Muhammad Shafique · Jörg Henkel

Hardware/Software Architectures for Low-Power Embedded Multimedia Systems

Springer

Hardware/Software Architectures for Low-Power Embedded Multimedia Systems

Muhammad Shafique • Jörg Henkel

Hardware/Software Architectures for Low-Power Embedded Multimedia Systems

Springer

Muhammad Shafique
Karlsruhe Institute of Technology
Haid-und-Neu-Str. 7
76131 Karlsruhe
Deutschland
muhammad.shafique@kit.edu

Jörg Henkel
Karlsruhe Institute of Technology
Haid-und-Neu-Str. 7
76131 Karlsruhe
Deutschland

ISBN 978-1-4419-9691-6 e-ISBN 978-1-4419-9692-3
DOI 10.1007/978-1-4419-9692-3
Springer New York Dordrecht Heidelberg London

Library of Congress Control Number: 2011931865

© Springer Science+Business Media, LLC 2011
All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.
The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Contents

1 Introduction 1
1.1 Trends and Requirements of Advanced Multimedia Systems 2
1.2 Trends and Options for Multimedia Processing 4
1.3 Summary of Challenges and Issues 9
1.4 Contribution of this Monograph 9
1.5 Monograph Outline 13

2 Background and Related Work 15
2.1 Video Coding: Basics and Terminology 15
2.2 The H.264 Advanced Video Codec: A Low-power Perspective 17
2.2.1 Overview of the H.264 Video Encoder and Its Functional Blocks 17
2.2.2 Low-Power Architectures for H.264/AVC Video Encoder 22
2.2.3 Adaptive and Low-Power Design of the Key Functional Blocks of the H.264 Video Encoder: State-of-the-Art and Their Limitations 24
2.3 Reconfigurable Processors 28
2.3.1 Fine-Grained Reconfigurable Fabric 29
2.3.2 Leakage Power of Fine-grained Reconfigurable Fabric and the Power-Shutdown Infrastructure 30
2.3.3 Custom Instructions (CIs): A Reconfigurable Processor Perspective 32
2.3.4 Reconfigurable Instruction Set Processors 33
2.3.5 Rotating Instruction Set Processing Platform (RISPP) 35
2.4 Low-Power Approaches in Reconfigurable Processors 43
2.5 Summary of Related Work 45

3 Adaptive Low-Power Architectures for Embedded Multimedia Systems 49
3.1 Analyzing the Video Coding Application for Energy Consumption and Adaptivity 49
3.1.1 Advanced Video Codecs: Analyzing the Tool Set 51
3.1.2 Energy and Adaptivity Related Issues in H.264/AVC Video Encoder 53
3.2 Energy- and Adaptivity Related Issues for Dynamically Reconfigurable Processors 56
3.3 Overview of the Proposed Architectures and Design Steps 59
3.4 Power Model for Dynamically Reconfigurable Processors 63
3.4.1 Power Consuming Parts of a Computation- and Communication-Infrastructure in a Dynamically Reconfigurable Processor 63
3.4.2 The Proposed Power Model 65
3.5 Summary of Adaptive Low-Power Embedded Multimedia System 67

4 Adaptive Low-Power Video Coding 69
4.1 H.264 Encoder Application Architectural Adaptations for Reconfigurable Processors 69
4.1.1 Basic Application Architectural Adaptations 69
4.1.2 Application Architectural Adaptations for On-Demand Interpolation 72
4.1.3 Application Architectural Adaptations for Reducing the Hardware Pressure 75
4.1.4 Data Flow of the H.264 Encoder Application Architecture with Reduced Hardware Pressure 77
4.2 Designing Low-Power Data Paths and Custom Instructions 80
4.2.1 Designing the Custom Instruction for In-Loop Deblocking Filter 81
4.2.2 Designing the Custom Instructions for Motion Estimation 85
4.2.3 Designing the Custom Instruction for Motion Compensation 86
4.2.4 Area Results for the Custom Instructions of H.264 Encoder 87
4.3 Spatial and Temporal Analysis of Videos Considering Human Visual System 87
4.3.1 HVS-based Macroblock Categorization 92
4.3.2 QP-based Thresholding 93
4.3.3 Summary of Spatial and Temporal Analysis of Videos Considering Human Visual System 95
4.4 An HVS-Based Adaptive Complexity Reduction Scheme 95
4.4.1 Prognostic Early Mode Exclusion 97

4.4.2 Hierarchical Fast Mode Prediction 99
4.4.3 Sequential RDO Mode Elimination 100
4.4.4 Evaluation of the Complexity Reduction Scheme 100
4.5 Energy-Aware Motion Estimation with an Integrated Energy-Budgeting Scheme 104
4.5.1 Adaptive Motion Estimator with Multiple Processing Stages 105
4.5.2 enBudget: The Adaptive Predictive Energy-budgeting Scheme 111
4.5.3 Evaluation of Energy-Aware Motion Estimation with an Integrated Energy-Budgeting Scheme 117
4.5.4 Comparing Adaptive Motion Estimator with and Without the enBudget Scheme 118
4.5.5 Comparing UMHexagonS with and Without the enBudget Scheme 118
4.6 Summary of Low-power Application Architecture 121

5 Adaptive Low-power Reconfigurable Processor Architecture 123
5.1 Motivational Scenario and Problem Identification 123
5.1.1 Summary of the Motivational Scenario and Problem Identification 126
5.2 Run-time Adaptive Energy Management with the Novel Concept of Custom Instruction Set Muting 126
5.2.1 Concept of Muting the Custom Instructions 127
5.2.2 Power-shutdown Infrastructure for the Muted Custom Instructions 128
5.2.3 Run-time Adaptive Energy Management 130
5.2.4 Summary of the Run-time Adaptive Energy Management and CI Muting 132
5.3 Determining an Energy-minimizing Instruction Set 133
5.3.1 Formal Problem Modeling and Energy Benefit Function 133
5.3.2 Algorithm for Choosing CI Implementation Versions 135
5.3.3 Evaluation and Results for Energy-Minimizing Instruction Set 139
5.3.4 Summary of Energy Minimizing Instruction Set 145
5.4 Selective Instruction Set Muting 146
5.4.1 Problem Description and Motivational Scenarios 147
5.4.2 Operational Flow for Selective Instruction Set Muting 148
5.4.3 Analyzing the Energy Benefit Function of Muting 150
5.4.4 Hot Spot Requirement Prediction: Computing Weighting Factors for CIs 152
5.4.5 Evaluation of Selective Instruction Set Muting 153
5.3.6 Summary of Selective Instruction Set Muting 154
5.5 Summary of Adaptive Low-power Reconfigurable Processor Architecture 155

6 Power Measurement of the Reconfigurable Processors 157
6.1 Power Measurement Setup 157
6.2 Measuring the Power of Custom Instructions 158
6.2.1 Flow for Creating the Power Model 158
6.2.2 Test Cases for Power Measurements 160
6.2.3 Results for Power Measurement and Estimation 162
6.3 Measuring the Power of the Reconfiguration Process 164
6.3.1 Power Consumption of EEPROM 165
6.3.2 Power Consumption of the Reconfiguration via ICAP 165
6.4 Summary of the Power Measurement of the Reconfigurable Processors 166

7 Benchmarks and Results 167
7.1 Simulation Conditions and Fairness of the Comparison 168
7.2 Adaptive Low-power Application Architecture 169
7.2.1 Comparing Complexity Reduction Scheme to State-of-the-art and the Exhaustive RDO-MD 169
7.2.2 Comparing the Energy-Aware Motion Estimation with Integrated Energy Budgeting Scheme to State-of-the-art 173
7.3 Adaptive Low-power Processor Architecture 175
7.3.1 Comparing the Adaptive Energy Management Scheme (Without Selective Instruction Set Muting) to RISPP with Performance Maximization [BSH08c] 175
7.3.2 Applying the Adaptive Energy Management Scheme (Without Selective Instruction Set Muting) to Molen [VWG+04] Reconfigurable Processor 176
7.3.3 Comparing the Adaptive Energy Management Scheme (with Selective Instruction Set Muting) to State-of-the-art Hardware-oriented Shutdown 177
7.4 Summary of the Benchmarks and Comparisons 180

8 Conclusion and Outlook 183
8.1 Monograph Summary 183
8.2 Future Work 187

Appendix 191

Bibliography 211

Index 221

Abbreviations and Definitions

ACCoReS	Adaptive Computational Complexity Reduction Scheme
ADI	Arbitrary Directional Intra
ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instruction Set Processor
AVC	Advanced Video Coding
AVS	Advanced Visual Systems
B-MB	Bi-directionally predicted → MB (i.e., a prediction is performed from the previous and future reference frames)
BC	Bus Connector: Connecting a → DPC to the Computation and Communication Infrastructure
BOPF	Buffer Overflow Prevention Factor
BU	Basic Unit, it is a group of → MBs; it defines the granularity at which the rate controller computes a new QP value
CABAC	Context-Adaptive Binary Arithmetic Coding
CAVLC	Context-Adaptive Variable Length Coding
CBR	Constant Bit Rate
CI	Custom Instruction
CIF	Common Intermediate Format (Resolution: 352×288)
CIP	Combined Intra Prediction
cISA	core Instruction Set Architecture: the part of the instruction set that is implemented using the core processor pipeline (i.e., non-reconfigurable); it can be used to implement can be used to implement → Custom Instructions
CLB	Configurable Logic Block: part of an → FPGA, contains multiple → LUTs
CPU	Central Processing Unit
DCSS	Dynamic Clock Supply Stop
DCT	Discrete Cosine Transform
DPC	Data Path Container: a part of the reconfigurable fabric that can be dynamically reconfigured to contain a Data Path, i.e., an elementary hardware accelerator

DVFS	Dynamic Voltage and Frequency Scaling
EAPR	Early Access Partial Reconfiguration
EE	Encoding Engine
EEPROM	Electrically Erasable Programmable Read Only Memory
enBudget	The run-time adaptive Energy Budgeting Scheme
EPZS	Enhanced Predictive Zonal Search
EQ	Energy-Quality
FI	Forecast Instruction: a trigger instruction that indicates a Forecast Block containing a set of → CIs with an information of the compile-time analysis (e.g., expected number of executions)
FIFO	First-In First-Out buffer
FIR	Finite Impulse Response
FME	Fractional-pixel Motion Estimation
FMO	Flexible Macroblock Ordering
FM-CI	Fully-Muted Custom Instruction
FPGA	Field Programmable Gate Array: a reconfigurable device that is composed as an array of → CLBs and switching matrices
FPS	Frames Per Second
FS	Full Search
GOP	Group of Pictures with one I-Frame followed by a series of P- and/or B-Frames
GPP	General Purpose Processor
HDTV	High Definition Television
HD720p	High Definition 720 Lines Progressive Scan (Resolution: 1280×720)
HEVC	High Efficiency Video Coding
HT	Hadamard Transform
HVS	Human Visual System
I-MB	Intra-predicted → MB (i.e., a prediction is performed from the reconstructed pixels of → MBs from the current frame; it is also called *spatial* prediction
I4×4	Macroblock is encoded as Intra with prediction is done at 4×4 block sizes
I16×16	Macroblock is encoded as Intra where the whole 16×16 block is predicted
ICAP	Internal Configuration Access Port
IDCT	Inverse Discrete Cosine Transform
IEC	International Electrotechnical Commission
IHT	Inverse Hadamard Transform
ILP	Integer Linear Programming
IME	Integer-pixel Motion Estimation
IP	Intellectual Property
IPred	Intra Prediction
IQ	Inverse Quantization
ISA	Instruction Set Architecture
ISO	International Organization for Standardization

ISS	Instruction Set Simulator
ITU	International Telecommunication Union
JVT	Joint Video Team
KB	Kilo Byte (also KByte): 1024 Byte
KD	Derivative Gain
KI	Integral Gain
KP	Proportional Gain
LF	Loop Filter
LUT	Look-Up Table: smallest element in an → FPGA, part of a → CLB; configurable as logic or memory
MAD	Mean of Absolute Differences
MB	Mega Byte (also MByte): 1024 → KB
MB	Macroblock, a 16×16 pixel block of a video frame
MBEE	Mean Bit Estimation Error
MC	Motion Compensation
MD	Mode Decision
ME	Motion Estimation
MIPS	Microprocessor without Interlocked Pipeline Stages
MPEG	Motion Picture Experts Group
MPSoC	Multiprocessor System-on-Chip
MSE	Mean Square Error
MV	Motion Vector
MVC	Multiview Video Coding
NM-CI	Non-Muted Custom Instruction
NMOS	N-type Metal-Oxide-Semiconductor Logic
P-MB	Inter-predicted → MB (i.e., a prediction is performed from the reconstructed pixels of → MBs from the previous frame; it is also called *temporal* prediction
P8×8	Macroblock is encoded as Inter with sub-block types sizes of 8×8 or below
P16×16	Macroblock is encoded as Inter where the whole 16×16 block is predicted
PC	Personal Computer
PID	Proportional-Integral-Derivative
PMOS	P-type Metal-Oxide-Semiconductor Logic
PSM	Programmable Switching Matrix
PSNR	Peak signal-to-noise ratio (units: db)
Q	Quantization
QCIF	Quarter Common Intermediate Format (Resolution: 176×144)
QP	Quantization Parameter
RAM	Random Access Memory
RC	Rate Controller
RD	Rate Distortion
RDO	Rate Distortion Optimization
REMiS	Run-time Adaptive Energy Minimization Scheme

RFU	Reconfigurable Functional Unit: denotes a reconfigurable region that can be reconfigured towards a Custom Instruction implementation
RISPP	Rotating Instruction Set Processing Platform
SAD	Sum of Absolute Differences
SATD	Sum of Absolute Transformed Differences
SI	Special Instruction
SIF	Source Input Format (Resolution: 352×240)
SPARC	Scalable Processor Architecture: processor family from Sun Microsystems; used for the → RISPP prototype
SQCIF	Sub-quarter Common Intermediate Format (Resolution: 128×96)
SRAM	Static Random Access Memory
SSE	Sum of Squared Differences
TH	Threshold
UMHexagonS	Unsymmetrical-cross Multi-Hexagon-grid Search
VBR	Variable Bit Rate
VCEG	Video Coding Experts Group
VISA	Virtual Instrument Software Architecture
VLC	Variable Length Coding: a computational kernel that is used in H-264 video encoder
VLIW	Very Large Instruction Word
VM-CI	Virtually-Muted Custom Instruction
XML	Extensible Markup Language
XST	Xilinx Synthesis Technology
YUV	A video format denoting one Luminance (Luma, Y) and two Chrominance (Chroma, UV) Components. A typical resolution given to video encoders is YUV4:20:0, i.e., a sampling method where the two chrominance components have just half the resolution in vertical and horizontal direction as the luminance component

Definitions

Level	define a constraint on key parameters, e.g., specific resolutions and bit rates.
Profile	defines a set of coding tools and algorithms, targeting a specific class of applications.
Residual	Difference of current data to the corresponding prediction data.
Slice	A frame is build up of a number of slices, each containing an integral number of MBs.

List of Figures

Fig. 1.1 Overview of different video services over time 3
Fig. 1.2 **a** Flexibility vs. efficiency comparison of different architectural options; **b** evolution trend of Xilinx Virtex FPGAs 5
Fig. 2.1 An overview of the digital video structure (showing group of pictures, frame, slice, MB) and different video resolutions 16
Fig. 2.2 Functional overview of the H.264/AVC video encoder 18
Fig. 2.3 Variable block sizes for inter-predicted MBs (P-MBs) in H.264/AVC 18
Fig. 2.4 A typical composition of a fine-grained reconfigurable fabric with a 2D-Array of CLBs and PSMs along with the internal details of a Spartan-3 Tile 29
Fig. 2.5 State-of-the-art in power-shutdown infrastructure 31
Fig. 2.6 Extending a standard processor pipeline towards RISPP and the overview of the RISPP run-time system 35
Fig. 2.7 Hierarchical composition of custom instructions: multiple implementation versions exist per custom instruction and demand data paths for realization 36
Fig. 2.8 Example control-flow graph showing forecasts and the corresponding custom instruction executions 40
Fig. 2.9 Execution sequence of forecast and custom instructions with the resulting error back propagation and fine-tuning 41
Fig. 2.10 Overview of the hardware infrastructure for computation (data path container) and communication (bus connector) showing the internal composition of a bus connector 42
Fig. 3.1 Overview of an H.324 video conferencing application with H.264/AVC codec 50
Fig. 3.2 Processing time distribution of different functional blocks in the H.324 video conferencing application 51
Fig. 3.3 Percentage distribution of energy consumption of different functional blocks in the H.264 video encoder 54

Fig. 3.4 Distribution of I-MBs in slow-to-very-high motion scenes (test conditions: group of pictures=IPPP…, CAVLC, quantization parameter=28, 30fps) 57
Fig. 3.5 Overview of the adaptive low-power application and processor architectures 59
Fig. 3.6 Highlighting different steps to be performed at design, compile, and run time at both application and processor levels 61
Fig. 3.7 Power-relevant components of the computation- and communication infrastructure to execute CI implementation versions 64
Fig. 3.8 Example for a custom instruction (CI) implementation version 64
Fig. 4.1 Basic application architectural adaptations to construct the benchmark application. **a** Adapting reference software. **b** Improving data structure. **c** Profiling and Designing Custom Instructions 70
Fig. 4.2 Arrangement of functional blocks in the H.264 encoder benchmark application 71
Fig. 4.3 Number of computed vs. required interpolated MBs for two standard test sequences for mobile devices 72
Fig. 4.4 Distribution of different interpolation cases in the carphone video sequence 73
Fig. 4.5 H.264 encoder application architecture with reduced hardware pressure 74
Fig. 4.6 Data flow diagram of the H.264 encoder application architecture with reduced hardware pressure 78
Fig. 4.7 Description and organization of major data structures 79
Fig. 4.8 Steps to create optimized data paths from the standard formulae 81
Fig. 4.9 4-Pixel edges in one macroblock 82
Fig. 4.10 Pixel samples across a 4×4 block horizontal or vertical boundary 82
Fig. 4.11 Custom instruction for in-loop deblocking filter with example schedule 83
Fig. 4.12 The data paths for filtering conditions and filtering operation constituting the for custom instruction for in-loop deblocking filter 84
Fig. 4.13 Custom instruction for SATD4×4 showing the transform and SAV data paths 85
Fig. 4.14 Custom instruction for motion compensation showing different data paths 86
Fig. 4.15 Mode distribution and video statistics in the 7th frame of American Football 89
Fig. 4.16 Optimal coding mode distribution in rafting and American Football sequences at different Quantization Parameter (QP) values 90

Fig. 4.17 Directional groups with respect to the edge direction angle and notion of spatial and temporal neighboring macroblocks 91
Fig. 4.18 Mode distribution of frame 4 in Rafting sequence using the exhaustive RDO-MD for two different QP values: *Left*: QP=16 and *Right*: QP=38 94
Fig. 4.19 Overview of the adaptive computational complexity reduction scheme (ACCoReS) showing different processing steps and MB categorizations 96
Fig. 4.20 Processing flow of the hierarchical fast mode prediction 99
Fig. 4.21 Percentage mode excluded in ACCoReS for various video sequences 101
Fig. 4.22 Distribution of mode processing for QP=28 101
Fig. 4.23 Comparison of total SAD computations for various video sequences 102
Fig. 4.24 Frame-level in-depth comparison for Susie sequence 102
Fig. 4.25 Frame-level in-depth evaluation of correct mode prediction 103
Fig. 4.26 MB-level mode comparison with the exhaustive RDO-MD: frame 17 of American Football. *Top*: ACCoReS [PSNR=33.28 dB], *Bottom*: Exhaustive RDO-MD [PSNR=34.52 dB] 103
Fig. 4.27 Motion vector difference distribution in Foreman sequence (256 kbps) for various predictors compared to the optimal motion vector (obtained using the full search algorithm) 106
Fig. 4.28 Predictor conditions for motion-dependent early termination 107
Fig. 4.29 Four search patterns used in the adaptive motion estimator and the pixel-decimation patterns for SAD computation 108
Fig. 4.30 Flow of the enBudget scheme for energy-aware motion estimation 111
Fig. 4.31 Energy-Quality (EQ) classes: energy-quality design space exploration showing various pareto points and the pareto curve 113
Fig. 4.32 SAD vs. energy consumption comparison of different motion estimation stages for Foreman sequence 114
Fig. 4.33 Energy and quality comparison for the adaptive motion estimator with and without the enBudget for various video sequences 118
Fig. 4.34 Energy and quality comparison for the UMHexagonS [CZH02] with and without the enBudget for various video sequences 118
Fig. 4.35 Frame-wise energy consumption of the energy-aware motion estimation 119
Fig. 4.36 Macroblock-wise energy consumption map of two exemplary frames in the SusieTableMix_QCIF sequence for a 90 nm technology 120

Fig. 4.37 Energy consumption of the energy-aware motion estimation for various FPGA fabrication technologies for various video sequences 120
Fig. 5.1 Simplified comparison of energy consumption, highlighting the effects of different reconfiguration decisions 125
Fig. 5.2 Infrastructure necessary to exert the proposed CI muting technique 129
Fig. 5.3 Muting the temporarily unused instruction set 129
Fig. 5.4 Overview of the proposed adaptive low-power reconfigurable processor with run-time adaptive energy management along with the design-, compile-, and run-time steps 130
Fig. 5.5 Search space of five CIs with their implementation versions at the corresponding levels and the path of the energy-minimizing instruction set 136
Fig. 5.6 Energy-performance design spaces: evaluation of the energy minimization space using the adaptive energy management scheme under various area and performance constraints for four fabrication technologies for an encoding of 40 QCIF (176×144) frames 141
Fig. 5.7 Comparison of energy components in different fabrication technologies under various area constraints 142
Fig. 5.8 Comparing energy-performance design spaces for different video resolutions when using the energy management scheme under various area and performance constraints for an encoding of 60 video frames 143
Fig. 5.9 CI Execution results for 30 fps on 65 nm showing a detailed breakdown of energy components highlighting the contribution of reconfiguration and leakage energy. The lower graph shows the detailed execution pattern of various CIs executing in different hot spots of the H.264 video encoder along with total energy consumption 144
Fig. 5.10 Comparing the energy requirements of virtually- & fully-muted CIs for two scenarios 147
Fig. 5.11 Time-line showing the execution sequence of hot spots and the situation for a CI muting decision 148
Fig. 5.12 Flow for selecting a muting mode for the custom instruction (CI) set 149
Fig. 5.13 Venn diagram showing the data path requirements of previous, current, upcoming hot spots 149
Fig. 5.14 Calculating the weighting factor for custom instructions w.r.t. the application context 153
Fig. 5.15 Summary of energy benefit of using selective instruction set muting 154
Fig. 6.1 **a** Measurement setup, **b** The in-house developed power supply board 158

Fig. 6.2 Flow for creating the measurement-based power model 159
Fig. 6.3 Test case and setup for measuring the power of an idle (empty) framework 159
Fig. 6.4 Different test cases for measuring the power of different components of a custom instruction (CI) implementation version 160
Fig. 6.5 Connection of FIFO between EEPROM and ICAP 164
Fig. 6.6 **a** EEPROM voltage drop while loading one Data Path Bitstream from EEPROM to FPGA. **b** VCC_{INT} voltage drop for transferring one Data Path bitstream to ICAP and performing the corresponding reconfiguration 165
Fig. 7.1 Comparing the energy savings and quality loss of the ACCoReS with several state-of-the-art fast mode decision schemes 170
Fig. 7.2 Energy savings and quality loss of the ACCoReS compared to the exhaustive RDO-MD for CIF resolution video sequences 170
Fig. 7.3 Energy savings and quality loss of the ACCoReS compared to the exhaustive RDO-MD for QCIF resolution video sequences 171
Fig. 7.4 Comparing the rate distortion curves for QCIF and CIF sequences 172
Fig. 7.5 Power test with a real battery using Mobile sequence 172
Fig. 7.6 Summary of energy savings of the enBudget scheme compared to various fast adaptive motion estimation schemes 173
Fig. 7.7 Comparing energy saving and PSNR loss of the proposed energy-aware motion estimation and the enBudget scheme with various fast adaptive motion estimators. (* negative PSNR loss actually shows the PSNR gain of the scheme) 174
Fig. 7.8 Energy comparison of the AEM_FM and RISPP_PerfMax schemes for 65 nm 175
Fig. 7.9 Average energy comparison of the AEM_FM and RISPP_PerfMax for three technologies 176
Fig. 7.10 Percentage energy saving of Molen [VWG+04] plus AEM_FM over Molen without AEM_FM for three technologies 177
Fig. 7.11 Comparing the energy breakdown of the adaptive energy management scheme (with selective instruction set muting) to [Ge04]-based pre-VM and [MM05]-based pre-FM 178
Fig. 7.12 Energy comparison of the adaptive energy management scheme with [Ge04]-based pre-VM and [MM05]-based pre-FM techniques for varying amount of reconfigurable fabric 179
Fig. 7.13 Energy savings of the adaptive energy management scheme compared to the [Ge04]-based pre-VM technique 180
Fig. A.1 Comparison of produced bits with and without rate control 192

Fig. A.2 The multi-level rate control scheme covering GOP, frame/slice, & BU levels along with image and motion based macroblock prioritization 195
Fig. A.3 Critical Ziegler-Nichols-point for American Football 196
Fig. A.4 Temporal distance based QP calculation for B frames/slices 196
Fig. A.5 Basic unit (BU) level RC with texture and motion based QP adjustments 198
Fig. A.6 RD-curves comparison of the proposed multi-level RC with RC-mode-3 for carphone (QCIF, IPPP) and American Football (SIF, IBBP) 199
Fig. A.7 MBEE comparison of the multi-level RC with three different RC modes 200
Fig. A.8 Frame-wise comparison of the multi-level RC with RC-mode-3 for fast motion combined CIF sequences encoded at 2 Mbps@30 fps 201
Fig. A.9 Frame-wise comparison of the multi-level RC with RC-mode-0 for Susie mixed CIF sequence (*Bright*, *Dark*, *Noisy*) at 2 Mbps@30 fps 201
Fig. A.10 Evaluating the image and motion based MB prioritizations (Note: All excerpts are 2× zoomed using nearest neighbor interpolation) 202
Fig. B.1 Simulation methodology showing various steps of the simulation procedure 203
Fig. B.2 Reconfigurable processor simulator with the extensions implemented in the scope of this monograph for run-time adaptive energy-management 204
Fig. B.3 **a** H.264 video encoder executing on the RISPP prototype; **b** Floorplan of the RISPP prototype implementation on the Xilinx Virtex-4 LX 160 FPGA 206
Fig. B.4 H.264 video encoder executing on the TI' DM6437 DSP board 207
Fig. B.5 Flow for porting H.264 Encoder on DM6437 digital signal processor 208
Fig. C.1 The CES video analyzer tool showing the research framework for motion estimation, video merging, and texture analysis 209

List of Tables

Table 2.1 High-level properties of implementation version and custom instruction 39
Table 3.1 Comparing the coding tool set of various video encoding standards 52
Table 4.1 Custom instructions and data paths for the H.264 video encoder 76
Table 4.2 Implementation results for various data paths of the H.264 video encoder 87
Table 4.3 Thresholds and multiplying factors used in ACCoReS 94
Table 4.4 Summary of PSNR, bit rate, and speedup comparison for various video sequences (each encoded using eight different QPs) 101
Table 4.5 Comparing the video quality of different SAD decimation patterns for encoding of Susie CIF video sequence (30fps@256 kbps) 110
Table 4.6 Configuration and energy consumption for the chosen Energy-Quality (EQ) classes 114
Table 4.7 Coefficients and thresholds used by the algorithm of enBudget in Algorithm 4.4 117
Table 4.8 Performance, area, and energy overhead of enBudget 121
Table 5.1 Various custom instruction (CI) muting modes 128
Table 5.2 Parameters and evaluation conditions with their corresponding reference sources 140
Table 5.3 Hardware implementation results for the energy management scheme on the RISPP prototyping platform (see Fig. 6.1 in Sect. 6.1) 145
Table 6.1 Different placement combinations of two transform Data Paths for power measurement 162
Table 6.2 Measured power results for various data paths & HT4 × 4 implementation versions 162

Table 6.3 Parameters of power model for the CI implementation versions 162
Table 6.4 Power consumption and latencies of different implementation versions (using different amount of DPCs) for various custom instructions for 65 nm and 40 nm technologies 163

List of Algorithms

Algorithm 4.1 The Filtering Process for Boundary Strength = 4 82
Algorithm 4.2 Pseudo-Code of Group-A for Prognostic Early Mode Exclusion 97
Algorithm 4.3 Pseudo-Code of Group-B for Prognostic Early Mode Exclusion 98
Algorithm 4.4 Pseudo code of the Run-Time Adaptive Predictive Energy-Budgeting Scheme 116
Algorithm 5.1 Pseudo code of Determining the Energy Minimizing Instruction Set 137
Algorithm 5.2 Pseudo Code for Finding a Data Path for Virtually-Muting Mode 151

Chapter 1
Introduction

The unremittingly increasing user demands and expectations have fueled the gigantic growth for advanced multimedia services in mobile devices (i.e., embedded multimedia systems). This led to the emergence of high-performance image/video signal processing in such mobile devices that are inherently constrained with limited power/energy availability. On the one hand, advanced multimedia services resulted in the evolution of new multimedia standards with adaptive processing while providing high quality, increased video resolutions, increased user-interactivity, etc. As a result, the next generation applications executing on the embedded multimedia systems exhibit high complexity and consume high energy to fulfill the end-user requirements. On the other hand, the battery capacity in mobile devices is increasing at a significantly slow rate, thus posing serious challenges on the realization of next-generation (highly-complex) multimedia standards on embedded devices. Further parameters that affect the design of an embedded multimedia system are long device charging cycles, cost, short time to market, mass volume production, etc. Besides these constraints and parameters, the intense market competition has created a multi-dimensional pressure on the industry/research to provide innovative hardware/software architectures for *high-performance embedded multimedia systems with low power/energy consumption*. Due to the context-aware processing in the emerging multimedia standards, the need for user-interactivity, and frequent product upgrades (in a short-time-to-market business model) have introduced a new dimension of *run-time adaptivity* to the overall requirements of the emerging embedded multimedia systems in order to react to the run-time changing scenarios (e.g., quality and performance constraints, changing battery levels).

