEVC applications also in the portable devices.

References

[1] Y. Baek, H. S. Oh, and H. K. Lee, "Efficient block-matching criterion for motion estimation and its VLSI implementation," *IEEE Trans. Consumer Electron.*, vol. 42, no. 4, Nov. 1996, pp. 885-892.

[2] G. Bjøntegaard, "Calculation of average PSNR differences between RD curves," *document VCEG-M33*, Austin, TX, USA, Apr. 2001, pp. 1-4.

[3] G. Blake, R. G. Dreslinski, and T. Mudge, "A survey of multicore processors," *IEEE Signal Processing Mag.*, vol. 26, no. 6, Nov. 2009, pp. 26-37.

[4] S. Borkar, "Design challenges of technology scaling," *IEEE Micro*, vol. 19, no. 4, Jul. 1999, pp. 23-29.

[5] F. Bossen, V. Drugeon, E. Francois, J. Jung, S. Kanumuri, M. Narroschke, H. Sasai, J. Sole, Y. Suzuki, T. K. Tan, T. Wedi, S. Wittmann, P. Yin, and Y. Zheng, "Video coding using a simplified block structure and advanced coding techniques," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 12, Dec. 2010, pp. 1667-1675.

[6] A. Bovik, *The Essential Guide to Video Processing*. Elsevier, Burlington, MA, USA, 2009, p. 755.

[7] CAST, Inc., "H264-MP-E: H.264/AVC HD & ED video encoder core," Available online: http://www.cast-inc.com/ip-cores/video/h264-mp-e/.

[8] H. C. Chang, J. W. Chen, C. L. Su, Y. C. Yang, Y. Li, C. H. Chang, Z. M. Chen, W. S. Yang, C. C. Lin, C. W. Chen, J. S. Wang, and J. I. Guo, "A 7 mW to 183 mW dynamic quality-scalable H.264 video encoder chip," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Dig. Tech. Papers, San Francisco, CA, USA, Feb. 2007, pp. 280-281.

[9] H. C. Chang, J. W. Chen, B. T. Wu, C. L. Su, J. S. Wang, and J. I. Guo, "A dynamic quality-adjustable H.264 video encoder for power-aware video applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 12, Dec. 2009, pp. 1739-1754.

[10] C. W. Chen, Z. Li, and S. Lian, *Intelligent Multimedia Communication: Techniques and Applications*. Springer, Berlin, Germany, 2010, p. 506.

[11] C. Y. Chen, S. Y Chien, Y. W. Huang, T. C. Chen, T. C. Wang, and L. G. Chen, "Analysis and architecture design of variable block-size motion estimation for H.264/AVC," *IEEE Trans. Circuits Syst. I*, vol. 53, no. 3, Mar. 2006, pp. 578-593.

[12] T. C. Chen, S. Y. Chien, Y. W. Huang, C. H. Tsai, C. Y. Chen, T. W. Chen, and L. G. Chen, "Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 6, Jun 2006, pp. 673-688.

[13] T. C. Chen, Y. H. Chen, C. Y. Tsai, S. F. Tsai, S. Y. Chien, and L. G. Chen, "2.8 to 67.2mW low-power and power-aware H.264 encoder for mobile applications," in

Proc. IEEE VLSI Circuits Symp., Dig. Tech. Papers, Kyoto, Japan, Jun. 2007, pp. 222-223.

[14] T. C. Chen, Y. H. Chen, S. F. Tsai, S. Y. Chien, and L. G. Chen, “Fast algorithm and architecture design of low-power integer motion estimation for H.264/AVC,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 5, May 2007, pp. 568-577.

[15] T. C. Chen, Y. W. Huang, and L. G. Chen, “Fully utilized and reusable architecture for fractional motion estimation of H.264/AVC,” in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, vol. 4, May 2004, pp. 9-12.

[16] Y. H. Chen, T. C. Chen, C. Y. Tsai, S. F. Tsai, and L. G. Chen, “Algorithm and architecture design of power-oriented H.264/AVC baseline profile encoder for portable devices,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 8, Aug. 2009, pp. 1118-1128.

[17] Y. K. Chen, E. Q. Li, X. Zhou, and S. Ge, “Implementation of H.264 encoder and decoder on personal computers,” *J. Vis. Commun. Image R.*, vol. 17, no. 2 , Apr. 2006, pp 509-532.

[18] Z. Chen, J. Xu, Y. He, and J. Zheng, “Fast integer-pel and fractional-pel motion estimation for H.264/AVC,” *J. Vis. Commun. Image R.*, vol. 17, no. 2, Apr. 2006, pp. 264-290.

[19] C. H. Cheung and L. M. Po, “A novel cross-diamond search algorithm for fast block motion estimation,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 12, Dec. 2002, pp. 1168-1177.

[20] N. M. Cheung, X. Fan, O. C. Au, and M. C. Kung, “Video coding on multicore graphics processors,” *IEEE Signal Processing Mag.*, vol. 27, no. 2, Mar. 2010, pp. 79-89.

[21] W. Choi and B. Jeon, “Hierarchical motion search for H.264 variable-block-size motion compensation,” *SPIE Opt. Eng.*, vol. 45, no. 1, Jan. 2006, pp. 1-9.

[22] Cisco, *Cisco Visual Networking Index: Forecast and Methodology, 2009-2014*, Jun. 2010, p. 17.

[23] Cisco, *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2009-2014*, Jun. 2010, p. 15.

[24] Coda Research Consultancy Ltd., *Worldwide Smartphone Sales Forecast to 2015*, May 2010, p. 36.

[25] L. F. Ding, W. Y. Chen, P. K. Tsung, T. D. Chuang, H. K. Chiu, Y. H. Chen, P. H. Hsiao, S. Y. Chien, T. C. Chen, P. C. Lin, C. Y. Chang, and L. G. Chen, “A 212 MPixels/s 4096 x 2160p multiview video encoder chip for 3D/quad HDTV applications,” in *Proc. IEEE Int. Solid-State Circuits Conf.*, Dig. Tech. Papers, San Francisco, CA, USA, Feb. 2009, pp. 154-155.

[26] L. F. Ding, W. Y. Chen, P. K. Tsung, T. D. Chuang, P. H. Hsiao, Y. H. Chen, H. K. Chiu, S. Y. Chien, and L. G. Chen, “A 212 Mpixels/s 4096 x 2160p multiview video encoder chip for 3D/quad full HDTV applications,” *IEEE J. Solid-State Circuits*, vol. 45, no. 1, Jan. 2010, pp. 46-58.

[27] L. Deng, W. Gao, M. Z. Hu, and Z. Z. Ji, “An efficient hardware implementation for motion estimation of AVC standard,” *IEEE Trans. Consumer Electron.*, vol. 51, no. 4, Nov. 2005, pp. 1360-1366.

[28] H. Everett, "Generalized Lagrange multiplier method for solving problems of optimum allocation of resources," *Oper. Res.*, vol. 11, no. 3, May-Jun 1963, pp. 399-417.

[29] Eyelytics, Inc., "Main/baseline profile HD H264 encoder FPGA/ASIC IP," Available online: http://www.eyelytics.com/download/H264Encoder.pdf, Eyelytics datasheet.

[30] X. Q. Gao, C. J. Duanmu, and C. R. Zou, "A multilevel successive elimination algorithm for block matching motion estimation," *IEEE Trans. Image Processing*, vol. 9, no. 3, Mar. 2000, pp. 501-504.

[31] M. Ghanbari, "The cross-search algorithm for motion estimation," *IEEE Trans. Commun.*, vol. 38, no. 7, Jul. 1990 pp. 950-953.

[32] H. Gharavi and M. Mills, "Blockmatching motion estimation algorithms - new results," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 37, no. 5, May 1990, pp. 649-651.

[33] S. W. Golomb, "Run-length encodings," *IEEE Trans. Inform. Theory*, vol. 12, no. 3, Jul. 1966, pp. 399-401.

[34] H265.net, Available online: http://www.h265.net/.

[35] W. J. Han, J. Min, I. K. Kim, E. Alshina, A. Alshin, T. Lee, J. Chen, V. Seregin, S. Lee, Y. M. Hong, M. S. Cheon, N. Shlyakhov, K. McCann, T. Davies, and J. H. Park, "Improved video compression efficiency through flexible unit representation and corresponding extension of coding tools," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 12, Dec. 2010, pp. 1709-1720.

[36] H. M. Hang, W. H. Peng, C. H. Chan, and C. C. Chen, "Towards the next video standard: high efficiency video coding," *Asia-Pacific Signal and Information Processing Association Annual Summit and Conf.*, Dec. 2010, pp. 609-618.

[37] B. G. Haskell and J. O. Limb, "Predictive Video Encoding using Measured Subjective Velocity," U.S. Patent 3 632 865, Jan. 1972.

[38] X. He, X. Fang, C. Wang, and S. Goto, "Parallel HD encoding on CELL," in *Proc. IEEE Int. Symp. Circuits Syst.*, Taipei, Taiwan, May 2009, pp. 1065-1068.

[39] F. J. Hens and J. M. Caballero, *Triple Play: Building the Converged Network for IP, VoIP and IPTV*. John Wiley & Sons Ltd, Chichester, UK, 2008, p. 416.

[40] B. K. P. Horn and B. G. Schunk, "Determining optical flow," *Artificial Intelligence*, vol. 17, 1981, pp. 185-203.

[41] S. Y. Huang, C. Y. Cho, and J. S. Wang, "Adaptive fast block-matching algorithm by switching search patterns for sequences with wide-range motion content," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 11, Nov. 2005, pp. 1373-1384.

[42] Y. W. Huang, C. Y. Chen, C. H. Tsai, C. F. Shen, and L. G. Chen, "Survey on block matching motion estimation algorithms and architectures with new results," *J. VLSI Signal Processing*, vol. 42, no. 3, Mar. 2006, pp. 297-320.

[43] Y. W. Huang, T. C. Chen, C. H. Tsai, C. Y. Chen, T. W. Chen, C. S. Chen, C. F. Shen, S. Y. Ma, T. C. Wang, B. Y. Hsieh, H. C. Fang, and L. G. Chen, "A 1.3TOPS H.264/AVC single-chip encoder for HDTV applications," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Dig. Tech. Papers, San Francisco, CA, USA, Feb. 2005, pp. 128-129.

[44] Y. W. Huang, B. Y. Hsieh, S. Y. Chien, S. Y. Ma, and L. G. Chen, “Analysis and complexity reduction of multiple reference frames motion estimation in H.264/AVC,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 4, Apr. 2006, pp. 507-522.

[45] Imagination Technologies, Ltd. “VXE382 video encoder IP core family,” Available online: http://www.imgtec.com/powervr/powervr-vxe.asp.

[46] Intel Corp., “Core i7-980X processor extreme edition,” Intel Product Information, Available online: http://ark.intel.com/Product.aspx?id=47932.

[47] Intel Corp., “Pentium 4 processor extreme edition,” Intel Product Information, Available online: http://ark.intel.com/Product.aspx?id=27489.

[48] International Technology Roadmap for Semiconductors (ITRS), 2009 Update: system drivers.

[49] ISO/IEC 11172-2, “Information technology - coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbits/s - part 2: video,” MPEG-1, *ISO*, 1993.

[50] ISO/IEC 13818-2, “Information technology - generic coding of moving pictures and associated audio information - part 2: video,” MPEG-2, *ISO*, 1996.

[51] ISO/IEC 14496-2, “Information technology - coding of audio-visual objects - part 2: visual,” MPEG-4, *ISO*, 1999.

[52] ITU-R Recommendation BT.601-6, “Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios,” *International Telecommunication Union*, 2007.

[53] ITU-T Recommendation G.992.5, “Asymmetric digital subscriber line (ADSL) transceivers - extended bandwidth ADSL2 (ADSL2+),” *International Telecommunication Union*, Jan. 2005.

[54] ITU-T (formerly CCITT) Recommendation H.120, “Codecs for videoconferencing using primary digital group transmission,” *International Telecommunication Union*, 1989.

[55] ITU-T Recommendation H.261, “Video codec for audiovisual services at p*x*64 kbits/s,” *International Telecommunication Union*, Mar. 1993.

[56] ITU-T Recommendation H.263, “Video coding for low bit rate communication,” *International Telecommunication Union*, Mar. 1996.

[57] ITU-T Recommendation H.264, “Advanced video coding for generic audiovisual services,” *International Telecommunication Union*, Mar. 2009.

[58] J. R. Jain and A. K. Jain, “Displacement measurement and its application in interframe image coding,” *IEEE Trans. Commun.*, vol. 29, no. 12, Dec. 1981 pp. 1799-1808.

[59] Joint Video Team Reference Software, ver. JM 17.0, Available online: http://iphome.hhi.de/suehring/tml/.

[60] JointWave, Inc., “Jointwave E760: H.264 encoder IP core,” Available online: http://www.jointwave.com/intro_doc/Jointwave%20H.264%20Encoder%20Brochure%20E760.pdf, JointWave datasheet.

[61] C. Kappler, *UMTS Networks and Beyond*. John Wiley & Sons, Chichester, UK, 2009, p. 386.

[62] L. J Karam, I. AlKamal, A. Gatherer, G. A. Frantz, D. V. Anderson, and B. L Evans, "Trends in multicore DSP platforms," *IEEE Signal Processing Mag.*, vol. 26, no. 6, Nov. 2009, pp. 38-49.

[63] M. Karczewicz, P. Chen, R. L. Joshi, X. Wang, W. J. Chien, R. Panchal, Y. Reznik, M. Coban, and I. S. Chong, "A hybrid video coder based on extended macroblock sizes, improved interpolation, and flexible motion representation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 12, Dec. 2010, pp. 1698-1708.

[64] N. Kehtarnavaz and M. N. Gamadia, *Real-time Image and Video Processing: from Research to Reality*. Morgan and Claypool Publishers, USA, 2006, p. 96.

[65] J. Kim and T. Park, "A novel VLSI architecture for full-search variable block-size motion estimation," *IEEE Trans. Consumer Electron.*, vol. 55, no. 2, May 2009, pp. 728-733.

[66] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion-compensated interframe coding for video conferencing," in *Proc. National Telecommunication Conf.*, New Orleans, LA, USA, 1981, pp. G5.3.1-5.3.5.

[67] H. Kruegle, *CCTV Surveillance: Analog and Digital Video Practices and Technology*. Elsevier, Burlington, MA, USA, 2007, p. 656.

[68] P. Kuhn, *Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999, p. 239.

[69] I. Kuon and J. Rose, *Quantifying and Exploring the Gap Between FPGAs and ASICs*. Springer, 2010, p. 180.

[70] J. Lainema and M. Karczewicz, "Skip mode motion compensation," *document JVT-C027*, Fairfax, VA, USA, May 2002, pp. 1-2.

[71] G. G. Lee, Y. K. Chen, M. Mattavelli, and E. S. Jang, "Algorithm/architecture co-exploration of visual computing on emergent platforms: overview and future prospects," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 11, Nov. 2009, pp. 1576-1587.

[72] J. Lee and K. Yoo, "Multi-algorithm targeted low memory bandwidth architecture for H.264/AVC integer-pel motion estimation," *in Proc. IEEE Int. Conf. Multimedia and Expo*, Hannover, Germany, Jun. 2008, pp. 701-704.

[73] J. B. Lee and H. Kalva, *The VC-1 and H.264 Video Compression Standards for Broadband Video Services*. Springer, 2009, p. 496.

[74] O. Lehtoranta, *Parallel Encoder Implementations for High Quality Video*. PhD Thesis, Tampere University of Technology, Finland, Jan. 2007.

[75] A. Leontaris, P. C. Cosman, and A. M. Tourapis, "Multiple reference motion compensation: a tutorial introduction and survey," *Foundations and Trends in Signal Processing*, vol. 2, no. 4, Apr. 2009, pp. 247-364.

[76] D. X. Li, W. Zheng, and M. Zhang, "Architecture design for H.264/AVC integer motion estimation with minimum memory bandwidth," *IEEE Trans. Consumer Electron.*, vol. 53, no. 3, Aug. 2007, pp. 1053-1060.

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

[78] Z. Li and A. M. Tourapis, "Motion estimation with entropy coding considerations in H.264/AVC," in *Proc. IEEE Int. Conf. on Image Process.*, San Diego, CA, USA, Oct. 2008, pp. 2140-2143.

[79] C. J. Lian, S. Y. Chien, C. P. Lin, P. C. Tseng, and L. G. Chen, "Power-aware multimedia: concepts and design perspectives," *IEEE Circuits Syst. Mag.*, vol. 7, no. 2, 2007, pp. 26-34.

[80] D. T. Lin and C. Y. Yang, "H.264/AVC Video encoder realization and acceleration on TI DM642 DSP," *Lecture Notes in Computer Science*, vol. 5414, Springer, Berlin, Heidelberg, Jan. 2009, pp. 910-920.

[81] Y. K. Lin, D. W. Li, C. C. Lin, T. Y. Kuo, S. J. Wu, W. C. Tai, W. C. Chang, T. S. Chang, "A 242 mW 10 mm^2 1080p H.264/AVC high profile encoder chip," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Dig. Tech. Papers, San Francisco, CA, USA, Feb. 2008, pp. 314-315.

[82] Y. K. Lin, C. C. Lin, T. Y. Kuo, and T. S. Chang, "A hardware-efficient H.264/AVC motion-estimation design for high-definition video," *IEEE Trans. Circuits Syst. I*, vol. 55, no. 6, Jul. 2008, pp. 1526-1535.

[83] B. Liu and A. Zaccarin, "New fast algorithms for the estimation of block motion vectors," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 3, no. 2, Apr. 1993, pp. 148-157.

[84] L. K. Liu and E. Feig, "A block-based gradient descent search algorithm for block motion estimation in video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no. 4, Aug. 1996, pp. 419-422.

[85] Z. Liu, L. Li, Y. Song, S. Li, S. Goto, and T. Ikenaga, "Motion feature and Hadamard coefficient-based fast multiple reference frame motion estimation for H.264," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, no. 5, May 2008, pp. 620-632.

[86] Z. Liu, Y. Song, M. Shao, S. Li, L. Li, S. Ishiwata, M. Nakagawa, S. Goto, and T. Ikenaga, "A 1.41W H.264/AVC real-time encoder SoC for HDTV1080p," in *Proc. IEEE VLSI Circuits Symp.*, Dig. Tech. Papers, Kyoto, Japan, Jun. 2007, pp. 12-13.

[87] Z. Liu, Y. Song, M. Shao, S. Li, L. Li, S. Ishiwata, M. Nakagawa, S. Goto, and T. Ikenaga, "HDTV1080p H.264/AVC encoder chip design and performance analysis," *IEEE J. Solid-State Circuits*, vol. 44, no. 2, Feb. 2009, pp. 594-608.

[88] S. Ma, W. Gao, and Y. Lu, "Rate-distortion analysis for H.264/AVC video coding and its application to rate control," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 12, Dec. 2005, pp. 1533-1544.

[89] D. Marpe, H. Schwarz, S. Bosse, B. Bross, P. Helle, T. Hinz, H. Kirchhoffer, H. Lakshman, T. Nguyen, S. Oudin, M. Siekmann, K. Sühring, M. Winken, and T. Wiegand, "Video compression using nested quadtree structures, leaf merging, and improved techniques for motion representation and entropy coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 12, Dec. 2010, pp. 1676-1687.

[90] L. Merritt and R. Vanam, "Improved rate control and motion estimation for H.264 encoder," in *Proc. IEEE Int. Conf. on Image Process.*, vol. 5, San Antonio, TX, USA, Sep. 2007, pp. 309-312.

[91] S. Mochizuki, T. Shibayama, M. Hase, F. Izuhara, K. Akie, M. Nobori, R. Imaoka, H. Ueda, K. Ishikawa, and H. Watanabe, "A 64 mW high picture quality

H.264/MPEG-4 video codec IP for HD mobile applications in 90 nm CMOS," *IEEE J. Solid-State Circuits*, vol. 43, no. 11, Nov. 2008, pp. 2354-2362.

[92] MPEG, " Vision, applications and requirements for high efficiency video coding (HEVC)," *ISO/IEC/JTC1/SC29/WG11 N11872*, Daegu, South Korea, Jan. 2011.

[93] K. M. Nam, J. S. Kim, R. H. Park, and Y. S. Shim, "A fast hierarchical motion vector estimation algorithm using mean pyramid," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, no. 4, Aug. 1995, pp. 344-351.

[94] Y. Neuvo, "Cellular phones as embedded systems," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Dig. Tech. Papers, vol. 1, San Francisco, CA, USA, Feb. 2004, pp. 32-37.

[95] V. G . Oklobdzija, *The Computer Engineering Handbook*, CRC Press, Boca Raton, FL, USA, 2002, p. 1408.

[96] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi, "Video coding with H.264/AVC: tools, performance, and complexity," *IEEE Circuits Syst. Mag.*, vol. 4, no. 1, 2004, pp. 7-28.

[97] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proc. IEEE*, vol. 96, no. 5, May 2008, pp. 879-899.

[98] D. C. Pham, T. Aipperspach, D. Boerstler, M. Bolliger, R. Chaudhry, D. Cox, P. Harvey, P. M. Harvey, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Pham, J. Pille, S. Posluszny, M. Riley, D. L. Stasiak, M. Suzuoki, O. Takahashi, J. Warnock, S. Weitzel, D. Wendel, and K. Yazawa, "Overview of the architecture, circuit design, and physical implementation of a first-generation Cell processor," *IEEE J. Solid-State Circuits*, vol. 41, no. 1, Jan. 2006, pp. 179-196.

[99] L. M. Po and K. Guo, "Transform-domain fast sum of the squared difference computation for H.264/AVC rate-distortion optimization," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 6, Jun. 2007, pp. 765-773.

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

[101] Qualcomm, Inc., "Snapdragon chipsets," Available online: http://www.qualcomm.com/snapdragon.

[102] K. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*. Academic Press, San Diego, CA, USA, 1990.

[103] Renesas Technology Corp., Available online: http://www.renesas.com/index.html.

[104] I. E. G. Richardson, *H.264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia*. John Wiley & Sons Ltd, Chichester, UK, 2003, p. 281.

[105] E. Salminen, T. Kangas, J. Riihimäki, V. Lahtinen, K. Kuusilinna, and T. D. Hämäläinen, "HIBI communication network for system-on-chip," *J. VLSI Signal Processing-Systems for Signal, Image, and Video Technology*, Springer, vol. 43, no. 2-3, Jun. 2006, pp. 185-205.