Besides image and graphic processing, video coding is a primitive application of a mobile multimedia system. Advances in video compression standards continue to enable high-end video applications (like video conferencing/video calls, personal video recording, digital TV, internet video streaming, etc.) with high video quality, bigger video resolutions, and lower bit rates on battery-constrained mobile devices. This may lead to a workload of more than 35 G operations per second at a power budget of less than 300 mW [Ber09][1]. Advanced video codecs may consume a sig-

[1] Today's Li-ion batteries provide about 800 mAh at 3.7 V, or nearly 3 W h [Ber09].

M. Shafique, J. Henkel, *Hardware/Software Architectures for Low-Power Embedded Multimedia Systems,*
DOI 10.1007/978-1-4419-9692-3_1, © Springer Science+Business Media LLC 2011

nificant amount of processing time and energy due to their adaptive processing to provide better compression. However, encoding effort highly depends upon the characteristics of the input video sequence and the target bit rates. Therefore, under changing scenarios of input data characteristics and available energy budgets, embedded solutions for video encoding need to consider run-time adaptivity.

1.1 Trends and Requirements of Advanced Multimedia Systems

Typically, mobile multimedia devices range from Laptops (24–49 W/h battery) to Tablets (5–10 W/h battery) to Pocket mobile devices (3–5 W/h battery) [Tex10a]. According a device survey "the Future Image WIRE" [Esp04], the sales growth of camera phones have exploded from 25 to 450 Million units. Typical multimedia applications executing on such mobile devices are:

- **Digital Video:** video calls/conferencing, personal video recording, video playback, digital TV, video pre-/post-processing (like adaptive noise filtering, deinterlacing, edge enhancement), etc.
- **Digital Image:** photography, image processing, etc.
- **Digital Audio:** voice calls, audio recording, audio playback, etc.
- **Games:** game processing, rendering, etc.
- **Display processing:** brightness and contrast adjustment, up-/down-scaling, etc.

The increasing complexity of multimedia applications requires extreme computational capability from the underlying hardware platform. Over the last two decades, the video coding standards have evolved from MPEG-1 to H.264 to Multiview Video Coding for 3D videos. Moreover, the video resolutions have been increased from QCIF (Quarter Common Intermediate Format, 176×144) to SD (Standard Definition, 720×480) to HDTV (High Definition, 1920×1080). A radical increase is foreseen leading towards the UDTV (Ultra high-definition resolutions) to Realistic TVs (see Fig. 1.1) requiring a computational complexity of approximately 10,000× (relative to MPEG-4 QCIF@30fps) [Hui10; MBNN10]. Note, H.264 [ITU05] is one of the latest video coding standards that provides double compression compared to previous coding standards (e.g., MPEG-2, H.263, etc.) at the cost of additional computational complexity and energy consumption ($\sim 10\times$ relative to MPEG-4 advance simple profile [OBL+04]). Besides higher resolutions, the key reason of increasing video coding complexity is the complex tool set of advanced video encoders. The authors in [MBNN10] state an expected increase in the video complexity by 2× every 2 years. Although, high resolutions are mainly targeted for high-end multimedia devices, multiview video conferencing or personal recording at HD (or Quad-HD, 3840×2160 or 4096×2304) resolution is foreseen within next 5 years on mobile devices [Nok10]. Industrial prototypes like [Nok10; WUS+08] have already demonstrated the feasibility of 3D-videos and Multiview Video Coding on mobile devices using two views. In short, realization of advanced video coding of high resolution

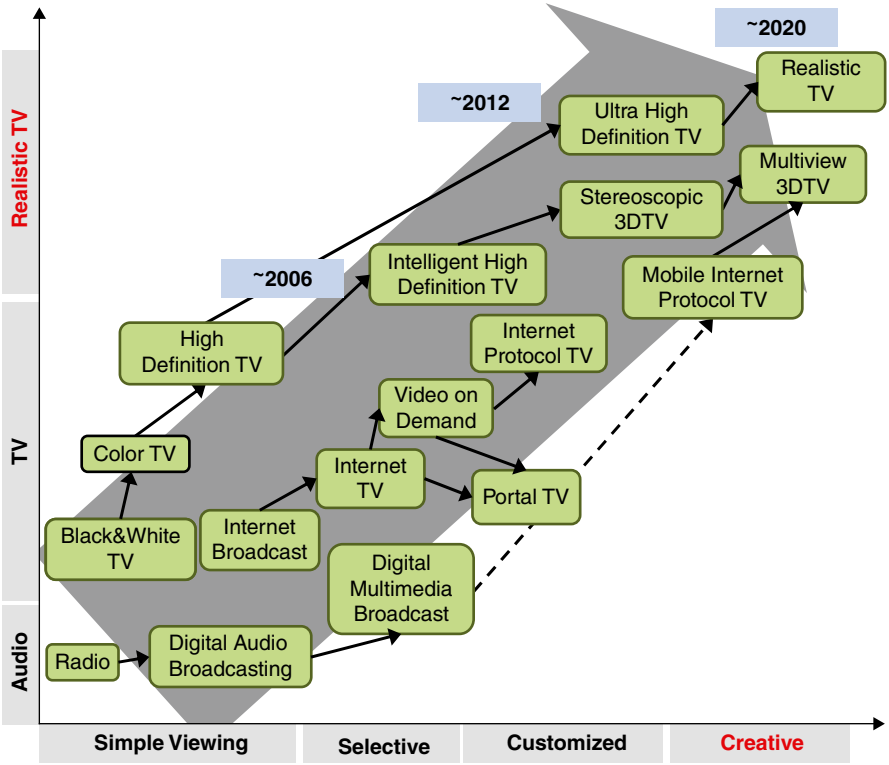


Fig. 1.1 Overview of different video services over time

videos on the battery-powered mobile devices demands high complexity reduction and low power consumption. Moreover, with the evolution of context-aware processing in advanced video coding standards, exploitation of parallelism is becoming more and more challenging [CK08; CLZG06].

Besides the above-discussed issues, *run-time adaptivity* has evolved as an important system attribute to facilitate user interaction and to react to the unpredictable scenarios in order to efficiently utilize the available energy resources. Moreover, scalability to different video resolutions and coding standards (i.e., different video algorithms) is required, which demands for an adaptive architecture for mobile devices.

Summarizing: The fundamental requirements of the next-generation embedded multimedia systems are:

- high performance to support bigger video resolution and higher frame rates
- high video quality at reduced bit rates
- (ultra) low power consumption
- adaptivity to the changing scenarios of available power resources, changing user constraints (different video resolutions, frame rates, bit rates, etc.)

- reduced chip area
- supporting different video format
- supporting multiple video coding standards
- programmability to have quick and easy application upgrades/updates
- reduced cost, (ultra) high production volumes, short time-to-market, and strong market competition

Considering the above-discussed design challenges, for a fast design turnaround time without entire system redesign, *adaptive low-power processor and application architectures* (with the support of hardware acceleration) for embedded multimedia are highly desirable.

1.2 Trends and Options for Multimedia Processing

Figure 1.2 shows traditional embedded approaches like Application Specific Integrated Circuits (ASICs), Digital Signal Processors (DSPs), Application-Specific Instruction Set Processors (ASIPs), and multimedia Multi-Processor System-on-Chip (MPSoCs) [Hen03]. These approaches do not necessarily meet all of the above-mentioned design challenges. Each of these has its own advantages and disadvantages, hence fails to offer a comprehensive solution to next generation complex mobile multimedia applications' requirements.

ASICs target specific applications where the '*performance per area*' and '*performance per power consumption*' can be optimized specifically. However, besides a high initial cost, the design process of ASICs is lengthy and is not an ideal approach considering short time-to-market. Moreover, ASICs lack flexibility (i.e., cannot perform tasks that were not considered while designing that ASIC and implementing modifications/enhancements may result in a costly redesign) and adaptivity, thus hard to adapt to standard evolutions and market/technology induced changes. Let us consider a scenario of video coding as a part of a mobile multimedia system. Advanced video codecs (like H.264 [ITU05], Microsoft VC1 [Mic10a, b], Chinese Audio Video Standard [YCW09]) exhibit a large set of tools to support a variety of scenarios and applications (e.g., low bit-rate video conferencing, high-quality personal video recording, HDTV, etc.). A generic ASIC for all tools is impractical and will be huge in size. In contrast, multiple ASICs for different applications have a longer design time and thus an increased Non-Recurring Engineering (NRE) cost. Moreover, when considering multiple applications (video encoder is just one application) running on one device, programmability is inevitable (e.g., to support task switching). Another use case scenario may be realized when (for example) H.264 video codec is an upgrade to the codec (from a previous generation) in a pre-existing video recording system. ASICs—due to lack of adaptivity and programmability—may perform inefficient or fail to support such scenarios. Therefore, programmable and reconfigurable/adaptive solutions for multimedia (especially video coding) have evolved as an attractive design approach.

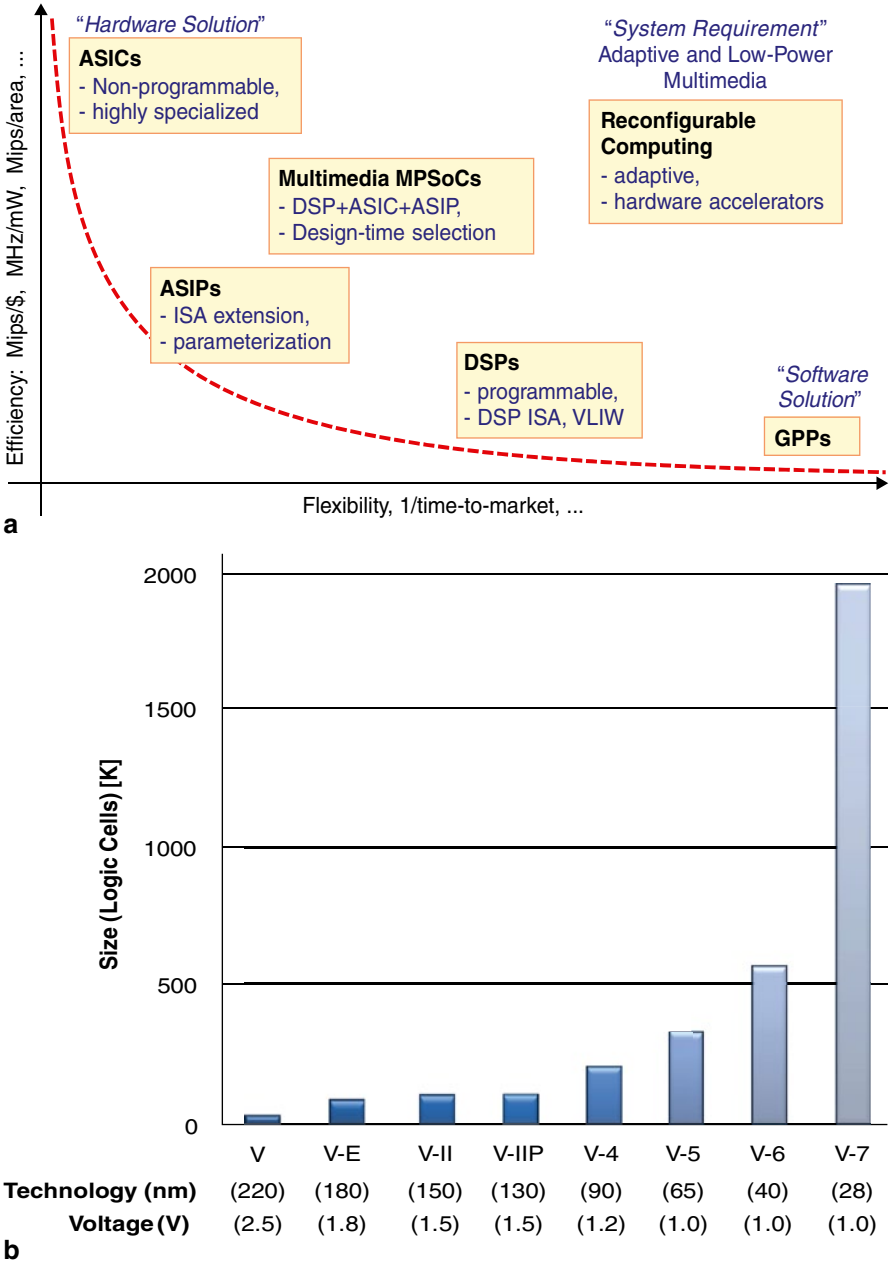


Fig. 1.2 **a** Flexibility vs. efficiency comparison of different architectural options; **b** evolution trend of Xilinx Virtex FPGAs

Unlike ASICs, **DSPs** offer high flexibility and a lower design time. Considering a software-based multimedia system, compared to General Purpose Processors (GPPs), DSPs provide better '*performance per area*' and '*performance per power consumption*'. It is because of their specialized assembly, specialized functional units, and exploitation of instruction level parallelism by using VLIW (Very Long Instruction Word) architecture, i.e., multiple instructions executing in parallel in the same cycle. Commercial solutions are from Philips Nexperia (PNX1500; PNX1700, Nexperia media processor PNX952x family) [Phi10] and from Texas Instruments (DaVinci and OMAP series) [Tex10a, b]. However, DSPs *alone* may not satisfy the power and/or performance challenges when considering the combination of tight power budgets on battery-powered mobile devices and intricate processing nature of next-generation multimedia algorithms. Moreover, DSP performance is limited by the available data bandwidth from the external memory [KRD+03; SHS08]. Although stream architecture [KRD+03] provides an efficient memory hierarchy to exploit the concurrency and data locality, it exploits a limited amount of parallelism (e.g., only data level parallelism) [Ste09]. Therefore, dedicated hardware accelerators are inevitable as they provide a high-degree of instruction and data level parallelism to meet applications' requirements with a limited power budget.

ASIPs overcome the shortcomings of DSPs and ASICs, with an application-specific instruction set that offers a high flexibility (than ASICs) in conjunction with a better efficiency in terms of '*performance per area*' and '*performance per power consumption*' (compared to GPP and DSPs). Tool suites and architectural IP for embedded customizable processors with different attributes are available from major vendors like Tensilica [Ten], CoWare [CoW], ARC [ARC], Stretch [Str], etc. ASIPs may offer a dedicated hardware implementation for *each* application kernel but this typically requires a large silicon footprint. However, for large applications featuring many kernels (instead of a few exposed ones), current ASIP concepts struggle. In fact, customization for many kernels may bloat the initial small processor core to considerably larger sizes (factors of the original core processor). Moreover, while scrutinizing carefully, in complex multimedia applications based on an advanced video codec (like H.264), it was noticed that these kernels are not active at the same time (see detailed analysis in Chap. 3). Still, the sequential execution pattern of the application execution may only utilize a certain portion of the additionally provided hardware accelerators at any time, thus resulting in an inefficient resource utilization and may become power inefficient.

Another trend is **heterogeneous Multimedia MPSoCs** that integrate several programmable processors (GPPs, DSPs), domain-specific weakly programmable coprocessors, and application-specific hardware accelerators (ASIPs, ASICs) using an on-chip communication structure to deliver higher performance. Commercial vendors have transformed their approaches from pure DSPs to multimedia MPSoCs where one or more DSP cores are coupled with programmable ARM cores and/or dedicated hardware accelerators. Prominent examples are Philips Nexperia [Phi10], Nomadik multimedia processor by STMicroelectronics [STM06], and Texas Instruments' DaVinci technology and OMAP processor series [Tex10a, b]. The selection

of cores in an MPSoC is determined at design time depending upon the requirements of a certain set of applications. Therefore, such an MPSoC does not provide the demanded efficiency when executing applications from different domains. Moreover, with a change in the application standard (e.g., a different video coding standard) the currently-used MPSoC becomes obsolete (as it may not fulfill the required performance and power design constraints for the end-product). Therefore, when using state-of-the-art multimedia embedded processors, the performance or power constraints may be achieved in a certain context, but the inability to react to the above-mentioned uncertainties (i.e., changing standards, unpredictable scenarios, and application behavior) and the resulting efficiency (in terms of power, performance, etc.) issues remain.

Field Programmable Gate Arrays (FPGAs) provide a platform solution with low NRE cost, faster time-to-market, and longer product lifetime, thus becoming more popular and mainstream [Te06]. With the continuing evolution of FPGAs (see Fig. 1.2, [Xil10a])[2], various architectures have emerged that embed a reconfigurable fabric (i.e., an embedded FPGA) within a core processor pipeline (e.g., MIPS, SPARC, VLIW) [Ama06; BL00; Bob07; CH02; Har01; HM09; TCW+05; VS07]. These so-called **dynamically reconfigurable processors** bridge the gap between ASICs/ASIPs and DSPs/GPPs by combining the performance and efficiency (due to their capability to exploit high degree of parallelism) of dedicated accelerators[3] (implemented using an embedded FPGA) with a high degree of adaptivity/flexibility (due to their programmability and hardware reconfigurability). The reconfigurable fabric can be reconfigured at run time to contain hardware accelerators, thus allowing a new dimension of adaptivity even after the fabrication and deployment. The adaptive nature of dynamically reconfigurable processors enables:

- feature updates of a given multimedia standard, e.g., a video encoder is enhanced with further coding tools to improve the compression efficiency,
- a standard upgrade, e.g., an existing H.263 video encoder is replaced by a newer version of an H.264 video encoder to provide better compression and quality,
- product upgrade, e.g., new hardware accelerators are loaded to expedite a new multimedia application in the next product release,
- improved product efficiency, e.g., an application designer can design new hardware accelerators of an existing application to expedite more kernels to achieve higher performance or improved video quality (like a new post-processing filter) that are determined by the new user market,
- hardware/software upgrades, e.g., new configuration bitstream of a hardware accelerator may replace the older one in order to provide low power and/or high performance,
- incremental design to cope with time-to-market issues.

[2] An increase of 20× in the logic density over the last 15 years [Xil10a].

[3] These accelerators are similar to those that are deployed by ASIPs.

This flexibility comes at the cost of increased area and power due to the reconfigurability and the structure of an FPGA-like fabric. Besides addressing the inefficient area utilization problem of ASIPs, Dynamically reconfigurable processors overcome the increased area issue by reusing the hardware in time-multiplex, while still providing a high-degree of parallelism. These processors partition their reconfigurable fabric into so-called *containers* that may load hardware accelerators at run time to implement so-called *Custom Instructions* that are then deployed to actually expedite the application's kernels. After the execution of a kernel is completed, the reconfigurable fabric may be allocated to Custom Instructions of other kernels or even to other applications by performing a dynamic reconfiguration of the accelerators. However, the process of reconfiguration incurs additional power overhead and latency (see details in Chap. 6). Moreover, with the evolution of sub-micron fabrication technologies, the consideration of leakage power/energy[4] has become imperative in the energy-aware design of reconfigurable processors.

Exploiting high-degree of parallelism allow dynamically reconfigurable processors to run at lower operating frequencies, thus providing a mean to low power consumption. Consequently, a high-degree of parallelism also corresponds to increased area and power (due to reconfiguration, leakage, and dynamic switching) requirements. Moreover, the execution frequency of the accelerators highly depends upon the input data that may significantly change at run time. Therefore, a tradeoff between the performance and power consumption needs to be evaluated at run time depending upon the system constraints (e.g., available hardware area, required application performance, input data, etc.). Furthermore, adaptivity provides a mean to react to the changing scenarios in order to efficiently exploit the available energy resource (due to changing battery levels).

State-of-the-art approaches in reconfigurable processors have mainly concentrated on improving the performance by reconfiguring application-specific hardware accelerators at run time to meet applications' demands and constraints. These processors lack of efficient energy management features. Lately, power reduction for reconfigurable fabric (like FPGAs) has become a key research interest as it will be discussed in Chap. 2. Similar to the low power approaches in ASICs, hardware shutdown may be performed to reduce the leakage energy of reconfigurable processors considering the usage of the reconfigurable hardware, i.e., statically determining the parts of a reconfigurable fabric to be shutdown. However, due to the adaptive nature and time-multiplexed usage of the reconfigurable fabric, it cannot be determined at compile time which hardware accelerators will be reconfigured on which parts of the reconfigurable fabric. Therefore, state-of-the-art hardware shutdown approaches may perform inefficient in such scenarios as they suffer from the limitation of inflexibility and are highly dependent upon the underlying shutdown policy. This monograph aims at raising the abstraction level of shutdown decision to the instruction set level (see details in Chap. 5) that enables a far higher potential

[4] The key reasons of increased leakage power in the sub-micron fabrication technologies are shorter device/transistor dimensions, reduced threshold voltage, high transistor density, etc.

for leakage energy savings and opens new avenues for researching efficient energy management schemes for dynamically reconfigurable processors.

1.3 Summary of Challenges and Issues

Multimedia systems with advanced video codecs (H.264, Microsoft VC1, etc.) employ a complex tool set to provide better quality/compression at the cost of significantly increased computational processing and power requirements (see details in Chaps. 2 and 3). Moreover, rapid standard evolution, users' demands for higher video resolution, incremental application upgrades on mobile devices pose additional research challenges related to adaptivity and low power consumption. Hence, for designing an adaptive low-power multimedia system there is a need to combat the above-discussed issues at all abstraction levels. Besides employing a low-power device technology (low-power cell library) and operating-system level power management, the low-power and adaptivity related issues need to address at both processor architecture and application architecture levels [FHR+10]. There are several scenarios that cannot be effectively predicted at design-/compile-time. In such scenarios, if an embedded multimedia system is not capable of adapting appropriately, it would give away some of the potential power savings [FHR+10]. Therefore, in order to cope with unpredictable scenarios, the next-generation low-power multimedia systems need to be able to adapt at run time to efficiently utilize the available energy resources, even though adaptivity comes at the cost of a power overhead. A tradeoff between the reconfiguration and leakage reduction needs to be evaluated at run time. This instigates the need for processor architectures with run-time reconfiguration and adaptation of the application architecture to exploit the low-power capabilities of the underlying processor architecture (run-time reconfiguration, high-degree of parallelism, higher abstraction level of power-shutdown, etc.).

This monograph aims at addressing the issues related to adaptivity and low power consumption jointly at processor and application levels under run time varying scenarios of available area, available energy budget, and user constraints. To support processor level adaptivity dynamically reconfigurable processors are used as a target computing platform.

1.4 Contribution of this Monograph

This monograph aims at achieving a high energy efficiency for dynamically reconfigurable processors (and reconfigurable computing in general) enabling *adaptive embedded multimedia systems with low power/energy consumption* to provide means for next-generation mobile multimedia applications and emerging multimedia standards. The **key goals** are to exploit the available potential of energy reduction in dynamically reconfigurable processors while meeting the performance constraint and keeping the video quality degradation unnoticeable, under run-time

varying scenarios (due to changing video properties, available energy resources, user-defined constraints, etc.). This monograph presents novel techniques for adaptive energy management *at both processor architecture and application architecture levels*, such that both hardware and software adapt together in order to minimize the overall energy consumption under design-/compile-time unpredictable scenarios.

The **adaptive low-power processor architecture** employs the novel concept of *Selective Instruction Set Muting* that allows to shun the leakage energy at the abstraction level of Custom Instructions, i.e., an instruction set oriented shutdown. State-of-the-art low-power schemes employ power-shutdown considering the state/usage of the hardware (i.e., a hardware-oriented shutdown) to reduce the leakage power/energy. As discussed earlier, when targeting reconfigurable processors, it cannot be determined at compile time which parts of the instruction set will be reconfigured on which part of the reconfigurable fabric. Therefore, unlike state-of-the-art, the proposed *Selective Instruction Set Muting* raises the abstraction level of shutdown to the instruction set level. Multiple Custom Instruction muting modes are introduced each providing a certain tradeoff between leakage energy saving and reconfiguration energy overhead. The proposed concept relates leakage energy to the execution context of an application, thus enabling a far higher potential for leakage energy savings. The associated potential energy savings have not been exploited by state-of-the-art approaches [CHC03; Ge04; MM05; Te06]. This monograph aims at exploiting this potential. It is especially beneficial for highly flexible Custom Instruction set architectures like in [Bau09; VWG+04]. Moreover, based on the concept of *Selective Instruction Set Muting*, a run-time adaptive energy management scheme investigates the tradeoff between leakage, dynamic, and reconfiguration energy for a given performance constraint, thus dynamically moving in the *energy-performance design space*.

The **adaptive low-power application architecture** employs the novel concept of *Energy-Quality Classes* and video properties dependent *adaptive complexity reduction* in order to realize the *adaptive low-power video encoding*. The proposed *Energy-Quality Classes* represent a particular Motion Estimation configuration that requires a certain energy while providing a certain video quality. It thereby enables a run-time tradeoff between the energy consumption and the resulting video quality.

In particular the novel contribution of this monograph are:

1. **Adaptive Low-power Video Coding Application Architecture:** At the application level, the adaptivity and energy reduction are demonstrated using an advanced video encoder (like H.264). An optimized application architecture is proposed for video encoders targeting dynamically reconfigurable processors. To reduce the energy requirements of different functional blocks of a low-power video encoder at run time, different algorithms have been developed as listed below:

   a. An analysis of spatial and temporal video properties with consideration of important Human-Visual System properties to categorize different video frames and their Macroblocks, such that different energy is spent on the encoding of Macroblocks with different texture and motion properties.

b. An adaptive complexity reduction scheme to reduce energy requirements of encoder by excluding improbable coding modes from the mode-decision process. It solves the issue of choosing the final coding mode out of hundreds of possible combination (without exhaustively searching the design space) by considering the spatial and temporal video properties. Unlike state-of-the-art, this scheme performs an extensive mode-exclusion before fast Mode Decision and Motion Estimation processes, thus providing a significant reduction in the computational complexity and energy consumption.
c. An energy-aware Motion Estimation with integrated energy-budgeting scheme in order to adaptively predict the energy quota for the Motion Estimation (that may consume up to 65% of the total encoding energy). It employs the novel concept of *Energy-Quality Classes* in order to realize the *adaptive low-power video encoding*. Each *Energy-Quality Class* corresponds to a particular Motion Estimation configuration that requires a certain energy while providing a certain video quality. It thereby enables a run-time tradeoff between the energy consumption and the resulting video quality. The energy-budgeting scheme chooses a certain *Energy-Quality Class* for different video frames considering the available energy, video frame characteristics, and user-defined coding constraints while keeping a good video quality.
d. For the blocks that are fixed by the standard and adaptivity is not possible, low-power hardware accelerators were designed.

2. **Novel Concept of Instruction Set Muting:** At processor level, a new way to save energy in dynamically reconfigurable processors is proposed in this monograph that allows to shun the leakage energy at the abstraction level of Custom Instructions. According to an execution context, the Custom Instruction set of a dynamically reconfigurable processor is selectively 'muted' at run time. It thereby relates leakage energy reduction to the execution context of an application, thus enabling a far higher potential for energy savings. The concept employs various so-called 'Custom Instruction muting modes' each leading to particular leakage energy savings. This enables a dynamic tradeoff between 'leakage energy saving' and 'reconfiguration energy overhead' considering the application execution behavior under run-time varying performance and area constraints (e.g., in a multi-tasking environment). Raising the abstraction level to instruction set addresses the above-discussed issues of hardware-oriented shutdown in dynamically reconfigurable processors where it cannot be determined at compile time which parts of the instruction set will be reconfigured on which part of the reconfigurable fabric. The key challenge is to determine which of the muting modes are beneficial for which part of the Custom Instruction set in a specific execution context.
3. **Adaptive Low-power Reconfigurable Processor Architecture:** To exploit the higher potential for energy savings due to the novel concept of *Instruction Set Muting* with multiple muting modes and to provide a high adaptivity (as demanded by multimedia applications with highly video data dependent processing considering changing scenarios of performance and area constraints, avail-

able energy resources, etc.), a run-time energy-management system is required. At the processor level, a run-time adaptive energy management scheme is employed that performs the following steps.

a. *Determine an energy minimizing instruction set:* First, an energy-minimizing instruction set for a dynamically reconfigurable processor is determined (considering leakage, dynamic, and reconfiguration energy) under run-time varying performance and area constraints.
b. *Perform the selective instruction set muting:* After choosing the energy-minimizing instruction set, a decision about the Custom Instruction muting mode is determined for the temporarily unused subset of the instruction set by considering the requirements and execution lengths of the compute-intensive parts of an application (i.e., the execution context of an application). It is determined at run time which subset of Custom Instructions should be put into which muting mode at which time by evaluating at run time the possible associated energy benefit (a joint function of leakage, dynamic, and reconfiguration energy).

For power estimation of dynamically reconfigurable processors, *a comprehensive power model* is developed, which is based on power measurements. Moreover, this monograph presents formal problem description and detailed evaluation of the proposed algorithms at processor and application levels. The superiority of the presented contribution is demonstrated by a fair comparison with state-of-the-art. In addition to the above-discussed scientific contribution, following has been developed in the scope of this work:

- A complete power-measurement setup for dynamically reconfigurable processors that consists of a power supply board, two oscilloscopes, an FPGA based prototyping board, and a control program (running on a laptop/desktop computer) for capturing the measurements from the oscilloscopes.
- A complete H.264 video encoder application with the proposed run-time algorithms and low-complexity data flow. The in-house developed H.264 encoder is currently executing on an in-house dynamically reconfigurable processor prototype [Bau09], Texas Instruments' multimedia processor, and laptops/desktop computers.
- A video analysis tool with an easy-to-use graphical user interface for quick and in-depth analysis of video sequences.

1.5 Monograph Outline

The monograph is outlined as follows: **Chap. 2** discusses the background for video coding (especially the advanced video codec H.264) and prominent related work on low-power encoder design and implementations. Afterwards, the background for (dynamically) reconfigurable processors is presented. The RISPP (Rotating In-

struction Set Processing Platform) dynamically reconfigurable processor [Bau09] is briefly described, which is used for detailed benchmarking of the novel contribution of this monograph. The formal model of *modular* Custom Instructions of RISPP is also discussed that will be used in the subsequent chapters for describing the algorithms of the low-power processor architecture in a clear and precise manner. Afterwards, state-of-the-art related work for different low-power techniques for reconfigurable computing is discussed.

Chapter 3 presents the requirement analysis of Video Coding for energy consumption and adaptivity. A case study of an H.324 Video Conferencing application is presented highlighting the video coding as the most compute-intensive task for mobile multimedia applications. The coding tool set of advanced video codecs is analyzed and common coding tools are explored. Different challenges and issues related to power consumption and adaptivity are discussed in the light of the H.264 video encoder. Afterwards, the overview of the proposed adaptive low-power application and processor architectures is presented along with different steps considered at design, compile, and run time. After discussing how the proposed concept addresses the challenges, a power model for dynamically reconfigurable processors is proposed in the scope of this monograph, which is used by the application and processor level energy management schemes that are proposed in Chaps. 4 and 5.

Chapter 4 presents the first key novel contribution of this monograph in detail, i.e., adaptive low-power video coding. First application architectural adaptations for video encoders are discussed targeting reconfigurable processors followed by the design of low-power Custom Instructions and hardware accelerators. Afterwards, an analysis of spatial and temporal video properties is presented that provides the foundation for adaptive complexity reduction and energy-aware Motion Estimation which are the primitive components of an adaptive low-power video encoder. The concept of adaptively excluding the less-probable coding modes is introduced that significantly reduces the computational requirements of the video encoder, thus saving the energy consumption. For the energy-aware Motion Estimation, the concept of *Energy-Quality Classes* is introduced that provides a run-time tradeoff between the energy consumption and the resulting video quality. A run-time energy-budgeting scheme is presented that allocates an energy quota for the Motion Estimation of different Macroblocks considering their spatial and temporal video properties.

In **Chap. 5** the novel concept of power-shutdown at the instruction set level (i.e., the so-called Custom Instruction muting) is introduced. A power-shutdown infrastructure is discussed that supports the Custom Instruction muting concept with multiple muting modes. Based on this, an adaptive low-power reconfigurable processor architecture is presented that employs a run-time adaptive energy management with *Selective Instruction Set Muting*. It provides a dynamic tradeoff between leakage, dynamic, and reconfiguration energy. Different components of this energy management scheme (i.e., run-time selection of an energy-minimizing instruction and Custom Instruction muting decisions) are discussed in the subsequent sections along with their formal model, algorithms, and evaluation for different fabrication technologies.

The power measurement setup, test cases for power measurements, and steps for creating the power model are explained in **Chap. 6**. Although the evaluation results for different components are already discussed in **Chaps. 4** and **5**, **Chap. 7** provides the detailed comparison of the application and processor level energy management with state-of-the-art. **Chapter 8** concludes this monograph and provides an outlook of the potential future works.

Appendix A briefly discusses the proposed *multi-level rate control*, which compensates the quality degradations that occurred as a result of the above-mentioned energy-aware adaptations. It provides smooth bit allocation which is critical for embedded multimedia systems. **Appendix B** presents the overview of the simulation environment. It further shows the in-house developed H.264 video encoder executing on an in-house dynamically reconfigurable processor prototype [Bau09] and Texas Instruments' multimedia processor. **Appendix C** shows the video analysis tool which was used for the analysis of spatial and temporal properties in Chap. 4.

Chapter 2
Background and Related Work

This monograph envisions adaptive low-power multimedia systems covering both the application and processor perspectives. Besides low power consumption, a special focus is on the support for adaptivity which is inevitable when considering the rapid evolution of the multimedia/video standards and high unpredictability due to user interactions, input data, and inclusion of adaptive algorithms in advanced standards. In order to support adaptivity dynamically reconfigurable processors are considered in this monograph. This chapter provides basics and terminology used in video coding and an overview of the H.264 video encoder which is one of the latest video coding standards. Afterwards, a general background of the reconfigurable processors and their low-power infrastructure is discussed in Sect. 2.3 followed by the prominent related work in dynamically reconfigurable processors and low-power approaches for reconfigurable computing. Especially, the RISPP processor [Bau09] is presented in detail as it is used for detailed benchmarking of the processor-level contribution of this monograph (i.e., adaptive low-power processor architecture).

2.1 Video Coding: Basics and Terminology

Figure 2.1 provides an overview of the structure of a video sequence. A video is composed of a sequence of frames of a scene captured at a certain *frame rate* (given as fps, frames per second) creating a smooth motion perception to the human eye. The basic unit of a video frame is a pixel (also called picture element or pel). The size of a video frame is denoted as its *resolution*, which is given as the number of pixels in one line (frame width, W) and number of lines (frame height, H). Different video resolutions can be seen in Fig. 2.1. Typical resolutions for mobile videos are CIF (Common Intermediate Format) and QCIF (Quarter CIF), while for entertainment quality (like in TVs, multimedia TVs, home cinema, etc.), the resolutions vary from SD (Standard-Definition) to HD (High-Definition).

M. Shafique, J. Henkel, *Hardware/Software Architectures for Low-Power Embedded Multimedia Systems,*
DOI 10.1007/978-1-4419-9692-3_2, © Springer Science+Business Media LLC 2011

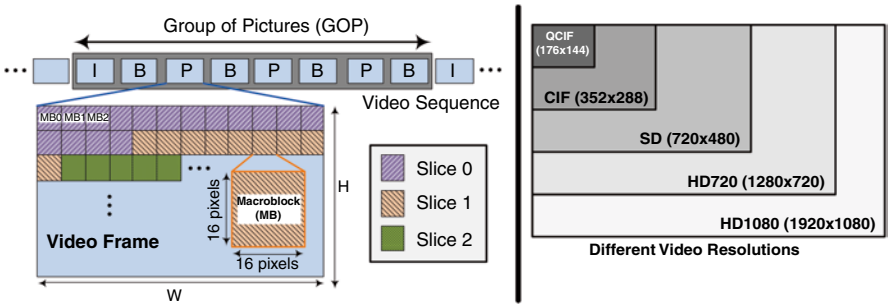


Fig. 2.1 An overview of the digital video structure (showing group of pictures, frame, slice, MB) and different video resolutions

Typically cameras capture a video frame in RGB[1] format which is then converted into YUV[2] (4:4:4) format for video encoding purpose. A video in *YUV format* consists of one *Luminance* (Y, also called Luma) and two *Chrominance* (UV, also called Chroma) components. YUV 4:4:4 denotes a full-sized Y, U, and V components. Since the human eye is more sensitive to brightness compared to the color, typically the Chroma components (U and V) are sub-sampled before encoding to obtain a resolution of YUV 4:2:0 where the size of Y component is W×H and the size of each of the U and V component is W/2×H/2. Note, the sub-sampling of the color components directly corresponds to a 50% reduction in the video data.

All the advanced video encoders are block-based encoders, i.e., the basic processing unit for an encoder is a 16×16 Luma pixel block which is called a Macroblock (MB). A group of MBs is called a *Slice*. A frame can be partitioned into several variable-sized slices (see Fig. 2.1). In an extreme case, one complete frame can also be a single slice. Depending upon the prediction direction, a slice/frame can be categorized as:

- **Intra-Predicted (I) Slice/Frame:** all MBs of this slice/frame are encoded using the *spatial* prediction, i.e., the prediction is performed using the reconstructed pixels of the neighboring MBs in the current slice/frame.
- **Inter-Predicted (P) Slice/Frame:** the MBs may be encoded using the *spatial* prediction or using the *temporal* prediction, i.e., the prediction is performed using the reconstructed pixels of the MBs in the previous slice/frame.
- **Bi-Predicted (B) Slice/Frame:** the MBs may be encoded using the *spatial* prediction or the *temporal* prediction from the previous and/or future slices/frames.

Although P- and B-Frames provide a higher compression compared to the I-Frames, the I-Frames are necessary in periodic intervals in order to provide random access to the video sequence and to avoid the propagation of the prediction error. The group

[1] RGB denotes Red, Green, Blue components of a video frame.

[2] The reason for using YUV space for video coding is its smaller correlation between the color components making the independent encoding of these components easier.

of frames between two I-Frames is called Group of Pictures. Typically a Group of Pictures defines the order of different frame types and the prediction structure. Note: the nomenclature used here is based on the H.264 video coding standard. However, most of the terminology is similar in the previous generations of the coding standards.

2.2 The H.264 Advanced Video Codec: A Low-power Perspective

The limited power resources of current/emerging mobile devices have led to the evolution of power-/energy-aware multimedia. Their video encoders demand huge amount of processing and energy from the underlying hardware, thus pose a challenge on low-cost/low-power embedded systems. In the following, before proceeding the state-of-the-art related work on adaptive and low-power video coding architectures, an overview of the H.264 video encoder which is one of the latest video coding standards.

2.2.1 Overview of the H.264 Video Encoder and Its Functional Blocks

The advanced video coding standard H.264/AVC[3] (Advanced Video Coding) [ITU05] was developed by the Joint Video Team (JVT) of the ITU-T VCEG and ISO/IEC MPEG to provide a bit rate reduction of 50% as compared to MPEG-2 with similar subjective visual quality [WSBL03]. However, this improvement comes at the cost of significantly increased computational complexity (~10× relative to MPEG-4 advance simple profile encoding, ~2× for decoding [OBL+04]), that directly corresponds to high energy consumption. This increased computational complexity and energy consumption of H.264 is mainly due to its complex prediction, Motion Estimation and Rate Distortion Optimized Mode Decision processes that operate on multiple (variable) block sizes (as shown in Fig. 2.3). It thereby poses serious challenges on the low-power encoder realizations for embedded multimedia systems.

Figure 2.2 presents the functional overview of the H.264/AVC video encoder. A sequence of uncompressed video frames in YUV 4:2:0 format is given as the input. Each frame is split into Macroblocks (MBs, i.e., blocks of 16×16 pixels). An MB can be further divided into 16×8, 8×16, or 8×8 blocks (see Fig. 2.3). Each 8×8 block can be further divided into 8×4, 4×8, or 4×4 sub-blocks. Altogether, there

[3] Also called MPEG-4 Part-10 (ISO/IEC 14496-10) or MPEG-4 Advanced Video Coding (AVC).

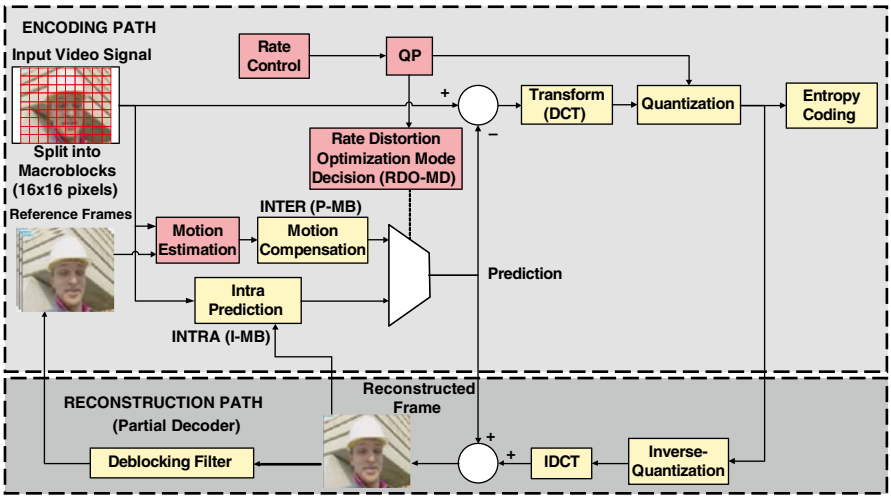


Fig. 2.2 Functional overview of the H.264/AVC video encoder

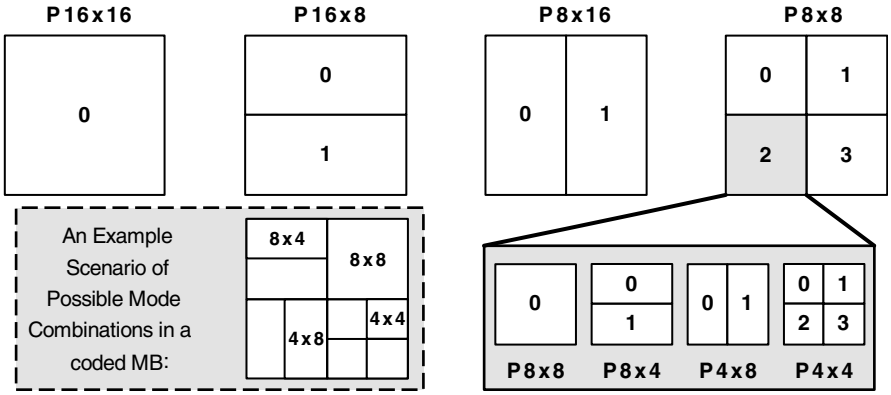


Fig. 2.3 Variable block sizes for inter-predicted MBs (P-MBs) in H.264/AVC

are seven different block types. The MBs of a frame are encoded in a raster scan order using one of the following three MB Types:

- **Intra-Predicted (I-MB):** the MB is encoded using a *spatial* prediction in the current frame.
- **Inter-Predicted (P-MB):** the MB is encoded using a *temporal* prediction from the previous frame.
- **Bi-Predicted (B-MB):** the MB is encoded using a *temporal* prediction from the previous & future frames.

The first frame of a Group of Pictures is called Intra-Frame where all of its MBs are encoded as I-MB. Intra Prediction in H.264 has been enhanced with multiple di-

rectional prediction modes which minimize the predictive error. For the Luminance (Luma, Y) component of an MB, the prediction may be formed for each 4×4 sub-block using nine prediction modes or for the complete MB (i.e., 16×16) with four prediction modes. Two 8×8 Chrominance (Chroma, UV) components are predicted by the same mode (out of 4). Therefore, the total number of Intra mode combinations for a single MB is given as $4*(9*16+4)$ that corresponds to 592 possible mode calculations for only Intra Mode Decision.

Remaining frames of a Group of Pictures are called Inter-Frames where their MBs can be encoded as I-MB or P-MB depending upon the decision of the Rate Distortion Optimized Mode Decision (RDO-MD). For P-MBs, Motion Estimation (ME) is performed for searching the current block in the reference frame (see Fig. 2.2) in order to find out the best match (i.e., the block with minimum distortion). The search is performed in a so-called (pre-defined) search window (a typical size is 33×33 pixels). The ME process consists of two stages: Integer-pixel ME (IME) and Fractional-pixel ME (FME). The IME uses Sum of Absolute Differences (SAD, see Eq. 2.1) to calculate the block distortion for an MB in the current frame (F_t) with respect to an MB in the reference frame (F_{t-1}) at integer pixel resolution.

$$SAD = \sum_{y=0}^{15} \sum_{x=0}^{15} |Current(x, y) - Reference(x, y)| \tag{2.1}$$

Once the best Integer-pixel Motion Vector (MV) is found, the FME stage refines the search to fractional pixel accuracy using Sum of Absolute Transformed Differences (SATD, Eq. 2.2) as the cost function to calculate the block distortion. It performs a 2-D Hadamard Transform (HT) on a 4×4 array of difference values. Compared to SAD, SATD provides a better MV. However, because of high computational load, SATD is only used in the FME stage.

$$SATD = \sum_{y=0}^{4} \sum_{x=0}^{4} |HT_{4\times4}\{Current(x, y) - Reference(x, y)\}| \tag{2.2}$$

HT4×4 is the 2-D 4×4 Hadamard Transform on a matrix D (in case of SATD, it is the differences between current and reference pixel values) and it is defined as:

$$HT_{4\times4} = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} [D] \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) /2 \tag{2.3}$$

Typically ME has to be performed for various block size combinations (altogether 20 different ME combinations per MB are evaluated in RDO-MD [GY05]). An example scenario is presented in Fig. 2.3. As a result, the ME process may consume up to 60% (1 reference frame) and >80% (5 reference frames) of the total encoding time [CZH02]. High computational load makes the ME module not only time

consuming but also energy/power demanding [YWV05]. Considering the variable-sized blocks in I- and P-MBs, each MB can be predicted using one of the following coding modes[4].

$$\begin{aligned} Mode_{P-MB} &\in \{SKIP, P16\times 16, P16\times 8, P8\times 16, P8\times 8, \\ &\qquad P8\times 4, P4\times 8, P4\times 4\} \\ Mode_{I-MB} &\in \{I16\times 16, I4\times 4\} \end{aligned} \tag{2.4}$$

The *exhaustive RDO-MD* in H.264 processes all possible P-MB and I-MB mode combinations in all possible block sizes. Therefore, RDO-MD is the most critical functional block in H.264, as it determines the number of ME iterations which is the most time and energy consuming part. RDO-MD in H.264 employs a Lagrange-based cost function that minimizes the Distortion (D) for a given Rate (R), as given below:

$$J(c, r, Mode|QP) = D(c, r, Mode|QP) + \lambda_{Mode} * R(c, r, Mode|QP) \tag{2.5}$$

'R' is the number of bits required to code the 'Mode' and 'D' is computed using SATD or SAD with respect to the current 'c' and the reference 'r' MBs. λ is the Quantization Parameter (QP)-based Lagrange Multiplier, such that: $\lambda = 0.85 * 2 * (QP - 12)/3$. The mode that provides the best prediction (i.e., minimizes the Eq. 2.5) is chosen as the final coding mode (i.e., the best mode).

For the selected best mode, the prediction data is generated according to the MB Type and the corresponding coding mode. This prediction data needs to be compliant to the standard specifications as the decoder creates an identical prediction for the decoding process. In case of the P-MB the prediction data is generated using Motion Compensation. In case the Motion Vector points to a fractional-pixel position, first, the samples at half-pixel positions (i.e., between the integer-position samples) in the Luma component (Y) of the reference frame are generated using a six-tap filter with weights *[1/32, –5/32, 20/32, 20/32, –5/32, 1/32]*. The samples at quarter-pixel positions are generated by Bilinear Interpolation using two horizontally and/or vertically adjacent half- or integer-pixel positions. This prediction data is subtracted from the current block to calculate the residue. Each 4×4 sub-block of the residue data is then transformed using a 4×4 integer-based 2-D Discrete Cosine Transform (DCT, Eq. 2.6). Note, the 4×4 DCT in H.264 is an integer transform (all operations can be carried out using integer arithmetic), therefore, it ensures zero mismatches between the encoder and the decoder.

$$DCT_{4\times 4} = CXC^T = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} [X] \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \tag{2.6}$$

[4] In this monograph, I8×8 is not considered as it is not used for the mobile devices. However, the contribution of this monograph is scalable to I8×8.

In case of an I-MB, the 16 DC components of an MB (one for each 4×4 block) are further transformed using a 4×4 Hadamard Transform (see Eq. 2.3). In case of Chroma components (U and V), the DC coefficients of each 4×4 block of Chroma coefficients are grouped in a 2×2 block (WDC) and are further transformed using a 2×2 Hadamard Transform as shown in Eq. 2.7.

$$HT_{2\times 2} = \left(\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} [W_{DC}] \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \right) \tag{2.7}$$

Afterwards, the transformed coefficients are quantized according to a QP value determined by a Rate Controller. The Rate Controller regulates the number of produced bits according to a given bit rate. Therefore it determines the QP value which is used for quantization as well as an input to the RDO-MD and ME. The quantized transformed coefficients are finally compressed using a lossless entropy coder. H.264/AVC employs a Context Adaptive Variable Length Coding (CAVLC) or a Context Adaptive Binary Arithmetic Coding (CABAC). In this monograph only CAVLC is considered.

A video encoder contains a model of the decoding process in order to reconstruct the encoded blocks for computing the prediction values for the subsequent blocks and upcoming frames. Therefore, the inverse quantization is performed on the quantized coefficients followed by an inverse transformation stage. The inverse DCT is given by Eq. 2.8 and it is orthogonal to the forward transform, i.e., $T^{-1}(T(X)) = X$.

$$IDCT_{4\times 4} = \left(\begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix} [X] \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{bmatrix} \right) \tag{2.8}$$

The inverse Hadamard Transforms are identical to the forward Hadamard Transforms (Eqs. 2.3, 2.7). After the inverse transformation, the prediction data is added into the inverse transformed values to obtain the reconstructed block. After the complete frame is encoded and reconstructed, H.264/AVC applies an *in-loop Deblocking Filter* on the reconstructed data to reduce blocking distortion by smoothening the block edges. The filtered frame serves as the reference frame, which is used for the Motion Estimation and Compensation of the upcoming frames. Note, for the Intra Prediction (i.e., in case of I-MBs), the prediction is formed using non-filtered values. Further details on the H.264/AVC standard can be found in [Ric03, 10; WSBL03].

It is worth mentioning that only the H.264/AVC decoder is fixed by the standard in order to ensure the compliance of the bitstream and syntax. Therefore, the prediction part, (inverse) transformation, and (inverse) quantization, entropy coding, and the Deblocking Filter need to be standard compliant at the encoder side. However, this leaves sufficient space for researchers and designers to incorporate their ideas in the Motion Estimation, Rate Control, and Mode Decision processes to obtain an efficient encoder application in terms of power and performance.

Now state-of-the-art related work is presented for adaptive and low-power architectures and algorithms for the complete H.264/AVC video encoder and its different functional blocks.

2.2.2 Low-Power Architectures for H.264/AVC Video Encoder

Majority of H.264 encoding solutions target ASIC implementation with a focus on either low power consumption of high resolution system. Few works have also targeted DSP-/ASIP-based and reconfigurable solutions. In the following the prominent related work is discussed for the complete H.264 encoder.