[106] P. Sangi, J. Heikkilä, and O. Silvén, "Selection of the Lagrange multiplier for block-based motion estimation criteria," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, vol. 3, Montreal, Canada, May 2004, pp. 325-328.

[107] O. Silvén and K. Jyrkkä, “Observations on power-efficiency trends in mobile communication devices,” *EURASIP J. Embed. Syst.*, vol. 2007, pp. 1-10.

[108] W. Simpson and H. Greenfield, *IPTV and Internet Video: Expanding the Reach of Television Broadcasting*. Elsevier, Burlington, MA, USA, 2007, p. 241.

[109] SMPTE Draft Standard for Television, “Proposed SMPTE standard for television: VC-1 compressed video bitstream format and decoding process,” *SMPTE Technology Committee C24 on Video Compression Technology*, Aug. 2005.

[110] Y. Song, Z. Liu, T. Ikenaga, and S. Goto, “Ultra low-complexity fast variable block size motion estimation algorithm in H.264/AVC,” *in Proc. IEEE Int. Conf. Multimedia and Expo*, Beijing, China, Jul. 2007, pp. 376-379.

[111] C. Stiller and J. Konrad, “Estimating motion in image sequences,” *IEEE Signal Processing Mag.*, vol. 16, no. 4, Jul. 1999, pp. 70-91.

[112] Y. Su and M. T. Sun, “Fast multiple reference frame motion estimation for H.264/AVC,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 3, Mar. 2006, pp. 447-452.

[113] G. J. Sullivan and T. Wiegand, “Rate-distortion optimization for video compression,” *IEEE Signal Processing Mag.*, vol. 15, no. 6, Nov. 1998, pp. 74-90.

[114] O. Takahashi, C. Adams, D. Ault, E. Behnen, O. Chiang, S. R. Cottier, P. Coulman, J. Culp, G. Gervais, M. S. Gray, Y. Itaka, C. J. Johnson, F. Kono, L. Maurice, K. W. McCullen, L. Nguyen, Y. Nishino, H. Noro, J. Pille, M. Riley, M. Shen, C. Takano, S. Tokito, T. Wagner, and H. Yoshihara, “Migration of Cell Broadband Engine™ from 65nm SOI to 45nm SOI,” in *Proc. IEEE Int. Solid-State Circuits Conf.*, Dig. Tech. Papers, San Francisco, CA, USA, Feb. 2008, pp. 86-87.

[115] T. K. Tan, G. Sullivan, and T. Wedi, “Recommended simulation common conditions for coding efficiency experiments,” *document VCEG-AA10*, Nice, France, Oct. 2005, pp. 1-5.

[116] M. Tanimoto, “Overview of free viewpoint television,” *Signal Proces. Image Commun.*, vol. 21, no. 6, Jul. 2006, pp. 454-461.

[117] TechArp, “CPU performance comparison guide rev. 1.7,” Available online: http://www.techarp.com/.

[118] Texas Instruments, Inc., “TMS320DM642: video/imaging fixed-point digital signal processor,” Texas Instruments datasheet, Available online: http://focus.ti.com/lit/ds/symlink/tms320dm642.pdf.

[119] Texas Instruments, Inc., “TMS320DM368: digital media system-on-chip (DMSoC),” Texas Instruments datasheet, Available online: http://focus.ti.com/lit/ds/symlink/tms320dm368.pdf.

[120] J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, “A novel unrestricted center-biased diamond search algorithm for block motion estimation,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 8, no. 4, Aug. 1998, pp. 369-377.

[121] T. Toivonen, *Efficient Methods for Video Coding and Processing*. PhD Thesis, University of Oulu, Oulu, Finland, Dec. 2007.

[122] A. M. Tourapis, O. C. Au, and M. L. Liou, “Highly efficient predictive zonal algorithms for fast block-matching motion estimation,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 10, Oct. 2002, pp. 934-947.

[123] Y. K. Tu, J. F. Yang, and M. T. Sun, "Efficient rate-distortion estimation for H.264/AVC coders," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 5, May 2006, pp. 600-611.

[124] K. Ugur, K. Andersson, A. Fuldseth, G. Bjøntegaard, L. P. Endresen, J. Lainema, A. Hallapuro, J. Ridge, D. Rusanovskyy, C. Zhang, A. Norkin, C. Priddle, T. Rusert, J. Samuelsson, R. Sjöberg, and Z. Wu, "High performance, low complexity video coding and the emerging HEVC standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 12, Dec. 2010, pp. 1688-1697.

[125] D. Vatolin, D. Kulikov, and A. Parshin, *MPEG-4 AVC/H.264 Video Codecs Comparison*, The Graphics & Media Lab Video Group, Moscow, Russia, Apr 2010, p. 74.

[126] R. Verma and A. Akoglu, "A coarse grained and hybrid reconfigurable architecture with flexible NoC router for variable block size motion estimation," in *Proc. IEEE Int. Parallel & Distributed Processing Symposium*, Miami, FL, USA, Apr. 2008, pp. 1-8.

[127] J. Wang and G. Hua, "Implementing high definition video codec on TI DM6467 SoC," in *Proc. IEEE Int. SoC Conf.*, Newport Beach, CA, USA, Sep. 2008, pp. 193-196.

[128] T. C. Wang, Y. W. Huang, H. C. Fang, and L. G. Chen, "Parallel 4×4 2D transform and inverse transform architecture for MPEG-4 AVC/H.264," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, Bangkok, Thailand, May 2003, pp. 800-803.

[129] T. Wedi and H. G. Musmann, "Motion- and aliasing-compensated prediction for hybrid video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, Jul. 2003, pp. 577-586.

[130] C. Wei, H. Hui, T. Jiarong, L. Jinmei, and Min Hao, "A high-performance reconfigurable VLSI architecture for VBSME in H.264," *IEEE Trans. Consumer Electron.*, vol. 54, no. 3, Aug. 2008, pp. 1338-1345.

[131] C. Wei, H. Hui, L. J. Mei, M. Z. Gang, T. J. Rong, and M. Hao, "A novel reconfigurable VLSI architecture for motion estimation," in *Proc. IEEE Int. Conf. on ASIC*, Guilin, China, Oct. 2007, pp. 774-777.

[132] T. Wiegand, J. R. Ohm, G. J. Sullivan, W. J. Han, R. Joshi, T. K. Tan, and K. Ugur, "Special section on the joint call for proposals on high efficiency video coding (HEVC) standardization," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 12, Dec. 2010, pp. 1661-1666.

[133] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G. J. Sullivan, "Rate-constrained coder control and comparison of video coding standards," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, Jul. 2003, pp. 688-703.

[134] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, Jul. 2003, pp. 560-576.

[135] W. Wolf, "Multiprocessor system-on-chip technology," *IEEE Signal Processing Mag.*, vol. 26, no. 6, Nov. 2009, pp. 50-54.

[136] x264, Available online: http://www.videolan.org/developers/x264.html.

[137] S. Y. Yap and J. V. McCanny, "A VLSI architecture for variable block size video motion estimation," *IEEE Trans. Circuits Syst. II*, vol. 51, no. 7, Jul. 2004, pp. 384-389.

[138] X. Yi, J. Zhang, N. Ling, and W. Shang, "Improved and simplified fast motion estimation for JM," *document JVT-P021*, Poznan, Poland, Jul. 2005, pp. 1-20.

[139] YouTube, "YouTube fact sheet," Available online: http://www.youtube.com/t/press.

[140] S. Zafar, Y. Q. Zhang, and J. S. Baras, "Predictive block-matching motion estimation for TV coding - part I: inter-block prediction," *IEEE Trans. Broadcast.*, vol. 37, no. 3, Sep. 1991, pp. 97-101.

[141] J. Zhang, X. Yi, N. Ling, and W. Shang, "Context adaptive Lagrange multiplier (CALM) for rate-distortion optimal motion estimation in video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 6, Jun. 2010, pp. 820-828.

[142] L. Zhang and W. Gao, "Reusable architecture and complexity-controllable algorithm for the integer/fractional motion estimation of H.264," *IEEE Trans. Consumer Electron.*, vol. 53, no. 2, May 2007, pp. 749-756.

[143] Y. Q. Zhang and S. Zafar, "Predictive block-matching motion estimation for TV coding - part II: inter-frame prediction," *IEEE Trans. Broadcast.*, vol. 37, no. 3, Sep. 1991, pp. 102-105.

[144] C. Zhu, X. Lin, and L. P. Chau, "Hexagon-based search pattern for fast block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 5, May 2002, pp. 349-355.

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

Publications

Publication 1

J. Vanne, E. Aho, T. D. Hämäläinen, and K. Kuusilinna, "A high-performance sum of absolute difference implementation for motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 7, Jul. 2006, pp. 876-883.

A High-Performance Sum of Absolute Difference Implementation for Motion Estimation

Jarno Vanne, Eero Aho, Timo D. Hämäläinen, and Kimmo Kuusilinna

***Abstract*—This paper presents a high-performance sum of absolute difference (SAD) architecture for motion estimation, which is the most time-consuming and compute-intensive part of video coding. The proposed architecture contains novel and efficient optimizations to overcome bottlenecks discovered in existing approaches. In addition, designed sophisticated control logic with multiple early termination mechanisms further enhance execution speed and make the architecture suitable for general-purpose usage. Hence, the proposed architecture is not restricted to a single block-matching algorithm in motion estimation, but a wide range of algorithms is supported. The proposed SAD architecture outperforms contemporary architectures in terms of execution speed and area efficiency. The proposed architecture with three pipeline stages, synthesized to a 0.18-μm CMOS technology, can attain 770-MHz operating frequency at a cost of less than 5600 gates. Correspondingly, performance metrics for the proposed low-latency 2-stage architecture are 730 MHz and 7500 gates.**

***Index Terms*—Early termination mechanism, motion estimation, sum of absolute difference (SAD), SAD architecture.**

I. Introduction

BLOCK-BASED motion estimation searches for the best matching block between the current and reference macroblocks (MBs). For the operation, the sum of absolute differences (SAD) is one of the most frequently employed criteria [1], [2]. As a result, motion estimation produces a motion vector (MV), which represents the motion of the MB.

Well known full-search (FS) is the simplest, but the most computation-intensive block-matching algorithm (BMA). To decrease the computational complexity, numerous optimized search algorithms have been developed including three-step search (TSS) [3] and new diamond search (DS) [4]. However, motion estimation still clearly dominates the whole encoding process. With DS algorithm, it is shown to be near 50% of all the complexity in an MPEG-4 video encoder [5]. Since a major part of motion estimation is pure SAD computation, it is well motivated to focus on SAD architectures.

An optimal hardware realization for SAD computation varies for different BMAs. Systolic architectures are a good match for FS due to simplicity and regularity [2], [6], but not for irregular BMAs including DS. Architectures, which are better tailored for irregular BMAs, are considered in [6]–[11].

Manuscript received May 9, 2005; revised September 9, 2005 and December 22, 2005. This work was supported in part by the Academy of Finland under Grant 104487, Tampere Graduate School in Information Science and Engineering, Nokia Foundation, Emil Aaltonen Foundation, Heikki and Hilma Honkanen Foundation, and HPY Foundation. This paper was recommended by Associate Editor L.-G. Chen.

J. Vanne, E. Aho, and T. D. Hämäläinen are with the Institute of Digital and Computer Systems, Tampere University of Technology, FI-33101 Tampere, Finland (e-mail: jarno.vanne@tut.fi).

K. Kuusilinna is with Nokia Research Center, FI-33721 Tampere, Finland.

Digital Object Identifier 10.1109/TCSVT.2006.877150

Compared to referenced SAD architectures, our proposal includes modified elements, which process arithmetic operations in a novel and efficient way. In addition, the proposed architecture provides several early termination mechanisms and sophisticated SAD computation control.

The rest of the paper is organized as follows. Section II describes the related work with SAD architectures supporting irregular BMAs. Section III considers the theory related to the proposed SAD algorithm. Section IV proposes our novel SAD architecture unit by unit. Performance comparisons between contemporary SAD architectures and our architecture are shown in Section V. Section VI concludes the paper.

II. Previous Work

This paper merely concentrates on SAD implementations. At the current block location (x, y), the SAD criterion is defined as

$$\mathrm{SAD}(x,y,i,j) = \sum_{l=0}^{M-1} \sum_{k=0}^{N-1} |A_{(x+l,y+k)} - B_{(x+i+l,y+j+k)}| \tag{1}$$

where $A_{(x+l,y+k)}$ and $B_{(x+i+l,y+j+k)}$ indicate pixels of the current block and the reference frame, respectively. The size of the block is $M \times N$ and SAD computation is performed in the search area location (i, j) which is the displacement of the candidate block compared to the current block. The candidate block yielding the minimum SAD value determines $\mathrm{MV} = (i, j)$ for the location (x, y).

SAD architecture can functionally be divided into three stages: absolute difference calculation, accumulation of absolute differences, and minimum SAD determination.

A. Absolute Difference Calculation

The purpose of the first stage is to calculate absolute differences between the candidate and current MB pixels.

Vassiliadis *et al.* in [7] introduce a unit customized for detecting and inverting the smaller one of the pixel values. This paper refers to the unit as *Vassiliadis' smaller operand inverter*. The unit calculates inversion with one's complement arithmetic. Hence, to obtain finally a proper two's complement SAD value, the introduced inversion error has to be compensated afterwards by an additional *correction term*.

Another absolute difference unit is presented by Jehng *et al.* in [6]. The unit produces an absolute difference with a one's complement adder surrounded by a few logic gates. The implementation is called *Jehng's absolute difference unit*.

The third conventional absolute difference unit is presented by Chen *et al.* in [10]. It is an improved embodiment of Jehng's unit and it is referred to as *Chen's absolute difference unit*. In the unit, the low performance end-around carry chain in one's complement adder is removed and the possible one-bit error is compensated later by a *correction bit*.

B. Accumulation of Absolute Differences

In the second stage, the produced absolute differences are accumulated. Parallelization that is exploited in the pixel accumulation can vary from purely serial to fully parallel.

Jehng *et al.* in [6] present an accumulation unit to be used with Jehng's absolute difference units. In this paper, the adder tree implementation is called *Jehng's adder tree unit*.

Despite the additional correction bits, Chen's absolute difference units are also well suited for the adder tree. The correction bit accumulation is implemented by substituting the correction bits to the carry-in inputs of the adders in the tree. The modified tree is referred to as *Chen's adder tree unit* [8].

The accumulation of absolute differences can also be implemented with *CSA tree unit* [12], [13]. The carry–save adder (CSA) tree compresses the incoming absolute differences to carry and sum vectors. The carry propagation is performed during the last stage with a fast adder. Designing the CSA tree structure with more complex calculation elements [14] than full- and half adders is out of the scope of this paper.

Chen *et al.* in [9] present a recursive CSA tree for SAD computation. It is hereafter called *Chen's compression array unit*. Besides absolute differences and correction bits, Chen's compression array is capable of compressing previously compressed carry and sum vectors which are fed back to the array. Contrary to the accumulation units presented above, Chen's compression array produces two output vectors which are added together in the minimum SAD determination stage.

Vassiliadis' smaller operand inverters could also be coupled to the presented four accumulation units. However, each Vassiliadis' unit produces two output values and a common correction term. Therefore, the used accumulation unit has to be extended for $2u+1$ input values in order to complete two's complement partial SAD value for u pixels in one stage.

C. Minimum SAD Determination

Typically, conventional architectures execute the minimum SAD determination stage in two successive phases. In the first phase, two partial SAD values are added together to compute the current SAD value, whereas the resulting SAD value is compared to the minimum SAD value in the second phase. To speed up the operation, Chen *et al.* [10] present an enhanced minimum SAD determination unit here called *Chen's MV determination unit*. It performs SAD value comparison without completing actual SAD values at all. SAD values are utilized for comparison purposes only and the MV is the output from the unit.

III. Proposed SAD Algorithm

Let us consider a single SAD operation performed between two $M \times N$ blocks. In this special case, locations (x,y) and (i,j) are fixed in (1). Therefore, (1) can be rewritten to

$$\mathrm{SAD} = \sum_{l=0}^{M-1}\sum_{k=0}^{N-1} |A_{l,k} - B_{l,k}| = \sum_{l=0}^{M-1}\sum_{k=0}^{N-1} \mathrm{ABS}_{l,k}. \quad (2)$$

Since all $A_{l,k}$ and $B_{l,k}$ are nonnegative n-bit numbers, all $\mathrm{ABS}_{l,k}$ can also be presented with n bits.

The proposed absolute difference calculation stage computes absolute differences according to Theorem 1.

Theorem 1: For a single operand pair $A, B \in [0, 2^n - 1]$, absolute difference $\mathrm{ABS} \in [0, 2^n - 1]$ can be expressed as

$$\mathrm{ABS} = |A - B| = \begin{cases} (A + B') + 1, & A > B \\ (A + B')' + 0, & A \le B \end{cases} \quad (3)$$

where $B' = 2^n - 1 - B$ and $(A+B')' = 2^n - 1 - (A + B')$ are one's complement representations of B and $(A + B')$, respectively.

Proof: For $A > B$,

$$A + B' = A + 2^n - 1 - B = |A - B| - 1 + 2^n. \quad (4)$$

Since $|A - B| \in [1, 2^n - 1]$, $(A + B') \in [2^n, 2^{(n+1)} - 2]$, i.e., $A + B'$ generates always a carry (2^n). Hence, operand 2^n can be ignored in (4) and 1 has to be added to $A + B'$ in order to yield the correct result $|A - B|$.

For $A \le B$, $(A + B') \in [0, 2^n - 1]$ and

$$A + B' = A + 2^n - 1 - B = 2^n - 1 - |A - B| \quad (5)$$

$$\Leftrightarrow (A + B')' = |A - B|. \quad (6)$$

As shown in (5) and (6), the correct result in this case is obtained by taking one's complement of $A + B'$. □

Let us assume that the proposed absolute difference calculation stage completes simultaneously M absolute differences, i.e., $\mathrm{ABS}_l = |A_l - B_l|$ for $l = 0, 1, \ldots, M-1$ are computed in parallel. Hence, M absolute differences can also be accumulated in parallel. Without loss of generality, let P out of M accumulated absolute differences be $(A > B)$ cases and $(M-P)$ absolute differences be $(A \le B)$ cases. The accumulation yields a SAD value $(\mathrm{SAD}M)$ for M absolute differences as

$$\begin{aligned}\mathrm{SAD}M &= \sum_{l=0}^{M-1} \mathrm{ABS}_l \\ &= \sum_{l=0}^{P-1} ((A_l + B'_l) + 1) + \sum_{l=P}^{M-1} ((A_l + B'_l)' + 0) \\ &= \sum_{l=0}^{P-1} 1 + \sum_{l=P}^{M-1} 0 + \sum_{l=0}^{P-1} (A_l + B'_l) + \sum_{l=P}^{M-1} (A_l + B'_l)'.\end{aligned} \quad (7)$$

Since addition of real numbers is an associative operation, the operands $((A_l + B'_l)', (A_l + B'_l), 1, 0)$ in (7) can be in arbitrary order.

To complete a SAD value (2) for $M \times N$ blocks, N $\mathrm{SAD}M_k$ values (for $k = 0, \ldots, N-1$) have to be computed and added together. As in [9], parallelism and time-multiplexed hardware reuse of the proposed accumulation stage are increased by utilizing a recursive implementation. Via recursion, a single $\mathrm{SAD}M_k$ value computation and the addition of it to the sum of the already-computed $\mathrm{SAD}M$ values $(= \mathrm{SAD}M_{k-1} + \cdots + \mathrm{SAD}M_1 + \mathrm{SAD}M_0)$ can be performed in parallel.

In practice, the proposed accumulation stage produces two addition terms called sum $(\mathrm{SAD\_S}_k)$ and carry $(\mathrm{SAD\_C}_k)$ vectors, where $\mathrm{SAD\_S}_k + \mathrm{SAD\_C}_k = \mathrm{SAD}_k = \mathrm{SAD}M_k + \cdots + \mathrm{SAD}M_1 + \mathrm{SAD}M_0$. The $\mathrm{SAD\_S}_k$ and $\mathrm{SAD\_C}_k$ values are recursively obtained as

$$\begin{aligned} &\mathrm{SAD\_S}_k + \mathrm{SAD\_C}_k \\ &\quad= \mathrm{SAD\_S}_{k-1} + \mathrm{SAD\_C}_{k-1} + \sum_{l=0}^{P-1} 1 + \sum_{l=P}^{M-1} 0 \\ &\qquad+ \sum_{l=0}^{P-1} (A_l + B_l') + \sum_{l=P}^{M-1} (A_l + B_l')' \end{aligned} \tag{8}$$

for $k = 0, 1, \ldots, N-1$ with $\mathrm{SAD\_S}_{-1} = \mathrm{SAD\_C}_{-1} = 0$. The complete SAD value equals $\mathrm{SAD}_{N-1} = \mathrm{SAD\_S}_{N-1} + \mathrm{SAD\_C}_{N-1}$.

In the proposed minimum SAD determination stage, $\mathrm{SAD\_S}_k$ and $\mathrm{SAD\_C}_k$ values are added together to complete SAD_k value. In parallel with the SAD_k computation, $\mathrm{SAD\_S}_k + \mathrm{SAD\_C}_k$ value is also compared against the current minimum SAD value (MIN_SAD). Theorem 2 determines a new minimum SAD value.

Theorem 2: A new minimum SAD value has been found, if

$$\mathrm{SAD\_S}_{N-1} + \mathrm{SAD\_C}_{N-1} + \mathrm{MIN\_SAD}'' < 2^{(n+q)} \tag{9}$$

i.e if the addition in (9) generates no carry. In (9), $\mathrm{MIN\_SAD}'' = 2^{(n+q)} - \mathrm{MIN\_SAD}$ is two's complement representation of MIN_SAD, $\mathrm{SAD\_S}_{N-1} + \mathrm{SAD\_C}_{N-1} \le 2^{(n+q)}$, $\mathrm{MIN\_SAD} \le 2^{(n+q)}$, and $q = \lceil \log_2(M \times N \times (2^n - 1)) \rceil - n$.