ASIC-Based Encoder Implementations: A hardware design methodology for H.264/AVC video coding system is described in [CLC06]. In this methodology, five major functions are extracted and mapped onto a four stage Macroblock (MB) pipelining structure. Reduction in the internal memory size and bandwidth is also proposed using a hybrid task-pipelining scheme. However, some functional blocks (e.g., Motion Compensation, DCT, and Quantization) are not considered for hardware mapping. The approach in [CCH+06] implements an H.264 encoder with a four-stage Macroblock (MB) level pipeline scheme, a memory hierarchy, and a dual-buffer entropy encoder. The prototype—implemented using UMC 180 nm—requires a relatively large footprint (922.8 KGates and 34.72 KB SRAM), thus resulting in a high power consumption of 581 mW for D1 and 785 mW for HD720p. This power consumption is infeasible for mobile applications according to [EY05]. The authors in [MSH+08] proposed a H.264 codec with high picture quality for mobile applications. They employed a Dynamic Clock Supply Stop (DCSS) system to reduce the power consumption. In order to solve the data dependencies between the Intra Prediction and the reconstruction loop tasks, a prediction scheme is presented that uses the original image instead of the reconstructed one. However, this approach inserts distortion to the final image. For Motion Estimation a *SKIP* algorithm is supported with SATD as the matching criteria. The hardware implementation is able to encode HD720p running at 144 MHz consuming 64 mW with an on-chip memory of 56 KB SRAM. In [CCT+09] a low-power H.264 encoder is proposed for portable devices. However, this work mainly focuses on the Motion Estimation and ignores other functional blocks of the encoder. The variable block size Integer-pixel Motion Estimation is based on the Four Step search and the Fractional-pixel Motion Estimation features the so-called One-Pass algorithm for three block sizes. However, such Motion Estimation search algorithms have a high tendency to trap in the local minima [CZH02; Tou02; YZLS05]. The proposed architecture encodes real-time (30fps) D1 resolution at 54 MHz consuming 43.5–67.2 mW. It uses 452.8 KGates and 16.95 KB SRAM in TSMC 180 nm technology. A partially quality-adjustable H.264 encoder is presented in [CCW+09] to provide fixed power vs. quality tradeoffs. A complete encoder hardware architecture is proposed with a control logic employing four quality modes. The design is implemented with TSMC 130 nm requiring 470 KGates and 13.3 KB SRAM. Although the main claim of this

work is the quality vs. power adaptivity, for higher resolutions it falls back to the lowest quality mode which is complementary to the system requirements, i.e., high resolution encoding typically requires high quality.

ASIP-/DSP-Based and Reconfigurable Encoder Implementations: In [KLHS06] an ASIP featuring Custom Instructions for Deblocking Filter and Intra Prediction are presented. Hardware accelerators for Motion Estimation/Compensation and entropy coding are also provided. A performance reduction of 20–25% is observed. Moreover, a small search range [−16, +15] is used which provides limited rate-distortion results for higher resolutions. The proposed implementation requires 76 KGates when synthesized using a Samsung SEC 180 nm technology. The authors in [SJJL09; ZWFS07] proposed encoding solutions using the TMS320DM642 VLIW processor executing at 600 MHz. Based on their complexity analysis, a Diamond search algorithm is deployed as the Motion Estimation search pattern, which is insufficient to capture high motion. Therefore, it results in a significant quality loss. Various DSP specific optimizations are programmed using the DM642 specialized instruction set for performance improvement. An energy efficient, instruction cell based, dynamically reconfigurable fabric combined with ANSI-C programmability, is presented in [MYN+06]. This architecture claims to combine the flexibility and programmability of DSP with the performance of FPGA. In [LK06] the authors have presented the XPP-S (Extreme Processing Platform-Samsung), an architecture that is enhanced and customized to suit the needs of multimedia application. It introduces a run-time reconfigurable architecture PACT-XPP that replaces the concept of instruction sequencing by configuration sequencing [May04; XPP02]. In [BKD+05; MVM05; VSWM05] the authors have mapped an H.264 decoder onto the ADRES coarse-grained reconfigurable array. In [BKD+05; VSWM05] the authors have targeted IDCT and in [MVM05] Motion Compensation optimizations are proposed using loop coalescing/merging, loop unrolling, etc. However, at the encoder side the scenario is different from that in decoder, because the interpolation for Luma component is performed on frame-level. Although the proposed optimizations in [MVM05] expedite the overall interpolation process, this approach does not avoid the excessive computations for those MBs that lie on integer-pixel boundary. A hardware co-processor for real time H.264 video encoding is presented in [MMFS06]. It provides only Context Adaptive Binary Arithmetic Coding (CABAC) and Motion Estimation in two different co-processors thus offers partial performance improvement.

Summarizing: the above-discussed encoding approaches primarily target ASIC-based solutions that lack flexibility and are not amenable to the standard evolution trends. Moreover, the above-discussed related work lacks run-time adaptivity when considering varying energy budgets and area/performance constraints. One of the key distinctions of the proposed contribution (in this monograph) is to enable the run-time configurability and tradeoff between the energy consumption and achieved video quality for dynamically varying energy budgets and area/performance constraints.

2.2.3 Adaptive and Low-Power Design of the Key Functional Blocks of the H.264 Video Encoder: State-of-the-Art and Their Limitations

A major research effort has been spent on designing individual blocks of the H.264 codec, e.g., Fast Mode Decision, Motion Estimation (ME), and Deblocking Filter. The state-of-the-art related work for the key functional blocks of the H.264 encoder is discussed in the following, highlighting the prominent work in adaptive and low-power algorithms and architectures.

Fast Mode Decision and Complexity Reduction Schemes: As discussed in Sect. 2.2.1, the *exhaustive RDO-MD* in H.264 investigates all possible P-MB and I-MB mode combinations in all possible block sizes to make a decision about the actual coding mode. Therefore, the *exhaustive RDO-MD* process is extremely compute-intensive and practically infeasible in real-world performance and/or power-critical embedded multimedia systems. Note, Mode Decision for P-MB modes is far more complex than that for I-MB modes due to the compute-intensive ME process. This fact becomes critical when after the RDO-MD the final coding mode comes out to be an I-MB mode, thus in this case the complete ME comes out to be unnecessary. To address the limitations of the *exhaustive RDO-MD*, fast RDO-MD schemes are employed. The basic idea of fast RDO-MD scheme is to select a set of coding mode candidates (which is much smaller than the set of all modes) such that the computational requirements of the RDO-MD process are significantly reduced while keeping the visual quality close to that of the *exhaustive RDO-MD*. State-of-the-art fast RDO-MD schemes can be categorized as fast P-MB MD [ADVLN05; GY05; JC04; KC07; LWW+03; PC08; SN06; WSLL07; Yu04], fast *SKIP*[5] MD [JL03], fast I-MB MD [MAWL03; PLR+05], and the combination of the above [ADVLN05; JL03]. These fast RDO-MD schemes either simplify the used cost function or reduce the set of candidate modes iteratively depending upon the output of the previous mode computation. The authors in [JC04] used Mean Absolute Difference of MB to reduce the number of candidate block types in ME. On average, it processes five out of seven block types. The approach in [KC07] uses the RD cost of neighboring MBs to predict the possible coding mode for the current MB. Similar approach is targeted by [PC08; WSLL07] that use the residue texture or residue of current and previously reconstructed MB for fast P-MB Mode Decision. The technique in [Yu04] uses the mode information from previous frame to predict the modes of MBs in the current frame. The technique in [SN06] provides a fast *SKIP* and $P16\times16$ prediction as an early predicted mode option. In [GY05], smoothness and SAD of the current MB are exploited to extend the *SKIP* prediction and exclusion of smaller block mode types. Even if all conditions are satisfied, still 152 out of 168 modes are evaluated, else all modes are evaluated as the *exhaustive RDO-MD*. The authors in [PLR+05] exploited the local edge information by creating an edge

[5] For a *SKIP* Macroblock, encoder does not send any motion and coefficient data and a *SKIP* Macroblock can be completely reconstructed at the decoder side.

map and an edge histogram for fast I-MB Mode Decision. Using this information, only a part of available I-MB modes are chosen for RDO, more precisely 4 instead of 9 $I4\times4$ and 2 out of the 4 $I16\times16$ are processed. The fast I-MB Mode Decision scheme in [MAWL03] uses partial computation of the cost function and selective computation of highly probable modes. $I4\times4$ blocks are down sampled and the predicted cost is compared to variable thresholds to choose the most probable mode. A limited work has been done that jointly performs fast Mode Decision for both I-MB and P-MB. In [AML07], a scalable mode search algorithm is developed where the complexity is adapted jointly by parameters that determine the aggressiveness of an early stop criteria, the number of re-ordered modes searched, and the accuracy of ME steps for the P-MB modes. At the highest complexity point, all P-MB and I-MB modes are processed with highest ME accuracy. The authors in [PYL06] proposed a scalable fast RDO-MD for H.264 that uses the probability distribution of the coded modes. It prioritizes the MB coding modes such that the highly probable modes are tried first, followed by less probable ones.

Most of these state-of-the-art RDO-MD schemes deploy a similar philosophy as they sequentially process mode by mode and exclude the modes depending upon the output of previously evaluated modes, i.e., modes are not excluded in the fast RDO-MD until some ME is done. Therefore, these approaches suffer from a limitation that—in worst case—all possible coding modes are evaluated. In average case, still significant (more than half of all) modes are computed or even in the best case at least one mode from both P-MB and I-MB is processed (see [GY05; JC04]). In any case, ME is always processed, thus the computational requirements of the state-of-the-art are still far too high, which makes them infeasible for low-power embedded multimedia systems. This monograph introduces an *Adaptive Computational Complexity Reduction Scheme* (see Sect. 4.4, p. 95) that addresses these issues by adaptively excluding as many coding modes as possible from the candidate mode set at run time, even before starting the actual fast RDO-MD and ME processes. It thereby provides a significant reduction in the computational complexity and energy consumption of the video encoder.

Once the coding mode is determined and the type is P-MB, the most energy consuming part of an encoder is Motion Estimation. The energy distribution of the encoding process will be discussed in Sect. 3.1 for an optimized video encoder implementation.

Motion Estimation (ME): The complexity and energy consumption of ME is directly proportional to the number of computed SADs to determine the best match (i.e., the MB with the minimum distortion). A *Full Search ME* (i.e., exhaustively searching all possible candidate positions[6] in the search window) provides the optimal match but requires a huge amount of energy (up to 65–90% of total encoding energy [YWV05]). As a result, it is not practicable for real-world applications. Many fast and adaptive ME schemes have been proposed to reduce the computational complexity, such as, *Unsymmetrical-cross Multi-Hexagon-grid Search*

[6] 1089 candidate positions per MB for a search window size of 33×33.

(UMHexagonS) [CZH02], *simple UMHexagonS* [YZLS05], *Enhanced Predictive Zonal Search* (EPZS) [Tou02], etc. However, these ME schemes do not consider available energy/power resources and only stop the search process when the quality constraint is met, thus they always compute a large number of SADs. The computation-aware ME schemes [KXVK06; THLW03; YCL05] stop the search process once the allocated computations are exhausted. Such approaches incorporate a rate-control like mechanism to determine the number of processed SADs and terminate the search once the allocated number of SADs are processed irrespective of whether a good match has been found or not. As a result, these approaches may suffer from severe quality artifacts. Moreover, these approaches are still energy-unaware. The works in [DGHJ05; RB05] provide various VLSI implementations to expedite the process of H.264 ME. Most of these hardware implementations are either suited for *Full Search* or *UMHexagonS*. An ASIP-based approach is considered in [MRS07] but it uses only spatial predictors and does not consider temporal information of the motion field. Moreover, it only uses cross and 3×3 square patterns that take longer to find the optimal Motion Vector (MV) in case of heavy or angular motions. The approach in [HP07] explores fixed and random sub-sampling patterns for computation reduction. The authors in [YWV05] presented power modeling for ME and evaluate it for eight different MEs. Some of the works have targeted the low-power issue in ME [SF04; SLIS07; WSK+07] but they focus on reducing the power either by changing the SAD formula [KSK06] or by eliminating candidates using partial distortion sorting [SLIS07]. Partial distortion sorting is itself an overhead and it excludes only a set of candidates from *Full Search*, which still results in a much larger number of candidates. The authors in [WSK+07] presented an ME scheme based on algorithmic noise tolerance. It uses an estimator based on input sub-sampling but this approach results in degradation especially in case of videos with small objects. Moreover, it uses a three-step search, which traps in local minima, and for modern Motion Estimators it is hard to track motion using sub-sampled input frame. The authors in [CCLR07] introduced a technique to reduce power in video communication by reducing the frame rate but it only works in case of very low motions. Moreover, it incurs a noticeable degradation in the quality of service. The technique in [SF04] exploits input data variations (not the changing levels of available energy) to dynamically configure the search-window size of *Full Search* but does not consider the energy/power level variations. Moreover, it targets *Full Search* which is far more energy consuming than state-of-the-art ME schemes.

State-of-the-art adaptive, fast, low-power, and scalable ME schemes either only consider a fixed quality-constrained solution or offer scalability with fixed termination rules that may lead to severe quality artifacts. These approaches do not provide run-time adaptivity when considering run-time varying energy budgets (i.e., whether there is sufficient energy budget to process the allocated number of SADs or not) and input video sequence characteristics. Additionally, these approaches ignore the user-defined constraints, e.g., required video quality level or encoding duration. As a result, these approaches are less energy-/power-efficient. Varying video sequence characteristics (motion type, scene cuts, etc.) and changing status of available energy budgets (due to a changing battery level or changing allocated energy

in a multi-tasking system) stimulate the need for a run-time adaptive energy-aware Motion Estimation scheme while exhibiting minimal loss in video quality. Note, the available energy budgets may change according to various application scenarios on mobile devices. This monograph introduces an *energy-aware Motion Estimation with integrated adaptive energy-budgeting scheme* (see Sect. 4.5, p. 104) that determines *'how much energy budget should be allocated to the Motion Estimation of one video frame or even one Macroblock when considering run-time varying scenarios'* while keeping a good video quality.

The proposed scheme is different from the above-discussed state-of-the-art as it comprehensively explores the tradeoff related to the energy consumption and video quality loss while considering the run-time varying scenarios. Unlike the above-discussed approaches (like [CCLR07; KSK06; SF04; SLIS07; WSK+07]), the proposed ME scheme moves in the energy-quality design space at run-time using the novel concept of *Energy-Quality Classes*, each requiring a different energy consumption and providing a different visual quality. These *Energy-Quality Classes* are selected at run time depending upon the available energy and user-defined controls (e.g., frame rate) to enable energy-aware adaptivity, that has not been targeted by others before. Moreover, novel search patterns are deployed that captures the large angular/irregular motions and further refines the motion search in close vicinity.

In-Loop Deblocking Filter: As the Deblocking Filter algorithm is fixed by the standard [ITU05], the key research focus in the Deblocking Filter is low-power, area-efficient, or high-throughput hardware acceleration. The approach in [MC07] uses a $2\times4\times4$ internal buffer and 32×16 internal SRAM for buffering the filtering operations with I/O bandwidth of 32-bits. All filtering options are calculated in parallel while the filtering conditions are computed in a control unit. This approach uses 1-D reconfigurable FIR filter (8 pixels in and 8 pixels out) but does not target the optimizations of actual filter Data Path. It requires 232 cycles to filter one MB. The authors in [SCL06] introduced a five-stage pipelined filter using two local memories. This approach suffers from the overhead of multiplexers to avoid pipeline hazards. It costs 20.9 K gate equivalents for 180 nm technology and requires 214–246 cycles/MB. A fast Deblocking Filter is presented in [PH06] that uses a Data Path, a control unit, an address generator, one 384×8 register file, two dual port internal SRAMs to store partially filtered pixels, and two buffers (input and output filtered pixels). The filter Data Path is implemented as a two-stage pipeline. The first pipeline stage includes one 12-bit adder and two shifters to perform numerical calculations like multiplication and addition. The second pipeline stage includes one 12-bit comparator, several two's complementers and multiplexers to determine conditional branch results. In worst case, this technique takes 6144 clock cycles to filter one MB. A pipelined architecture for the Deblocking Filter is illustrated in [CC07] that incorporates a modified processing order for filtering and simultaneously processes horizontal and vertical filtering. The performance improvement majorly comes from the reordering pattern. For 180 nm synthesis this approach costs 20.84 K gate equivalents and requires 192 (memory)+160 (processing) cycles. The authors in [AKL+07] mapped the H.264 Deblocking Filter on

the ADRES coarse-grained reconfigurable array [BKD+05; VSWM05]. It achieves 1.15× and 3× speedup for overall filtering and kernel processing, respectively.

The Deblocking Filter (Sect. 4.2.1) approach proposed in this monograph is different from the above approaches because it targets first the optimization of core filtering Data Paths in order to reduce the total number of primitive operations in one filtering. In addition to this, all conditions in one Data Path are collapsed and two generic conditions are calculated that decide the filtering output. A parallel scheme for filtering one 4-pixel edge is incorporated. A latest technique in the Deblocking Filter inspired from the proposed approach is presented in [NWKS09]. Similar to the proposed approach, the technique of [NWKS09] also performs operation reduction for the area reduction and high throughput. It additionally gates the clock of the unused processing units to reduce the dynamic power. Since this technique is synthesized using a 180 nm technology, the leakage power factor is not considered.

2.3 Reconfigurable Processors

In order to enable run-time adaptivity at the processor level, dynamically reconfigurable processors are deployed as the target platform in this monograph. These processors embed a reconfigurable fabric within a core pipeline (MIPS, SPARC, VLIW, etc.). Depending upon its reconfiguration granularity, a fabric can be categorized into coarse- and fine-grained reconfigurable fabric. A coarse-grained reconfigurable fabric consists of an array of word-level reconfigurable Arithmetic Logic Units (ALUs). It is amenable to data-flow dominant application kernels with word-level processing (typically 16 or 32 bit). A fine-grained reconfigurable fabric employs Look-Up Tables (LUTs) with bit-level reconfigurability. A typical example of such fabric is Field Programmable Gate Arrays (FPGAs). Fine-grained reconfigurable fabrics are amenable to highly-parallel processing of byte-level operations (as typically required in image and video processing), state machines (sub-byte level), and bit-level operations (e.g., bit-shuffling, packing and merging operations, condition computations, etc.). A detailed discussion on these approaches can be found in [VS07]. Detailed surveys and overview of different approaches in reconfigurable computing are provided in [Ama06; BL00; Bob07; CH02; Har01; HM09; TCW+05; VS07].

In the scope of this monograph, the focus is placed on dynamically reconfigurable processors with a fine-grained reconfigurable fabric. These processors provide a high adaptivity and flexibility (due to their hardware reconfigurability and programmability) combined with the performance and efficiency of dedicated hardware accelerators (by exploiting a high degree of parallelism using an embedded FPGA). Now the basic structure of a fine-grained reconfigurable fabric will be explained.

2.3.1 Fine-Grained Reconfigurable Fabric

Figure 2.4 illustrates the internal structure of a fine-grained reconfigurable fabric that consists of Configurable Logic Blocks (CLBs) and Programmable Switching Matrices (PSMs) connected with dedicated wires [Te06; Xil08a]. The internal details are shown for a Xilinx Spartan FPGA. A CLB-PSM pair is typically referred as

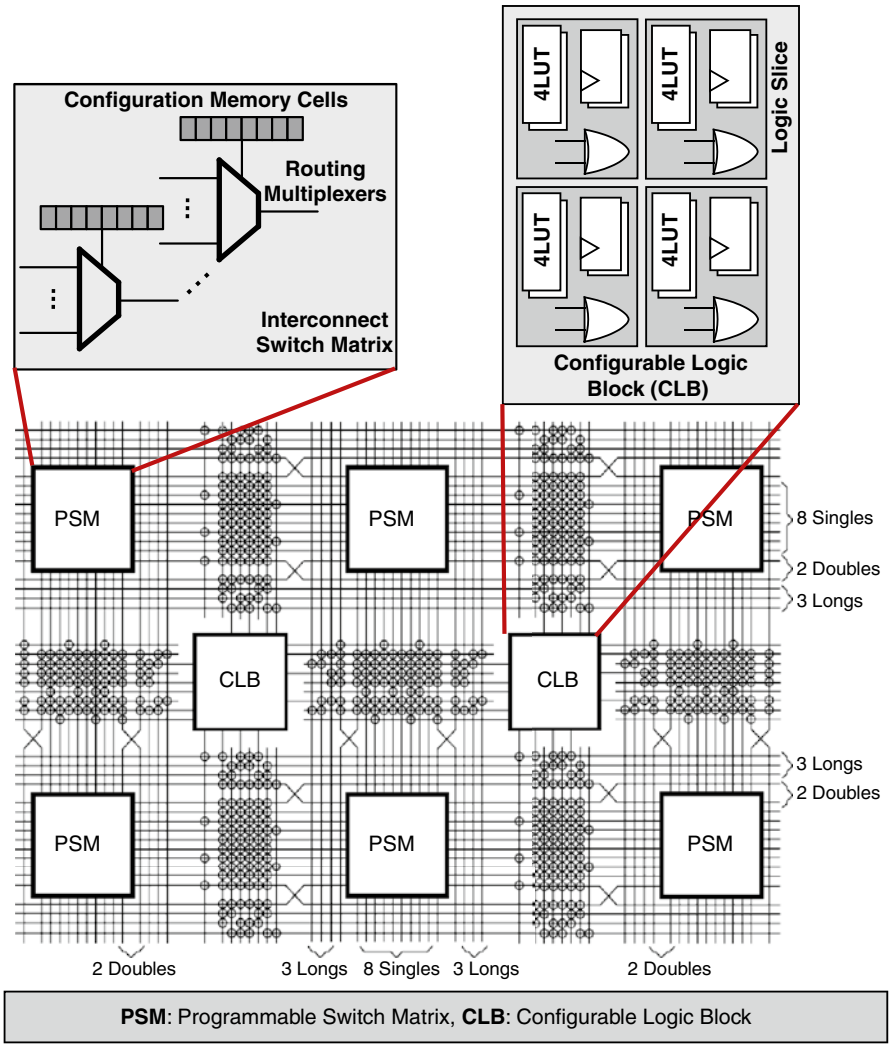


Fig. 2.4 A typical composition of a fine-grained reconfigurable fabric with a 2D-Array of CLBs and PSMs along with the internal details of a Spartan-3 Tile

a *Tile*. The CLBs consist of Look-Up Tables (LUTs) and flip flops, while the PSMs constitute configurable interconnects and routing multiplexers[7]. Such a fine-grained reconfigurable fabric is efficient in implementing bit-/byte-level operations, control logic (finite state machines), and small memories. The fabric can be reconfigured at run time to implement different hardware accelerators. A case where only a region of the fabric is reconfigured at run time is referred as *partial run-time reconfiguration*[8]. In this case, the regions that are not reconfigured remain active and functional. Typically a reconfiguration is performed via an on-chip mechanism for accessing the configuration memory. In case of Xilinx Virtex FPGAs, it is the Internal Configuration Access Port (ICAP [Xil09]). The reconfiguration latency depends upon the size of the configuration data and the reconfiguration bandwidth (for a reconfiguration bandwidth of 36 MB/s and a 40 KB configuration data, i.e., Data Path bitstream, the reconfiguration latency corresponds to **54,254** cycles at 50 MHz).

2.3.2 Leakage Power of Fine-grained Reconfigurable Fabric and the Power-Shutdown Infrastructure

Recently, low-power design, especially the leakage power reduction, has become a key research focus in reconfigurable computing. An overview of low-power techniques for FPGAs is presented in [LL08]. A detailed analysis of leakage power of the fine-grained reconfigurable fabric in Xilinx FPGAs is performed in [TL03], highlighting the significance of leakage reduction in FPGAs especially when considering mobile devices. An analysis of dynamic power consumption in Virtex-II FPGAs is presented in [SKB02]. An FPGA architecture evaluation framework for power efficiency analysis is presented in [LCHC03] predicting leakage to be dominant for future technologies.

Lately, **power-gating** (i.e., hardware-oriented power-shutdown) has been introduced in FPGAs to reduce the leakage power by switching-off the power supply to the reconfigurable regions with the help of high-V_t mid-oxide sleep transistor [Ge04; Te06]. Besides an area overhead, sleep transistors typically introduce a performance penalty due to their on-resistance in case the circuit is active and operating. Therefore, the granularity of the power-shutdown (i.e., the smallest hardware block that can be independently shutdown) is one of the main design decisions. In the following, different power-shutdown infrastructures are discussed for the fine-grained reconfigurable fabric.

The hardware-oriented power-shutdown scheme of [CHC03] uses three sleep transistors for CLBs and one for routing resources in order to obtain a fine-grained leakage control. It employs a power-shutdown of each independent LUT, flip flop, and routing switch. Such an approach provides a relatively higher power saving at

[7] Further details on the Xilinx internal structure and its usage can be found in [Xil07].

[8] Most of the architectures use the Xilinx FPGAs and tools [LBM+06] to prototype partial run-time reconfiguration.

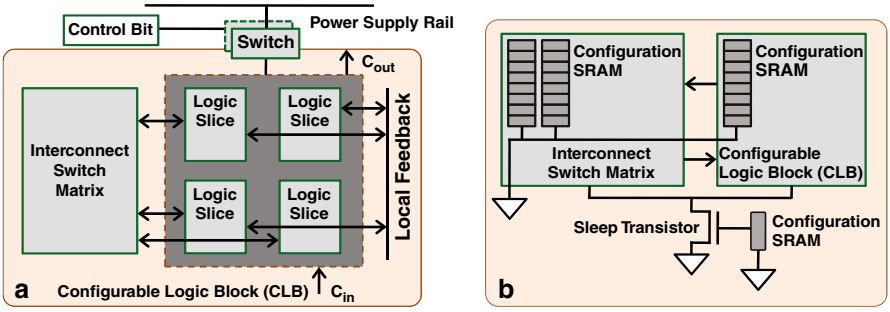


Fig. 2.5 State-of-the-art in power-shutdown infrastructure

the cost of significantly increased area overhead. The approach of [Ge04] targets power-shutdown of clusters of several tiles (see Fig. 2.5a, [Ge04]) that may be controlled by a single sleep transistor in order to provide leakage power reduction while keeping the area overhead low [PSB05; Te06]. However, in this case the number of powered-off tiles is reduced, as some clusters are 'partially used', i.e., they contain tiles that are used (thus need to remain active) and other tiles unused (thus could be powered-off). Therefore, when considering a cluster-level power-shutdown, such 'partially used' clusters cannot be shutdown and a relatively lower power savings are obtained. To address this issue, the approach of [Ge04] proposes a region-constrained placement to increase the number of clusters that be shutdown.

Xilinx research labs introduced the *Pika* low-power FPGA (90 nm) for battery-powered applications that supports voltage scaling, hardware power-shutdown, and a low-leakage Configuration SRAM [Te06]. The power-shutdown infrastructure of *Pika* provides a compromise between [CHC03] and [Ge04] by providing sleep transistors at the level of individual tiles (see Fig. 2.5b) [Te06]. The authors showed that (on average) 25% tiles are idle when evaluated for over 100 benchmarks. NMOS sleep transistors are considered in *Pika* as they provide better speed characteristics compared to the PMOS sleep transistors. Unlike using the thin-oxide high-V_t transistors for power-shutdown [MSM+96] that incurs high leakage at 90 nm (and below) technology nodes, *Pika* employs mid-oxide sleep transistors. A design space exploration is performed in [Te06] to select a transistor size by exploring the power and delay behavior of various transistor designs. Mid-oxide sleep transistor has been selected for the *Pika* FPGA as it provides a leakage reduction by over 1000× at the cost of 10% performance degradation. *Pika* incurs an 8% area increase due to their power-shutdown infrastructure [Te06]. Kindly note that Xilinx has not yet introduced such an infrastructure in their commercial products, however, it is envisaged to be in their future product lines.