Proof: The smaller one of the operands $\mathrm{SAD}_{N-1} (= \mathrm{SAD\_S}_{N-1} + \mathrm{SAD\_C}_{N-1})$ and MIN_SAD is detected with the following inequality checking:

$$\begin{aligned} &\mathrm{SAD\_S}_{N-1} + \mathrm{SAD\_C}_{N-1} < \mathrm{MIN\_SAD} && (10) \\ \Leftrightarrow &\mathrm{SAD\_S}_{N-1} + \mathrm{SAD\_C}_{N-1} < 2^{(n+q)} - \mathrm{MIN\_SAD}'' && (11) \\ \Leftrightarrow &\mathrm{SAD\_S}_{N-1} + \mathrm{SAD\_C}_{N-1} + \mathrm{MIN\_SAD}'' < 2^{(n+q)}. && (12) \end{aligned}$$

□

IV. Proposed Hardware Implementation

The proposed units are designed for the system in which $N = M = 16$ and $n = q = 8$. However, the units can be modified to support other bit widths and levels of parallelism as well.

A. Proposed Absolute Difference Unit

The *proposed absolute difference unit* is depicted in Fig. 1. The unit implements the functionality presented in (3) for the pixels $A(a_{7..0})$ and $B(b_{7..0})$, where bit widths are marked with MSB..LSB notation. If $A \le B$, the adder in Fig. 1. yields a result $S = A + B' = s_{8..0} < 2^n$(5). Hence, no carry bit

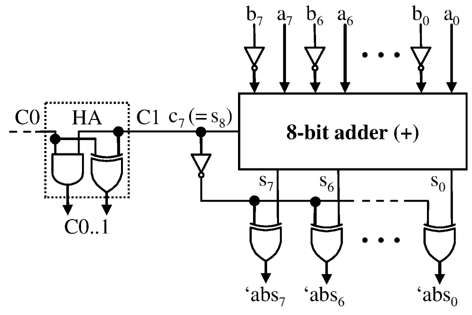


Fig. 1. Proposed absolute difference unit.

is generated (c_7 = '0') and the inverted c_7 bit is used to bit-invert (one's complement) the result $s_{7..0}$ to a correct absolute difference. Otherwise, $A > B$ and the result $S = A + B' = s_{8..0} \ge 2^n$ (4) implies that c_7 = '1'. In that case, the result $s_{7..0} = A - B - 1$ and c_7 bit can be forwarded as a correction bit. Notation $'\mathrm{ABS}l = ('\mathrm{abs}_{7..0})l$ $(l = 0, 1, \ldots, 15)$ signifies that the correction bit (= '0'/'1') is missing from the result.

As in Chen's implementation, a low-performance end-around carry chain used in Jehng's absolute difference unit is eliminated in the proposed unit. However, contrary to Chen's unit, the carry-in of the adder can also be removed in the proposed implementation. In addition, the structure of the proposed unit enables immediate correction bit addition without increased delay. As depicted in Fig. 1, the correction bits (C0 and C1) of the two adjacent absolute difference units are added together with a half adder (HA) in parallel with output XORs of the units. The generated 2-bit correction bit vectors (C0..1, ..., C14..15) are well suited to the proposed compression array in the subsequent stage.

B. Proposed Compression Array Unit

The *proposed compression array unit* presented in Fig. 2 implements the functionality of (8). It executes absolute difference and correction bit vector accumulations using separate CSA trees. A 6-stage CSA tree is designed to receive and compress all the $'\mathrm{ABS}l$ values, as well as C0..1, $\mathrm{SAD\_S}_{k-1}$, and $\mathrm{SAD\_C}_{k-1}$ vectors. A 3-stage CSA tree is targeted for partial accumulation of the other correction bit vectors. In all the CSAs, the outputs on the left produce partial carries, whereas the right-hand outputs are for partial sums.

Before a new SAD value accumulation can begin, $\mathrm{SAD\_S}_{k-1}$ and $\mathrm{SAD\_C}_{k-1}$ vectors are initialized to zero. In Fig. 2, an initialization structure controlled by a single signal (INIT) is depicted inside the square.

During operation, the sum and carry bits produced by the 3-stage CSA tree are inserted one by one into proper CSA inputs available in the 6-stage CSA tree. In Fig. 2, each letter (A to F) indicates a connection required between the two CSA trees. The bit stages are indicated with a bold font. In turn, LSBs of $\mathrm{SAD\_S}_{k-1}$ $(\mathrm{ss}_{7..0})$ and $\mathrm{SAD\_C}_{k-1}$ vectors $(\mathrm{sc}_{7..3})$ are immediately inserted to the first stage of the 6-stage CSA tree. The intermediate bits of these vectors ($\mathrm{ss}_{11..8}$ and $\mathrm{sc}_{11..8}$) are individually inserted into the tree, whereas the MSBs of these vectors ($\mathrm{ss}_{15..12}$ and $\mathrm{sc}_{15..12}$) are added in the final stage.

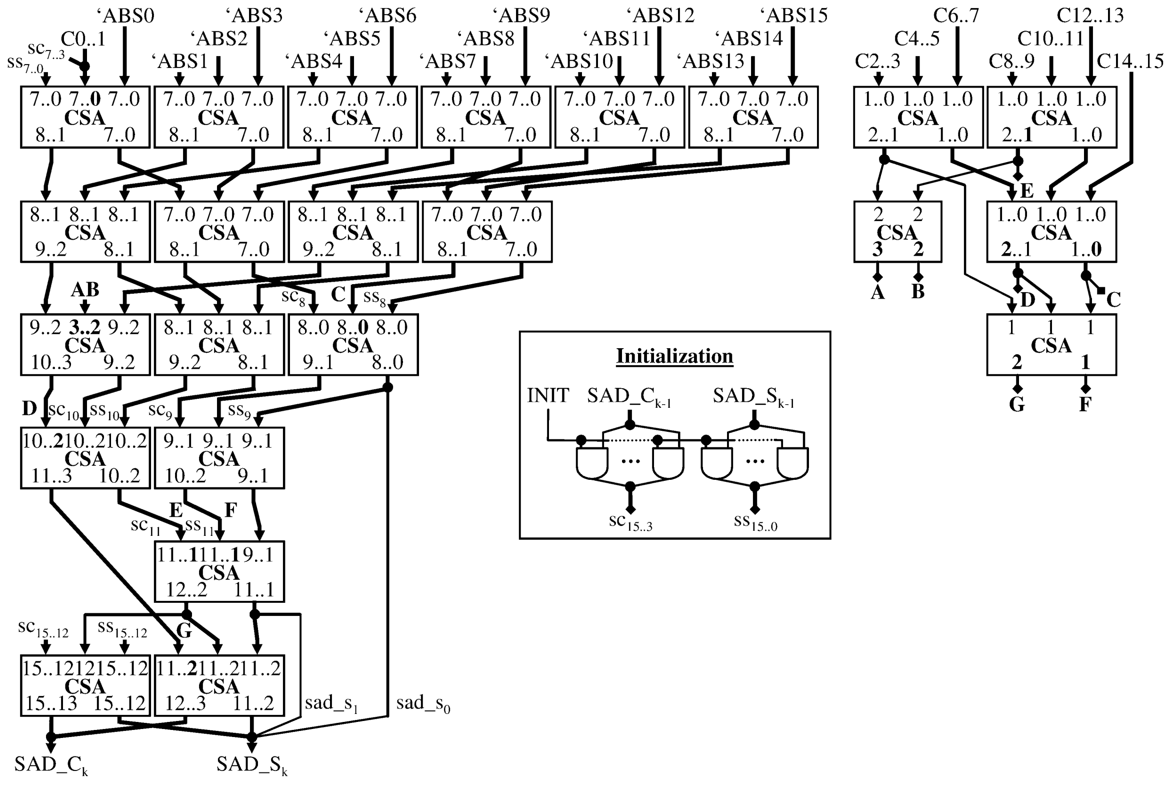


Fig. 2. Proposed compression array unit.

The proposed compression array reduces the number of operands at the earliest opportunity. Although this causes some area overhead, it also diminishes *common bit positions* involved in $\mathrm{SAD\_S}_k$ and $\mathrm{SAD\_C}_k$ vectors. In this context, a bit position is common for two vectors, if the bit position belongs to the index ranges of both vectors. Due to the enhanced correction bit accumulation, the proposed compression array is capable of compressing the number of common bit positions to 13 (i.e., $\mathrm{sad\_s}_{15..3}$ and $\mathrm{sad\_c}_{15..3}$). Without the correction bit accumulation, the proposed compression array could reduce the number of common bit positions to 12. Hence, the inclusion of the correction bits involves only one additional common bit position.

C. Proposed Minimum SAD Determination Unit

The *proposed minimum SAD determination unit* in Fig. 3 includes a novel structure for a parallel SAD value comparison and SAD value calculation. Contrary to Chen's approach, the proposed unit produces a complete minimum SAD value which can be exploited by other parts of the video encoder including INTRA/INTER mode decision [1]. The motion vector determination is excluded from this paper.

The unit performs the comparison presented in (9) for the operands $\mathrm{SAD\_S}_k$, $\mathrm{SAD\_C}_k$ and MIN_SAD. A 2-input OR gate manages the constant one bit addition which is required to convert the bit-inverted MIN_SAD value to two's complement representation. The CSA compression yields two difference vectors ($\mathrm{dif\_s}_{15..1}$ and $\mathrm{dif\_c}_{16..1}$) which are added together. The adder outputs only the most significant sum bit (s_{16}) which signifies the result of the inequality checking.

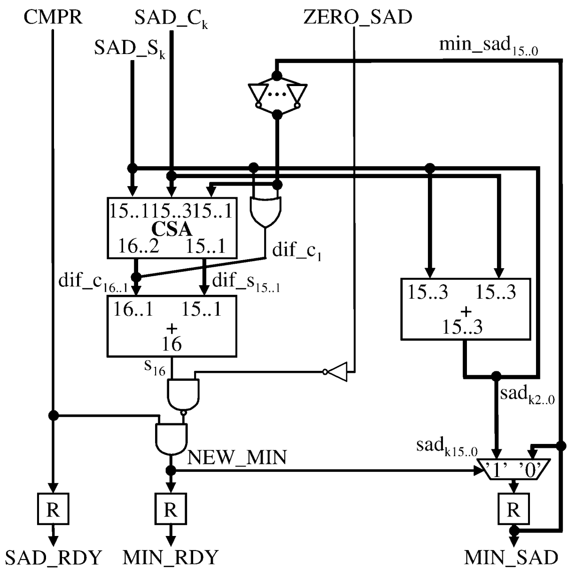


Fig. 3. Proposed minimum SAD determination unit without TM0-TM3.

In parallel with SAD value comparison, SAD_k value is calculated by adding $\mathrm{SAD\_S}_k$ and $\mathrm{SAD\_C}_k$ values together. A complete SAD value (SAD_{15}) computation requires 16 passes through the unit. Since an operationally decisive SAD value comparison is performed between SAD_{15} and MIN_SAD values, a specific control input (CMPR) indicating the proper time for comparison is only asserted during the final pass. If s_{16} = '0' after the final pass comparison,

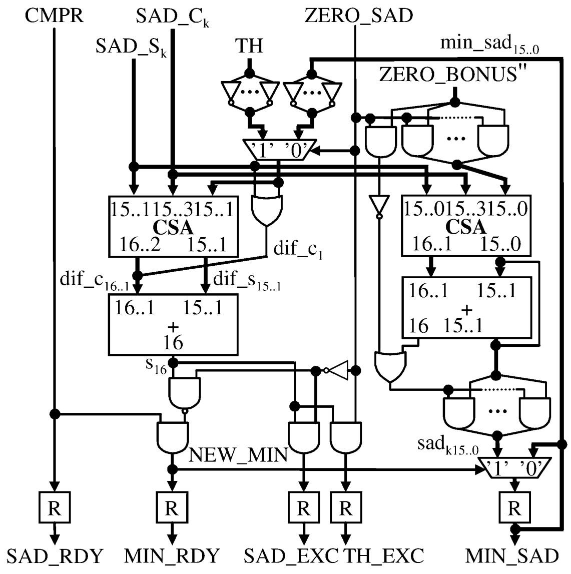


Fig. 4. Proposed minimum SAD determination unit with TM0-TM3.

$\mathrm{SAD}_{15} < \mathrm{MIN\_SAD}$ and SAD_{15} value is selected by a multiplexer ($\mathrm{NEW\_MIN} =$ '1') and stored in the registers as the new minimum SAD value.

The first SAD value computed for a zero motion vector (zero MV) is processed differently due to the absence of the minimum SAD value. This special case is indicated by one of the input control signals (ZERO_SAD). Regardless of the comparison result, the calculated SAD value for the zero MV is selected as the new minimum SAD value. In the considered cases, a new minimum SAD value activates two output control signals (SAD_RDY and MIN_RDY). Otherwise, the minimum SAD value is maintained in registers and only one control signal (SAD_RDY) is active after the comparison.

D. Implemented Early Termination Mechanisms

Early termination mechanisms eliminate unnecessary calculations in SAD computation. The proposed architecture implements four different early termination mechanisms (TM0-TM3). TM0-TM2 are also considered in [11], but the implementation techniques differ from the proposed one.

TM0 monitors the temporary SAD value accumulated by the compression array. The mechanism interrupts the ongoing SAD computation if the accumulated SAD value exceeds a predetermined threshold value. In the proposed compression array, a desired output bit of $\mathrm{SAD\_C}_k$ vector can be selected as an interrupt signal. As discussed in [11], the utilization of the threshold value can enable a narrower compression array implementation. However, the bit width reduction without over restricting SAD values causes diminutive area savings in the proposed compression array. In addition, the bit width reduction maintains the array height, so the delay of the array remains the same. Thus, a full width compression array is a reasonable solution. The other mechanisms are included in the proposed minimum SAD determination unit shown in Fig. 4.

TM1 monitors the most significant sum bit (s_{16}) of the adder. $s_{16} =$ '1' before or during comparison denotes that $\mathrm{SAD}_k > \mathrm{MIN\_SAD}$. The violation is indicated by a single control signal

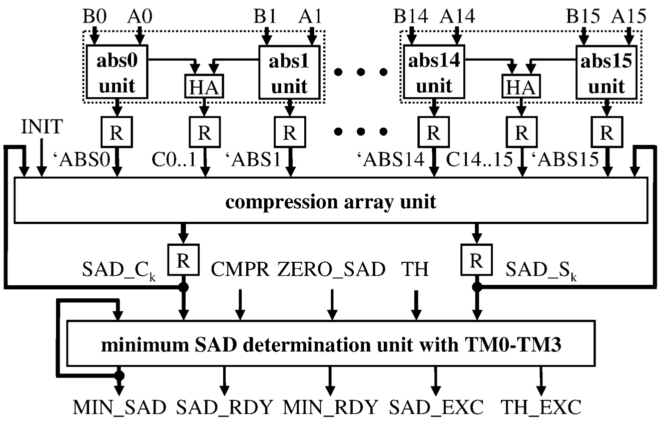


Fig. 5. Proposed 3-stage SAD architecture.

(SAD_EXC). Due to the absence of the valid minimum SAD value, the violation is ignored during SAD computation of the zero MV.

TM2 examines whether the current SAD value is under the predetermined threshold value. The mechanism is only supported within SAD computation for the zero MV. Threshold exceeding detection is accomplished by the logic used normally for the comparison of SAD_k and $\mathrm{MIN\_SAD}$ values (9). Hence, an additional multiplexer is required to select either $\mathrm{MIN\_SAD}$ value or threshold (TH) value for the comparison logic. In turn, an activated control signal (TH_EXC) indicates an exceeded TH value.

The proposed unit also supports the use of multiple threshold values. After the current TH value is exceeded, the subsequent, larger, TH value can be fed in during the next cycle. The one cycle delay between successive TH values prevents the examinations of the remaining larger TH values if the TH value is exceeded during the final cycle before the SAD value completion. Unexamined, larger, TH values may, in the worst case, induce unnecessary calculations, but a proper result is still returned. On the other hand, the detection of the exceeded threshold value could also be performed completely in parallel by duplicating the comparison part of the unit. Parallel threshold value detection would reduce the unit delay due to the avoided multiplexer and would remove the delay between successive threshold value examinations. However, the area increase would be relatively high due to the duplicated comparison logic.

TM3 favors SAD value for the zero MV. In practice, a predefined constant referred to as *zero bonus* is subtracted from SAD value for the zero MV. Here, the two's complemented zero bonus ($\mathrm{ZERO\_BONUS}''$) is assumed to be determined at design time, although it could also be one of the unit inputs being adjustable at run time. Before calculating SAD_k value for the zero MV, an additional CSA performs an addition between $\mathrm{SAD\_S}_k$, $\mathrm{SAD\_C}_k$, and $\mathrm{ZERO\_BONUS}''$. The SAD_k value is clipped to 0 if $\mathrm{SAD\_S}_k + \mathrm{SAD\_C}_k + (\mathrm{ZERO\_BONUS}'') < 0$.

E. Proposed Overall SAD Architecture

Fig. 5 depicts the high-level structure containing all the proposed units as well as all the discussed early termination mechanisms. The architecture is divided into three pipeline stages. The first pipeline stage is composed of 16 proposed absolute

TABLE I
AREA AND DELAY ESTIMATES FOR THE REFERENCED AND PROPOSED UNITS OF THE SAD ARCHITECTURE

Unit Label	Unit Description	Compatible Units	Delay(τ) RCA	Delay(τ) CLA	Area(CU) RCA	Area(CU) CLA
ABS_0	Vassiliadis' smaller operand inverter	-	18	8	2320	2776
ABS_1	Jehng's absolute difference unit	-	35	14	2832	6416
ABS_2	Chen's absolute difference unit	-	19	9	2353	5769
ABS_3	Proposed absolute difference unit	-	18	8	2305	4761
ACC_0	Tailored adder tree unit	ABS_0	31	37	3625	10273
ACC_1	CSA tree unit	ABS_0	39	30	3506	3660
ACC_2	Tailored compression array unit	ABS_0	25		3853	
ACC_3	Jehng's adder tree unit	ABS_1	27	27	1714	4827
ACC_4	Tailored CSA tree unit	ABS_1	32	24	1736	1890
ACC_5	Tailored compression array unit	ABS_1	19		1944	
ACC_6	Chen's adder tree unit	ABS_2	29	28	1834	6183
ACC_7	Tailored CSA tree unit	ABS_2	38	25	1960	2319
ACC_8	Chen's compression array unit	ABS_2	19		2148	
ACC_9	Tailored adder tree unit	ABS_3	29	28	1834	6183
ACC_10	Tailored CSA tree unit	ABS_3	36	25	2018	2325
ACC_11	Proposed compression array unit	ABS_3	19		2182	
MIN_0	Chen's MV determination unit	ACC_8	48	20	953	1147
MIN_1	Proposed MIN SAD unit	ACC_11	39	16	643	1292
MIN_2	Proposed MIN SAD unit (TM0-TM3)	ACC_11	42	19	956	1786

difference units. In turn, the second and third stages include the proposed compression array and the minimum SAD determination unit, respectively. The pipeline registers are depicted only after the first two stages, since the registers of the last stage are included in the minimum SAD determination unit.

V. PERFORMANCE ANALYSIS

This analysis is restricted to SAD implementations which operate 16 pixels in parallel.

A. *Theoretical Analysis*

As in [15], the area cost for s-input basic gates is assumed to be s cost-units (CUs), expect for XOR and XNOR gates having $2s$ CUs. In turn, the delay for all the gates is supposed to be 1 τ. Registers, interconnections, and fan-out/fan-in related issues are excluded from the calculations.

Table I tabulates the theoretical delay and area estimates for the evaluated units. The units are divided into three groups: absolute difference units ($\mathrm{ABS\_0} - \mathrm{ABS\_3}$), accumulation units ($\mathrm{ACC\_0} - \mathrm{ACC\_11}$), and minimum SAD determination units ($\mathrm{MIN\_0} - \mathrm{MIN\_2}$). Compatibility between successive units is tabulated in the third column, e.g., the ACC_11 unit is compatible with the ABS_3 unit.

Available 2-operand adders in the units are analyzed as being implemented with ripple-carry adders (RCAs) and carry-lookahead adders (CLAs). Theoretical performance estimates for CLAs are averages of 2-input and s-input gate versions of CLA. For the compression array units (ACC_2, ACC_5, ACC_8, and ACC_11), only single performance metrics are reported, since these units are analyzed as being implemented completely with CSAs.

Let us first consider the reported results of the absolute difference units. In all the cases, the area cost is calculated for 16 units. The proposed ABS_3 unit attains higher performance and evidently better area efficiency over Jehng's (ABS_1) and Chen's (ABS_2) implementations. Performance of the proposed ABS_3 unit is even equal to Vassiliadis' ABS_0 unit, which only detects the larger one of the input operands. Hence, the CLA-based ABS_3 unit is a desirable solution for high-performance applications, whereas the RCA-based ABS_3 unit is targeted for area efficient systems.

Three types of compatible accumulation units are evaluated per each absolute difference unit: adder tree, CSA tree, and compression array. For example, the ACC_0–ACC_2 units are particularly designed for the ABS_0 unit. Word "tailored" attached to the accumulation unit description denotes that the unit is modified from the original implementation in order to be compatible with the targeted absolute difference unit. In addition, the functionally essential initialization logic for the sum and carry vector is attached to Chen's ACC_8 unit. Although some of the adder tree and CSA tree configurations have higher area efficiency than the compression arrays, the ACC_2, ACC_5, ACC_8, and ACC_11 units evidently outperform respective approaches in terms of execution speed. Compared to the other compression arrays, the ACC_2 unit has significantly lower performance. In turn, although the ACC_5 unit is as fast as the ACC_8 and ACC_11 units, the performance of the compatible ABS_1 unit is not sufficient. Hence, the ACC_8 and ACC_11 units with the compatible absolute difference units are the most competitive approaches. Compared to the ACC_8 unit, the number of common bit positions involved in the output vectors is reduced in the proposed ACC_11 unit. The common bit position reduction enables the usage of narrower adders and registers in the following minimum SAD determination unit. Hence, the ACC_11 unit is the best solution since a diminutive area overhead of it is more than compensated by area savings in the minimum SAD determination unit.

The analyzed minimum SAD determination units are particularly tailored for the preferred ACC_8 and ACC_11 units. In addition, the logic required for the motion vector determination

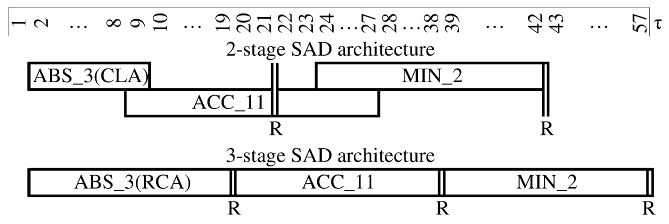


Fig. 6. Pipeline schemes for the proposed SAD architecture.