Note, the above-presented approaches [CHC03; Ge04; Te06] only shutdown the logic area (i.e., CLBs and PSMs) while keeping the Configuration SRAM powered-on to retain the configuration state. These approaches consider a low-leakage SRAM with high-V_t transistors. The authors in [LLH07] showed that high-V_t transistors for leakage reduction in the Configuration SRAM result in increased SRAM

write time (i.e., increased reconfiguration time), that leads to a higher reconfiguration energy. The authors in [MM05] proposed fine-grained leakage optimization by shutting down the Configuration SRAM of unused LUTs. A hardware-oriented power-shutdown technique with four sleep modes (achieved by applying different bias to the footer device) is proposed in [Ae06] to provide a tradeoff between wakeup overhead and leakage savings.

In order to activate the circuit, sleep transistors (requiring wakeup time and wakeup energy) are switched-on. The Xilinx *Pika* project demonstrated a study of wakeup characteristics for the mid-oxide sleep transistors where the wakeup time is given as approximately 100 ns. A comparison of energy consumption is performed for active and standby modes. This study demonstrates that a circuit is beneficial to shutdown if the minimum sleep period is less than 2 μs in order to amortize the wakeup energy.

2.3.3 Custom Instructions (CIs): A Reconfigurable Processor Perspective

A Custom Instruction (CI) is an assembly instruction that implements the functionality of a compute-intensive kernel of an application for its accelerated execution. As discussed earlier, reconfigurable processors deploy so-called CIs (composed of hardware accelerators, see Sect. 2.3.5) in order to expedite application's computational hot spots[9]. Typically, a tool flow is used for designing hardware accelerators and CIs for fine-grained reconfigurable processors, which is similar to that for ASIPs/extensible processors. Therefore, the related work for automatic detection and high-level synthesis of CIs and hardware accelerators from extensible processors can be used for creating reconfigurable CIs, too. For example, in case of H.264 video encoder, CIs can be designed to accelerate the *Discrete Cosine Transform*, *Sum of Absolute Difference* (SAD) for Motion Estimation, *In-Loop Deblocking Filter*, etc.

Reconfigurable processors partition their reconfigurable fabric into so-called *Reconfigurable Functional Units* (RFUs, connected to the core processor pipeline) that are reconfigured at run time to (re-)load CI implementations for a hot spot. After the execution of a hot spot is completed, the reconfigurable fabric may be allocated to the CIs of other hot spots, thus demanding a dynamic reconfiguration of the RFUs, which may consume a noticeable amount of energy due to the reconfiguration process. Therefore, unlike extensible processors that statically provide all CIs, reconfigurable processors use the fine-grained reconfigurable fabric in a time-multiplexed manner to dynamically implement CIs of different hot spots of an application or even CIs of different applications from diverse domains. To be able to reconfigure any CI into any RFU, the interface between the core pipeline and

[9] Throughout this paper, a *hot spot* denotes a *computational hot spot* that contains compute-intensive application parts (i.e., kernels).

all RFUs is identical. So-called prefetching instructions are inserted to trigger the upcoming reconfigurations [LH02]. Further details on reconfigurable CIs and the differences between CIs for reconfigurable and extensible processors can be found in [Bau09].

In the following, prominent state-of-the-art reconfigurable processors are discussed (in chronological order) that can benefit from the novel contribution of this monograph.

2.3.4 Reconfigurable Instruction Set Processors

The **OneChip and OneChip98** projects [WC96; CC01; JC99] couple RFUs with the core processor pipeline for implementing multi-cycle CIs. RFUs may access the main memory, while the core processor pipeline continues executing, in order to expedite streaming applications [CC01]. However, it may result in memory inconsistencies. The OneChip98 project proposed a hardware support to automatically resolve nine different types of memory inconsistencies. The RFUs provide six configuration contexts which are loaded using the direct memory access.

The **CoMPARE** processor [SGS98] couples the pipeline with an RFU having four inputs and two outputs. The RFU cannot implement a state-machine or data-feedback logic as it does not provide registers or latches. Only one CI can be reconfigured in the RFU. Therefore, if more than one CIs are executing within a computational hot spot, reconfiguration are performed during their execution.

The **Proteus** processor [Dal99, 03] couples the core processor pipeline with a fine-grained reconfigurable fabric divided into multiple RFUs each containing one CI at a time. The main research focus of Proteus is on the Operating System support considering task switching and the opcode management for CIs in a multi-tasking environment. Shared CIs among different tasks are assigned the same opcode. In case of tasks with disparate processing behavior and insufficient number of available RFUs, some CIs execute in software thus resulting in a steep performance degradation.

The **Molen** processor [PBV06, 07; VWG+04] couples a core processor pipeline with a fine-grained reconfigurable fabric via a dual-port register file and an arbiter for shared memory. These dedicated exchange registers are used to provide parameters to the CIs. Additional control instructions are provided to manage the reconfiguration requests and CI executions. The decision of run-time reconfiguration is determined at compile time using these control instructions. A generic instruction is used to execute all CIs. The address of the configuration bitstream for the demanded CI is provided to this generic instruction.

Bauer [Bau09] identified three different problems for state-of-the-art reconfigurable processors (e.g., Proteus [Dal99, 03] and Molen [PBV06, 07; VWG+04]).

1. **Data Path Sharing Problem:** The above-mentioned reconfigurable processors employ *monolithic* CIs, i.e., a CI is implemented as a dedicated hardware block using the reconfigurable fabric. These CIs may be either entirely loaded onto the

reconfigurable fabric or not at all. As a result, sharing of common Data Paths is not possible in these reconfigurable processors. The concept of *monolithic* CIs has two additional drawbacks as given below.

2. **RFU Fragmentation Problem:** In order to dynamically reconfigure CI implementations into any RFU, all RFUs need to be of equally-sized and share a common connection interface. If an CI implementation is smaller than the RFU, the remaining resources within the RFU cannot be used to implement another CI. Therefore, it leads to a noticeable fragmentation which potentially results in increased area requirements.
3. **Longer Reconfiguration Time Problem:** Although *monolithic* CIs provide higher performance (due to more parallelism), they result in significantly longer reconfiguration time due to more hardware. Depending upon the expected number of CI executions and the reconfiguration frequency, the reconfiguration overhead may degrade the potential performance improvement. Reconfigurable processor typically address this problem by: (a) prefetching CI implementations to start the corresponding reconfigurations as early as possible [LH02], and/or (b) using core Instruction Set Architecture (cISA) to execute a CI when it is not available in the RFUs.

Bauer [Bau09] proposed the **Rotating Instruction Set Processing Platform (RISPP)** that—instead of *monolithic* CIs—employs *modular* CIs based on a hierarchical composition. These *modular* CIs are composed of elementary Data Paths (as hardware accelerators) and support multiple Implementation Versions per CI. It addresses the above-discussed three problems of sharing, fragmentation, and longer reconfiguration time. The concept of *modular* CIs enables an efficient utilization of the reconfigurable fabric. It provides the potential for a higher adaptivity and efficiency than *monolithic* CIs. RISPP incorporates a **run-time system** and a **hardware infrastructure** (for computation and communication) to utilize the potential of *modular* CIs in order to increase the efficiency and performance.

Note that *monolithic* CIs are actually a special case of *modular* CIs, i.e., they provide exactly one implementation per CI which is implemented using one large Data Path. Therefore, in subsequent chapters, RISPP is used to motivate and apply adaptive low-power processor and application architectures and run-time energy-management. RISPP is a more sophisticated and advanced reconfigurable processor compared to previous state-of-the-art (like Molen and Proteus). However, for the evaluation, dynamically reconfigurable processors with both *modular* and *monolithic* CIs are used, i.e., RISPP and Molen.

Before discussing the concepts and techniques for low-power approaches in reconfigurable computing, in the following, an overview of RISPP is provided to a level of detail necessary to understand the novel contribution of this monograph (for further details please refer to Bauer [Bau09]). The details for other reconfigurable processors can be found in [Ama06; BL00; Bob07; CH02; Har01; HM09; TCW+05; VS07].

2.3.5 Rotating Instruction Set Processing Platform (RISPP)

The RISPP architecture [Bau09] embeds a fine-grained partially and dynamically reconfigurable fabric with a five-stage pipeline RISC processor (e.g., MIPS [ASI] or Sparc-V8 [Aer]) as shown in Fig. 2.6 [BSTH07]. The reconfigurable fabric is connected to the Execution stage as a functional unit. A hardware infrastructure is realized for the computation and communication within the reconfigurable fabric (details are explained later in this section). The communication part is a set of multiple segmented busses as a fixed interconnect structure (i.e., non-reconfigurable). The computation part consists of partially reconfigurable regions, i.e., so-called Data Path Containers (DPCs). These DPCs can be dynamically reconfigured to contain any particular Data Path on individual basis.

The principle distinction of RISPP compared to other reconfigurable processors is the concept of *modular* Custom Instructions (CIs) composed of elementary Data Paths and a run-time system to support them. These Data Paths can be reconfigured independently and can be shared among different CIs. This enables different implementations options per CI (from pure software, i.e., using cISA, to various hardware implementations), thus providing different tradeoffs between the reconfigurable fabric area and the achieved performance. The RISPP run-time system performs online monitoring of the CI executions, dynamically selects one implementation for each CI of the currently executing computational hot spot, and schedules the Data Path reconfigurations. In order to avoid potential memory inconsistencies and to simplify the memory arbiter implementation, RISPP stalls the pipeline during the execution of a CI.

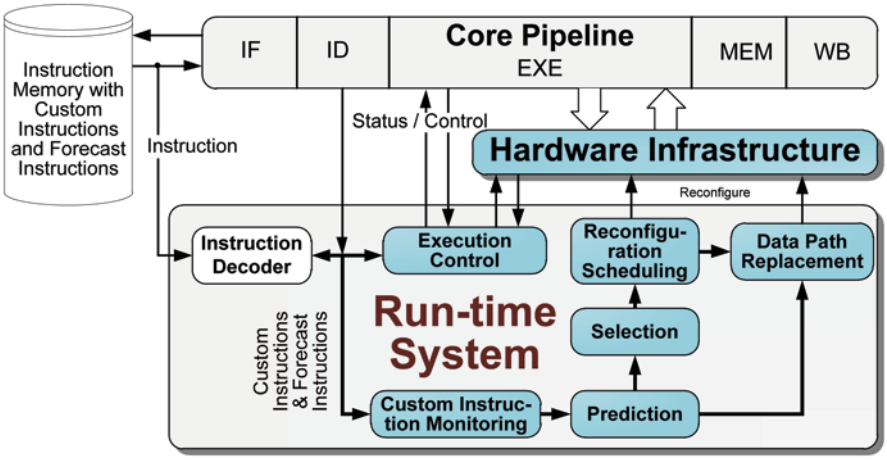


Fig. 2.6 Extending a standard processor pipeline towards RISPP and the overview of the RISPP run-time system

2.3.5.1 Modular Custom Instructions (CIs) with Hierarchical Composition

Figure 2.7 presents an overview of the hierarchical composition of *modular* CIs, consisting of the following three levels [BSKH07; BSTH07]:

Data Path: A *Data Path* corresponds to an elementary hardware accelerator module that can be (re-)loaded onto the reconfigurable fabric at run time. These Data Paths exploit a high level of operator parallelism and operator chaining. Data Paths of different *types* exhibit different computational properties, thus different latency and power consumption. A Data Path *instance* corresponds to an instantiation of a Data Path type. A reconfigurable fabric is typically partitioned (typically rectangular in shape) into so-called **Data Path Containers (DPCs)** that can be dynamically reconfigure to contain any particular Data Path. A DPC is similar to RFUs but typically smaller in size. Note, multiple instances of a certain Data Path type can be available at the same time, i.e., loaded into different DPCs.

Implementation Version: An *Implementation Version* of a CI is typically composed of multiple instances of different Data Paths types. The CI Implementation Versions differ in the amount of reconfigurable hardware they need to allocate and their resulting execution time. Therefore, these Implementation Versions provide different tradeoff points between performance and reconfiguration overhead. An Implementation Version is available if all of its demanded Data Paths are completely loaded onto the reconfigurable fabric. A particular Data Path can be shared among different Implementation Versions of different CIs (as shown in Fig. 2.7) because at most one CI executes at a time. The Implementation Versions of a CI feature different energy (leakage, dynamic, and reconfiguration) characteristics (as it will be discussed in Chap. 4 supported by actual measurements in Chap. 6). For each CI there exists exactly one Implementation Version (the slowest one) that executes by using only the cISA, i.e., without any accelerating Data Paths (see Fig. 2.7). It is activated by a synchronous exception (trap) that is automatically triggered if the CI shall execute and if the required Data Paths are not yet ready to execute (e.g., because

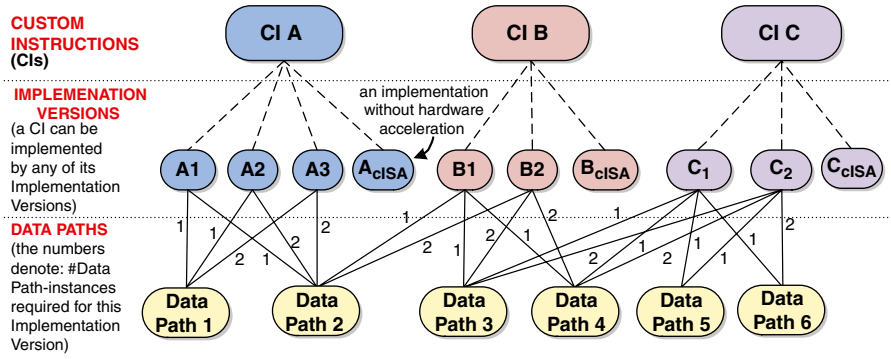


Fig. 2.7 Hierarchical composition of custom instructions: multiple implementation versions exist per custom instruction and demand data paths for realization

of reconfiguration delay). Depending upon the available hardware resources, a CI may execute using cISA or on the reconfigurable fabric (depending upon what is more efficient and which Data Paths are loaded at a certain point in time during the application execution).

Custom Instruction: A Custom Instruction (CI) is an assembler instruction that is executed to expedite a computational hot spot, thus the application execution. After a hot spot has completed its execution, the DPCs may be allocated to the CIs of other hot spots, which requires a run-time reconfiguration of the DPCs to load new Data Paths. The functionality of a CI is typically composed of several Data Paths that are loaded into the DPCs. A *modular* CI (as in the case of RISPP) has multiple Implementation Versions. As soon as a Data Path is completed loading onto a DPC, it may be used to execute a faster Implementation Version. This enables the gradual upgrading and downgrading of CIs during run time by switching between different Implementation Versions. On the contrary, a *monolithic* CI (as in the case of Molen) has only one Implementation Version (i.e., a specialized case of *modular* CIs) and it may be either entirely loaded onto the reconfigurable fabric or not at all.

The concept of *modular* CIs diminishes the above-discussed three problems of sharing, fragmentation, and longer reconfiguration time. The sharing problem is alleviated by Data Path sharing among different Implementation Versions of the same and/or different CIs. The fragmentation is reduced by using small-sized Data Paths instead of relatively large-sized *monolithic* CIs. The maximum fragmentation is limited by the size of a DPC, which is typically small. The problem of longer reconfiguration delay is alleviated by gradual upgrading, i.e., as soon as one Data Path finished reconfiguration, a faster Implementation Version might become available to accelerate the CI execution. Additionally, the concept of *modular* CIs provides enhanced performance and adaptivity by exploiting the tradeoff between performance and reconfiguration overhead at run time depending upon the application execution context. In order to enable this, a run time system and hardware infrastructure is employed in RISPP which will be explained after the formal model of *modular* CIs.

2.3.5.2 Formal Model of the Modular Custom Instructions

The formal model of the hierarchical CI composition is summarized here in order to clearly formulate the pseudo-codes in Chap. 4.

A data structure $(\mathbb{N}^n, \cup, \cap)$ is defined such that $\mathbb{N}^n$ is the set of all Implementation Versions and n is the number of different Data Path types. A CI is represented as a set of Implementation Versions. Let $\vec{m}, \vec{o}, \vec{p} \in \mathbb{N}^n$ be Implementation Versions with $\vec{m} = (m_0, \ldots, m_{n-1})$ where m_i denotes the amount of required instances of Data Path type A_i to implement the Implementation Version (similarly for $\vec{o}$ and $\vec{p}$). The total number of Data Paths required to implement an Implementation Version $\vec{m}$ is given as its determinant, i.e., $|\ |: \mathbb{N}^n \rightarrow \mathbb{N}; |\vec{m}| := \sum_{i=0}^{n-1} m_i$.

The cISA Implementation Version of a CI is represented as $|\vec{m}| := 0$. The operators $\cup$ and $\cap$ are used to combine two Implementation Versions (typically of different CIs). The operator $\cup$ (see Eq. 2.9) provides the Implementation Version that contains the Data Paths required to implement both Implementation Versions of its input. The operator $\cap$ (see Eq. 2.10) provides the Implementation Version[10] with shared Data Paths between the both Implementation Versions of its input.

$$\cup : \mathbb{N}^n \times \mathbb{N}^n \rightarrow \mathbb{N}^n;\ \vec{o} \cup \vec{p} := \vec{m};\ m_i := \max\{o_i, p_i\} \tag{2.9}$$

$$\cap : \mathbb{N}^n \times \mathbb{N}^n \rightarrow \mathbb{N}^n;\ \vec{o} \cap \vec{p} := \vec{m};\ m_i := \min\{o_i, p_i\} \tag{2.10}$$

To express the Data Paths that are additionally required to realize an Implementation Version $\vec{p}$ when the Data Paths of an Implementation Version $\vec{o}$ are already available, the upgrade operator $\triangleright$ is used (Eq. 2.11).

$$\triangleright : \mathbb{N}^n \times \mathbb{N}^n \rightarrow \mathbb{N}^n;\ \vec{o} \triangleright \vec{p} := \vec{m};\ m_i := \begin{cases} p_i - o_i, if\ p_i - o_i \geq 0 \\ 0,\ else \end{cases} \tag{2.11}$$

The comparison relation between two Implementation Versions is defined as $\vec{o} \leq \vec{p}$ (see Eq. 2.12). As the relation is reflexive, anti-symmetric, and transitive, $(\mathbb{N}^n, \leq)$ is a partially ordered set.

$$\vec{o} \leq \vec{p} := \begin{cases} true, \text{if } \forall i \in [1, n] : o_i \leq p_i \\ false, \text{else} \end{cases} \tag{2.12}$$

In addition to the formal model for *modular* CIs, additional functions are defined for the ease of description of pseudo-codes in Chap. 4. The syntax of these functions is oriented at the object oriented programming style, such that an Implementation Version is seen as an object and the function is called for that particular object. Table 2.1 provides an overview of all these functions with the corresponding high-level properties with their syntax and an explanation [Bau09]. Additionally, a list $dpc = (dpc_1, ..., dpc_n)$ is defined where n is the total number of DPCs in this list and $dpc_k.getLoadedDP()$ returns the loaded Data Path in the kth DPC. Note this *dpc* list is a software data structure and it is not related to the hardware DPCs. It is defined to facilitate the management of different Data Paths and hardware DPCs along with their power states. The functions $dpc.add(dpc_k)$ and $dpc.remove(dpc_k)$ are used to add and remove the kth DPC to the dpc list. $dpc.find(i).location()$ finds the (first) DPC location corresponding to the ith Data Path.

2.3.5.3 Run-Time System of RISPP

The run-time system (see Fig. 2.6) exploits the potential of *modular* CIs and controls the hardware infrastructure to determine the set of Implementation Versions and the corresponding reconfiguration decisions as well as to control the CI execu-

[10] Not necessarily of a particular Custom Instruction, it is rather a representation of the set of Data Paths of two Implementation Versions.

Table 2.1 High-level properties of implementation version and custom instruction

Operator for Implementation Version $\vec{m}$ or Custom Instruction s	Description
$\text{int}\, l = \vec{m}.getLatency()$	An Implementation Version has certain execution latency (in cycles)
$s = \vec{m}.getCI()$	This function returns the CI of an Implementation Version along with the corresponding information
$\vec{m} = s.getCISAImpV()$	Returns that Implementation Version of a CI that uses the cISA for execution
$\vec{m} = s.getFastestAvailImpV(\vec{a})$	This function returns the fastest Implementation Version $\vec{m}$ of the CI s that can be implemented with the available Data Paths $\vec{a}$
$\text{int}\, f = s.getExpectedExecutions()$	This function returns the expected number of execution of the CI s in a particular computational hot spot. This frequency depends upon the input data and it is estimated by an online-monitoring scheme
$\vec{s}.add(i)$	Add one instance of the ith Data Path in the vector $\vec{s}$
$\vec{s}.remove(i)$	Remove one instance of the ith Data Path from the vector $\vec{s}$

tions. The goal of algorithms for the run-time system are to maximize the performance for a given area of the reconfigurable fabric. The area of the reconfigurable fabric (i.e., number of DPCs), the core pipeline, CI formats and free opcode space, and the implementation of the run-time system are fixed at design-time. The composition of Implementation Versions[11], cISA implementations, insertion of forecast instructions, and opcodes for CIs are determined at compile-time.

At compile time, the so-called **Forecast Instructions (FI)** are inserted in the application binary[12] to trigger the run-time system. These FI contains the information about the CIs that are expected to be executed next, thus triggering the prefetching. However, eventually the run-time system determines the reconfiguration decisions for a set of Implementation Versions for the CIs as mentioned in the FIs. An offline-profiling is used to predict the execution frequency of a CI, i.e., the so-called **Forecast Value (FV)** of an FI. Since the CI execution frequency may depend on input data, the FV needs to be updated at run time to reflect recent CI execution frequencies. The run-time system of RISPP (see Fig. 2.6) uses an **Online-Monitoring** and a **Prediction** scheme to dynamically update the expected CI execution frequencies.

The **Decoder** triggers the modules of the run-time system when CIs or FIs are encountered in the instruction stream. For a CI, the **Execution Control** manages the execution mode (i.e., using cISA or a hardware Implementation Version) and the Online-Monitoring counts the CI executions. The Prediction module uses the difference between the initial FV and the monitored executions to adjust the FV for

[11] Which Implementation Versions is used to implement a CI is determined at run time, but the composition of individual Implementation Versions are not affected, therefore, they can be prepared at compile time.

[12] FIs and CIs are programmed as inline assembly. The assembler is extended to know about the instruction formats and opcodes of all FIs and CIs occurring in the assembly code.

the next execution. The predicted CI execution frequency is used by the **Selection** to determine a set of Implementation Versions that maximizes the performance for the given size of the reconfigurable fabric (i.e., the number of Data Paths that can be reconfigured onto it at the same time). Afterwards, a set of required Data Paths is determined from the selected Implementation Versions and the **Scheduling** determines the reconfiguration sequence of these Data Paths. In case there is no empty DPC available in the hardware infrastructure, the **Replacement** determines which Data Path should be replaced to load the new one. In the following, the key parts of the run-time system are explained briefly.

Online-Monitoring: A *fair* distribution of the reconfigurable fabric within a computational hot spot depends upon the execution frequencies of CIs. An offline-profiling provides average-case FVs (i.e., the prediction of expected CI executions) for a particular application. Since the CI execution frequency may vary at run time, an online-monitoring in conjunction with an error back-propagation scheme (based on Temporal Difference) is used for fine-tuning/updating the FVs. It thereby enables the adaptivity to changing CI requirements, e.g., a faster Implementation Version is selected for a CI which is predicted to be executed more often compared to the previous execution run. Typically the FVs of all CIs of a hot spot are indicated as a so-called Forecast Block (FB, i.e., a set of predictions) to trigger the run-time system once per hot spot. Figure 2.8 shows an example scenario using two Forecast Blocks FB_1 and FB_2 where FB_1 forecasts the execution of the CI_A within a hot spot and FB_2 predicts that the hot spot execution is finished and the CI_A is no longer required [Bau09]. Figure 2.9 demonstrates the idea of fine-tuning the FVs by representing the control-flow graph of Fig. 2.8 as a linear chain of FBs for two iterations of the outer loop [Bau09]. The CI executions between two Forecast Blocks FB_t and FB_{t+1} are monitored as $M(FB_{t+1})$. Whenever the FB_{t+1} is encountered, the difference between $FV(FB_t)$ and the monitoring value $M(FB_{t+1})$ is computed. In addition to this difference, the Forecast Block FB_{t+1} is also considered for computing the error $E(FB_{t+1})$, as it may also predict some executions of that CI to come soon. Equation 2.13 shows the computation of this error using parameter $\gamma \in [0,1]$ to weigh the contribution of $FV(FB_{t+1})$. Afterwards, the error is back-propagated to the preceding FB (see Eq. 2.14) where the parameter $\alpha \in [0,1]$ is used to control the strength of this back propagation. A moderate α value avoids thrashing and

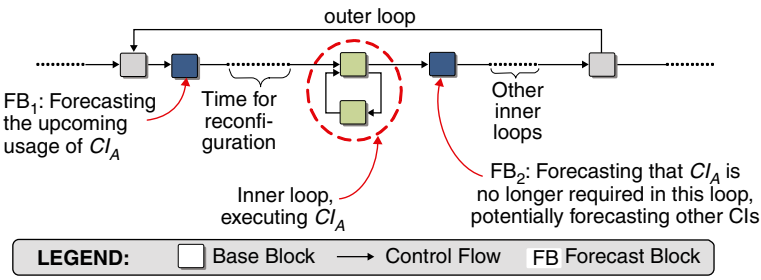


Fig. 2.8 Example control-flow graph showing forecasts and the corresponding custom instruction executions

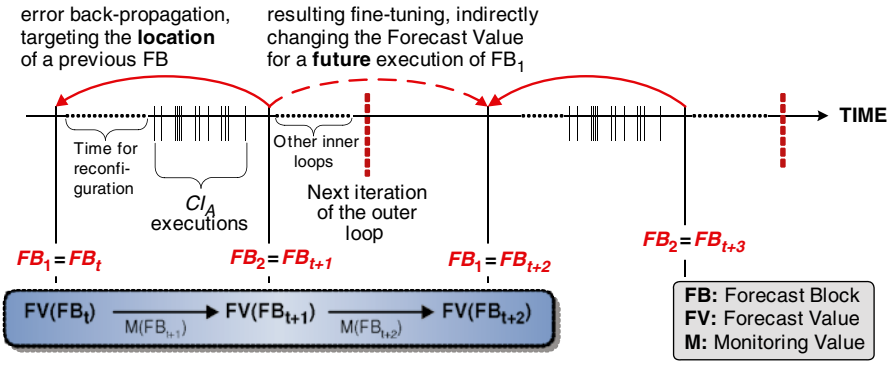


Fig. 2.9 Execution sequence of forecast and custom instructions with the resulting error back propagation and fine-tuning

provides smooth variations in the prediction. The so-called static prefetching (i.e., no fine-tuning at run time [LH02]) can be realized as a special case of this model by using $\alpha=0$. Note, fine-tuning the FVs for multiple CIs is done independent of each other.

$$E(FB_{t+1}) := M(FB_{t+1}) - FV(FB_t) + \gamma FV(FB_{t+1}) \tag{2.13}$$

$$FV(FB_t) := FV(FB_t) + \alpha E(FB_{t+1}) \tag{2.14}$$

Implementation Version Selection: The concept of *modular* CIs allows RISPP to dynamically determine which Implementation Version shall be used to implement a CI, i.e., distributing the reconfigurable fabric among different CIs depending on the run-time varying application requirements. The Selection is triggered for each computational hot spot by Forecast Instructions. It determines the set of Implementation Versions (to implement the forecasted CIs) that maximizes the overall performance while considering the given size of the reconfigurable fabric. RISPP incorporates a greedy algorithm for Selection that uses a profit function which considers the CI execution frequency, the latency improvement, and the reconfiguration delay of an Implementation Version. The coefficients of the profit functions are empirically computed. After the Implementation Versions are selected, the reconfiguration sequence for the required Data Paths is determined as only one reconfiguration may be performed at a time.

Reconfiguration-Sequence Scheduling: It determines the sequence in which Data Paths are reconfigured. This sequence is important (in terms of performance) as it determines which Implementation Versions are available first to expedite the computational hot spot. In RISPP, four different strategies are explored. The Highest Efficiency First (HEF) scheduler is finally used due to its better performance over other strategies. The HEF scheduler determines the upgrade Implementation Version which is the most beneficial one (in terms of performance) on a scheduling path while considering the latency improvement, the CI execution frequency, and

the amount of additionally demanded Data Paths. If a new Data Path is going to be reconfigured and there is no free DPC available, some Data Path need to be replaced which is determined by the Replacement.

Replacement: The run-time system in RISPP employs a Minimum Degradation replacement policy that considers the potential performance degradation for CIs when replacing a Data Path. It replaces the one that leads to the overall smallest performance degradation for the CIs. This policy aims to keep all CIs in a good performance by searching the downgrade step with the smallest performance impact.

2.3.5.4 Hardware Infrastructure for Communication and Computation

The Hardware Infrastructure (see Fig. 2.10) is partitioned into so-called Data Path Containers (DPCs) and Bus Connectors (BCs) [Bau09]. The DPCs can be dynamically reconfigured to contain one Data Path at a time, without affecting the other parts. Each DPC is connected to a dedicated BC via so-called Bus Macros[13] [Xil05]

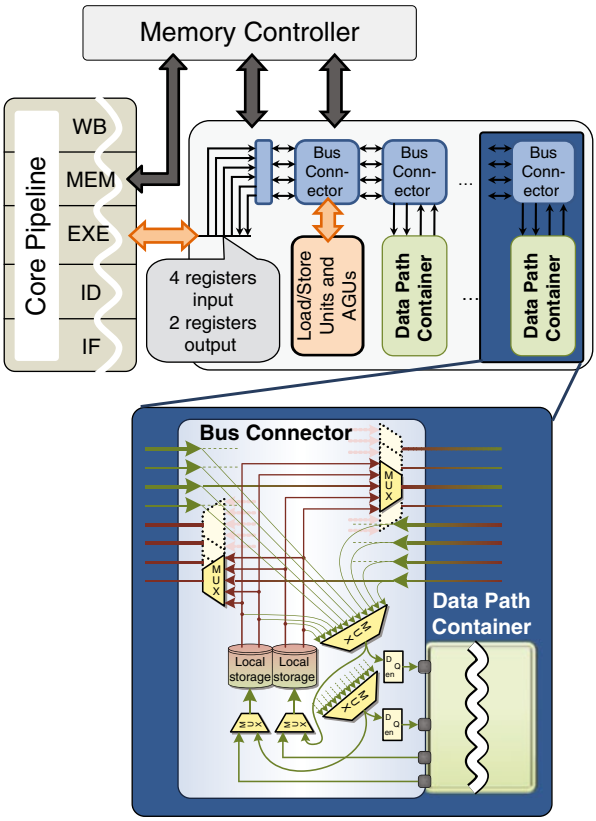


Fig. 2.10 Overview of the hardware infrastructure for computation (data path container) and communication (bus connector) showing the internal composition of a bus connector

[13] Bus Macros are used to establish communication between the partially reconfigurable part (i.e., DPC) and the non-reconfigurable part (i.e., BC).

such that the communication resources are available during the reconfiguration of the DPC (see connection details in [BSH08a]). The BC is connected to the adjacent BCs using unidirectional segmented buses (4 buses in each direction) in order to provide high-bandwidth parallel communication with low latency (single cycle).

Each BC contains two 4×32-bit local storages (1 write and 2 read ports each) to temporarily store the Data Path output. A DPC may receive inputs from segmented buses and/or local storages via BC-DPC latched input connections[14]. The data from segmented busses can be directly written to the local storages and alternatively the output of a local storage may drive any BC output. The control signals are provided to BCs in each cycle using 1024-bit Very Long Control Words (VLCWs) in order to determine the connections between different DPCs. This Hardware Infrastructure is connected to the general-purpose register file of the core pipeline. Each DPC provides a fixed interface with two 32-bit inputs, two 32-bit outputs, a 6-bit control signal (provided in the VLCW), a clock signal and an 8-bit output to notify the system about the currently loaded Data Path. Two 128-bit Load/Store Units are used to access two independent high-bandwidth memory ports in parallel and the Address Generation Units provide addresses to the Load/Store Units.

2.4 Low-Power Approaches in Reconfigurable Processors

Previous approaches in reconfigurable processors (like OneChip [WC96], CoMPARE [SGS98], Proteus [Dal03], Molen [VWG+04], and RISPP [BSH08b; BSH08c]) have mainly concentrated on improving the performance by reconfiguring application-specific hardware accelerators at run time to meet applications' demands and constraints. This reconfiguration process may consume a noticeable amount of energy. Consequently, the major shortcoming of these reconfigurable processors is their high energy consumption compared to ASICs and lack of efficient energy management features [Te06]. Moreover, with the evolution of submicron fabrication technologies, the consideration of leakage power/energy has become imperative in the energy-aware design of reconfigurable processors. A basic shutdown infrastructure is required to provide a foundation to exert high-level power and energy management schemes like the one proposed in this monograph (Chap. 5). Several academic and industrial research projects have already made the case for such shutdown infrastructure (see details in Sect. 2.3.2). In the following, prominent design-, compile-, and run-time low-power related work for FPGAs and reconfigurable processors is presented.

Design-Time Approaches: Several design-time low-power architectural approaches for FPGAs are presented in [CWL+05; Te06]. A 90 nm low-power FPGA for battery-powered applications is introduced in [Te06]. It supports voltage

[14] The latch disconnects the Data Path of the DPC from the not-demanded external inputs, thus avoiding unnecessary toggles and reducing the dynamic power consumption of the Data Paths.

scaling, power-shutdown, and a low-leakage configuration SRAM. The authors in [CWL+05] presented a trace-based timing and power evaluation method for device and architecture co-optimization for FPGA power reduction by exploring the design space of V_t and V_{dd}. The approach in [RP04] uses body biasing, multi-V_t logic, and gate biasing to reduce the leakage in FPGAs. A fine-grained leakage optimization techniques is presented in [MM05] that performs shutdown of the configuration SRAM of the unused LUTs. The authors in [LLH07] showed that too-high V_t transistors for leakage reduction in the configuration SRAM result in an increased reconfiguration time, thus leading to higher reconfiguration energy. These design-time approaches (e.g., [CWL+05; Te06]) provide the infrastructure to enable run-time adaptive energy management schemes, the domain in which the contribution of this monograph lies.

Compile-Time Approaches: Besides design-time approaches, compile-time approaches typically target power-aware placement [Ge04], software partitioning and mapping [GE08], co-processor selection [Ge07] etc. A region-constrained placement is presented in [Ge04] to reduce leakage energy in FPGAs by increasing the number of unused regions to be switched off. An energy-optimal software partitioning scheme for heterogeneous multiprocessor systems is presented in [GE08] incorporating a resource model considering the time and energy overhead of run-time mode switching. The authors in [GE08] optimized the software partitioning at compile-time by formulating it as an Integer Linear Programming (ILP) problem. An ILP-based energy-aware co-processor selection for reconfigurable processors is proposed in [Ge07]. However, compile-time techniques for power-reduction (placement, partitioning, co-processor selection etc.) are not able to react to run-time changing scenarios, thus they perform inefficient in that respect.

Run-Time Approaches: The authors in [Ge04] additionally proposed a *time-based power-shutdown scheme* for run-time leakage minimization. However, [Ge04] does not consider which parts of the reconfigurable fabric are beneficial to shutdown at what time when considering *partial run-time reconfiguration*. A run-time approach in [Ne08] incorporates operand isolation and selective context fetching to reduce the power in reconfigurable processors. The authors in [PP08] presented a methodology for energy-driven application's self-adaptation using run-time power estimation. However, these approaches target reducing dynamic energy and ignore the leakage energy and power-shutdown. Several approaches in dynamic energy management incorporate Dynamic Voltage and Frequency Scaling (DVFS) techniques. The authors in [Qe07] employed configuration pre-fetching and configuration parallelism (using multiple configuration controllers) to create excessive system idle time and then employs voltage scaling on the configuration process to reduce the configuration energy in run-time reconfigurable processors. A low-power version of the Warp Processor [LSV06] is proposed in [Lys07]. It performs online profiling and online synthesis to automatically determine and synthesize suitable hardware accelerators at run time with a support of DVFS to dynamically reduce the power consumption. The approach in [HL07] co-schedules the hardware and software tasks at

run-time by using slack time. The slack time is introduced by reusing hardware task configurations to trigger the voltage scaling such that the preceding software tasks consume lesser power. DVFS techniques target on finding out the slack time in the execution pattern to reduce voltage and frequency. DVFS alone would not solve the problem of energy-minimizing instruction set in reconfigurable processors (see Sect. 5.3, p. 133) especially when considering the changing execution frequencies of Custom Instructions. The main challenge here is to minimize the overall energy under run-time varying constraints while considering the power-shutdown decision at a higher abstraction level (as discussed in Chap. 5). Yet, DVFS schemes (e.g., [Qe07; Te06]) may be integrated with the proposed contribution to achieve even further energy reduction.

2.5 Summary of Related Work

The application-/algorithm-level related work on H.264 encoder either reduce the coding complexity by sequentially excluding the improbable candidate coding modes or by employing adaptive fast Motion Estimation schemes. However, state-of-the-art adaptive, fast, low-power, and scalable Mode Decision and Motion Estimation approaches either only consider a fixed quality-constrained solution or offer scalability with fixed termination rules that may lead to severe quality artifacts. These approaches do not provide run-time adaptivity when considering run-time varying energy-budgets, input video sequence characteristics, and user-defined constraints. As a result, these approaches are less energy-/power-efficient.

For designing an adaptive low-power multimedia system there is a need to combat the power and adaptivity related issues at both application and processor levels [FHR+10]. Majority of the multimedia systems are designed for heterogeneous MPSoCs, where different applications components (like video encoder/decoder, audio encoder/decoder, etc.) are implemented as an ASIC or ASIP. The programmability is mainly achieved by deploying DSPs, where the flexible parts are executed on the DSP. Most of the video encoding solutions (considering the advanced H.264 video encoder which is 10× more complex compared to the previous generations of encoding standards) target ASIC implementations to achieve low power. However, several works have also considered DSP or ASIP based implementations. Moreover, there are several ASIC-based solutions for different functional blocks of the H.264 video encoder that exploit pipelining and parallelism. On the one hand, ASIC-based implementations are less flexible and are not amenable to the run-time varying constraints and standard evolution trends, especially considering short time-to-market. On the other hand, the major shortcoming of multimedia MPSoCs is their design-time selection of cores depending upon an initial set of application requirements. Since the cores are optimally selected for a set of initial requirements, these MPSoCs may not fulfill the required performance and/or power constraints when there is a change in the application requirements, design constraints, standard change, etc.

Moreover, such MPSoCs may not handle the advanced multimedia standards (that exhibit unpredictable computational behavior and/or subjected to run-time varying constraints of available energy) in a power efficient way, especially when considering short time-to-market and short-term standard evolutions and product upgrades. Furthermore, the previous approaches lack run-time adaptivity when considering varying energy budgets and area/performance constraints.

Dynamically reconfigurable processors provide an alternate solution by exploiting the high degree of parallelism along with a high degree of adaptivity. Previous approaches in reconfigurable processors have mainly focused on improving the performance by reconfiguring application-specific hardware accelerators at run time to meet applications' demands and constraints. However, due to the reconfiguration process and the fabric nature to support high configurability, these reconfigurable processors suffer from high energy consumption compared to ASICs and lack of efficient energy management features. Recently, low-power design, especially the leakage power reduction, has become a key research focus in reconfigurable computing. Several academic and industrial research projects have already made the case for power-shutdown infrastructure. Such an infrastructure is required to enable run-time adaptive energy management schemes, as the one proposed in this monograph.

State-of-the-art low-power approaches employ a hardware-oriented shutdown based on the state of a particular hardware. However, in dynamically reconfigurable processors, such a technique will perform inefficient as it cannot be determined at compile time which Custom Instructions will be reconfigured on which parts of the reconfigurable fabric. Moreover, state-of-the-art techniques do not evaluate the tradeoff between leakage, dynamic, and reconfiguration energy at run time which is inevitable when considering design-/compile-time unpredictable scenarios of application execution (changing performance constraints, input data, etc.) and available area and energy budgets.

The low-power and adaptivity concerns for multimedia systems with advanced video codecs (subjected to unpredictable scenarios) are addressed by the proposed adaptive low-power reconfigurable processor architecture and an energy-aware H.264 video coding application architecture. At the processor level the novel concept of *Selective Instruction Set Muting* (with multiple muting modes) allows to shun the leakage energy at the abstraction level of Custom Instructions. This enables a far higher potential for leakage energy saving. Furthermore, the proposed adaptive energy-management scheme comprehensively explores the tradeoff related to leakage, dynamic, and reconfiguration energy under run-time varying performance and area constraints. At the application architecture level, the novel concept of *Energy-Quality Classes* enables a run-time tradeoff between the energy consumption and the resulting video quality. The concept of *Energy-Quality Classes* along with an adaptive energy-budgeting scheme provides a foundation for energy-aware Motion Estimation. The energy-aware Motion Estimation and an adaptive complexity reduction scheme realize an *adaptive low-power video encoder* application architecture. The detailed issues and energy analysis at both application and processor level are discussed in Chap. 3. This chapter also provides an overview of the proposed

processor and application architectures followed by the power model used by both architectures for adaptive energy management. In Chap. 4 the adaptive low-power video encoding is discussed. Chapter 5 presents the adaptive low-power reconfigurable processor architecture with run-time adaptive energy management scheme. The comparison with sastate-of-the-art is presented in Chap. 7.

Chapter 3
Adaptive Low-Power Architectures for Embedded Multimedia Systems

In this chapter an overview of the proposed application and processor architectures for embedded multimedia systems is presented, highlighting different steps performed at design, compile, and run time. The details of these architectures are provided in Chaps. 4 and 5. First, Sect. 3.1 discusses an H.324 video conferencing application and provides the processing time distribution of different computational hot spots of various application tasks. In Sect. 3.1.1, the coding tool set of advanced video codecs is analyzed and similarities between different coding standards are highlighted, while corroborating the selection of the H.264/AVC video coding standard for this monograph. In Sect. 3.1.2, energy and adaptivity related issues in the H.264 video encoder application are analyzed and discussed. Together with these, other issues for dynamically reconfigurable processors are discussed in Sect. 3.2. Afterwards, Sect. 3.3 presents an overview of the proposed application and processor architectures along with different steps to be performed at design, compile, and run time. At the end, the proposed power model for dynamically reconfigurable processors is discussed in Sect. 3.4, highlighting different power consuming components from the computation and communication infrastructure of the processor.

3.1 Analyzing the Video Coding Application for Energy Consumption and Adaptivity

In current and emerging mobile devices, energy/power is a critical design parameter and multimedia is a major application domain. These multimedia applications with advanced video encoders—due to their huge amount of processing and energy requirements—pose a serious challenge on low-cost/low-power embedded systems. In the following a video conferencing application is discussed highlighting the dominance of an advanced video encoder with respect to its computational and energy requirements along with the inherent adaptivity. The inherent adaptivity will be discussed to highlight issues that can be exploited for run-time energy management at both application and processor architecture levels.

M. Shafique, J. Henkel, *Hardware/Software Architectures for Low-Power Embedded Multimedia Systems,*
DOI 10.1007/978-1-4419-9692-3_3, © Springer Science+Business Media LLC 2011

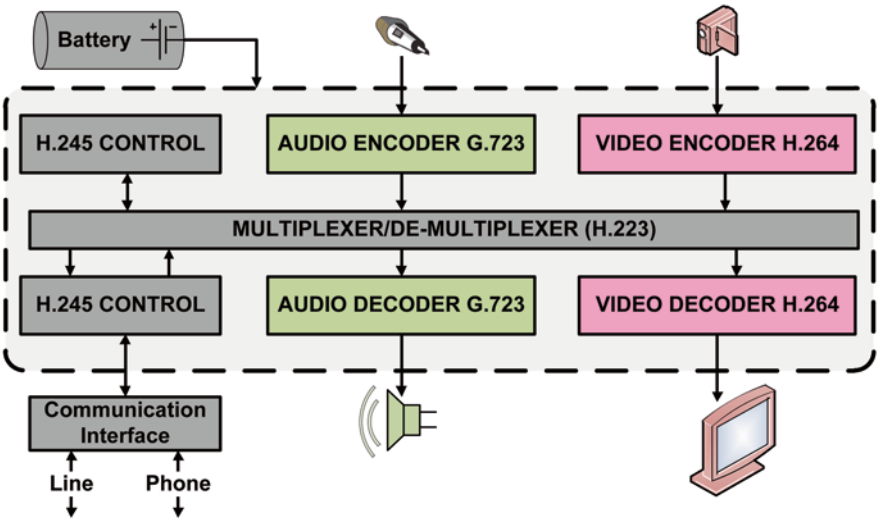


Fig. 3.1 Overview of an H.324 video conferencing application with H.264/AVC codec

Figure 3.1 shows the block diagram of a video conferencing application, which is envisaged to be an important application in future mobile devices for video calls. A video conferencing is composed of a video encoder/decoder with video pre-/post-processing modules, an audio encoder/decoder, a multiplexer, and a communication protocol. In order to achieve a high compression and better video quality, typically an advanced video encoder (like H.264/AVC [ITU05]) is employed. In order to find out the computational hot spots and their relative complexity, profiling has been performed.

Figure 3.2 illustrates the average-case distribution of the processing time (in percentage) of various tasks of the video conferencing application. It is noticed that more than 70% computations are consumed by the H.264 video codec (encoder consumes >60%). The remaining computational quota is allocated to the video pre- and post-processing and the G.723 audio codec. Less than 10% is reserved for the remaining tasks. It shows that video coding is the dominant application task in the video conferencing application. This statement holds true for various other multimedia applications, like personal video recording, etc. Therefore, in the remaining part of this monograph, video encoding is considered as the key application for energy reduction. Moreover, unlike video decoding (which is fixed by the standard), video encoding exhibits a great potential for energy reduction at the application level due to non-normative parts (e.g., Motion Estimation and Mode Decision are not fixed by the standard), as it will be discussed in the remaining sections and Chap. 4.

In the following, the coding tool set of various video coding standards is compared in order to justify the selection of H.264/AVC. Afterwards, the energy and adaptivity related issues in the H.264 video encoder will be discussed in Sect. 3.1.2.

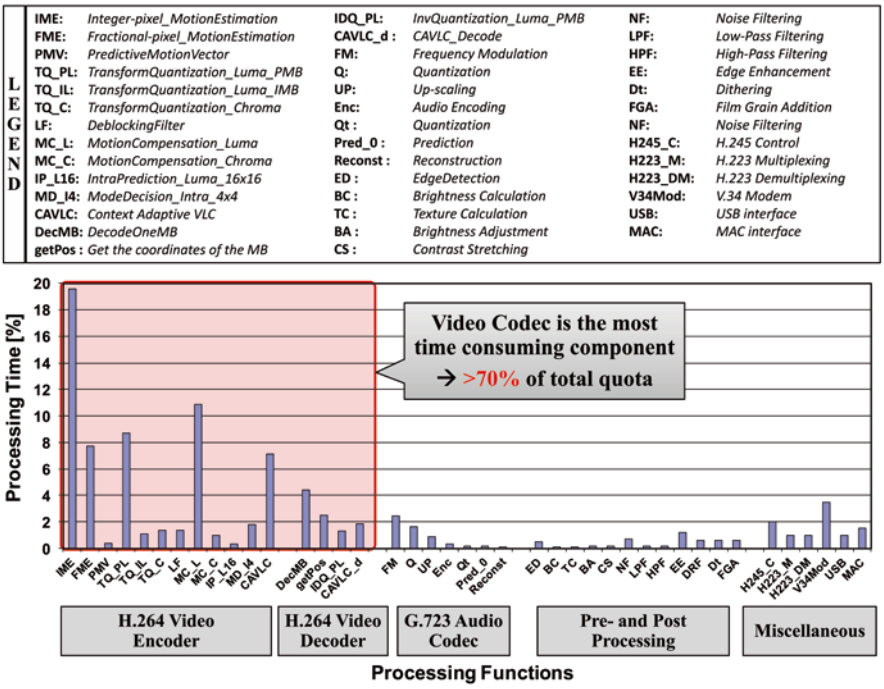


Fig. 3.2 Processing time distribution of different functional blocks in the H.324 video conferencing application

3.1.1 Advanced Video Codecs: Analyzing the Tool Set

Table 3.1 presents a comparison of the coding tool set of various advanced video encoding standards [ITU05, 09; Joi08, 10; KL07; Mic10a, b; Ric03, 10; YCW09]. The Microsoft VC-1 standard and Chinese Audio Video Standard (AVS) belong to the same generation period as of the H.264/AVC. Multiview Video Coding is the latest extension (finalized in 2008 [Joi08]) of H.264 for 3D-videos where a 3D-scene is captured by multiple cameras. H.265/HEVC (High Efficiency Video Coding) belong to the next generation of video codecs and it is expected to be standardized by the end of 2012 [Joi10]. It is worthy to note that within the same generation, the coding tool set of H.264/AVC is the most complex one that also results in a relatively better coding efficiency. As discussed in Chap. 2, Motion Estimation and Rate Distortion Optimized Mode Decision are the most critical components of a video encoder from computation complexity and energy reduction point of views. It can be noted that within the same generation, H.264/AVC offers the highest number of coding mode options (see variable-block sized Motion Estimation and Intra Prediction modes in Table 3.1). Moreover, the coding options offered by AVS and VC-1 are approximately a subset of the coding options of H.264/AVC. Therefore, the application-level energy reduction algorithms and the application architecture for

Table 3.1 Comparing the coding tool set of various video encoding standards

Coding tools	Advanced video encoding standards				
	H.264/AVC (Advanced Video Coding)	Audio Video Standard (AVS)	Microsoft VC-1	Multiview Video Coding (MVC)	H.265/HEVC (High Efficiency Video Coding)
Frame type	I, P, B, SP, SI	I, P, B	I, P, B, BI, Skipped P	I, P, B	I, P, B, …
Variable-block sized motion estimation	16×16, 16×8, 8×16, 8×8, 4×8, 8×4, 4×4	16×16, 16×8, 8×16, 8×8	16×16, 16×8, 8×16, 8×8, 4×8, 8×4, 4×4	16×16, 16×8, 8×16, 8×8, 4×8, 8×4, 4×4	Geometry Block Partitioning Asymmetric Block Partitioning
Transform size	4×4, 8×8	8×8	4×4, 8×8, 8×4, 4×8	4×4, 8×8	Adaptive (may also have 8×8)
Transform	Integer DCT, Hadamard	Integer DCT	DCT	Integer DCT, Hadamard	Large Transform (16×16–64×64), Rotational/Mode Dependent Directional Transform
Motion vector resolution	1/4-pixel (6-tap and Bilinear filter)	1/4-pixel (4-tap filter)	1/4-pixel (4-tap filter)	1/4-pixel (6-tap and Bilinear filter)	1/8-pixel (separable, non-separable, or directional adaptive interpolation filter), adaptive motion vector resolution
SKIP mode	MB-Level	MB-Level	Frame-Level	MB-Level	MB-Level
Maximum number of reference frames	16 each way	2 each way	1 each way	16 each way	Adaptive warped reference
Intra prediction modes & block sizes	Luma: 16×16 (4 modes), 4×4 (9 modes); Chroma: 8×8 (4 modes)	8×8 (5 modes for Luma and 4 modes for Chroma)	None	Luma: 16×16 (4 modes), 4×4 (9 modes); Chroma: 8×8 (4 modes)	Adaptive reference sample smoothing, planar or angular prediction, arbitrary directional Intra (ADI), Combined Intra Prediction (CIP)
Entropy coding mode	CAVLC, CABAC	CA-2D-VLC, CABAC	Adaptive VLC	CAVLC, CABAC	Low-complexity entropy coding with VLC codes, high coding efficiency with V2V codes
In-loop deblocking filter	5 strength cases	3 strength cases	Yes	5 strength cases	Adaptive post-loop filters, Intra planar mode filtering

adaptive low-power video coding are equally applicable to AVS and VC-1. Since Multiview Video Coding is an extension of H.264, the proposed contribution can be easily extended towards Multiview Video Coding. Further energy reduction can be obtained by extending the analysis to 3D, i.e., by exploiting the extensive correlation space of the 3D-neighborhood (see Sect. 8.2 for future works).

When considering the evolution of the video coding standards, it can be noticed in Table 3.1, that H.265/HEVC extends the computation and coding mode model of H.264 by providing further adaptivity, thus further extending the conventional data dominant processing to control dominant processing. High adaptivity is planned to be employed in the H.265 standard to achieve 2× higher coding efficiency compared to the H.264/AVC coding standard [ITU05]. Since adaptivity is the key property in various algorithms employed in different functional blocks of the H.265/HEVC standard, the proposed adaptive low-power processor architecture will provide a good foundation for researching energy-efficient multimedia solutions. Moreover, the adaptive low-power video coding concepts (as proposed in Chap. 4) can also be extended towards further adaptivity, especially the proposed concept of *Energy-Quality Classes* for energy-aware Motion Estimation (see details in Sect. 4.5) will be equally beneficial for H.265/HEVC. However, the SAD computation unit need to be replaced according to the new block partitioning structure. Furthermore, the Macroblock categorization based on the spatial and temporal video properties while considering the Human Visual System (see details in Sects. 4.3 and 4.4) will also be beneficial. However, its usage may be adapted depending upon the final set of coding options adopted by the standardization committee.

As corroborated by the above discussion, H.264/AVC is considered for researching the adaptive low-power video coding which is a primitive component of current and upcoming embedded multimedia systems. In the subsequent chapters, the discussion will be more focused towards the H.264/AVC encoding.

3.1.2 Energy and Adaptivity Related Issues in H.264/AVC Video Encoder

Video encoding consumes a significant amount of processing time and energy. Encoding effort highly depends upon the characteristics of the input video sequence and the target bit rates. Moreover, the available energy budgets may change according to various application scenarios on mobile devices. Under changing scenarios of input data characteristics and available energy budgets, embedded solutions for video encoding require run-time adaptivity.