TABLE II
AREA AND DELAY ESTIMATES FOR THE MOST COMPETITIVE SAD ARCHITECTURES

SAD Architecture	Stages	Delay(τ)	Area(CU)
Chen's	2	24	9956
Proposed	2	21	9051
Proposed (TM0-TM3)	2	21	9569
Chen's	3	20	8112
Proposed	3	19	8071
Proposed (TM0-TM3)	3	19	8589

is excluded from Chen's $\mathrm{MIN\_0}$ unit. According to reported results, the proposed $\mathrm{MIN\_1}$ unit provides the fastest implementation. As shown by the $\mathrm{MIN\_2}$ unit, the presented early termination mechanisms can be included in the proposed unit very efficiently. In addition, delay increase in RCA-based implementations tends to be far more significant than area savings. Hence, the CLA-based $\mathrm{MIN\_2}$ unit is considered the best.

B. Pipelining

Fig. 6 depicts 2- and 3-stage pipelining schemes for the best overall SAD architecture $(\mathrm{ABS\_3} + \mathrm{ACC\_11} + \mathrm{MIN\_2})$. For illustration purposes, "as late as possible" scheduling is used in the schemes. The pipeline stages (R) are theoretically balanced in adder level granularity. In the proposed 2-stage scheme, the optimal location for the intermediate pipeline stage is after the fourth CSA stage of the compression array. In turn, the 3-stage pipelining follows the implementation in Fig. 5. The RCA-based ABS_3 units are adequately fast to be used in 3-stage schemes, but faster CLA-based units are required for 2-stage schemes.

The theoretical area and delay metrics for the proposed SAD architecture are tabulated in Table II, which also summarizes performance metrics for the most competitive SAD architectures: Chen's architecture $(\mathrm{ABS\_2} + \mathrm{ACC\_8} + \mathrm{MIN\_0})$ and the proposed one without early termination mechanisms $(\mathrm{ABS\_3} + \mathrm{ACC\_11} + \mathrm{MIN\_1})$. Both 2- and 3-stage pipelining schemes are analyzed. Registers are also included in the calculations in order to take pipelining overheads into account.

Compared to Chen's 2- and 3-stage architectures, the performance improvement of the proposed architectures is over 10% and 5%, respectively. The performance gap is wider with 2-stage architectures, since the proposed architectures can efficiently overlap the execution of successive units (Fig. 6). The inclusion of early termination mechanisms causes a slight area overhead in 3-stage architecture. In other cases, the proposed architecture is also more area efficient than Chen's approach.

The importance of the proposed optimizations is emphasized by the major role of SAD computation in video encoding. E.g., if DS algorithm is applied for full-pixel block-matching, encoding 16CIF (1408×1152) format at 30 frames per second requires approximately 3 million SAD operations per second. The proposed 3-stage architecture completes the 3 million SAD values 50 $\mathrm{M}\tau$ (5%) faster than corresponding Chen's architecture. The delay gap is widened to 150 $\mathrm{M}\tau$ (10%) between 2-stage architectures. Inclusion of fractional pixel estimation would increase the delay gap further.

TABLE III
SYNTHESIS RESULTS FOR THE PROPOSED SAD ARCHITECTURE

SAD Architecture	Stages	Freq(MHz)	Area(Cells)
		730	7509
Proposed (TM0-TM3)	2	575	5462
		420	4085
		774	5624
Proposed (TM0-TM3)	3	581	4377
		420	4078

C. Synthesis Results

The area and timing results based on logic synthesis are provided in Table III for the proposed 2- and 3-stage architectures $(\mathrm{ABS\_3} + \mathrm{ACC\_11} + \mathrm{MIN\_2})$. The applied technology is $0.18\ \mu\mathrm{m}$ CMOS process. The area values (gate count) are based on equivalent 2-input NAND gates, whereas the delay values represent the critical path in the pipelined architectures. Registers are included in the results.

With the selected technology, the proposed 3-stage SAD architecture is capable of operating at a frequency of 770 MHz, and costs the equivalent of 5600 NAND gates. Correspondingly, the 2-stage architecture can be clocked at 730 MHz at a cost of 7500 NAND gates. Lowering the operating frequency requirements could be used to decrease the silicon area.

To conclude the analysis, the proposed 3-stage architecture is a very good implementation for high throughput and area efficient systems in which three cycle latency is acceptable. In turn, the proposed 2-stage architecture is a feasible solution for low-latency systems as well as for 400-MHz systems and below. Very irregular block-matching algorithms and efficient use of early termination mechanisms favors the selection of 2-stage architecture.

VI. CONCLUSION

In a video encoder, motion estimation can consume up to half of the execution time. To reduce resources and computation power required for motion estimation, an optimized and efficient SAD architecture was presented in this paper. In addition, sophisticated control and several early termination mechanisms were presented for the architecture. The theoretical analysis of the paper illustrate that the proposed SAD architecture outperforms contemporary approaches. Furthermore, the synthesis results verify that the proposed architecture achieves very high execution speed at a cost of manageable silicon area.

REFERENCES

[1] V. Bhaskaran and K. Konstantinides, *Image and Video Compression Standards: Algorithms and Architectures*, 2nd ed. Amsterdam, The Netherlands: Kluwer Academic, 1997.

[2] P. Kuhn, *Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation*. Amsterdam, The Netherlands: Kluwer Academic, 1999.

[3] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion compensated interframe coding for video conferencing," in *Proc. Nat. Telecommun. Conf.*, New Orleans, LA, USA, 1981, pp. G5.3.1–5.3.5.

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

[5] O. Lehtoranta and T. D. Hämäläinen, "Complexity analysis of spatially scalable MPEG-4 encoder," in *Proc. Int. Symp. System-on-Chip*, Tampere, Finland, Nov. 2003, pp. 57–60.

[6] Y. S. Jehng, L. G. Chen, and T. D. Chiueh, "An efficient and simple VLSI tree architecture for motion estimation algorithms," *IEEE Trans. Signal Process.*, vol. 41, no. 2, pp. 889–900, Feb. 1993.

[7] S. Vassiliadis, E. A. Hakkennes, S. Wong, and G. G. Pechanek, "The sum-absolute-difference motion estimation accelerator," in *Proc. 24th Euromicro Conf.*, Västerås, Sweden, Aug. 1998, pp. 559–566.

[8] Q. Shu and H. Chen, "An efficient implementation of motion estimation algorithms," in *Proc. 4th Int. Conf. Solid-State and Integr. Circuit Technol.*, Oct. 1995, pp. 697–699.

[9] H. Chen and Q. Shu, "Apparatus for implementing a block matching algorithm for motion estimation in video image processing," U.S. Patent 5 864 372, Jan. 26, 1999.

[10] H. Chen and Q. Shu, "Adaptive block-matching motion estimator with a compression array for use in a video coding system," U.S. Patent 5 838 392, Nov. 17, 1998.

[11] D. Guevorkian, A. Launiainen, and P. Liuha, "Method for performing motion estimation in video encoding, a video encoding system and a video encoding device," U.S. Patent Appl. Publication 2003/0043911 A1, Mar. 6, 2003.

[12] C. Wallace, "A suggestion for parallel multipliers," *IEEE Trans. Electron. Comput.*, vol. EC-13, pp. 14–17, 1964.

[13] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford, U.K.: Oxford Univ. Press, 1999.

[14] L. Dadda, "Some schemes for parallel multipliers," *Alta Freq.*, vol. 34, pp. 349–356, May 1965.

[15] A. R. Omondi, *Computer Arithmetic Systems: Algorithms, Architecture and Implementations*. Hertfordshire, U.K.: Prentice Hall Int., 1994.

Publication 2

J. Vanne, E. Aho, T. D. Hämäläinen, and K. Kuusilinna, “A parallel memory system for variable block-size motion estimation algorithms,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, no. 4, Apr. 2008, pp. 538-543.

A Parallel Memory System for Variable Block-Size Motion Estimation Algorithms

Jarno Vanne, Eero Aho, Timo D. Hämäläinen, and Kimmo Kuusilinna

***Abstract*—This paper proposes an efficient parallel memory system for algorithms applied in fixed and variable block-size motion estimation (VBSME). The proposed system is implemented by a novel combination of two parallel memory architectures. The distribution of data among the memory modules is modified over contemporary approaches and the optimized address computation unit enables a rapid address generation for accessed memory locations. Furthermore, the introduced data permutation scheme organizes data efficiently for storage and retrieval. The proposed system enables up to $4\times$ speedup in data storage and retrieves data up to 55% faster for VBSME compared with the reference implementations. With a 0.18-μm CMOS technology, the proposed memory addressing and data permutation scheme can be clocked at 980 MHz operating frequency with a cost of less than 6 kgates. On FPGA, the system can operate at 200 MHz with less than 700 logic elements. The results show that the proposed system is applicable to real-time VBSME at HDTV resolution.**

***Index Terms*—Address computation, data permutation, motion estimation, parallel memory.**

I. Introduction

BLOCK-BASED motion estimation searches for the best matching block between the current and reference blocks. In traditional fixed block-size motion estimation (FBSME), the well-known full-search (FS) is the simplest, but the most data-intensive is block-matching algorithm (BMA). Numerous optimized BMAs [1], [2] developed for FBSME operate with significantly reduced memory bandwidth, but they also need more arbitrary memory accesses than FS. Furthermore, variable block-size motion estimation (VBSME) makes use of seven block sizes (4×4, 4×8, 8×4, 8×8, 8×16, 16×8, and 16×16), which introduces completely new data accessing and memory bandwidth requirements compared with FBSME.

Contemporary flexible memory systems targeted for FBSME can retrieve data for various BMAs but they are either inappropriate for VBSME [3]–[6] or suffer from limited data storage capability [7], [8]. In turn, the memories introduced for VBSME [2], [9] cannot support data accesses of optimized BMAs [10]. This is the motivation for designing an efficient memory system that supports several BMAs and block sizes.

In our proposal, the reference and current block data are stored in local on-chip memories which are implemented with two parallel memory architectures. To the best of our knowledge, the proposed *dual parallel memory architecture* is the first memory system that provides maximal data storage capability and supports multiple optimized BMAs applied both in FBSME and VBSME. Therefore, it is applicable to multiple video coding standards including MPEG-4, H.263, and H.264/AVC.

The remainder of this paper is organized as follows. Section II describes the related work. Section III introduces our new methods in on-chip memory data distribution, address computation, and data permutation. Section IV presents the implemented dual parallel memory architecture. Performance comparisons are shown in Section V. Section VI concludes the paper.

II. Related Work

To reduce the bandwidth from the external memories, the reference (search area) and the current block data are typically stored in local on-chip memories [2]–[6]. In FBSME, a word-addressable memory having word length of L pixels ($L \in \{4, 8, 16\}$) can be an efficient storage for the current block but not for the search area since optimized BMAs tend to address the search area in arbitrary (unaligned) locations.

Retrieving an unaligned L-pixel data with a single access demands a pixel-addressable memory where an L-pixel access can start from any pixel. The pixel-addressable memories can be implemented with parallel memories [11] in which a *module assignment function* (S) divides the memory space between the N memory modules and an *address function* (a) defines the addresses inside the memory modules. The pixels accessed in parallel can be defined by an *access format* (F) whereas a *scanning point* (r) assigns the placement of F in the memory. If every pixel accessed by F is physically located in a different memory module, F is *conflict-free* within S.

The pixel-addressable search area memories in [3]–[6] support only conflict-free row access format for L adjacent pixels, so they are incompatible with 4×4 blocks in VBSME, if $L > 4$. In [2] and [9], the VBSME compatibility for the row access format with $L > 4$ is enabled by partitioning a distortion computation unit to process a single L-pixel access as a group of parallel 4-pixel accesses. Since the search paths of adjacent blocks tend to diverge in optimized BMAs [10], such partitioned unit is only compatible with regular FS.

The block addressable memories in [7] and [12] can support conflict-free block access formats of 4×4 pixels when $N = 16$, but they only provide parallel storage of 4 pixels. That is ineffective in data storage, since data is normally transmitted from the external memories in the standard row-major order.

Instead, a memory system in [13] can be parametrized to allow arbitrary access of a 4×4 block and a row of 16 adjacent

Manuscript received July 17, 2006; revised October 24, 2006 and January 10, 2007. This work was supported in part by the Academy of Finland, Tampere Graduate School in Information Science and Engineering, Nokia Foundation, Emil Aaltonen Foundation, Heikki and Hilma Honkanen Foundation, and HPY foundation. This paper was recommended by Associate Editor K.-H. Tzou.

J. Vanne and T. D. Hämäläinen are with the Department of Computer Systems, Tampere University of Technology, FI-33101 Tampere, Finland (e-mail: jarno.vanne@tut.fi).

E. Aho and K. Kuusilinna are with the Nokia Research Center, FI-33721 Tampere, Finland.

Digital Object Identifier 10.1109/TCSVT.2008.918273

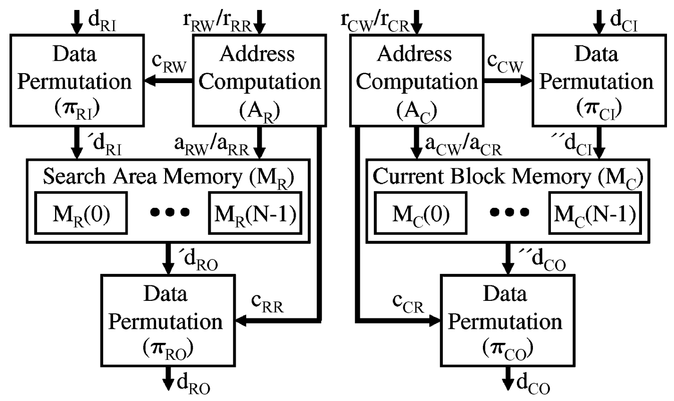


Fig. 1. Fundamental operating principle of the proposed memory system.

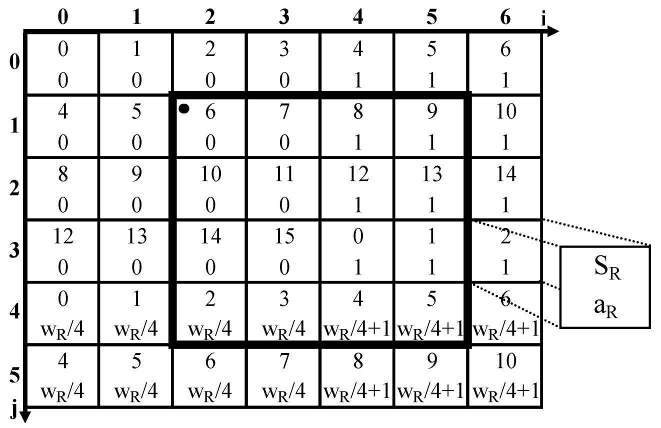


Fig. 2. Module assignment and address functions for the M_R memory.

pixels when $N = 16$. Its complex address computation unit is partitioned in a shared global address calculation logic and separate address completion logic for each memory module. In [14], the address computation of [13] is simplified, but an address routing circuit is required for the movement of addresses.

In the proposed dual parallel memory architecture, both memories adopt S and a from [13], a compact address computation unit is implemented without address routing circuit, and the amount of data permutation networks is the same as in the single parallel memory architectures in [13] and [14].

III. Proposed Memory System

Fig. 1 depicts an unoptimized block diagram of the proposed system. It consists of two parallel memory architectures: one for the search area and the other for the current block data.

In order to store reference frame data (d_{RI}) to the M_R memory, d_{RI} along with the respective scanning point (r_{RW}) are delivered to the system. With r_{RW}, the A_R circuit produces control (c_{RW}) for the π_{RI} network and the memory write addresses (a_{RW}) for the permuted write data $('d_{RI})$. Storing current frame data (d_{CI}) in the M_C memory is performed correspondingly.

The system supplies search area data (d_{RO}) and current block data (d_{CO}) simultaneously for distortion computation. In order to retrieve requested data from the memories, both the A_R and A_C circuits calculate read addresses (a_{RR} and a_{CR}) from the respective scanning points (r_{RR} and r_{CR}). The accessed search area data $('d_{RO})$ and current block data $(''d_{CO})$ are permuted with the π_{RO} and π_{CO} networks which are controlled (c_{RR} and c_{CR}) by the A_R and A_C circuits, respectively.

A. M_R Memory Organization

Let the design-time configurable size of the M_R memory be $w_R \times h_R$, where the dimensions w_R and h_R are multiples of 16. They define the memory area in i- j-directions, respectively.

To permit efficient row-major ordered data storage, the M_R memory supports a row access format (F_{RW}) for 16 adjacent pixels. Filling the M_R memory along 16-pixel wide vertical strips requires a nonoverlapping covering placement set

$$P_{RW}(S_R, F_{RW}) = \{(i_{RW}, j_{RW}) | i_{RW} \bmod 16 = 0 \wedge i_{RW} \in [0, w_R - 1] \wedge j_{RW} \in [0, h_R - 1]\} \quad (1)$$

for F_{RW}. In (1), the modulo condition implies the allowed conflict-free i coordinate values for $r_{RW}(i_{RW}, j_{RW})$.

Since a row access format is not optimal for VBSME, the M_R memory supports a 4×4 block access format (F_{RR}) for the data retrieval. To support arbitrary accesses of optimized BMAs, F_{RR} requires an unrestricted placement set

$$P_{RR}(S_R, F_{RR}) = \{(i_{RR}, j_{RR}) | i_{RR} \in [0, w_R - 1] \wedge j_{RR} \in [0, h_R - 1]\}. \quad (2)$$

The linear module assignment function providing (1) for F_{RW} and (2) for F_{RR} is

$$S_R(i, j) = (i + 4j) \bmod 16. \quad (3)$$

The admissible address function for $S_R(i, j)$ is

$$a_R(i, j) = a_R(i, 0) + a_R(0, j) = \left\lfloor \frac{i \bmod w_R}{4} \right\rfloor + \frac{w_R}{4} \times \left\lfloor \frac{j \bmod h_R}{4} \right\rfloor \quad (4)$$

where $a_R(i, 0)$ and $a_R(0, j)$ are the mutually orthogonal row and column address components of $a_R(i, j)$, respectively. In (4), $\bmod w_R$ and $\bmod h_R$ represent the circular property of the M_R memory. They make the M_R memory compatible with the search area reuse schemes presented, e.g., in [6] and [8].

The M_R memory is implementable with 16 1-pixel-wide modules $(M_R(0), \ldots, M_R(15))$. Fig. 2 illustrates a part (7×6 pixels) of the M_R memory. An example location for F_{RR} is indicated with a square and $r_{RR}(i_{RR}, j_{RR}) = (2, 1)$ marked with a dot "•" is located in the memory module 6 with the address 0.

B. M_C Memory Organization

Let the size of the M_C memory be $16 \times h_C$, where h_C is multiple of 16. Like the M_R memory, the M_C memory supports a 16-pixel-wide row access format (F_{CW}) for data storage and 4×4 block access format (F_{CR}) for data retrieval.

Since 4×4 pixel block boundaries in the M_C memory are not exceeded when executing typical BMAs, the M_C memory is implementable with four 4-pixel-wide modules $(M_C(0), \ldots, M_C(3))$ instead of 16 1-pixel-wide modules.

In the M_C memory, the placement sets for F_{CW} and F_{CR} are

$$P_{CW}(S_C, F_{CW}) = \{(i_{CW}, j_{CW}) | i_{CW} = 0 \wedge j_{CW} \in [0, h_C - 1]\} \tag{5}$$

$$P_{CR}(S_C, F_{CR}) = \{(i_{CR}, j_{CR}) | i_{CR} \in [0,3] \wedge j_{CR} \bmod 4 = 0 \wedge j_{CR} \in [0, h_C - 1]\}. \tag{6}$$

The restrictions in (5) and (6) simplify the module assignment and address functions which are defined as

$$S_C(i,j) = (i+j) \bmod 4 \tag{7}$$

$$a_C(i,j) = i + 4 \times \left\lfloor \frac{j}{4} \right\rfloor. \tag{8}$$

C. Address Computation

The A_R circuit (Fig. 1) uses (3) and (4) to compute

$$a_{RR}(a_{RR}(0), \ldots, a_{RR}(15))$$

and

$$a_{RW}(a_{RW}(0), \ldots, a_{RW}(15))$$

for the pixels accessed by F_{RR} and F_{RW}. Correspondingly, the A_C circuit (Fig. 1) applies (7) and (8) in

$$a_{CR}(a_{CR}(0), \ldots, a_{CR}(3))$$

and

$$a_{CW}(a_{CW}(0), \ldots, a_{CW}(3))$$

computation. Since (1), (6), and (5) restrict the placements of F_{RW}, F_{CR}, and F_{CW}, the logic for a_{RW}, a_{CR}, and a_{CW} computations is negligible. Hence, only a_{RR} computation is considered here.

Let us determine an address $a_{RR}(K)$ for a specific memory module $M_R(K)$ ($K \in [0, 15]$) when $r_{RR}(i_{RR}, j_{RR})$ of F_{RR} is known. In the M_R memory, let a pixel at location (i_K, j_K) be stored in $M_R(K)$, i.e., $K = S_R(i_K, j_K)$. Since (3) is conflict-free with respect to F_{RR}, each of the 16 $(i_K, j_K) \in F_{RR}$ locations are mapped in distinct $M_R(K)$. Now, $a_{RR}(K)$ for a specific $M_R(K)$ can be individually computed, i.e., each $M_R(K)$ is properly addressed without additional address routings. According to (4), $a_{RR}(K) = a_R(i_K, j_K)$ is the sum of separable components

$$ai_{RR}(K) = a_R(i_K, 0) = \left\lfloor \frac{(i_{RR} + \Delta i_{RR}(K)) \bmod w_R}{4} \right\rfloor \tag{9}$$

$$aj_{RR}(K) = a_R(0, j_K) = \frac{w_R}{4} \times \left\lfloor \frac{(j_{RR} + \Delta j_{RR}(K)) \bmod h_R}{4} \right\rfloor \tag{10}$$

where $\Delta i_{RR}(K)$ and $\Delta j_{RR}(K)$ extend the placement of $r_{RR}(i_{RR}, j_{RR})$ stored in $S_R(r_{RR}) = (i_{RR} + 4j_{RR}) \bmod 16$ to the position $(i_K, j_K) \in F_{RR}$ stored in $M_R(K)$. The shape (4×4 pixels) of F_{RR} limits that $\Delta i_{RR}(K) \in [0,3]$ and $\Delta j_{RR}(K) \in [0,3]$.