The advanced video coding standard H.264/AVC [ITU05] provides double compression compared to previous coding standards (MPEG-2, H.263, etc.) [WSBL03] at the cost of additional computational complexity (~ 10× relative to MPEG-4 advance simple profile encoding [OBL+04]). This directly corresponds to high energy consumption. This increased energy consumption of H.264 is mainly due to

its complex Motion Estimation (ME) and Rate Distortion Optimized Mode Decision (RDO-MD) processes. It is worthy to note that RDO-MD is the most critical functional block in H.264, as it determines the number of ME iterations. Therefore, complexity reduction at the stage of RDO-MD is required first, before proceeding to the energy reduction at the ME stage.

As discussed in Sect. 2.2.3, many efforts have been made in developing fast Mode Decision schemes for H.264 to reduce the complexity of encoding [ADV-LN05; GY05; JC04; JL03; KC07; LWW+03; MAWL03; PC08; PLR+05; SN06; WSLL07; Yu04]. However, most of these state-of-the-art RDO-MD approaches sequentially process and eliminate the modes depending upon the result of the previously evaluated modes. Therefore, modes are not excluded in the fast RDO-MD until some ME is not done. As a result, these approaches still compute a significant (more than half of all) number of modes or even in the best case at least one mode from both P-MB and I-MB is processed (see [GY05; JC04]). Since the ME is always processed in any case, the computational requirements of the state-of-the-art are still far too high, which makes them infeasible for low-power embedded multimedia systems. Therefore, **there is a dire need for a complexity reduction scheme** that can adaptively exclude as many coding modes as possible from the candidate mode set (as used for the Mode Decision process) at run time even before starting the actual fast RDO-MD and ME processes.

Such a scheme is more critical for low-power video encoding solutions as it may avoid the complete ME process (the most energy consuming functional block of an encoder, see Fig. 3.3). A good complexity reduction scheme needs to predict the possible coding mode with a high accuracy that requires an in-depth knowledge of the spatial and temporal properties of the video data. It requires a relationship between the video properties and the probable coding mode. A high accuracy of the coding mode prediction also requires a joint consideration of the spatial and temporal video properties. For low power consumption, an aggressive mode exclusion may be desirable that needs to consider the properties of the Human Visual Sys-

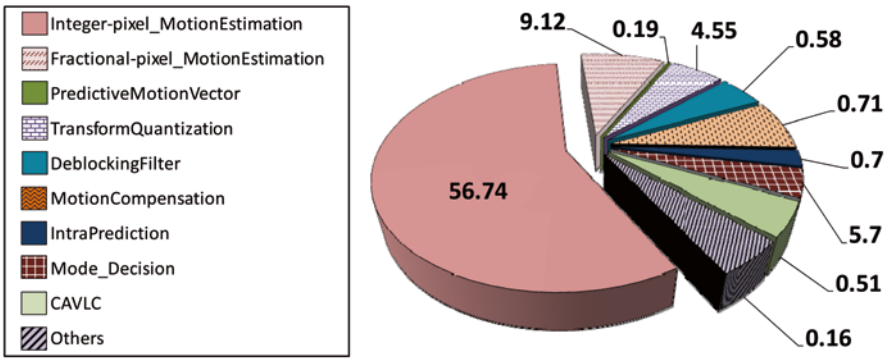


Fig. 3.3 Percentage distribution of energy consumption of different functional blocks in the H.264 video encoder

tem in order to analyze the subjective impact of the coding modes. It is worthy to note that for extracting the spatial and temporal video properties, additional image processing operations are required that incur a power and performance overhead. Therefore, the overhead of the extra processing must be amortized by the significant complexity and energy reduction offered by the scheme. After a coding mode is predicted, the most energy consuming part of an encoder is the Motion Estimation process.

Figure 3.3 shows the distribution of energy consumption for different functional blocks[1] of an H.264 encoder. Figure 3.3 shows that ME is one of the most compute-intensive and energy demanding functional blocks of an H.264 encoder. It can be noticed that ME may consume up to 65% (Integer-pixel-ME = 56%, Fractional-pixel ME = 9%) of total encoding energy. The energy consumption of ME is directly proportional to the number of computed SADs (Sum of Absolute Differences) to determine the best match (i.e., the MB with the minimum distortion). As discussed in Sect. 2.2.3, state-of-the-art adaptive, fast, low-power, and scalable ME schemes provide a fixed quality-constrained solution or alternatively offer scalability with fixed termination rules that may lead to severe quality artifacts. These approaches do not provide run-time adaptivity when considering the following run-time varying scenarios:

1. available energy (may change due to a changing battery level or changing allocated energy in a multi-tasking system for different application cases)
2. video sequence characteristics (motion type, scene cuts, etc.)
3. user-defined coding conditions (duration, quality level, etc.)

As a result, these approaches are less energy-/power-efficient. Therefore, **an energy-aware Motion Estimation scheme is desirable** that dynamically adapts its configuration considering the above-mentioned run-time varying scenarios while keeping a good video quality (PSNR). Since the video data has diversity (i.e., different frames and/or different MBs in a frame have different spatial and temporal properties), such an energy-aware ME needs to provide a tradeoff between the available energy budget and resulting video quality. Therefore, the key challenge here is: *how much energy budget should be allocated to the ME of one video frame or even one MB when considering run-time varying scenarios*. The number of SAD computations are then determined from the allocated energy-budget of an MB. It needs to be considered that more energy should be allocated to a fast moving and highly-textured MB compared to a stationary or slow moving MB. Since a less ME effort for a fast moving textured MB may result in noticeable quality degradation, *carefully allocating the energy budget to different frames and MBs is crucial*. Therefore, the energy-aware ME needs to be equipped with **an integrated run-time adaptive energy-budgeting scheme**.

[1] In this experiment the fast adaptive motion estimator UMHexagonS [CZH02] is used to have a realistic distribution.

3.2 Energy- and Adaptivity Related Issues for Dynamically Reconfigurable Processors

In Chaps. 1 and 2, it was motivated that *dynamically reconfigurable processors* provide means for run-time adaptivity at the processor level and they are particularly beneficial in applications with hard-to-predict behavior where conventional embedded processors operate inefficiently with respect to energy/power consumption. However, previous approaches in reconfigurable processors (like OneChip [WC96], CoMPARE [SGS98], Proteus [Dal03], Molen [VWG+04], and RISPP [BSH08b, c]) have mainly focused on performance improvement and efficient area utilization while meeting applications' demands and constraints. These reconfigurable processors suffer from the overhead of reconfiguration energy and high leakage due to their fabric structure. Consequently, the major shortcoming of these processors is their high energy consumption (compared to ASICs) and lack of efficient energy management features [Te06]. Moreover, with the evolution of sub-micron fabrication technologies, the consideration of leakage power/energy has become imperative in the energy-aware design of reconfigurable processors. **Efficient high-level energy management schemes are required** that utilize the underlying power shutdown infrastructure (as proposed by [Ge04; MM05; Te06]) to achieve relatively higher leakage reduction. When targeting a high-level energy management scheme, it needs to be considered that, in reconfigurable processors it cannot be determined at compile time which Custom Instruction (CIs) will be reconfigured on which parts of the reconfigurable fabric. This depends upon many factors, for instance, due to:

1. Application-level unpredictability: the execution frequency of CIs of a computational hot spot may vary due to the changing input video sequence characteristics (e.g., texture properties, motion type, scene cuts) or user-defined coding conditions (e.g., quality level, target bit rate).
2. Compile-/design-time unpredictable scenarios in a multi-tasking system
   a. which task will obtain which share of the reconfigurable fabric
   b. what is the task priority (may change at run time)
   c. which task will run under which performance constraint, e.g., due to changing user preferences (e.g., desired frames per second in case of the H.264 application)
3. available energy (may change due to a changing battery level or changing allocated energy in a multi-tasking system)

Let us analyze the case of application level unpredictability in detail using an analytical study of H.264 video encoder showing the varying computational requirements of the H.264 video encoder application (as discussed in Sect. 3.1.2) due to various coding modes, MB types, and ME configurations. Figure 3.4 illustrates the analysis for the distribution of MB types for different sequences with diverse motion properties. In case of high motion, the ratio of I-MBs is dominant. In case of slow-to-medium motion, the number of P-MB is dominant. Kindly note that, such

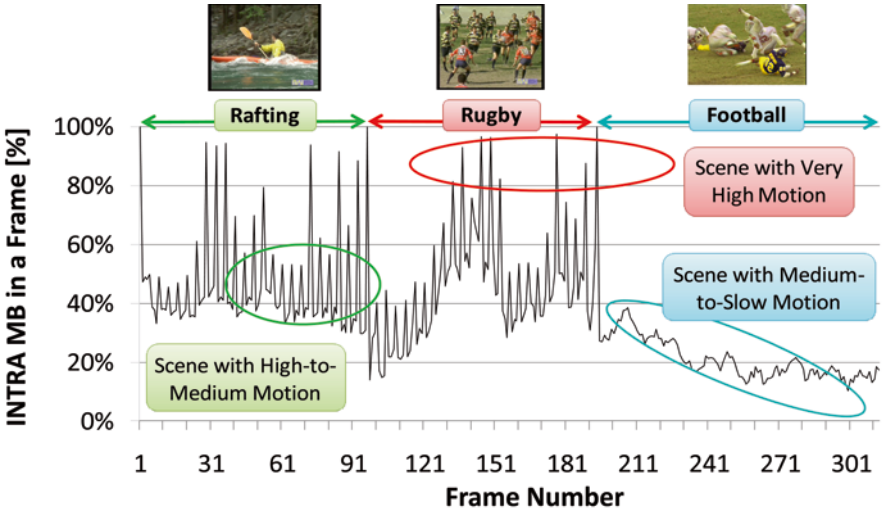


Fig. 3.4 Distribution of I-MBs in slow-to-very-high motion scenes (test conditions: group of pictures=IPPP…, CAVLC, quantization parameter=28, 30fps)

a distribution of I-MB/P-MB cannot be predicted at compile time as the input video sequence is typically unknown. It should be noted that the CIs for I-MBs and P-MBs require different kinds of Data Paths (i.e., elementary hardware accelerators). Therefore, for changing I-MB/P-MB distribution, the corresponding CIs (and Data Paths) will be executed in different frequencies. Summarizing, it is hard to predict at design/compile time that which share of available reconfigurable fabric will be used to accelerate which CIs and where on the fabric their Data Paths will be reconfigured.

Considering the unpredictability from various sources (as discussed above) and significant reconfiguration and leakage power of the reconfigurable processors, **several challenging issues should be addressed by a high-level energy management scheme at run time,** which is the key to realize an adaptive low-power reconfigurable processor architecture. These challenging issues are:

1. Is it beneficial to shutdown regions of the reconfigurable fabric to reduce its leakage (and execute CIs using the core Instruction Set Architecture) or to use a larger share of the reconfigurable fabric to decrease the application execution time at the cost of a higher reconfiguration energy?

   a. This highly depends upon the performance constraints, application characteristics, and the input data properties.
   b. Therefore, it is not trivial to decide under which circumstances the execution using a reconfigurable fabric is energy-efficient or not.

2. How to predict which set of Custom Instructions (CIs) will minimize the energy consumption of a given computational hot spot when considering leakage,

reconfiguration, and dynamic energy under scenarios of run-time changing performance and/or area constraints?

a. At some point in time leakage energy may dominate, while at some other points in time (e.g., due to changed system constraints), reconfiguration energy may dominate.
b. Decisions made solely at design/compile time will therefore with high certainty lead to energy-inefficient scenarios.
c. Hence, *a run-time adaptive scheme* is desirable that chooses an *energy minimizing* set of CIs under varying constraints and then apply shutdown to the temporarily unused set of CIs, such that the total energy consumption is minimized.
d. A comprehensive power model for dynamically reconfigurable processors is required to facilitate an energy management scheme.

3. At which level the power-shutdown decision should be determined?

a. State-of-the-art approaches (as discussed in Sect. 2.4) employ a hardware-oriented shutdown, i.e., the power-shutdown signal is issued based on the usage/state of a particular hardware. However, as discussed above, in dynamically reconfigurable processors such a technique will perform inefficient as it cannot be determined at compile time which CIs will be reconfigured on which parts of the reconfigurable fabric.
b. Therefore, there is a need to raise the abstract level of the power-shutdown decision to the abstraction level of CIs (i.e., an instruction set oriented shutdown) considering the execution length of computational hot spots, i.e., the execution context of an application.

4. Given that the logic and configuration SRAM can be independently shutdown (supported by an appropriate shutdown infrastructure, see Sect. 5.2.2), what kind of different shutdown modes can be realized?

a. Given multiple shutdown modes (as it will be discussed in Chap. 5), how to determine which shutdown mode is beneficial for which set of CIs at what time under run-time varying application contexts?
b. Which muting (i.e., shutdown) modes for CIs will bring more energy reduction while jointly considering the leakage, dynamic, and reconfiguration energy?
c. This decision depends upon the execution length of the computational hot spots during which different CIs are used for the application acceleration in different execution frequencies.
d. Moreover, this decision also depends upon the requirements of upcoming hot spot executions and the performance constraints (i.e., more or less reconfigurable fabric is required to accelerate hot spots).
e. Therefore, *a Selective Instruction Set Muting technique* is required.

Chapters 4 and 5 of this monograph provide algorithms and strategies to address the above-mentioned challenging issues at the application and processor architecture

levels, respectively. In the following, a brief overview of the proposed architectures is provided highlighting the design-, compile-, and run-time steps and requirements from both application architecture and processor architecture perspectives.

3.3 Overview of the Proposed Architectures and Design Steps

Figure 3.5 presents an overview of the monograph contribution for adaptive low-power application and processor architectures in order to address the above-mentioned challenging issues. *Adaptive low-power video encoding* is realized by incorporating adaptive algorithms at the application level that react to the changing battery status, video properties, and user constraints at run time.

At the application level the energy reduction is performed by adaptively reducing the computational requirements of various algorithms used by different functional blocks in the H.264 video encoder (see Chap. 4 for details). First, application architectural adaptations are performed targeting the reconfigurable processors (see Sect. 4.1) and various low-power Custom Instructions and Data Paths are designed (see design in Sect. 4.2). During the application execution, the complexity reduction is performed by adaptively excluding the highly improbable mode options from

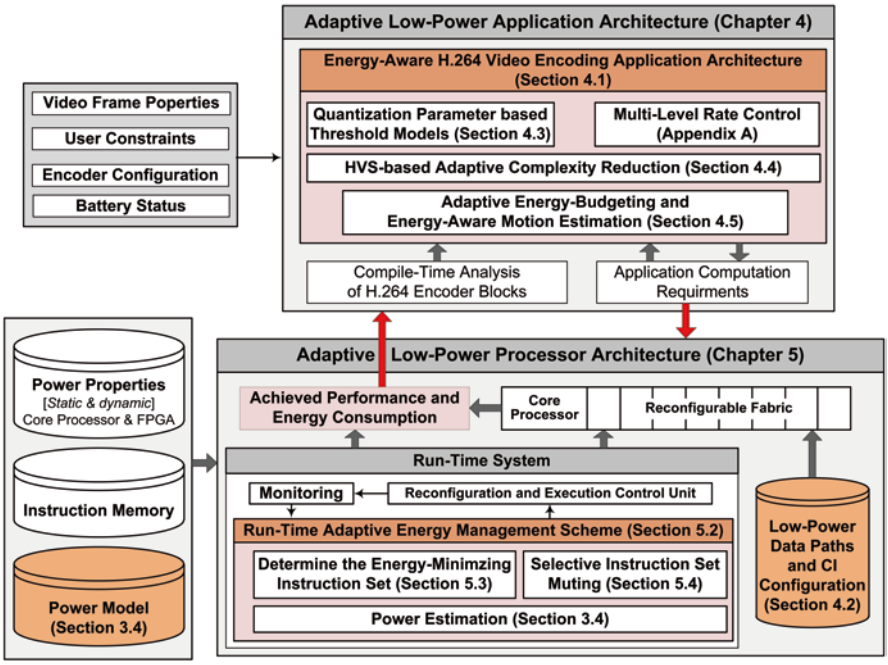


Fig. 3.5 Overview of the adaptive low-power application and processor architectures

the candidate coding mode set (see Sect. 4.4) using Human-Visual System based Macroblock categorization (see Sect. 4.3). Quantization Parameter based threshold models are developed to obtain precise categorization depending upon the coding configuration. Once the final coding mode candidates are determined, adaptive energy-budgeting is performed for the Motion Estimation corresponding to each candidate coding mode. The predicted energy budget is forwarded to the energy-aware Motion Estimation (see details in Sect. 4.5) to select an appropriate *Energy-Quality Class* (i.e., the Motion Estimation configuration). Since the energy-aware adaptations incur a quality loss as a side-effect, in order to compensate this quality loss, a multi-level rate control is designed which determines the Quantization Parameter value for each Macroblock considering its spatial and temporal properties (see Appendix A). It allocates more bits to the complex Macroblocks and less bits to the less-complex ones.

At the processor level, the energy reduction is performed by determining an energy-minimizing instruction set for given area and performance constraints (see details in Sect. 5.3). It requires a power-model for power estimation (see Sect. 3.4). Afterwards, the temporarily unused set of Custom Instructions is muted (i.e., shutdown) to further reduce the leakage energy (see details in Sect. 5.4). For overall processing, low-power Data Paths are reconfigured on the reconfigurable fabric. The consumed energy is fed back to the application level algorithms and also to the processor level algorithms for further adaptations based on the monitored results (achieved performance, energy consumption, etc.).

Figure 3.6 shows an overview of the design-, compile-, and run-time steps at both application and processor levels. These steps are discussed in the following.

At the application level, the coding tool set is finalized at design time and different application architectural adaptations are performed along with the data structures and the data flow (see Sect. 4.1 for details). The finalized application architecture with optimized data flow is implemented. The algorithms for extracting the spatial and temporal video properties are analyzed and a set of algorithms for important video properties is selected (see Sect. 4.3). Afterwards, the algorithms for different functional blocks of the video encoder are designed and implemented.

At compile time, the energy consumption analysis of these functional blocks is performed and the energy consumption is characterized. Depending upon the chosen Motion Estimation algorithm, different patterns and predictor sets are analyzed and selected before exploring the design space of the *Energy-Quality Classes* (see details in Sect. 4.5). A set of common optimal *Energy-Quality Classes* is obtained by performing the design space exploration for various test video sequences. Based on the analysis of the video properties and optimal coding modes for various video sequences, different Quantization Parameter based threshold equations are formulated (see Sect. 4.3.2). Additionally, low-power Custom Instructions (CIs) and their composing Data Paths are designed at compile-time (see Sect. 4.2) based on the *modular* CI composition model of RISPP [Bau09] and opcodes are assigned to different CIs. Moreover, each CI is implemented using core Instruction Set Architecture (cISA).

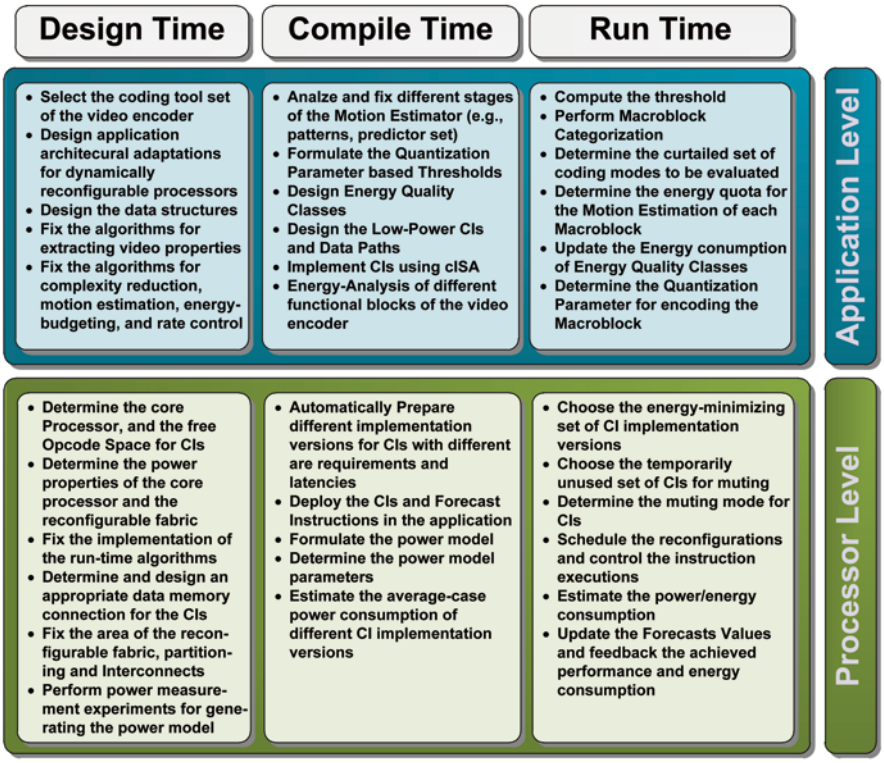


Fig. 3.6 Highlighting different steps to be performed at design, compile, and run time at both application and processor levels

At run time, the Quantization Parameter based threshold equations are used to obtain thresholds that are deployed to partition Macroblocks in different categories with consideration of important Human-Visual System properties (see Sect. 4.3). The Macroblock categories are used for performing the adaptive complexity reduction that excludes improbable coding modes from the mode-decision process to avoid unnecessary energy wastage (see details in Sect. 4.4). Afterwards, the adaptive energy-budgeting is performed that provides the predicted energy budget for the Motion Estimation of one Macroblock. Depending upon the predicted energy budget an appropriate *Energy-Quality Class* is selected and the corresponding configuration is forwarded to the energy-aware Motion Estimation (see details in Sect. 4.5). After the Motion Estimation is completed, the energy of *Energy-Quality Classes* is updated depending upon the current video statistics. For the actual encoding, the Quantization Parameter is determined by a multi-level rate control algorithm that allocates a bit budget to the Group of Pictures and then distributes this budget to different frames within this Group of Pictures. It afterwards determines the final Quantization Parameter value for each Macroblock inside a frame considering its spatial and temporal properties (see Appendix A).

At the processor level, first the architectural parameters (i.e., the core processor, the area of the reconfigurable fabric, connection of the core processor and the reconfigurable fabric, the data memory connection of CIs, etc.) are determined at design time. The leakage and dynamic power properties of the core processor and the reconfigurable fabric are required for designing the power model. As the power model is based on the actual power measurements, the experimental setup is designed and the power of various CI Implementation Versions in hardware is measured (see Chap. 6). The design of the run-time algorithm is also fixed at the design-time.

At compile time, the power model is formulated and the parameters of the model are estimated (see Sect. 3.4 and Chap. 6 for details). Furthermore, the key input (i.e., the multiple Implementation Versions for each CI with area vs. performance/power tradeoff) for the run-time algorithms is prepared automatically at compile time. Note, which Implementation Version for which CI will be used in a given execution scenario cannot be determined at compile time as it depends upon various unpredictable factors (changing performance constraints, available reconfigurable fabric area, input data properties, etc.), as discussed in Chap. 1 and Sect. 3.2. However, the composition of an Implementation Version (i.e., its schedule of Data Path usages) does not change at run time. Afterwards, the average-case power/energy of each CI Implementation Version is estimated by considering various placement cases for different Data Paths of the CI on the reconfigurable fabric (see Sect. 3.4 for the details on how the placement of a Data Path may affect the power consumption of a CI Implementation Version). The CI Implementation Versions and their performance, area, and energy properties are provided to the run-time energy management system. These CIs along with the Forecast Instructions (see details in Sect. 2.3.5) are programmed in the application using inline assembly[2].

At run time, an energy minimizing instruction set is chosen depending upon the monitored CI execution frequency, performance constraint, and the available area of the reconfigurable fabric (see details in Sect. 5.3). Since the execution frequency of CIs may change at run time (depending upon the changing control flow or computational properties of an application or changing performance constraints, as analyzed in Sects. 3.1 and 3.2), the number of actual CI execution is monitored at run time. After choosing the energy-minimizing instruction set, the temporarily unused set of the CIs is determined which is the candidate for muting (i.e., power-shutdown) to reduce the leakage energy. Depending upon the Data Path requirements of the currently executing and the upcoming computational hot spots, a particular muting mode is determined for each CI (see details in Sect. 5.4). Afterwards, the shutdown signals to the corresponding sleep transistors are issued. The Data Paths of the selected CIs are reconfigured on the fabric. Since the actual placement of a Data Path of a CI is determined at run time depending upon the reconfiguration scheduling and replacement (see Sects. 2.3.5 and 3.4), the actual power consumption is estimated at run time.

[2] Note, the assembler is extended to identify the CIs and the Forecast Instructions as it needs to know which instruction format and opcode shall be used for the corresponding CI and Forecast Instruction.

Now, the power model for dynamically reconfigurable processors is presented which is required for explaining the key contribution of this monograph. Details of the power measurement setup, steps for creating the power model, and different test cases to measure the power of individual components will be discussed in Chap. 6.

3.4 Power Model for Dynamically Reconfigurable Processors

Diverse efforts have been undertaken in the estimation and modeling of power consumption in FPGAs [AN04; CJMP03; HLLW08; PWY05; Ze07]. The authors in [HLLW08] presented a technique for rapid estimation of the dynamic power consumption of a hybrid FPGA with coarse-grained and fine-grained units. A dynamic power estimation model for an FPGA-based soft-core processor has been presented in [Ze07]. The authors in [AN04; PWY05] presented more detailed power models for FPGA. An analysis of dynamic power consumption in Virtex-II FPGAs is presented in [CJMP03]. The authors in [BHU03] presented power estimation and power consumption for Xilinx Virtex FPGAs, highlighting the tradeoffs between measured dynamic power and reconfiguration power of different applications. However, none of them comprehensively covers a highly-adaptive reconfigurable processor, i.e., an ASIC-based core Instruction Set Architecture (cISA) in conjunction with an embedded FPGA that supports run-time choices of multiple Implementation Versions per Custom Instruction (CI).

In this section a comprehensive power model for a dynamically reconfigurable processor[3] considering *modular* CIs (like the one discussed in Sect. 2.3.5) is presented. The main challenge is to estimate the power/energy of the *modular* CIs executing on the reconfigurable fabric considering run-time choices of multiple CI Implementation Versions. Before proceeding to the proposed power model, different power consuming parts of a typical computation- and communication-infrastructure on a dynamically reconfigurable processor (like in [BSH08a]) will be investigated.

3.4.1 Power Consuming Parts of a Computation- and Communication-Infrastructure in a Dynamically Reconfigurable Processor

To estimate the dynamic power consumption of an executing CI (P_{CI_dyn}), its specific realization on the reconfigurable fabric (depending on the Implementation Version) needs to be considered. Figure 3.7 shows an abstract schematic of the hardware infrastructure for computation and communication (see details in Sect. 2.3.5) that partitions the reconfigurable fabric into Data Path Containers (DPCs) [BSH08a].

[3] An ASIC-based core Instruction Set Architecture with an embedded FPGA.