TABLE I
ASSIGNING $\Delta j_{RR}(K)$ VALUES AS A FUNCTION OF $S_R(R_{RR})$ ($t \in [0,3]$)

$(S_R(r_{RR})+t) \bmod 16$	**G_t**			
	$\Delta j_{RR}(3-t)$	$\Delta j_{RR}(7-t)$	$\Delta j_{RR}(11-t)$	$\Delta j_{RR}(15-t)$
0-3	0	1	2	3
4-7	3	0	1	2
8-11	2	3	0	1
12-15	1	2	3	0

The linearity [11] of (3) guarantees that the displacement $(\Delta i_{RR}(K), \Delta j_{RR}(K))$ is constant in the whole scanning field for a specific $S_R(r_{RR}) \in [0, 15]$ and $M_R(K)$ pair ($K \in [0, 15]$). For example, $(\Delta i_{RR}(10), \Delta j_{RR}(10)) = (1, 2)$ is always valid for $S_R(r_{RR}) = 1$ (Fig. 2). Hence, the displacement of each $M_R(K)$ from a known $S_R(r_{RR})$ can be defined with static values at design time.

Let us now consider (10), where $\Delta j_{RR}(K)$ values could be separately predefined for each of the 256 combinations of $M_R(K)$ and $S_R(r_{RR})$. However, the proposed method is able to reduce the combinations to 64, since each $M_R(K)$ has always the same $\Delta j_{RR}(K)$ value within four successive values of $S_R(r_{RR})$. E.g., $\Delta j_{RR}(5) = 3$ when $S_R(r_{RR})$ is 6–9 (Fig. 2). Table I tabulates the unambiguously definable $\Delta j_{RR}(K)$ values for $M_R(0), \ldots, M_R(15)$. The modules are divided into the four groups G_t ($t \in [0,3]$), where $M_R(3)$, $M_R(7)$, $M_R(11)$, and $M_R(15)$ belong to the G_0, $M_R(2)$, $M_R(6)$, $M_R(10)$, and $M_R(14)$ belong to the G_1, etc. The depicted ranges of $S_R(r_{RR})$ are appropriate for the G_0 whereas the ranges of $S_R(r_{RR})$ used by the $G_1 - G_3$ are converted to the depicted ones by incrementing $S_R(r_{RR})$ by t. E.g., for $M_R(2)$ in G_1, Table I assigns that $\Delta j_{RR}(2) = 3$ when $S_R(r_{RR})$ is 3–6 since $(S_R(r_{RR}) + 1) \bmod 16$ is 4–7.

A similar approach could also be used to compute (9), but the regularities of (3) and (4) simplify (9) further. Since w_R is a multiple of 16 and $\Delta i_{RR}(K) \in [0,3]$, (9) can be rewritten as

$$ai_{RR}(K) = \left(\left\lfloor \frac{i_{RR}}{4} \right\rfloor + \left\lfloor \frac{i_{RR} \bmod 4 + \Delta i_{RR}(K)}{4} \right\rfloor \right) \bmod \frac{w_R}{4} = (ai_{RR}(S_R(r_{RR})) + \Delta ai_{RR}(K)) \bmod \frac{w_R}{4} \tag{11}$$

where $ai_{RR}(S_R(r_{RR})) = \lfloor i_{RR}/4 \rfloor$ is an address of $S_R(r_{RR})$ and $\Delta ai_{RR}(K) \in [0,1]$ is an offset of a specific $M_R(K)$. Due to (3) and (4), all the modules belonging to a same G_t have an equivalent $\Delta ai_{RR}(K)$ value that is directly derivable from $i_{RR} \bmod 4$.

D. Data Permutation Scheme

Fig. 3(a) presents a data flowchart for the system in Fig. 1. To store the row-major ordered $d_{RI}(d_{RI}(0), \ldots, d_{RI}(15))$ to the M_R memory, the π_{RI} network permutes d_{RI} according to (3). The permutation to attain $'d_{RI}('d_{RI}(0), \ldots, 'd_{RI}(15))$ is

$$'d_{RI}(k) = d_{RI}([k - S_R(r_{RW})] \bmod 16) \tag{12}$$

where $S_R(r_{RW}) = c_{RW}$ and $'d_{RI}$ is obtained by shifting d_{RI} by $S_R(r_{RW})$ pixels right and substituting the rightmost $S_R(r_{RW})$ pixels of d_{RI} as the leftmost $S_R(r_{RW})$

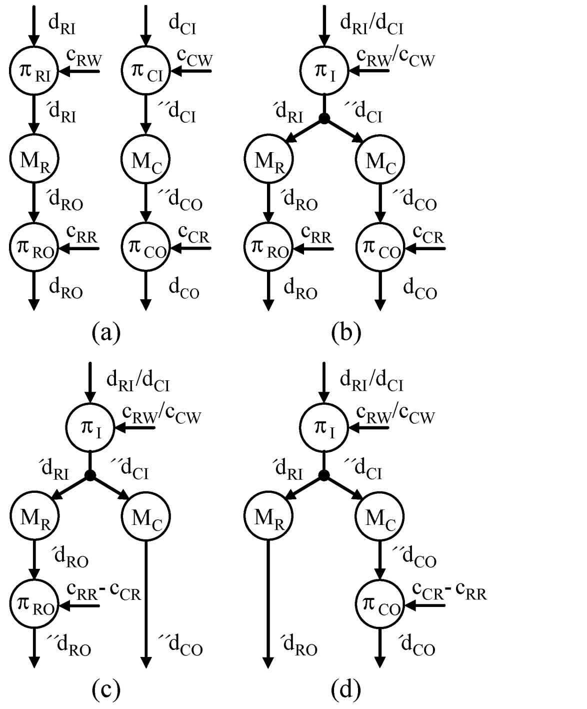


Fig. 3. Data flowcharts. (a) Basic. (b) Shared input. (c) Shared input and $'d_{RO}$ ordered as $''d_{CO}$. (d) Shared input and $''d_{CO}$ ordered as $'d_{RO}$.

pixels of $'d_{RI}$. Correspondingly, the π_{CI} network permutes $d_{CI}(d_{CI}(0), \ldots, d_{CI}(15))$ as

$$''d_{CI}(k) = d_{CI}\left([k - 4 \times S_C(r_{CW})] \bmod 16\right) \quad (13)$$

where $4 \times S_C(r_{CW}) = c_{CW}$.

Since $c_{RW} \in \{0,4,8,12\}$ and $c_{CW} \in \{0,4,8,12\}$, the π_{RI} and π_{CI} networks have equal functionality. Hence, a shared data permutation network (π_I) is used for d_{RI} and d_{CI} in Fig. 3(b). Sharing a data path between d_{RI} and d_{CI} causes performance penalty equal to the time required for filling the M_C memory. However, duplication of the data permutation network is avoided.

In Fig. 3(a) and (b), the π_{RO} and π_{CO} networks permute the retrieved pixels back to the original (row-major) order. However, the original order of pixels is unnecessary in distortion computation because only corresponding pixels are compared to each other. Hence, only $'d_{RO}$ or $''d_{CO}$ has to be permuted to the order of the other. Fig. 3(c) depicts a flowchart which permutes $'d_{RO}$ according to $''d_{CO}$, whereas $''d_{CO}$ is permuted according to $'d_{RO}$ in Fig. 3(d). The respective permutations are

$$''d_{RO}(k) = 'd_{RO}\left([k + S_R(r_{RR}) - S_C(r_{CR})] \bmod 16\right) \quad (14)$$
$$'d_{CO}(k) = ''d_{CO}\left([k + S_C(r_{CR}) - S_R(r_{RR})] \bmod 16\right) \quad (15)$$

where $S_R(r_{RR}) = c_{RR} \in [0,15]$ and $S_C(r_{CR}) = c_{CR} \in [0,15]$.

Due to this new idea, only one permutation network is required for the output data. The selection between the flowcharts in Fig. 3(c) and (d) depends on an application. In Fig. 3(c), the retrieved best match can be permuted appropriately for the next design level, whereas the computation complexity around the M_R and M_C memories is better balanced in Fig. 3(d).

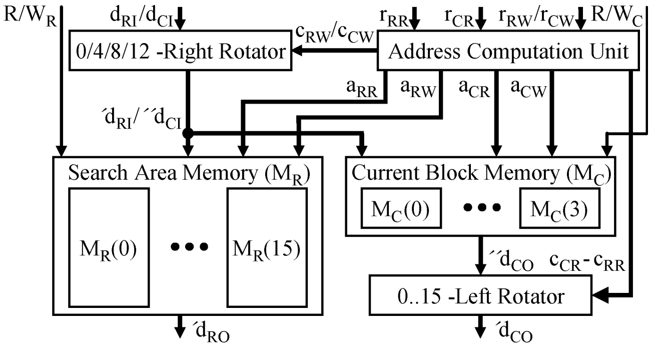


Fig. 4. Implemented memory system.

IV. Proposed Memory Implementation

Fig. 4 depicts the implemented memory system. It is based on Fig. 3(d). The M_R and M_C memories are controlled by read/write signals (R/W_R and R/W_C). The π_I and π_{CO} networks are implemented with a *0/4/8/12-Right Rotator* and a 0..15-*Left Rotator*, respectively. The A_R and A_C circuits (Fig. 1) are replaced with a single *Address Computation Unit* which includes separate units for a_{RR}, a_{RW}, a_{CR}, and a_{CW} decoding.

A. Address Computation Unit

Fig. 5 depicts an a_{RR} *Decode Unit.* Four adders compute 2 MSBs of $S_R(r_{RR})$ and its increments from $r_{RR}(i_{RR}, j_{RR})$. The functionality of Table I is mapped to four Δj_{RR} *Decoding Circuits*, which together consume only 20 basic logic gates. The circuits decode $\Delta j_{RR}(0), \ldots, \Delta j_{RR}(15)$ values that are used to generate $aj_{RR}(0), \ldots, aj_{RR}(15)$ values according to (10).

In Fig. 5, a Δi_{RR} *Decoding Circuit* produces $\Delta ai_{RR}(G_t)$ values for the $G_1 - G_3$ and $ai_{RR}(G_t)$ values are computed according to (11). Since $\Delta ai_{RR}(G_0)$ is always zero, no logic is used to generate $ai_{RR}(G_0)$. The Δi_{RR} *Decoding Circuit* is implemented with two basic logic gates.

Finally, the a_{RR} *Decode Unit* completes $a_{RR}(K)$ by adding the computed $ai_{RR}(G_t)$ and $aj_{RR}(K)$ values together (Fig. 5). The logic needed for a_{RW}, a_{CR}, and a_{CW} *Decode Units* is negligible.

B. Data Rotators

The 0..15-*Left Rotator* (Fig. 4) consists of 0/8-, 0/4-, 0/2-, and 0/1—rotator stages, whereas the 0/4/8/12-*Right Rotator* includes 0/8- and 0/4-rotator stages. In these 16-input, 16-output rotators, data is passed through a single $0/l$-stage without rotation or data is rotated by l pixels. A single $0/l$-stage is realized with an array of simplified 1-bit 2-to-1 multiplexers [13].

V. Performance Analysis

The first evaluation compares the proposed and contemporary flexible search area memories [3]–[7]. In addition, the implementations in [12]–[14] are evaluated as candidates for the search area memory. The memories are parametrized to $N = 16$, if N is not limited by the reference memory. A single

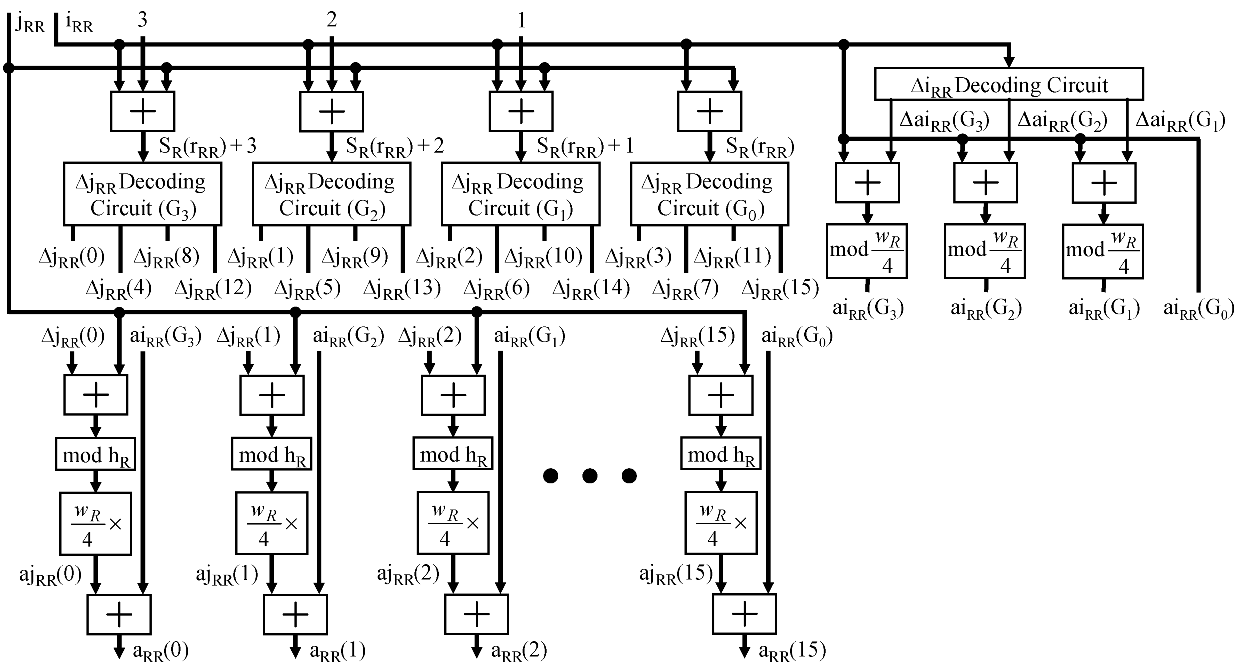


Fig. 5. a_{RR} *Decode Unit.*

TABLE II
NUMBER OF SEARCH AREA MEMORY ACCESSES PER SECOND

BMA	Operation	Million accesses/s								
		Gupta [3]	Gong [4]	Chao [5]	Tanskanen [6]	Jehng [7]	Kuzmanov [12]	Morrin [13]	Park [14]	Proposed
		N = 16	N = 16	N = 8	N = 8	N = 16	N = 16	N = 16	N = 16	N = 16
DS/ FBSME	Data storage	0.17	0.17	0.34	0.34	0.67	0.67	0.17	0.17	0.17
	Data retrieval	0.70	0.70	1.41	1.41	0.70	0.70	0.70	0.70	0.70
	Total	**0.87**	**0.87**	**1.74**	**1.74**	**1.38**	**1.38**	**0.87**	**0.87**	**0.87**
DS/ VBSME	Data storage	0.17	0.17	0.34	0.34	0.67	0.67	0.17	0.17	0.17
	Data retrieval	11.14	11.14	12.54	12.54	4.88	4.88	4.88	4.88	4.88
	Total	**11.31**	**11.31**	**12.88**	**12.88**	**5.56**	**5.56**	**5.05**	**5.05**	**5.05**

N-pixel data bus from external memories is assumed to be available for data storage. Reuse of search area data is also utilized.

Table II tabulates the average number of memory accesses per second when performing FBSME and VBSME with the DS algorithm [1]. The reported values are based on QCIF (176×144) size sequences "Container," "Foreman," "News," and "Silent" encoded at 30 frames/second (fps). The size of the search are is 48×48 pixels and the checking point counts of DS in the sequences are obtained from [10].

Compared with Gupta's [3] and Gong's [4] approaches, the proposed memory requires an equal number of accesses in FBSME, but consumes 55% less accesses in VBSME. The data accessing capabilities of Chao's [5] and Tanskanen's [6] memories are similar to Gupta's and Gong's approaches. However, they are restricted to $N = 8$.

Jehng's [7] and Kuzmanov's [12] memories perform data retrieval for FBSME and VBSME with a number of memory accesses that is equal to the proposed one. However, the proposed memory can store row-major ordered data four times faster than those approaches. The reductions in total memory access count in FBSME and VBSME are 37% and 9%, respectively. Without search area reuse, the respective reductions would be even higher: 53% and 19%. The speedup in data storage decreases the time the global data bus is reserved for motion estimation.

The proposed approach and those of Morrin [13] and Park [14] require an equal amount of accesses in all the examined cases. Next, these three implementations are further compared.

A. Address Computation Comparison

The proposed, Morrin's, and Park's approaches include two similar data rotators, but they implement address computation differently. Hence, only the proposed a_{RR} *Decode Unit* (Fig. 5) is compared with Morrin's and Park's address computation units which are parametrized at design time to support F_{RR} and F_{RW}. The units are configured for $w_R = h_R = 64$.

Table III tabulates the theoretical critical path delay and area estimates for the units. As in [15], the area cost for s-input basic gates is assumed to be s cost-units (CUs), expect for XOR and XNOR gates having $2s$ CUs. The delay for all the 2-input gates is assumed to be 1 τ. Registers, interconnections, and fan-out/fan-in related issues are not considered.

TABLE III
AREA AND DELAY ESTIMATES FOR THE ADDRESS COMPUTATION UNITS

Unit	$w_R \times h_R$	Delay(τ)	Area(CU)	Rotation(Hor / Ver)
Park [14]	64 x 64	20	3798	Yes / Yes
Morrin [13]	64 x 64	24	1361	Yes / Yes
$A6464_{RR}$	64 x 64	12	753	Yes / Yes
$A6464h_{RR}$	64 x 64	12	753	Yes / No
$A6448_{RR}$	64 x 48	14	849	Yes / Yes
$A6448h_{RR}$	64 x 48	12	753	Yes / No
$A4848_{RR}$	48 x 48	24	2227	Yes / Yes
$A4848h_{RR}$	48 x 48	17	1563	Yes / No

TABLE IV
ASIC AND FPGA SYNTHESIS RESULTS FOR THE PROPOSED MEMORY SYSTEM

Config	Memory	ASIC		FPGA	
	Size(KB)	Freq(MHz)	Area(Cells)	Freq(MHz)	Area(LEs)
6464	4.4	980	5749	202	685
6464h	4.4	980	5749	202	685
6448	3.3	893	5795	179	717
6448h	3.3	980	5749	202	685
4848	2.6	588	6528	168	880
4848h	2.6	719	6184	171	726

Compared with Park's and Morrin's units, the proposed unit $(\mathrm{A6464_{RR}})$ computes memory addresses for the used access format 1.7 and 2.0 times faster and consumes 80% and 45% less resources, respectively. The proposed unit is also configured for $w_R = 64$, $h_R = 48$ $(\mathrm{A6448_{RR}})$ and for $w_R = h_R = 48$ $(\mathrm{A4848_{RR}})$. Since the search area reuse typically requires only horizontally circular memory, the a_{RR} *Decode Units* supporting merely the horizontal (Hor) memory rotation $(\mathrm{A6464h_{RR}}$, $\mathrm{A6448h_{RR}}$, and $\mathrm{A4848h_{RR}})$ are also evaluated. Excluding a vertical (Ver) rotation is of importance when $w_R = h_R = 48$.

B. ASIC and FPGA Synthesis

The ASIC and FPGA results based on logic synthesis are tabulated in Table IV for the proposed system (Fig. 4). The configuration determines the dimensions of the M_R memory.

The technology for ASIC synthesis is 0.18-μm CMOS process. The area (cells) is based on equivalent 2-input NAND gates and the operating frequency represents the critical path in the pipelined systems. The ASIC-based system includes three pipeline stages. The registers and all the units except memory modules are included in the results. In the 6464, 6464h, and 6448h configurations, pipelining consumes 52% of all of silicon area, whereas the data rotators and address computation consume 40% and 8% of the area, respectively. In 4848 configuration, the respective metrics are 46%, 35%, and 19%.

The FPGA-based system is synthesized to Altera Stratix EP1S40F780C5 logic device. The memory modules are synthesized using the embedded memory modules on the FPGA. The system includes a balanced four-stage pipeline. All the configurations consume less than 3% of the FPGA resources.

With optimal 100% utilization, the proposed system clocked at 1 MHz can deliver data for DS in FBSME at QCIF size (real-time, 30 fps), whereas D1 (720×576) and HDTV (1280×720) resolutions need operating frequencies of 15 and 33 MHz, respectively. In VBSME, the respective metrics are 6, 84, and 185 MHz. Hence, the ASIC-based system coupled to an efficient distortion computation unit such as [16] meets the real-time requirements in all the examined resolutions. At HDTV resolution, the FPGA-based system targeted for VBSME presumably needs performance enhancement which can be achieved by increasing the pipeline depth, utilizing faster BMAs [10], or duplicating the memory system.

VI. CONCLUSION

This paper proposes a novel dual parallel memory architecture for FBSME and VBSME. Compared with the reference approaches, the proposed memory addressing doubles the performance and consumes almost half of the silicon area. In addition, the introduced data permutation scheme halves the number of permutation networks. Our proposal offers up to four times speedup in data storage and retrieves data up to 55% faster than the reference search area memories. It can process FBSME and VBSME in real-time at HDTV resolution.

REFERENCES

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

[2] P. Kuhn, *Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation*. Boston, MA: Kluwer, 1999.

[3] G. Gupta and C. Chakrabarti, "Architectures for hierarchical and other block matching algorithms," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, no. 6, pp. 477–489, Dec. 1995.

[4] D. Gong and Y. He, "A new programmable video signal processor for motion estimation and motion compensation," in *Proc. SPIE-VCIP*, San Jose, CA, Jan. 2001, vol. 4310, pp. 920–931.

[5] W. M. Chao, C. W. Hsu, Y. C. Chang, and L. G. Chen, "A novel hybrid motion estimator supporting diamond search and fast full search," in *Proc. IEEE Int. Symp. Circuits Syst.*, Phoenix–Scottsdale, AZ, May 2002, vol. 2, pp. 492–495.

[6] J. K. Tanskanen, T. Sihvo, and J. Niittylahti, "Byte and modulo addressable parallel memory architecture for video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 14, no. 11, pp. 1270–1276, Nov. 2004.

[7] Y. S. Jehng, L. G. Chen, and T. D. Chiueh, "An efficient and simple VLSI tree architecture for motion estimation algorithms," *IEEE Trans. Signal Process.*, vol. 41, no. 2, pp. 889–900, Feb. 1993.

[8] Y. K. Lai, L. G. Chen, H. T. Chen, M. J. Chen, Y. P. Lee, and P. C. Wu, "A novel video signal processor with programmable data arrangement and efficient memory configuration," *IEEE Trans. Consumer Electron.*, vol. 42, no. 3, pp. 526–534, Aug. 1996.

[9] C. Y. Chen, S. Y. Chien, Y. W. Huang, T. C. Chen, T. C. Wang, and L. G. Chen, "Analysis and architecture design of variable block-size motion estimation for H.264/AVC," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 53, no. 3, pp. 578–593, Mar. 2006.

[10] W. Choi and B. Jeon, "Hierarchical motion search for H.264 variable block-size motion compensation," *SPIE Opt. Eng.*, vol. 45, no. 1, pp. 1–9, Jan. 2006.

[11] M. Gössel, B. Rebel, and R. Creutzburg, *Memory Architecture & Parallel Access*. Amsterdam, The Netherlands: Elsevier Science, 1994.

[12] G. Kuzmanov, G. Gaydadjiev, and S. Vassiliadis, "Multimedia rectangularly addressable memory," *IEEE Trans. Multimedia*, vol. 8, no. 2, pp. 315–322, Apr. 2006.

[13] T. H. Morrin and D. C. van Voorhis, "Method and apparatus for accessing horizontal sequences and rectangular sub-arrays from an array stored in a modified word organized random access memory system," U.S. Patent 3 938 102, Feb. 10, 1976.

[14] J. W. Park, "An efficient memory system for image processing," *IEEE Trans. Comput.*, vol. C-35, no. 7, pp. 669–674, Jul. 1986.

[15] A. R. Omondi, *Computer Arithmetic Systems: Algorithms, Architecture and Implementations*. Hertfordshire, U.K.: Prentice-Hall Int., 1994.

[16] J. Vanne, E. Aho, T. D. Hämäläinen, and K. Kuusilinna, "A high-performance sum of absolute difference implementation for motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 7, pp. 876–883, Jul. 2006.

Publication 3

J. Vanne, E. Aho, K. Kuusilinna, and T. D. Hämäläinen, "A configurable motion estimation architecture for block-matching algorithms," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 4, Apr. 2009, pp. 466-476.

A Configurable Motion Estimation Architecture for Block-Matching Algorithms

Jarno Vanne, *Member, IEEE*, Eero Aho, *Member, IEEE*, Kimmo Kuusilinna, *Member, IEEE*, and Timo D. Hämäläinen, *Member, IEEE*

Abstract—**This paper introduces a configurable motion estimation architecture for a wide range of fast block-matching algorithms (BMAs). Contemporary motion estimation architectures are either too rigid for multiple BMAs or the flexibility in them is implemented at the cost of reduced performance. The proposed architecture overcomes both of these limitations. The configurability of the proposed architecture is based on a new BMA framework that can be adjusted to support the desired set of BMAs. The chosen framework configuration is implemented by an intelligent control logic which is integrated to an efficient parallel memory system and distortion computation unit. The flexibility of the framework is demonstrated by mapping five different BMAs (BBGDS, DS, CDS, HEXBS, and TSS) to the architecture. The total execution time of the mapped BMAs is shown to be almost directly proportional to the number of tested checking points in the search area, so the architecture is very tolerant of different BMA-specific search strategies and search patterns. In addition, a run-time switching between supported BMAs can be done without performance compromises. With a 0.13-μm CMOS technology, the proposed architecture configured for HEXBS, BBGDS, and TSS requires only 14.2 kgates and 2.5 KB of memory at 200 MHz operating frequency. A performance comparison to the reference programmable architectures reveals that only the proposed implementation is able to process real-time (30 fps) fixed block-size motion estimation (1 reference frame) at full HDTV resolution (1920 × 1080).**

Index Terms—**Block-matching algorithms (BMAs), BMA framework, configurable architecture, motion estimation.**

I. Introduction

BLOCK-BASED motion estimation has been widely adopted by the current video compression standards such as MPEG-1/2/4 and H.261/263/264. In block-based motion estimation, a block-matching algorithm (BMA) searches for the best matching block for the current macroblock from the reference frame. During the searching procedure, the checking point yielding the minimum block distortion (MBD) determines the displacement of the best matching block. For the block distortion computation, the sum of absolute differences (SAD) is one of the most frequently employed criteria. After finding the MBD point, motion estimation delivers a motion vector (MV) of the current block and prediction residues. The MV of the current block equals the displacement of the best matching block.

The well-known full-search (FS) is the simplest, but the most computation-intensive BMA, which exhaustively tests all the checking points in the search area. Numerous fast BMAs have been developed to reduce huge computation load of FS [1], [2]. They can be roughly classified into lossy and lossless BMAs. Lossless BMAs such as successive elimination-based algorithms [3], [4] have the same search quality as FS, but they speed up FS by eliminating unnecessary checking points as early as possible. Techniques utilized by lossy BMAs include simplification of matching criterion [5] and decimation of checking points. Particularly, decimation of checking points is utilized by various fast BMAs including three-step search (TSS) [6], block-based gradient descent search (BBGDS) [7], new diamond search (DS) [8], [9], hexagon-based search (HEXBS) [10], cross-diamond search (CDS) [11], predictive MV field adaptive search technique (PMVFAST) [12], and unsymmetrical-cross multi-hexagon-grid search (UMHexagonS) [13]. Decimation of checking points may sometimes produce suboptimal result, i.e., BMA may drop into the local minimum point in the search area. However, these BMAs achieve clearly faster convergence than the lossless ones. The latest video compression standard H.264/AVC further increases complexity of motion estimation with variable block sizes and multiple reference frames. Decimation of checking points is also an applicable method for reducing the computation load of variable block-size motion estimation [14].

Existing motion estimation implementations are mainly tailored to a single BMA [15]–[22], since flexibility normally degrades efficiency. However, these tailored architectures are too rigid for general-purpose usage, since none of the presented BMAs provides both high search quality and fast processing speed for a mixture of rapid, moderate, and slow motion. Therefore, rather than using a single BMA, an intelligent motion estimation architecture should adaptively [23] select the most appropriate BMA at run time according to the motion content being processed.

The contemporary architectures supporting multiple BMAs are either fixed to a limited set of BMAs [1], [24]–[29] or the programmability [30]–[32] is implemented at a cost of performance and area. Particularly, their performance is a limiting factor when processing high-resolution video sequences. This

Manuscript received September 28, 2007; revised May 9, 2008. First version published March 4, 2009; current version published May 20, 2009. This work was supported in part by the Academy of Finland, Tampere Graduate School in Information Science and Engineering, Nokia Foundation, Emil Aaltonen Foundation, Heikki and Hilma Honkanen Foundation, and HPY foundation. This paper was recommended by Associate Editor L.-G. Chen.

J. Vanne and T. D. Hämäläinen are with the Department of Computer Systems, Tampere University of Technology, FI-33101 Tampere, Finland (e-mail: jarno.vanne@tut.fi).

E. Aho and K. Kuusilinna are with Nokia Research Center, FI-33721 Tampere, Finland.

Digital Object Identifier 10.1109/TCSVT.2009.2014012

is the motivation for designing a motion estimation architecture that provides real-time (30 fps) processing speed for a specific BMA up to HDTV resolution, but is still flexible enough to be used with a wide set of BMAs.

Our proposal presents a new *BMA framework* that is compatible with a predefined set of BMAs at run time. In addition, it is parametrizable to new BMAs at design time. The proposed framework is dedicated to fast BMAs that utilize decimation of checking points, so it is not suited for BMAs that rely on successive elimination and subsampling. Although this paper considers only single reference frame motion estimation with fixed block size (16×16 pixels), the basic principles presented within the framework are also applicable in variable block-size motion estimation with multiple reference frames. The flexibility of the proposed framework is based on an introduced *separable search path generation* and *parametrizable search strategy control*. Together they enable that an execution time of each BMA is almost directly proportional to the number of tested checking points in the search area. In our motion estimation architecture, the proposed framework is realized with a control unit that is seamlessly coupled to an earlier introduced parallel memory system [33] and distortion computation unit [34]. The memory system and the distortion computation unit are customized for an optimized SAD algorithm that is able to provide adequate performance for the architecture.

The rest of the paper is organized as follows. Section II describes the related work on motion estimation architectures. Section III introduces our new BMA framework. Section IV presents the implemented configurable motion estimation architecture. Performance comparisons between the proposed and contemporary architectures are shown in Section V. Section VI concludes the paper.

II. Related Work

Numerous VLSI architectures have been introduced for motion estimation during the last decades. The presented architectures can be classified as *BMA-specific*, *flexible*, and *programmable architectures* [1].

A. BMA-Specific Architectures

BMA-specific architectures support only a single BMA. The majority of contemporary BMA-specific architectures implement FS [15]–[17] because of its regular data flow and low control overhead. FS is typically implemented with a systolic mesh-connected array which offers high throughput through parallel processing, pipelining, and data reuse. Due to the huge inherent complexity of FS, a large array of processing elements (PEs) is needed to achieve high performance.

Contrary to the FS case, systolic arrays are not well suited for the fast BMAs, which include unpredictable data flow, irregular memory addressing, and hardly parallelizable sequential control. Prior-art BMA-specific architectures for fast BMAs are mainly concentrated on TSS and its derivatives [18]–[20]. Although these implementations achieve good cost efficiency over the FS architectures, they are too rigid for a broad range of applications [25]–[32].

B. Flexible Architectures

Flexible architectures implement a specific set of BMAs. Jehng *et al.* [24], Zhang and Tsui [26], and Kuhn [1] introduced architectures that support both FS and TSS. Kuhn's architecture also supports half-pixel motion estimation.

Dutta and Wolf [25] developed an architecture that can change its configurable interconnection network and PEs according to used BMA. The architecture is designed to receive a control sequence for an executed BMA in advance through simulations. The flexibility of the architecture is demonstrated by mapping, e.g., FS and TSS, to it.

Chao *et al.* [27] proposed a flexible architecture for fast FS and DS. In the DS mode, the implementation uses a ROM-based approach to reduce redundant testing of checking points.

Li *et al.* [28] presented an architecture for two predictive BMAs: PMVFAST and EPZS (enhanced predictive zonal search). The architecture includes nine PEs. Since each PE computes distortion of a separate checking point, there are unused PEs if fewer than nine checking points are tested simultaneously. Different search patterns are configured with variable delay units. The architecture does not execute all the operations of BMAs, but complicated control parts of BMAs are managed by an RISC processor.

Dias *et al.* [29] introduced a configurable architecture for FS and a limited set of fast BMAs such as TSS and DS. The architecture implements BMAs by filling lookup tables with predefined data structures that contain all the BMA-specific information about the search patterns and search paths. These run-time reconfigurable data structures can simultaneously contain multiple BMAs. However, they are not capable of expressing more complex BMAs such as PMVFAST.

Although flexible architectures are more generic than BMA-specific architectures, they only support a limited set of BMAs. Their control and computational structures tend to be incompatible with new BMAs or additional BMA features.

C. Programmable Architectures

Programmable architectures are designed to support a practically unlimited set of BMAs. However, all the reference architectures consider only BMAs that utilize pixel decimation techniques.

Lin *et al.* [30] presented a programmable motion estimation architecture that applies macro-commands to execute BMAs. The architecture processes macro-commands interactively instead of executing fixed search patterns in batches. As in [28], part of the BMA control is managed by a host processor.

Gong and He [31] developed a programmable video signal processor for fast BMAs. Their BMA framework includes five parameters per BMA and the control is implemented with an RISC processor. The programmability is verified by mapping, e.g., FS and TSS, to the processor. The architecture also supports half-pixel motion estimation.

Dias *et al.* [32] introduced an application-specific instruction set processor (ASIP) that is capable of executing various fast BMAs. In addition, a set of software tools were developed to program BMAs on the processor. The implemented ASIP has an instruction set of eight instructions that are specially tailored for motion estimation. BMAs mapped to the processor include FS, TSS, DS, and MVFAST. The processor can manage both fixed and variable block sizes.

The presented programmable architectures are suitable for general-purpose usage. However, the programmability in them is achieved at a cost of increased silicon area and performance degradation.

The proposed architecture can be categorized as a hybrid of flexible and programmable architectures. It supports a predefined set of BMAs at run time, but it is easily upgradeable to completely new or slightly modified BMAs at design time. The proposed architecture implements BMAs with predefined data structures as in [29], but a more compact representation for these data structures is introduced in this paper. In addition, compared to the architecture in [29], the proposed extensions to BMA parametrization enable that more sophisticated BMAs, such as predictive BMAs, can be implemented with the proposed BMA framework.

III. Proposed BMA Framework

When a BMA is executed, the parameters of the proposed BMA framework control the search path generation in the search area. In addition, the parameters adjust search strategy control which is responsible for BMA-specific search directions and search patterns.

A. Proposed Search Path Generation

Typically, the search of the best matching block is restricted to a search area around the original location of the block [1]. Let a size of a two-dimensional search area be $w \times h$ pixels, where $w \in \{48, 80, 112, 144\}$ and $h \in \{48, 80, 112, 144\}$ are the practical dimensions of the area in i and j directions, respectively. For video sequences with small motion content, w and h are usually 48, whereas an enlarged search area is beneficial for larger frame sizes and for sequences having rapid motion content.

The proposed architecture is designed to access a block of 4×4 search area pixels simultaneously, so 16 accesses are required per candidate MB (checking point). In the search area, the accessed 4×4 base block is indicated by a scanning point $(r_{RR}(ri_{RR}, rj_{RR}))$ [33], where $ri_{RR} \in [0, w-1]$ and $rj_{RR} \in [0, h-1]$. The proposed fundamental principle of r_{RR} composition is presented in Fig. 1, where $w = h = 48$. Since BMAs find the MBD point via subsequent search steps during which the search path is updated, r_{RR} can be composed of several independent and parametrizable partial offsets.

The first partial offset of r_{RR} is an *initial offset* $(\Delta r_\alpha(\Delta ri_\alpha, \Delta rj_\alpha))$ that points to the center of the search area. After each searching procedure, the center of the search area is shifted according to used search area reuse scheme [18], i.e., Δri_α is incremented so that the successive searching procedures can partially reuse the same search area data. The proposed system supports search area reuse schemes that are implementable when $\Delta ri_\alpha \in [0, w-1]$ and $\Delta rj_\alpha \in [0, h-1]$ are multiples of 16.

With center-biased BMAs (e.g., DS, HEXBS, and TSS), Δr_α determines the search center from which the

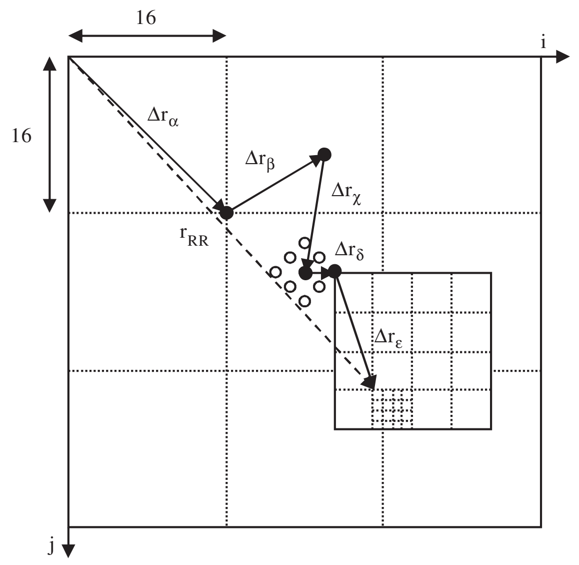


Fig. 1. Proposed separable composition of r_{RR} in the 48×48 pixel search area.

searching procedure is started. The predictive BMAs such as PMVFAST [12] determine the search center according to the predicted MV. For these BMAs, Δr_α is incremented with a *prediction-based offset* $(\Delta r_\beta(\Delta ri_\beta, \Delta rj_\beta))$. This optional offset is particularly targeted for non-center-biased video sequences.

The BMA displacement from $\Delta r_\alpha + \Delta r_\beta$ is indicated by a *BMA movement offset* $(\Delta r_\chi(\Delta ri_\chi, \Delta rj_\chi))$. The offset is automatically updated by the architecture when the center of the search pattern is moved.

The support for different search patterns is enabled by an adjustable *checking point offset* $(\Delta r_\delta(\Delta ri_\delta, \Delta rj_\delta))$. It points to the top-left corner of the candidate MB and determines a displacement of a tested checking point from the pattern center. An example search pattern resembling a large diamond search pattern (LDSP) [8], [9] of DS is depicted in Fig. 1, where Δr_δ equals the displacement between the pattern center and the right-most checking point of the pattern.

A *base block offset* $(\Delta r_\varepsilon(\Delta ri_\varepsilon, \Delta rj_\varepsilon))$ represents a displacement of a 4×4 base block from the top-left corner of the candidate MB. In Fig. 1, the small grid illustrates the accessed 4×4 base block. The size of the MB restricts that $\Delta ri_\varepsilon \in \{0, 4, 8, 12\}$ and $\Delta rj_\varepsilon \in \{0, 4, 8, 12\}$.

Adding the partial offsets together yields $r_{RR}(ri_{RR}, rj_{RR})$ as

$$ri_{RR} = (\Delta ri_\alpha + \Delta ri_\beta + \Delta ri_\chi + \Delta ri_\delta + \Delta ri_\varepsilon) \bmod w \quad (1)$$

$$rj_{RR} = (\Delta rj_\alpha + \Delta rj_\beta + \Delta rj_\chi + \Delta rj_\delta + \Delta rj_\varepsilon) \bmod h. \quad (2)$$

When $\Delta r_\varepsilon = (0, 0)$, the respective MV(MV$i$, MV$j$) is derived with Δr_α and r_{RR} as

$$\mathrm{MV}i = \left(\left(\frac{w-16}{2} + ri_{RR} - \Delta ri_\alpha\right) \bmod w\right) - \frac{w-16}{2} \quad (3)$$

$$\mathrm{MV}j = \left(\left(\frac{h-16}{2} + rj_{RR} - \Delta rj_\alpha\right) \bmod h\right) - \frac{h-16}{2}. \quad (4)$$

Incrementing Δri_α after each searching procedure requires that the search area is circular: e.g., when $w = 48$ and

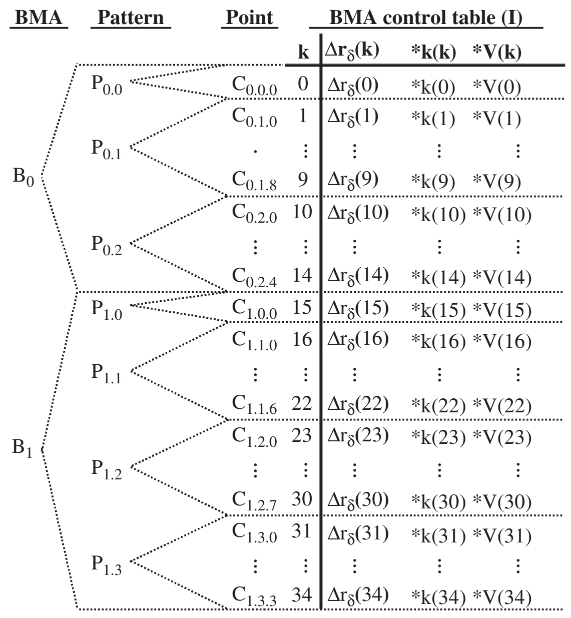
$\Delta ri_\alpha = 32$, ri_{RR} may reach w during the searching procedure, in which case it is clipped to zero. In (1)–(4), the circular property of the search area is realized by mod w, mod h, $\frac{w-16}{2}$, and $\frac{h-16}{2}$. In addition, the following restrictions have to hold:

$$\Delta ri_\beta, \Delta ri_\chi, \Delta ri_\delta, \mathrm{MV}i \in \left[-\frac{w-16}{2}+1, \frac{w-16}{2}-1\right]$$

$$\Delta rj_\beta, \Delta rj_\chi, \Delta rj_\delta, \mathrm{MV}j \in \left[-\frac{h-16}{2}+1, \frac{h-16}{2}-1\right]$$

Otherwise, the MV exceeds the boundaries of the search area.

Composing r_{RR} from the mutually independent partial offsets enables adaptive and easily controllable search path generation for BMAs. BMA-specific features related to search area reuse schemes, search center, search patterns, and base block accessing can be separately adjusted via Δr_α, Δr_β, Δr_δ, and Δr_ε. A single BMA feature typically applies only to a single partial offset, and the rest of the offsets can be maintained constant. As presented later in this paper, the introduced separable composition of r_{RR} is also beneficial for hardware implementation.



BMA	Pattern	Point	k	$\Delta r_\delta(k)$	$*k(k)$	$*V(k)$
B_0	$P_{0.0}$	$C_{0.0.0}$	0	$\Delta r_\delta(0)$	$*k(0)$	$*V(0)$
	$P_{0.1}$	$C_{0.1.0}$	1	$\Delta r_\delta(1)$	$*k(1)$	$*V(1)$
		·	⋮	⋮	⋮	⋮
		$C_{0.1.8}$	9	$\Delta r_\delta(9)$	$*k(9)$	$*V(9)$
	$P_{0.2}$	$C_{0.2.0}$	10	$\Delta r_\delta(10)$	$*k(10)$	$*V(10)$
		⋮	⋮	⋮	⋮	⋮
		$C_{0.2.4}$	14	$\Delta r_\delta(14)$	$*k(14)$	$*V(14)$
B_1	$P_{1.0}$	$C_{1.0.0}$	15	$\Delta r_\delta(15)$	$*k(15)$	$*V(15)$
	$P_{1.1}$	$C_{1.1.0}$	16	$\Delta r_\delta(16)$	$*k(16)$	$*V(16)$
		⋮	⋮	⋮	⋮	⋮
		$C_{1.1.6}$	22	$\Delta r_\delta(22)$	$*k(22)$	$*V(22)$
	$P_{1.2}$	$C_{1.2.0}$	23	$\Delta r_\delta(23)$	$*k(23)$	$*V(23)$
		⋮	⋮	⋮	⋮	⋮
		$C_{1.2.7}$	30	$\Delta r_\delta(30)$	$*k(30)$	$*V(30)$
	$P_{1.3}$	$C_{1.3.0}$	31	$\Delta r_\delta(31)$	$*k(31)$	$*V(31)$
		⋮	⋮	⋮	⋮	⋮
		$C_{1.3.3}$	34	$\Delta r_\delta(34)$	$*k(34)$	$*V(34)$

Fig. 2. Fundamental operating principle of the proposed BMA control table.

B. Proposed Search Strategy Control

A flexible architecture having a hardwired control for each of the supported BMAs achieves high performance, but even a diminutive BMA modification requires redesigning the control. Since the search strategies and patterns tend to vary between BMAs, a parametrizable search strategy control is a necessity for the proposed configurable architecture.

In the search area (Fig. 1), a generation of Δr_χ is dependent on the selected search strategy and search patterns. The proposed architecture processes all the BMAs according to the introduced *general BMA execution flow* which assigns the available search strategies and search patterns. The first step in the flow is common for all the BMAs, whereas the next search steps are BMA specific. The general BMA execution flow is summarized for a specific BMA as:

Step 1. The first search pattern is positioned in the search center and only the search center is tested and selected as the initial MBD point. The searching procedure proceeds to Step 2.

Step 2. All the valid checking points surrounding the pattern center are tested. If no new MBD point is found, i.e., the MBD point is still found in the pattern center or if a single-pass pattern is used, the searching procedure proceeds to Step 3. Otherwise, the recursive search pattern is repositioned so that the new MBD point is at the center of the pattern and Step 2 is recursively repeated.

Step 3. If all the available search patterns have already been executed, the MBD point found in Step 2 is selected and the searching procedure is stopped. Otherwise, the search pattern is switched to a next search pattern which is positioned so that the MBD point found in Step 2 is in the center of it and the searching procedure proceeds to Step 2.

The general BMA execution flow is realized with a proposed *BMA control table*, which can consist of $b_{\max}$ number of BMAs. Each BMA B_b ($b \in [0, b_{\max}-1]$) can include $p_{\max}$ number of search patterns which can be either single pass or recursive. In each B_b, the first search pattern ($P_{b.0}$) always includes only a search center ($C_{b.0.0}$) whereas the rest of the search patterns $P_{b.p}$ ($p \in [1, p_{\max}-1]$) can have $c_{\max}$ checking points $C_{b.p.c}$ ($c \in [0, c_{\max}-1]$). Upper limits $b_{\max}$, $p_{\max}$, and $c_{\max}$ are positive integers that are design-time selectable in an implementation specific range.

Fig. 2 depicts an example of the BMA control table (I) including two BMAs (B_0 and B_1) which have three ($P_{0.0}, \ldots, P_{0.2}$) and four ($P_{1.0}, \ldots, P_{1.3}$) search patterns, respectively. Each checking point $C_{b.p.c}$ is logically mapped to the location $I(k)$ in the table, where k is resolved as a function of b, p, and c.

Three parameters are associated with each $I(k)$: $\Delta r_\delta(k)$ (see Fig. 1), an index of the next search pattern ($*k(k)$), and a vector identifying valid checking points in the next search pattern ($*V(k)$). According to the general BMA execution flow, a searching procedure always starts from the search center that is tested and the next search pattern is selected based on it. In the consecutive search patterns, only valid checking points are tested. A valid checking point k yielding the MBD after a single search pattern iteration is marked with k_{MBD} and the search pattern is moved in the search area according to $\Delta r_\delta(k_{MBD})$. The used search pattern is either reselected or switched to the next one according to $*k(k_{\mathrm{MBD}})$ that indexes a center point of the destination search pattern. Valid checking points surrounding the destination pattern center are determined by the vector $*V(k_{\mathrm{MBD}})$ which includes $c_{\max}$–1 number of validation flags, one flag for each surrounding checking point. To avoid redundant testing, a checking point belonging to the destination pattern is invalidated, if it has already been tested by the used pattern. In [27], the similar invalidation

BMA	Pattern	Point	k	$\Delta r_\delta(k)$	*k(k)	*V(k)
DS	Center	$C_{0.0.0}$	0	(0, 0)	1	11111111
	LDSP	$C_{0.1.0}$	1	(0, 0)	10	00001111
		$C_{0.1.1}$	2	(–2, 0)	1	11000111
		$C_{0.1.2}$	3	(–1, –1)	1	00000111
		$C_{0.1.3}$	4	(0, –2)	1	00011111
		$C_{0.1.4}$	5	(1, –1)	1	00011100
		$C_{0.1.5}$	6	(2, 0)	1	01111100
		$C_{0.1.6}$	7	(1, 1)	1	01110000
		$C_{0.1.7}$	8	(0, 2)	1	11110001
		$C_{0.1.8}$	9	(–1, 1)	1	11000001
	SDSP	$C_{0.2.0}$	10	(0, 0)	10	00000000
		$C_{0.2.1}$	11	(–1, 0)	10	00000000
		$C_{0.2.2}$	12	(0, –1)	10	00000000
		$C_{0.2.3}$	13	(1, 0)	10	00000000
		$C_{0.2.4}$	14	(0, 1)	10	00000000

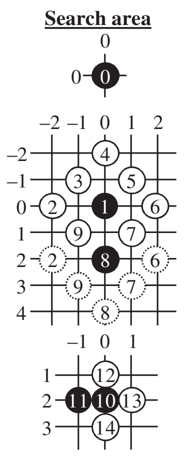


Fig. 3. BMA control table parametrized for DS and an example search path.

method is applied with DS and almost all redundant testing is avoided. In the destination pattern, the pattern center always overlaps with the MBD point of the previous search pattern, so the pattern center is never tested. However, the pattern center is selected as the MBD point, if tested surrounding checking points do not yield a new MBD point.

The BMA control table is implementable with a ROM and associated control logic. Storing BMA parameters in the ROM enables fast and low-cost implementation for a wide range of BMAs. The search strategy is completely controllable with the table parameters, so BMAs can be executed without processor control.

A detailed usage of the BMA control table is presented in Fig. 3 in which B_0 (Fig. 2) is replaced with DS and an example search path of DS is shown in a two-dimensional search area. Only the essential coordinates of the search area are presented within each search pattern. In addition, each location $I(k)$ of the BMA control table is associated with a checking point k in the search area.

The searching procedure is started with center pattern that is used to test the search center ($C_{0.0.0}$) mapped to $I(0)$. The pointers k and k_{MBD} are set according to moving condition $*k(0)$, i.e., $k_{\mathrm{MBD}} = k = *k(0) = 1$, so a center of LDSP ($C_{0.1.0}$) mapped to $I(1)$ is selected after $C_{0.0.0}$ is tested. Since the current pattern center always overlaps with the MBD point of the previous search pattern, $C_{0.0.0}$ and $C_{0.1.0}$ represent the same checking point. Hence, the next untested checking point mapped to $I(2)$ is selected, i.e., $k = 2$ is set. Besides the checking point mapped to $I(2)$, the other surrounding checking points mapped to $I(3), \ldots, I(9)$ are valid ('1') according to the respective 1-bit flags of the vector $*V(0)$. The vector $*V(0)$ is interpreted from right to left, i.e., the rightmost flag controls $I(2)$ and so on. The checking points are tested by incrementing k one by one.

In the search path example (Fig. 3), a checking point 8 of the first LDSP is a new MBD point. To follow the example, it is assumed that $k_{\mathrm{BDM}} = 8$ is found after the checking points of the first LDSP are tested. Based on $*k(8) = 1$, the center of LDSP is moved by $\Delta r_\delta(8) = (0,\ 2)$ and LDSP is reselected. In addition, $k_{\mathrm{BDM}} = k = 1$ are set. In the reselected LDSP, the checking points mapped to $I(2), I(6), I(7), I(8)$, and $I(9)$ are valid according to $*V(8)$, whereas the center of LDSP and invalid ('0') surrounding checking points mapped to $I(3), I(4)$, and $I(5)$ are omitted. In this example, no new k_{BDM} is found by the reselected LDSP. Therefore, the center of LDSP mapped to $I(1)$ is chosen as the MBD point, $*k(1) = 10$ switches LDSP to SDSP (small diamond search pattern), and $k_{\mathrm{BDM}} = k = 10$ is set. According to $*V(1)$, the checking points mapped to $I(11), \ldots, I(14)$ are valid. Since SDSP includes only four checking points in total, the four leftmost flags in $*V(1)$ are invalid. The checking points mapped to $I(11), \ldots, I(14)$ are tested and $k_{\mathrm{BDM}} = 11$ is found in this example. Since $*V(11)$ indicates only invalid checking points, the search stops.

IV. Proposed Implementation

Fig. 4 presents an exemplary system architecture for a video encoder in which the proposed motion estimation architecture is connected to other system components (CPU, frame memory, and other accelerators) via an on-chip communication network. The other function-specific accelerators may include hardware modules for fractional pixel motion estimation, DCT, quantization, etc. The motion estimation architecture is best suited for a communication network that is composed of 128-bit wide data and 8-bit wide command buses. Depending on the system requirements, a network topology can vary from a shared bus to highly scalable network such as HIBI [35].

The network-specific wrappers are used to integrate the system components to the network. The motion estimation architecture requires a wrapper that reconciles its unidirectional data buses ($data_{IN}$, $data_{OUT}$) and control/status signals ($ctrl_{IN}$, $ctrl_{OUT}$) to the bidirectional data and command buses of the system, respectively. Besides interface integration, the wrapper of the motion estimation architecture implements protocols to retrieve reference and current frame data directly from the frame memory. It also forwards the result of the motion estimation directly to the CPU and other target components.

Fig. 5 depicts a high-level structure of the proposed motion estimation architecture. The main components of the architecture are a *control unit*, a *memory system* [33], and a *SAD unit* [34]. The 128-bit wide $data_{IN}$ and $data_{OUT}$ buses of the architecture are controlled with the $ctrl_{IN}$ (*BMA_id*, *vld*, *reuse*, *ack*) and $ctrl_{OUT}$ (*new*, *stored*, *id*, *rdy*) signals.

If the architecture is configured to support multiple BMAs, BMA switching can be performed at run time with *BMA_id* input. The $data_{IN}$ bus is time-multiplexed between reference frame data (d_{RI}), current frame data (d_{CI}), and Δr_β which adjusts a search center of a prediction-based BMA. The control unit uses the one-hot coded 3-bit *new* signal to request new d_{RI}, d_{CI}, and Δr_β. It detects valid input data (d_{RI}, d_{CI}, or Δr_β) by monitoring the 3-bit *vld* signal. The *reuse* signal determines whether to store the whole search area or only a search area strip because of search area reuse. During data storage, the control unit produces scanning points (r_{RW} and r_{CW}) and respective storage control signals ($ctrl_M$) for the memory system. The motion estimation architecture has a separate

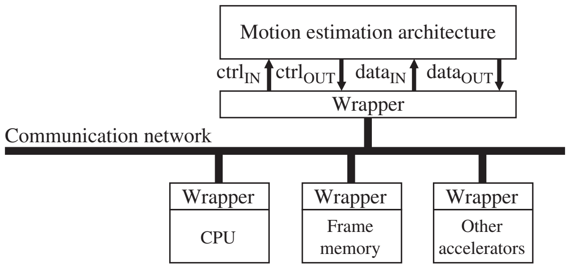


Fig. 4. Exemplary system architecture for a video encoder.

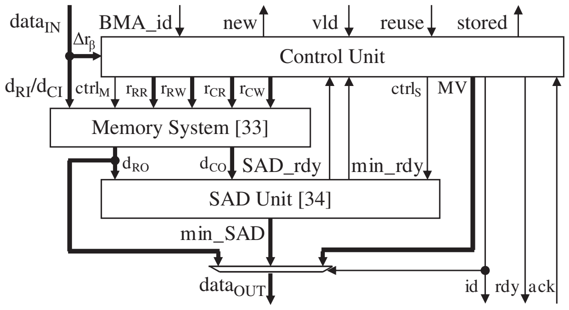


Fig. 5. Implemented motion estimation architecture.

two-stage pipeline for data storage. After d_{RI}, d_{CI}, and Δr_β are stored, all the bits in the 3-bit *stored* signal are asserted.

During the searching procedure, the control unit composes BMA-specific scanning points (r_{RR} and r_{CR}) for the search area data (d_{RO}) and current block data (d_{CO}). The control unit controls data delivery ($ctrl_M$) from the memory system as well as SAD computation ($ctrl_S$). Pixels are retrieved from the memory system so that all the PEs in the SAD unit can always participate in distortion computation. Hence, moderately high utilization of PEs is reached. The SAD unit computes partial distortion of 16 pixels in parallel; i.e., it completes a SAD value for a candidate MB in 16 clock cycles. The SAD unit informs the control unit of the SAD value completion (*SAD_rdy*) and whether the finished SAD value is the new minimum (*min_rdy*). The control unit, the memory system, and the SAD unit all include three pipeline stages, so the pipeline depth of the architecture is nine.

After the final MBD point is found, the control unit uses the *rdy* signal to request permission for data delivery. The permission is acknowledged with the *ack* signal after which the minimum SAD value (*min_SAD*), the motion vector (*MV*), and the best matching block are delivered. The $data_{OUT}$ bus is time-multiplexed between *min_SAD*, *MV*, and d_{RO}. They are identified with the one-hot coded 3-bit *id* signal.

A. Control Unit

The main components of the control unit and essential signals during the searching procedure are depicted in Fig. 6. A *main controller* together with the *BMA control table* comprises a core of the searching procedure control. Fig. 7 presents a flowchart of the main controller. The BMA control table implements the functionality described in Fig. 2.

The main controller stays in the data store mode until d_{RI} and d_{CI} are stored in the memory system (Fig. 5) and Δr_β is delivered to an r_{RR} *computation unit*. When data storage is completed, the main controller initializes the searching procedure for the BMA selected with *BMA_id* input.

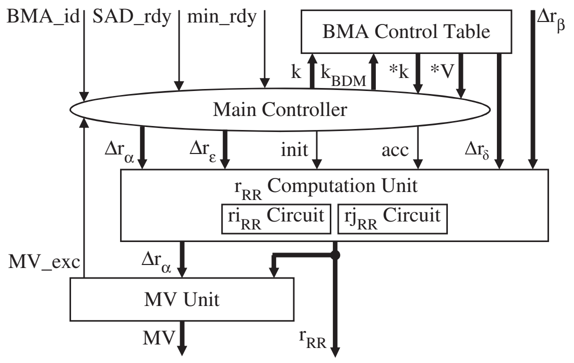


Fig. 6. Control unit.

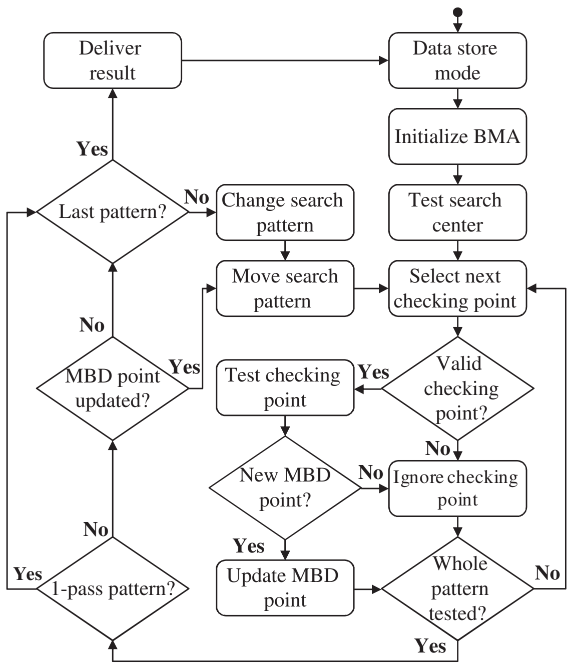


Fig. 7. Flowchart of the main controller.

The searching procedure is started by testing the search center after which all the valid surrounding checking points of the search pattern are tested. The main controller produces appropriateΔr_α, Δr_ε, and control signals (*init*, *acc*) for the r_{RR} computation unit. In addition, it computes the pointer k that is used to retrieve Δr_δ from the BMA control table. The valid checking points are selected according to the vector $*V$. The main controller is informed of the SAD value completion by the *SAD_rdy* signal. If the *min_rdy* signal is active simultaneously with the *SAD_rdy* signal, a tested checking point is a new MBD point and the main controller updates k_{MBD} accordingly.

The *MV unit* determines an MV of a tested checking point according to (3) and (4). If the determined MV exceeds the boundary of the search area, the unit asserts a signal (*MV_exc*) which interrupts the checking point testing and the main controller selects a next valid checking point.

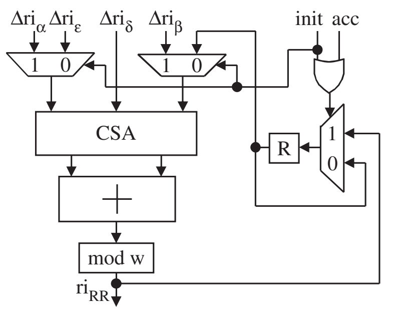


Fig. 8. ri_{RR} circuit.

After the whole search pattern is tested, the main controller determines whether a single-pass or a recursive search pattern has been used. If a new MBD point has been found with a recursive search pattern, the searching procedure continues with the same search pattern that is repositioned according to the new MBD point. Otherwise, the search using the current pattern is completed and the search pattern is switched to another. In both of these cases, the main controller accesses the BMA control table with k_{MBD} to retrieve corresponding Δr_δ, $*k$, and $*V$. In the case of the last pattern, the searching procedure is completed and the result is delivered.

The r_{RR} computation unit includes identical ri_{RR} and rj_{RR} *circuits* for (1) and (2) computation, respectively. Fig. 8 depicts the ri_{RR} circuit. The circuit is controlled by the initialize (*init*) and accumulate (*acc*) signals. It selects partial offsets (Fig. 1) with multiplexers and reduces the selected offsets with CSA (carry save adder) to two operands before final addition.

When the ri_{RR} circuit determines a search center, the *init* signal is set to '1'. The circuit computes the search center as $ri_{RR} = (\Delta ri_\alpha + \Delta ri_\delta + \Delta ri_\beta) \bmod w$ and the result is stored in the register (R). Since $\Delta ri_\delta = 0$ for the search center, $R = ri_{RR} = (\Delta ri_\alpha + \Delta ri_\beta) \bmod w$. In order to determine the checking points around the pattern center, the *init* signal is set to '0', in which case $R = (\Delta ri_\alpha + \Delta ri_\beta) \bmod w$ is maintained constant and a tested checking point is computed as $ri_{RR} = (\Delta ri_\varepsilon + \Delta ri_\delta + R) \bmod w$.

After the first search pattern is completed, the ri_{RR} circuit modifies the search path by accumulating R. For the accumulation, Δri_δ corresponding to the MBD point of the completed pattern has to be redelivered to the circuit. The circuit recomputes $ri_{RR} = (\Delta ri_\varepsilon + \Delta ri_\delta + R) \bmod w$, where $\Delta ri_\varepsilon = 0$. The asserted *acc* signal updates the register R as $R = ri_{RR} = (\Delta ri_\delta + R) \bmod w$. Now, the search path is prepared for the next search pattern.

Utilizing the same hardware for the r_{RR} computation and the search path modification makes the circuit very area-efficient while maintaining high performance. The ri_{RR} and rj_{RR} circuits are able to compute a new r_{RR} in every clock cycle, and only one clock cycle delay is needed for the search path modification.

B. Memory System

The memory system (Fig. 5) includes two separate local on-chip memories: a search area memory for d_{RI} and a current block memory for d_{CI} [33]. The memories include 16 1-pixel wide and four 4-pixel wide single-port parallel memory modules [36], respectively. The search area memory is configurable for the search areas of $w \in \{48, 80, 112, 144\}$ and $h \in \{48, 80, 112, 144\}$ at design time. In addition, search area reuse is supported.

To achieve maximal data storage capability, the memory system supports a 16-pixel wide row access format for data storage. Data is retrieved from the memories with a 4×4 block access format (Fig. 1) which is compatible with variable block sizes. Since unrestricted memory accesses are vital for fast BMAs, the block format can be arbitrarily placed in the pixel-addressable search area memory.

Besides the memory modules, the memory system includes an address computation unit as well as data rotators for the write and read data. The memory system is presented in detail in [33].

C. SAD Unit

The SAD unit (Fig. 5) uses SAD as a block distortion measure. At the current block location $(x,\ y)$, the SAD criterion is defined as

$$\text{SAD}(x, y, i, j) = \sum_{q=0}^{Q-1} \sum_{u=0}^{U-1} |A_{(x+q,y+u)} - B_{(x+i+q,y+j+u)}| \tag{5}$$

where $A_{(x+q,y+u)}$ and $B_{(x+i+q,y+j+u)}$ indicate pixels of the current block and the reference frame, respectively. The size of the block is $Q \times U$ and SAD computation is performed in the search area location $(i,\ j)$ which is the displacement of the candidate block from the current block.

The SAD unit implements an optimized SAD algorithm which processes arithmetic operations of (5) in an efficient way. A flexible control with multiple available early termination mechanisms and threshold values enhances execution speed and makes the architecture compatible with various BMAs.

The SAD unit is divided into three stages. The first stage is composed of 16 absolute difference units, the second stage includes a compression array for a partial SAD value accumulation, and the third stage is for a minimum SAD determination unit. The SAD unit is thoroughly described in [34].

V. Performance Analysis

The performance of the proposed motion estimation architecture (Fig. 5) is evaluated with TSS [6], BBGDS [7], DS [8], [9], HEXBS [10], and CDS [11]. All the five BMAs follow the proposed BMA execution flow, but the search strategies and patterns vary between them. TSS uses single-pass search patterns and a coarse-to-fine search strategy in which search step size is hierarchically converged. The amount of repeated steps is determined by the size of the search area. BBGDS utilizes a square pattern of 3×3 checking points which recursively advances towards the MBD point. The search strategy and patterns of DS are presented in Fig. 3. HEXBS resembles DS, expect that it replaces LDSP with

TABLE I
PERFORMANCE OF THE PROPOSED MOTION ESTIMATION ARCHITECTURE

Sequence	Format	BMA	PSNR	Points/MB	PSNR diff.	Speed-up	I_{size}(bits)	t_T/MB	t_D/MB	t_D/point	MHz @ 30 fps
Salesman	QCIF	TSS	40.28	33.0	−0.03	29	851	685	569	17.2	3
		DS	40.30	13.0	−0.01	74	270	345	230	17.6	2
		CDS	40.29	9.1	−0.02	106	960	272	156	17.2	1
		BBGDS	40.30	9.1	−0.01	106	130	272	156	17.2	1
		HEXBS	40.24	11.0	−0.07	87	208	313	197	17.9	1
Foreman	CIF	TSS	31.18	33.0	−1.08	29	851	680	569	17.2	9
		DS	31.21	18.7	−1.05	51	270	450	338	18.1	6
		CDS	31.10	16.8	−1.17	57	960	417	305	18.2	5
		BBGDS	31.12	16.6	−1.14	58	130	412	301	18.2	5
		HEXBS	30.68	14.1	−1.59	68	208	372	260	18.4	5
F1 Car	D1	TSS	22.75	33.0	−1.60	29	851	678	569	17.2	33
		DS	22.13	22.1	−2.22	44	270	509	400	18.1	25
		CDS	22.08	21.5	−2.27	45	960	502	393	18.3	25
		BBGDS	21.65	18.5	−2.70	52	130	448	339	18.3	22
		HEXBS	22.00	15.8	−2.35	61	208	404	295	18.6	20
Pedestrian	HDTV	TSS	29.93	33.0	−0.52	29	851	677	569	17.2	166
		DS	29.00	27.5	−1.46	35	270	623	516	18.7	153
		CDS	28.98	26.0	−1.48	37	960	597	489	18.8	147
		BBGDS	28.55	27.3	−1.91	35	130	615	507	18.6	151
		HEXBS	28.83	19.0	−1.62	51	208	473	365	19.3	116

hexagon-shaped search pattern of six checking points. CDS has the most sophisticated search strategy, which employs several search patterns and halfway-stop techniques. CDS begins with a cross-shaped search pattern after which the search optionally continues as DS.

A. Experimental Results

The executed four video sequences "Salesman" (176 × 144, 449 frames), "Foreman" (352 × 288, 300 frames), "F1 Car" (720 × 576, 220 frames), and "Pedestrian" (1920 × 1080, 50 frames) vary in motion content as well as in frame size. All the simulations are consistently performed with a single reference frame and with fixed block size of 16 × 16 pixels. MV is restricted to [−15, 15] ($w = h = 48$), and the searching procedures are always started from the search center without prediction. The proposed architecture is configured to prefer a selection of the search center; i.e., the value of 100 is subtracted from the SAD value of zero MV (zero bonus). Other thresholds or early termination mechanisms [34] are not used in the simulations.

Table I tabulates the simulation results. An average peak signal-to-noise ratio (PSNR) of motion-compensated frames and an average number of checking points per MB are used as the measure of BMA search quality and search speed, respectively. In all the sequences, TSS has a fixed amount of checking points, whereas the motion content affects the search path lengths of DS, CDS, BBGDS, and HEXBS. In addition, average PSNR degradation (PSNR diff.) and speed-up ratio over FS are reported for each BMA. Compared to FS, the evaluated BMAs suffer from reduced search quality when motion content and resolution are higher. However, they also achieve clearly better search speed in all the examined cases. For example, HEXBS achieves average speed-up ratio of 67. More accurate performance comparisons of these BMAs are presented in [1]–[14].

The size (I_{size}) of the BMA control table varies between BMAs. The control tables of BBGDS and HEXBS resemble that of DS (Fig. 3). The control table of TSS is the most regular one, whereas CDS requires the most complex table. However, even the most complex table is implementable with small overhead (I_{size} < 1 Kbits). The usage of the parameter $*V(k)$ enables the proposed control table to be much more compact than the data structure presented in [29]. For example, the height of the BMA control table for DS is 15 (Fig. 3), whereas the corresponding data structure for DS requires 53 positions in [29].

A total clock cycle count (t_T) per MB includes cycles consumed by distortion computation (t_D), data storage, and data delivery. However, search area extension used in image boundaries is assumed to be performed beforehand so that it does not burden the architecture. Derived from t_T/MB and t_D/MB, the utilization of the computation units varies from 57% to 84%. With the evaluated fast BMAs, the time consumed by data storage and data delivery is emphasized since only a minority of checking points is tested. Masking the delays of data storage and delivery would require dual-port search area and current block memories as well as an additional data buffer for the best matching block.

Normalized execution times of different BMAs are tabulated as computation cycles consumed per checking point (t_D/point). The proposed architecture is very tolerant of different types of BMAs. It processes all the test cases with only 12% deviation in t_D/point metric, which means that a total cycle count is almost directly proportional to the number of checking points tested. Hence, the BMA-specific metrics (PSNR and points/MB) are the most significant criteria when determining a preferred set of BMAs for the proposed architecture.

In the "Salesman" sequence, the small motion content means that most of the blocks are stationary; i.e., MBD points are mainly located in the search center. All the evaluated

TABLE II
COMPARISON OF THE PROPOSED AND EXISTING ARCHITECTURES

Architecture	Supported BMAs	Avg.t_T/MB	Freq (MHz)	Memory (KB)	Area (kgates)	Power (mW)	♯ of PEs	Process (μm)	Voltage (V)	Search range
Proposed	HEXBS/BBGDS/TSS	390/437/680	200	2.5	14.2	59	16	0.13	1.2	[−15, 15]
Jong [18]	TSS	794	40	1.7	13.6	n.a.	9	n.a.	n.a.	[−7, 7]
Chen [20]	TSS	851	50	1.5	11.6	350	9	0.80	5.0	[−7, 7]
Lakamsani [19]	Enhanced TSS	2 368	40	0.6	4.1	n.a.	3	n.a.	n.a.	[−5, 5]
Zhang [26]	TSS/FS	204/1 591	n.a.	1.9	24.0	n.a.	48	n.a.	n.a.	[−8, 8]
Kuhn [1]	TSS/FS	948/16 399	100	2.4	22.3	n.a.	16	0.25	2.5	[−16, 15]
Chao [27]	DS/Fast FS	437/2 879	50	3.5	9.0	224	8	0.35	3.3	[−16, 15]
Li [28]	PMVFAST/EPZS	1 042/1 042	200	n.a.	17.5	n.a.	9	0.18	n.a.	n.a.
Lin [30]	Programmable	n.a.	66	0.8	31.3	<300	72	0.50	3.3	Programmable
Gong [31]	Programmable	1392 (TSS)	133	39.1	30.0	n.a.	16	0.25	n.a.	n.a.
Dias [32]	Programmable	5376 (DS)	144	n.a.	13.3	48	1	0.18	1.8	[−8, 7]

n.a. = not available.

BMAs consume the smallest available number of checking points with stationary blocks, so an average computational complexity (points/MB) of each BMA is close to its theoretical minimum. Since all the BMAs produce almost equal PSNR values, BBGDS and CDS with the lowest cycle counts (t_T/MB) are the preferred ones. With BBGDS and CDS, the architecture clocked at 1 MHz meets real-time performance (30 fps).

The higher motion content in the "Foreman" sequence increases computational complexity of DS, CDS, BBGDS, and HEXBS over the "Salesman" sequence. HEXBS has the lowest search quality, but the increase in cycle count is more moderate with it (19%) than with DS (30%), CDS (53%), and BBGDS (52%). The cycle count of TSS remains approximately the same.

The rapid motion content in the "F1 Car" sequence considerably decreases the average PSNR values. In addition, the computational complexities of TSS and other BMAs are further converged. Compared to the "Salesman" sequence, increase in cycle count is the lowest with HEXBS (29%). In this case, quality favors TSS, but HEXBS is the best in terms of speed.

When processing the high-resolution "Pedestrian" sequence, significant increase in cycle count over the "Salesman" sequence exists with DS (81%), CDS (119%), and BBGDS (126%). Instead, HEXBS achieves a reasonable average PSNR value with moderate growth in cycle count (51%). TSS produces the highest search quality. Real-time operating frequencies demanded for HEXBS and TSS are 116 and 166 MHz, respectively.

B. Synthesis Results and Comparison

According to the above analysis, BBGDS is well suited for sequences with small or moderate motion content, TSS for low bit rate purposes due to its moderately good search quality, and HEXBS for fast execution. Hence, the proposed BMA framework is configured to support HEXBS, BBGDS, and TSS operating modes.

The area and timing results based on logic synthesis as well as other characteristics are tabulated for the proposed configuration in the first row of Table II. A 0.13-μm HCMOS9 standard cell library by STMicroelectronics was applied in the synthesis and the results are reported under the nominal operating conditions (1.2 V, 25 °C). The area values (gate count) are based on equivalent 2-input NAND gates, whereas the clock frequency represents the critical path delay in the pipelined architectures. The pipeline registers and all the units except the memory modules are included in the area results. The average cycle counts (avg. t_T/MB) for HEXBS, BBGDS, and TSS are derived from the simulation experiments (Table I).

The proposed architecture (Fig. 5) clocked at 200 MHz is implementable with 14.2 kgates and 2.5 KB SRAM memory when the search range is $[-15, 15]$. At 200 MHz, the power consumption of the implementation is 59 mW, of which 29 mW is consumed by the memories. The BMA control table configured for HEXBS, BBGDS, and TSS costs only 0.4 kgates which is below 3% of the total gate count.

High-resolution sequences would particularly benefit from the larger search range. Enlarging the search range of the proposed architecture clocked at 200 MHz to $[-31, 31]$ increases memory consumption to 6.5 KB and the gate count to 15.2 kgates. The respective metrics with the search range of $[-47, 47]$ are 12.5 KB and 15.5 kgates. For higher speed requirements, the architecture having the search range of $[-15, 15]$ can be accelerated with a synthesis tool (under stricter delay constraints) to 500 MHz at a cost of 16.6 kgates. The power consumption in this case is 123 mW, of which the memories consume 60 mW.

Table II also summarizes the characteristics of the contemporary architectures. The reported results are based on the given search range and a single reference frame. In addition, area metrics for all the architectures are tabulated without memory. The transistor counts reported in [20] and [30] are converted into gate counts by supposing four transistors per a single 2-input NAND gate. Correspondingly, six transistors are assumed per an on-chip SRAM bit.

Compared to Jong's [18] and Chen's [20] BMA-specific architectures, the proposed approach uses over four times larger a search area and still decreases cycle count (t_T/MB) for TSS by 14% and 20%, respectively. In addition, hardware overhead is low. Lakamsani's [19] BMA-specific architecture implements enhanced TSS with a small silicon area, but at a cost of considerably increased cycle count.

Zhang's [26] flexible architecture requires 70% less cycles with TSS, but it uses approximately a quarter of the search area and consumes 69% more logic gates than the proposed architecture. Kuhn's [1] flexible architecture requires approximately the same amount of memory for the same search range than the proposed one. However, Kuhn's architecture uses only a quarter of the available search area with TSS and still suffers from 39% overhead in cycle count. The support for half-pixel motion estimation increases the gate count in Kuhn's approach.

Chao's [27] flexible architecture requires 5.2 k less logic gates but 1 KB more memory than the proposed approach. In addition, the proposed architecture (HEXBS mode) consumes 11% less cycles than Chao's architecture (DS mode). When the architectures process the same "Foreman" sequence (CIF) using the same DS mode, the cycle count of the proposed architecture (Table I) is 8% lower than the respective metric in [27]. Utilizing all the available early termination mechanisms in the proposed architecture would increase the delay gap further. Li's [28] flexible architecture supports sophisticated BMAs, but it has 53–167% higher cycle count, it executes complicated control parts with an RISC processor, and it requires 23% more logic gates than the proposed architecture. Dias' [29] configurable architecture is excluded from Table II since only implementation results on FPGA are available for it. In VIRTEX-II Pro device, it consumes 2213 slices and the operating frequency of 66 MHz is reported to be sufficient for real-time execution of CIF resolution.

Lin's [30] and Gong's [31] programmable architectures require over 110% more logic gates than the proposed approach. Lin's architecture has a fully programmable search range and it has low memory consumption, but cycle count is not reported for any BMA. In Gong's approach, cycle count is separately given for TSS. Although the cycles for half-pixel motion estimation, prediction, and compensation are excluded, it still requires twice the number of cycles with TSS than the proposed approach. In addition, Gong's architecture consumes a lot of memory. Dias' [32] programmable implementation is evaluated with DS. It consumes less area and power than the proposed approach, but with a quarter of the search area it requires approximately 10 times more cycles to execute a BMA than the proposed one.

According to Tables I and II, the proposed architecture can perform fixed block-size motion estimation for full HDTV (1920×1080) sequences in real time (30 fps). A fair overall performance comparison with the other architectures would require that their operating frequencies and voltages are scaled to the 0.13-μm technology. However, although an assumed 43% increase [37] in operating frequency per CMOS process generation would be taken into account, none of the reference programmable architectures would be able to process full HDTV sequences at 30 fps with the examined BMAs (Table I).

The proposed architecture outperforms the reference architectures in terms of performance because of its efficient memory system and SAD unit. Furthermore, the control unit with the parametrizable search strategy control (Fig. 2) and separable search path generation (Fig. 1) makes the architecture adaptive to various search strategies, search paths, and search patterns. Despite the support of multiple BMAs, area cost of the proposed architecture is still close to BMA-specific architectures.

VI. Conclusion

This paper has introduced a motion estimation architecture with a novel BMA framework that is adjustable for a wide range of fast BMAs. Parameters of the BMA framework are responsible for the search path generation and search strategy control of BMAs, which are processed according to the introduced general BMA execution flow. The architecture supports a predefined set of BMAs at run time and provides low latency as well as high throughput for all the mapped BMAs independent of the search strategy, search path, or search patterns. The total execution time of each BMA is almost directly proportional to the number of checking points tested. Hardware utilization is also moderately high since all the processing elements always participate in distortion computation.

Experimental results show that the proposed architecture provides higher performance than the reference BMA-specific, flexible, or programmable architectures. In addition, the silicon area cost is close to BMA-specific architectures. Mapping HEXBS, BBGDS, and TSS to the architecture clocked at 200 MHz costs only 14.2 kgates and 2.5 KB of memory. The architecture is able to process real-time (30 fps) fixed block-size motion estimation (1 reference frame) at full HDTV resolution with all the examined BMAs (BBGDS, DS, CDS, HEXBS, and TSS). Hence, the requested flexibility is provided without sacrificing performance.

Mapping a new or modified BMA to the architecture only requires reparametrization of the BMA framework. Therefore, the proposed architecture can be applied as a reusable IP block in the contemporary and future video encoders.

References

[1] P. Kuhn, *Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation*. Boston, MA: Kluwer, 1999, p. 239.

[2] Y. W. Huang, C. Y. Chen, C. H. Tsai, C. F. Shen, and L. G. Chen, "Survey on block matching motion estimation algorithms and architectures with new results," *J. VLSI Signal Process.*, vol. 42, no. 3, pp. 297–320, Mar. 2006.

[3] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *IEEE Trans. Image Process.*, vol. 4, no. 1, pp. 105–107, Jan. 1995.

[4] X. Q. Gao, C. J. Duanmu, and C. R. Zou, "A multilevel successive elimination algorithm for block matching motion estimation," *IEEE Trans. Image Process.*, vol. 9, no. 3, pp. 501–504, Mar. 2000.

[5] B. Liu and A. Zaccarin, "New fast algorithms for the estimation of block motion vectors," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 3, no. 2, pp. 148–157, Apr. 1993.

[6] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion-compensated interframe coding for video conferencing," in *Proc. Nat. Telecommunication Conf.*, New Orleans, LA, 1981, pp. G5.3.1–5.3.5.

[7] L. K. Liu and E. Feig, "A block-based gradient descent search algorithm for block motion estimation in video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no. 4, pp. 419–422, Aug. 1996.

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

[9] J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, "A novel unrestricted center-biased diamond search algorithm for block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 8, no. 4, pp. 369–377, Aug. 1998.

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

[11] C. H. Cheung and L. M. Po, "A novel cross-diamond search algorithm for fast block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 12, pp. 1168–1177, Dec. 2002.

[12] A. M. Tourapis, O. C. Au, and M. L. Liou, "Highly efficient predictive zonal algorithms for fast block-matching motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 10, pp. 934–947, Oct. 2002.

[13] Z. Chen, J. Xu, Y. He, and J. Zheng, "Fast integer-pel and fractional-pel motion estimation for H.264/AVC," *J. Vis. Commun. Image Represent.*, vol. 17, no. 2, pp. 264–290, Apr. 2006.

[14] W. Choi and B. Jeon, "Hierarchical motion search for H.264 variable-block-size motion compensation," *SPIE Opt. Eng.*, vol. 45, no. 1, pp. 1–9, Jan. 2006.

[15] T. Komarek and P. Pirsch, "Array architectures for block matching algorithms," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 36, no. 10, pp. 1301–1308, Oct. 1989.

[16] L. D. Vos and M. Stegherr, "Parameterizable VLSI architectures for the full-search block-matching algorithm," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 36, no. 10, pp. 1309–1316, Oct. 1989.

[17] K. M. Yang, M. T. Sun, and L. Wu, "A family of VLSI designs for the motion compensation block-matching algorithm," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 36, no. 10, pp. 1317–1325, Oct. 1989.

[18] H. M. Jong, L. G. Chen, and T. D. Chiueh, "Parallel architectures for 3-step hierarchical search block-matching algorithm," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, no. 4, pp. 407–416, Aug. 1994.

[19] P. Lakamsani, "An architecture for enhanced three step search generalized for hierarchical motion estimation algorithms," *IEEE Trans. Consum. Electron.*, vol. 43, no. 2, pp. 221–227, May 1997.

[20] T. H. Chen, "A cost-effective three-step hierarchical search block-matching chip for motion estimation," *IEEE J. Solid-State Circuits*, vol. 33, no. 8, pp. 1253–1258, Aug. 1998.

[21] Y. W. Huang, S. Y. Chien, B. Y. Hsieh, and L. G. Chen, "Global elimination algorithm and architecture design for fast block matching motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 14, no. 6, pp. 898–907, Jun. 2004.

[22] C. Y. Chen, S. Y Chien, Y. W. Huang, T. C. Chen, T. C. Wang, and L. G. Chen, "Analysis and architecture design of variable block-size motion estimation for H.264/AVC," *IEEE Trans. Circuits Syst. I*, vol. 53, no. 3, pp. 578–593, Mar. 2006.

[23] S. Y. Huang, C. Y. Cho, and J. S. Wang, "Adaptive fast block-matching algorithm by switching search patterns for sequences with wide-range motion content," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 11, pp. 1373–1384, Nov. 2005.

[24] Y. S. Jehng, L. G. Chen, and T. D. Chiueh, "An efficient and simple VLSI tree architecture for motion estimation algorithms," *IEEE Trans. Signal Process.*, vol. 41, no. 2, pp. 889–900, Feb. 1993.

[25] S. Dutta and W. Wolf, "A flexible parallel architecture adapted to block-matching motion-estimation algorithms," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no. 1, pp. 74–86, Feb. 1996.

[26] X. D. Zhang and C. Y. Tsui, "An efficient and reconfigurable VLSI architecture for different block matching motion estimation algorithms," in *Proc. IEEE Int. Conf. Acoustic, Speech, and Signal Processing*, Munich, Germany, vol. 1, Apr. 1997, pp. 603–606.

[27] W. M. Chao, C. W. Hsu, Y. C. Chang, and L. G. Chen, "A novel hybrid motion estimator supporting diamond search and fast full search," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, Phoenix-Scottsdale, AZ, May 2002, pp. 492–495.

[28] T. Li, S. Li, and C. Shen, "A novel configurable motion estimation architecture for high-efficiency MPEG-4/H.264 encoding," in *Proc. Asia and South Pacific Design Automation Conf.*, vol. 2, Shanghai, China, Jan. 2005, pp. 1264–1267.

[29] T. Dias, N. Roma, L. Sousa, and M. Ribeiro, "Reconfigurable architectures and processors for real-time video motion estimation," *J. Real-Time Image Process.*, vol. 2, no. 4, pp. 191–205, Dec. 2007.

[30] H. D. Lin, A. Anesko, and B. Petryna, "A 14-Gops programmable motion estimator for H.26X video coding," *IEEE J. Solid-State Circuits*, vol. 31, no. 11, pp. 1742–1750, Nov. 1996.

[31] D. Gong and Y. He, "A new programmable video signal processor for motion estimation and motion compensation," in *Proc. SPIE-VCIP*, vol. 4310, San Jose, CA, Jan. 2001, pp. 920–931.

[32] T. Dias, S. Momcilovic, N. Roma, and L. Sousa, "Adaptive motion estimation processor for autonomous video devices," *EURASIP J. Embed. Syst.*, vol. 2007, no. 1, pp. 1–10, May 2007.

[33] J. Vanne, E. Aho, T. D. Hämäläinen, and K. Kuusilinna, "A parallel memory system for variable block-size motion estimation algorithms," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, no. 4, pp. 538–543, Apr. 2008.

[34] J. Vanne, E. Aho, T. D. Hämäläinen, and K. Kuusilinna, "A high-performance sum of absolute difference implementation for motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 7, pp. 876–883, Jul. 2006.

[35] E. Salminen, T. Kangas, J. Riihimäki, V. Lahtinen, K. Kuusilinna, and T. D. Hämäläinen, "HIBI communication network for system-on-chip," *J. VLSI Signal Process.- Syst. Signal, Image, Video Tech.*, vol. 43, no. 2–3, pp. 185–205, Jun. 2006.

[36] M. Gössel, B. Rebel, and R. Creutzburg, *Memory Architecture & Parallel Access*. Amsterdam, The Netherlands: Elsevier, 1994, p. 246.

[37] S. Borkar, "Design challenges of technology scaling," *IEEE Micro*, vol. 19, no. 4, pp. 23–29, Jul. 1999.

Jarno Vanne (M'02) received the M.S. degree in information technology from Tampere University of Technology (TUT), Tampere, Finland, in 2002.

He is currently pursuing the Ph.D. degree as a research scientist in the DACI research group, the Department of Computer Systems, TUT. His research interests include video coding systems, motion estimation, and parallel memories.

Eero Aho (M'02) received the M.S. degree in electrical engineering and the Ph.D. degree in information technology, from Tampere University of Technology, Tampere, Finland, in 2001 and 2007, respectively.

Currently, he works in the Energy Efficient Computing Group, the Nokia Research Center, Tampere. His research interests include memory systems and parallel processing.

Kimmo Kuusilinna (M'02) received the Ph.D. degree in information science from Tampere University of Technology, Tampere, Finland, in 2001.

He currently works as a Principal Member of Research Staff at the Nokia Research Center, Tampere. His main research interests include system-level design for mobile devices, on-chip interconnections, and parallel memories.

Timo D. Hämäläinen (M'95) received the M.S. and Ph.D. degrees in electrical engineering from Tampere University of Technology (TUT), Tampere, Finland, in 1993 and 1997, respectively.

He has been a Professor in the Department of Computer Systems, TUT, since 2001. He heads the DACI research group that focuses on wireless sensor networks as well as multi-processor system-on-chip architectures, modeling and design tools. He is the author of over 50 journal papers and 180 conference publications and holds several patents on wireless systems.

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FI-33101 Tampere, Finland

ISBN 978-952-15-2628-2
ISSN 1459-2045