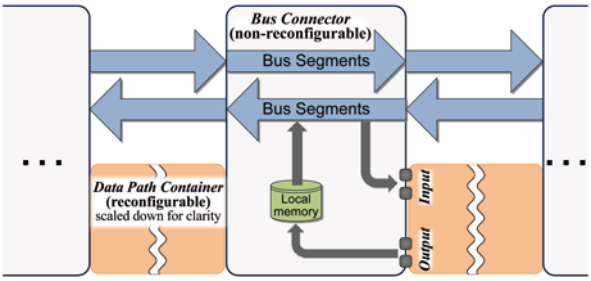


Fig. 3.7 Power-relevant components of the computation- and communication infrastructure to execute CI implementation versions

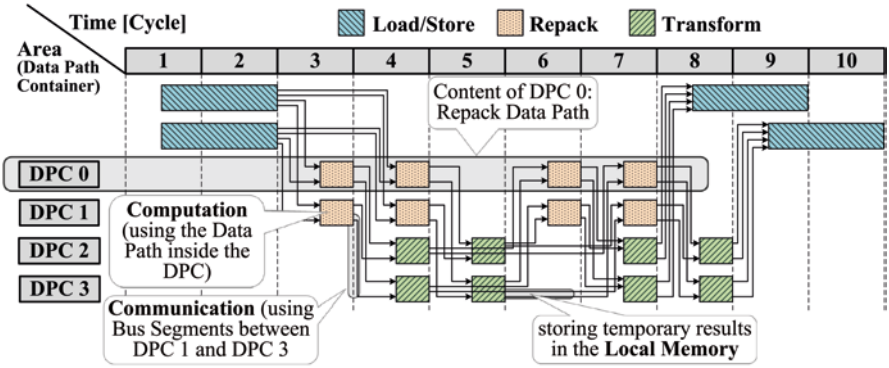


Fig. 3.8 Example for a custom instruction (CI) implementation version

Each DPC is attached to a Bus Connector with small local storage and connected to segmented buses.

Figure 3.8 shows the realization of a certain CI Implementation Version using the hardware infrastructure for the *Hadamard Transform* (from the H.264 encoder application) of a 4×4 input array that is loaded from the data memory. In addition to the actual transformation (*Transform* Data Path, implementing a butterfly using eight 8-bit additions along with bit-level rewiring), the CI requires a rearrangement of the input data and the intermediate results on sub-word level (*Repack* Data Path). The presented Implementation Version in Fig. 3.8 uses two instances of each of the *Transform* Data Path and the *Repack* Data Path, resulting in a CI execution time of 10 cycles. The CI can also be implemented if only one instance of *Transform* and *Repack* is available (e.g., because the reconfigurations of the other Data Paths are not yet completed), resulting in a CI execution time of 15 cycles. The fastest Implementation Version for this CI uses four instances of *Transform* and *Repack* and executes in eight cycles.

Note: different Data Path types typically differ in their required execution energy (e.g., *Repack* requires less energy than *Transform*). Furthermore, the Data Paths need to communicate, for instance, in Fig. 3.8 the result of *Repack* in cycle 3 is the input of *Transform* in cycle 4. However, the result might not be used immediately,

for instance, the result of *Transform* in cycle 5 is only required two cycles later, so it needs to be temporarily stored.

Summarizing: to determine the dynamic power consumption of a CI Implementation Version execution, the following needs to be considered:

- The types of Data Paths and how often they are executed.
- The number of write/read accesses on the local storage.
- The number of bus segments necessary for communicating the intermediate results. This value depends on the relative placement of the communicating Data Paths on the reconfigurable fabric.

Typically, the computation and communication activities during the execution of an Implementation Version vary per cycle: in cycle 3, for instance (see Fig. 3.8), two *Repack* Data Paths are demanded whereas in cycle 5 two *Transform* Data Paths are necessary (similar differences exist for the local storages and the bus lines). Now, the details of the proposed power model are presented in the following.

3.4.2 The Proposed Power Model

The power of a dynamically reconfigurable processor consists of the following components:

$$P_{ReconfProc} = P_{CI_dyn} + \sum P_{DPC_leak} + \sum P_{DPC_reconf} + P_{cISA_dyn} + P_{cISA_leak} \tag{3.1}$$

3.4.2.1 Dynamic Power When Executing a Custom Instruction (CI)

To study the effect of different constituting parameters on the power of a CI Implementation Version, various measurements using an in-house developed FPGA prototyping platform (see Chap. 6 for details) are conducted. As discussed above, the total dynamic power consumption of a CI Implementation Version (see Eq. 3.2) comprises the power of computing Data Paths ($P_{DataPath}$), communicating bus segments (P_{SegBus}), and read/write from the local storage (P_{Memory}). Considering these parameters, the dynamic power of a CI Implementation Version (P_{CI_dyn}) is modeled as:

$$P_{CI_dyn} = \alpha * P_{DataPath} + \beta * P_{SegBus} + \gamma * P_{Memory} + \delta \tag{3.2}$$

On FPGA running at frequency 'F', the energy consumption of an Implementation Version with a latency of 'L' cycles is calculated as: $E_{CI_dyn} = P_{CI_dyn} * (L/F)$. α, β, γ, δ are the model coefficients (see details in Sect. 6.2). δ accounts for the measurement noise. $P_{DataPath}$, P_{SegBus}, and P_{Memory} are explained as below:

Data Path Power ($P_{DataPath} = \left(\Sigma_{i=1}^{n} N_i P_{DataPath_i}\right)/L$): the average power of Data Paths (for an Implementation Version with a latency of L cycles) depends upon the types of Data Paths and how often they are executed. N_i is the number of cycles for which the ith Data Path type is used to realize the Implementation Version. Due to their distinct processing nature, different Data Path types generally differ in their power consumption $P_{DataPath_i}$ (see Sect. 6.2.3 for the measured power results).

Bus Power ($P_{SegBus} = BusSeg_{avg} * P_{bus}$): Data Paths communicate with each other over segmented buses (see Figs. 3.7 and 3.8). The number of bus segments required for communicating the intermediate results depends on the relative position of the communicating Data Paths on the reconfigurable fabric. $BusSeg_{avg}$ is the average number of bus segments employed per cycle and P_{bus} is the average power consumption of one bus segment.

Memory Power ($P_{Memory} = Mem_{avg} * P_{RW}$): The output of a Data Path is temporarily stored in the local memory of the Bus Connector (see Fig. 3.7). Mem_{avg} is the average number of local memory accesses (read or write) per cycle and P_{RW} is the power consumption of a single read or write operation.

$P_{DataPath_i}$, P_{bus}, and P_{RW} are the measured values (see Chap. 6), while the values of N_i, $BusSeg_{avg}$, and Mem_{avg} depend upon a particular CI Implementation Version.

3.4.2.2 Leakage Power of Data Path Containers (DPCs)

P_{DPC_leak} denotes the leakage power of a DPC. Each DPC is treated as a group of Configurable Logic Blocks (CLBs) that are powered-off with sleep transistors (power-shutdown infrastructure will be discussed in Sect. 5.2.2). The power shutdown decision depends upon the temporarily unused set of CIs (see details in Chap. 5).

3.4.2.3 Reconfiguration Power

P_{DPC_reconf} represents the power when reconfiguring a DPC (i.e., a Data Path is loaded onto a DPC). Differently sized Data Paths may require different reconfiguration time due to their varying bitstream lengths. The reconfiguration energy is given by: $E_{DPC_reconf} = T_{reconf} * P_{DPC_reconf}$. The procedure for measuring the reconfiguration power will be discussed in Sect. 6.3.

3.4.2.4 Dynamic and Leakage Power of the core Instruction Set Architecture (cISA)

P_{cISA_dyn} and P_{cISA_leak} denote the dynamic and leakage power consumption of the so-called core Instruction Set Architecture (cISA), respectively. A five-stage pipeline

processor Leon2 with a SPARC V8 instruction set is used in the current prototype platform as the cISA.

3.5 Summary of Adaptive Low-Power Embedded Multimedia System

This chapter has analyzed different issues related to the energy consumption and adaptivity of an advanced H.264/AVC video encoder in a video conferencing application from both application and processor architecture perspectives. It was identified that the H.264/AVC video codec requires more than 70% of the total computational load and energy consumption of a video conferencing application. Since most of the advanced video encoders share a similar computational model and tool set as of the H.264/AVC encoder, it was selected as the target multimedia application in this monograph. Afterwards, the energy and adaptivity related issues in the H.264/AVC application were analyzed. It was found that Mode Decision and Motion Estimation are the most critical components of a video encoder. Moreover, different run-time varying constraints were mentioned. Afterwards, considering the application-level unpredictability, other adaptivity and energy related issues were explored for the dynamically reconfigurable processors. After the analysis, an overview of the proposed processor and application architectures is presented along with the design-, compile-, and run-time steps. At the end, a novel power model for dynamically reconfigurable processors is proposed. This power model considers different types of Data Paths, their placement on the fabric, and memory accesses to estimate the power of Custom Instructions with various Implementation Versions. Moreover, this model considers the leakage and dynamic power of the core processor and the reconfigurable fabric along with the power consumed by the reconfiguration process. The details of the power measurements and model generation methodology will be discussed in Chap. 6. This power model is later on used for energy estimation and run-time energy management at both application and processor architecture levels.

Chapter 4
Adaptive Low-Power Video Coding

This chapter presents the novel adaptive low-power application architecture of advanced H.264 video encoder. It employs an adaptive complexity reduction scheme and an energy-aware Motion Estimation scheme using the novel concept of *Energy-Quality Classes* to realize *adaptive low-power video encoding*.

Section 4.1 presents the H.264 encoder application architectural adaptations for reconfigurable processors. First the basic application architectural adaptations are performed (Sect. 4.1.1) for improving the data flow and data structures. Afterwards, adaptations for reduced computations and reduced hardware pressure are discussed in Sects. 4.1.2 and 4.1.3, respectively. The detailed data flow for the optimized application architecture is discussed in Sect. 4.1.4. The design of low-power Custom Instructions and Data Paths is discussed in Sect. 4.2. The analysis of spatial temporal video properties is explained in Sect. 4.3. Based on this analysis and relevant Human Visual System properties, Macroblock categorization is performed (Sect. 4.3.1) which employs Quantization Parameter based thresholding in order to react to the changing bit rate scenarios (Sect. 4.3.2). This analysis is used by the adaptive computational complexity reduction scheme (Sect. 4.4) to remove the improbable coding modes from the candidate mode set. Section 4.5 presents the energy-aware Motion Estimation scheme. First an adaptive Motion Estimator with multiple processing stages is proposed in Sect. 4.5.1. Afterwards, Sect. 4.5.2 discusses how an energy budget is computed for different Macroblocks and how the *Energy-Quality Classes* are designed and deployed.

4.1 H.264 Encoder Application Architectural Adaptations for Reconfigurable Processors

4.1.1 Basic Application Architectural Adaptations

The JM software of the H.264/AVC video encoder [JVT10] contains a large set of tools to support a variety of applications (video conferencing to HDTV) and uses complex data structures to facilitate all these tools. For that reason, the reference

M. Shafique, J. Henkel, *Hardware/Software Architectures for Low-Power Embedded Multimedia Systems,*
DOI 10.1007/978-1-4419-9692-3_4, © Springer Science+Business Media LLC 2011

software is not a suitable base for research and development of a low-power video encoder.

Therefore, the application architecture of the JM software [JVT10] is passed through a series of following **basic application architectural adaptations** in order to obtain a good starting point, i.e., a so-called H.264 encoder *Benchmark Application*. This Benchmark Application provide a foundation for researching the application architectural adaptations amenable to the reconfigurable processors. The details of these basic application architectural adaptations are as follows:

1. First, the reference software is adapted to contain only *Baseline-Profile* tools (Fig. 4.1a) considering multimedia applications executing on mobile devices. The Baseline-Profile is further truncated/curtailed by excluding Flexible Macroblock Ordering (FMO) and multiple slice (i.e., complete video frame is one slice).
2. Afterwards, the data structure of this application is improved by replacing, for example, multi-dimensional arrays with one-dimensional arrays to improve the memory accesses (Fig. 4.1b). The basic data flow of the application is additionally improved and the inner loops are unrolled to enhance the compiler optimization space and to reduce the amount of jumps.

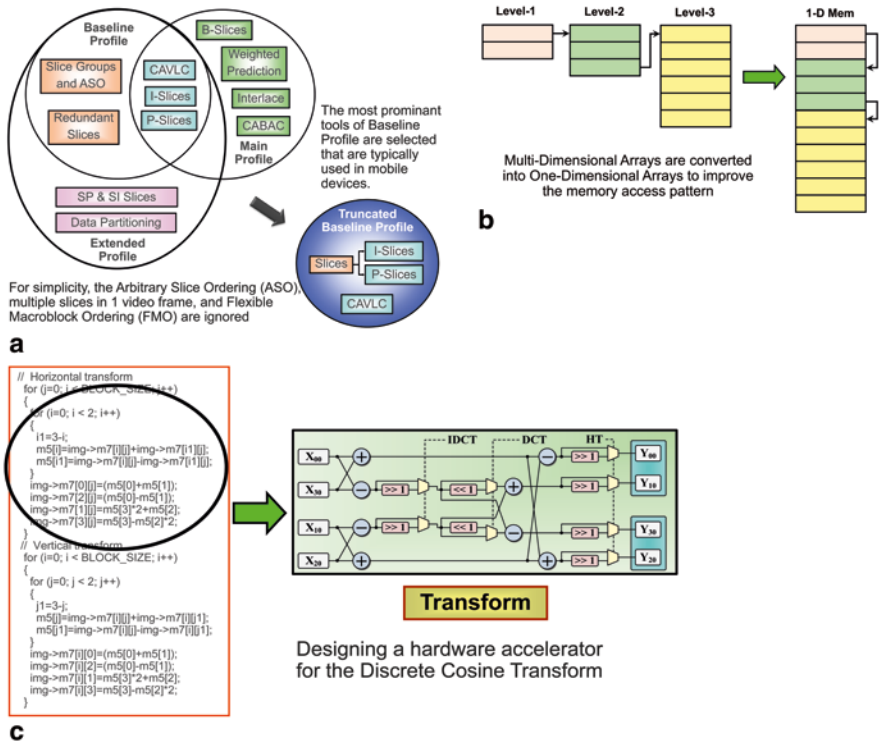


Fig. 4.1 Basic application architectural adaptations to construct the benchmark application. **a** Adapting reference software. **b** Improving data structure. **c** Profiling and Designing Custom Instructions

3. The reference software uses a *Full Search* Motion Estimator which is not practicable in real-world applications and it is only used for quality comparison. Therefore, real-world applications necessitate a low-complexity Motion Estimator. A low-complexity fast and adaptive Motion Estimator called *UMHexagonS* [CZH02] was used to reduce the processing loads of ME process while keeping the visual quality closer to that of Full Search. Full Search requires on average 107811 SADs/frame for Carphone QCIF video sequence (256 kbps, 16 Search Range and 16×16 Mode). On the contrary, UMHexagonS requires on average 4424 SADs/frame. Note, UMHexagonS will also be used as a competitor for the proposed energy-aware Motion Estimation scheme in Sect. 4.5 (p. 104).
4. Afterwards, this application is profiled to detect the computational hot spots (Fig. 4.1c). Several *modular* Custom Instructions (CIs, according the CI model of RISPP as discussed in Sect. 2.3.5) along with their composing low-power Data Paths (i.e., elementary hardware accelerators) are designed and implemented to expedite the hot spots of the H.264 encoder (see details in Sect. 4.2, p. 80). This adapted and optimized application then serves as the Benchmark Application for further architectural adaptations that are amenable to the reconfigurable processors.

Figure 4.2 shows overview of the hot spots (with various functional blocks) in the Benchmark Application. It consists of three main hot spots:

- **Interpolation for Motion Compensation:** An upscaled frame with half-pixel and quarter-pixel values is generated using a six-tap filter and a bilinear filter (see details in [ITU05]). This interpolated frame is then used in the Motion Compensation process. Note, the interpolated data may be used by the Motion Estimation process, however, it is not fixed by the standard.
- **Macroblocks (MB) Encoding Loop:** the main loop for encoding an MB. It consists of:
  - Motion Estimation (ME) using Sum of Absolute Differences (SAD) and Sum of Absolute Transformed Differences (SATD)
  - Motion Compensation (MC)
  - Intra Prediction (IPred)

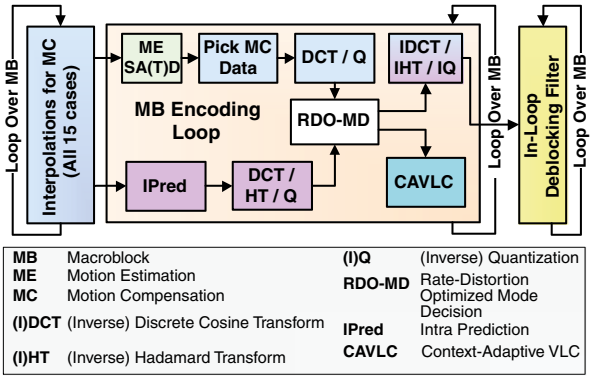


Fig. 4.2 Arrangement of functional blocks in the H.264 encoder benchmark application

- Rate Distortion Optimized Mode Decision (RDO-MD)
- Discrete Cosine Transform (DCT) and Inverse Discrete Cosine Transform (IDCT)
- Hadamard Transform (HT) and Inverse Hadamard Transform (IHT)
- Quantization (Q) and Inverse Quantization (IQ)
- Context Adaptive Variable Length Coding (CAVLC)

• **In-Loop Deblocking Filter:** the filter for removing the blocking artifacts.

These functional blocks operate at the MB-level where an MB can be of type Intra (I-MB: uses IPred for the *spatial* prediction) or Inter (P-MB: uses MC for the *temporal* prediction).

4.1.2 Application Architectural Adaptations for On-Demand Interpolation

Figure 4.3 shows a statistical study on different mobile video sequences with low-to-medium motion considering the fact that the H.264 encoder Benchmark Application interpolates MBs before entering the main MB Encoding Loop (see Fig. 4.2). It is noticed that in each frame the number of MBs for which an interpolation was actually required to process MC is much less than the number of MBs processed for interpolation by the Benchmark Application. After analysis, it was found that the significant gap between the processed and the actually required interpolations is due to the stationary background, i.e., the motion vector (which determines the need for

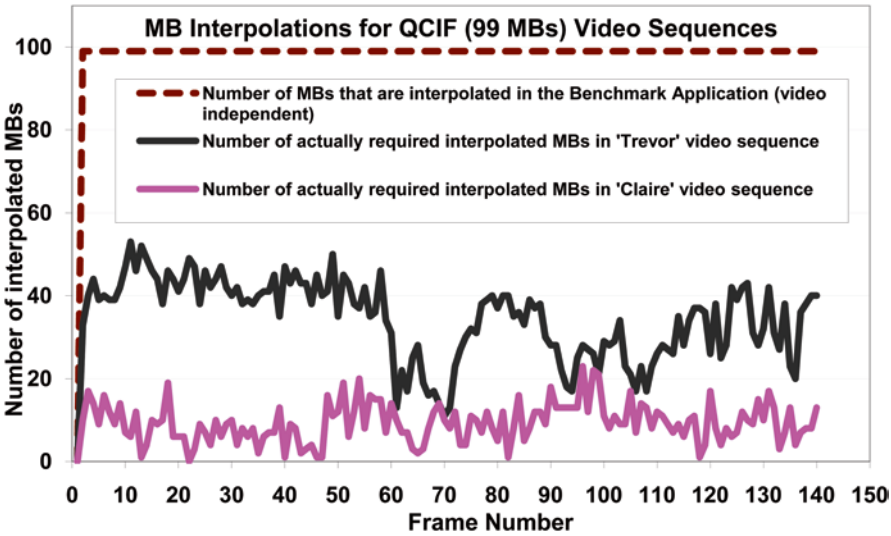


Fig. 4.3 Number of computed vs. required interpolated MBs for two standard test sequences for mobile devices

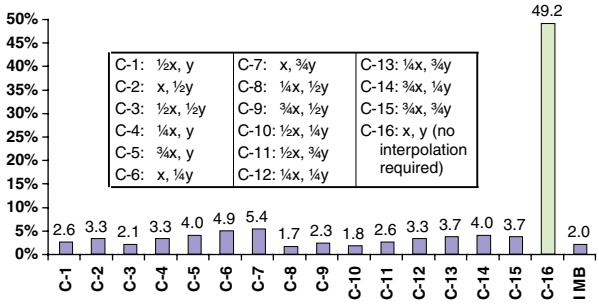


Fig. 4.4 Distribution of different interpolation cases in the carphone video sequence

interpolations) is zero. The interpolation is only required for MBs with motion vectors (given by the ME) with fractional-pixel accuracy. Additionally, even for those MBs that require an interpolation, only one of the 15 possible interpolation cases is actually required (indeed one interpolation case is needed per Block, potentially a sub-part of a MB), which shows the enormous saving potential. The last two bits of the motion vector hereby determine the required interpolation case.

Figure 4.4 shows the distribution of interpolation cases in the 'Carphone' sequence (a standard videophone test sequence with the highest interpolation computation load in the QCIF test-suite). Figure 4.4 demonstrates that in total 48.78% of the total MBs require one of these interpolation cases (C-1 to C-15). The case C-16 is for those MBs where the last two bits of the motion vector are zero (i.e., integer pixel resolution or stationary) such that no interpolation is required. The I-MBs (for Intra Prediction) actually do not require an interpolation either. One of main challenges is to eradicate this problem by shifting the process of interpolation after the ME computation. This enables to determine and process only the required interpolations, i.e., so-called on-demand interpolation. Figure 4.5 shows the application architectural adaptation to reduce the overhead of excessive interpolations. After performing the ME, the motion vector is obtained, which allows to perform only the required interpolation. The Fractional-pixel ME might additionally require interpolations, but it is avoided in most of the cases (C-16) due to the stationary nature of these MBs. The proposed application architecture maintains the flexibility for the designer to choose any low-complexity interpolation scheme for Fractional-pixel ME, e.g., [SJ04].

One of the side effects of shifting the interpolation after ME is that it increases the number of functional blocks inside the MB Encoding Loop. It is noted that—besides the interpolation—there are already several functional blocks inside the MB Encoding Loop (Fig. 4.2). As discussed in Sect. 2.3, due to the significant reconfiguration time, the fabric in the reconfigurable processors is not reconfigured between the processing of a hot spot, i.e., within processing of each MB. Therefore, not all Data Paths of the CIs in the MB Encoding Loop may be supported in the available reconfigurable fabric (depending upon its size). The bigger number of Data Paths required to expedite a computational hot spot corresponds to a high **hardware pressure** inside this hot spot (i.e., a large-sized reconfigurable fabric has to be provided to expedite the hot spot). A higher *hardware pressure* results in:

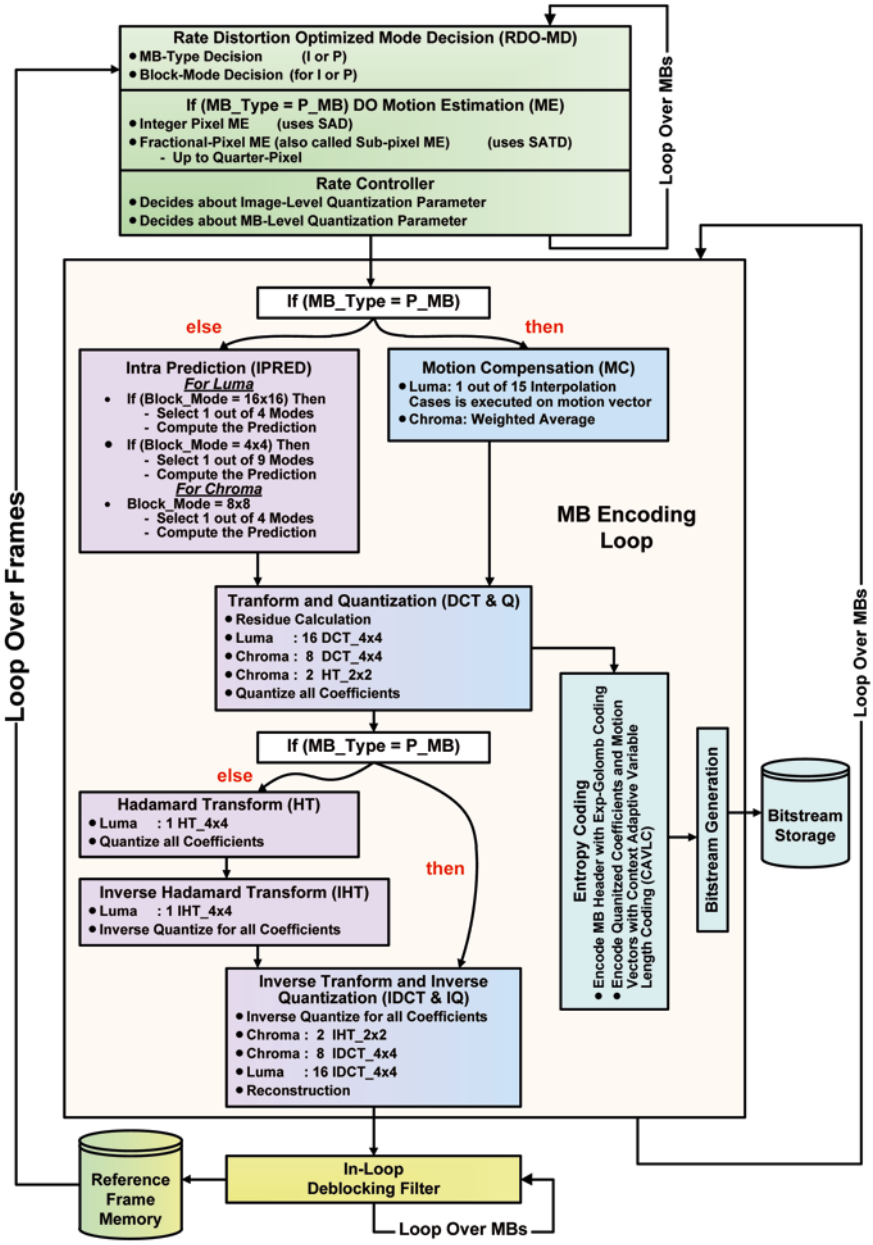


Fig. 4.5 H.264 encoder application architecture with reduced hardware pressure

- more Data Paths that might be required (for meeting the performance constraint) within a hot spot than actually fit into the reconfigurable fabric. Therefore, not all hot spots might be expedited and the CIs are executed using the Core Instruction Set Architecture (cISA) instead, and
- increased reconfiguration overhead (latency, energy), as the reconfiguration time depends on the amount of fabric that needs to be reconfigured.

Both points lead to performance degradations for the reconfigurable processors, depending on the magnitude of *hardware pressure*. This is a drawback for the class of reconfigurable processors and therefore further application architectural adaptations are required to counter this drawback. In the following, the application architectural adaptations to reduce the *hardware pressure* inside the MB Encoding Loop are proposed that introduce the concept of decoupling the Motion Estimation and Rate Distortion Optimized Mode Decision from the main MB Encoding Loop.

4.1.3 Application Architectural Adaptations for Reducing the Hardware Pressure

Although the application architectural adaptation for on-demand interpolation (Sect. 4.1.2) results in a significant reduction of performed interpolations, it further increases the *hardware pressure* of the MB Encoding Loop, as the hardware for the Motion Compensated Interpolation is now shifted inside this loop. A higher *hardware pressure* has a negative impact when the encoder application is executed on a reconfigurable processor. This is due to the fact that the amount of hardware required to expedite the MB Encoding Loop (i.e., the *hardware pressure*) is increased and not all Data Paths can be accommodated in the available reconfigurable fabric. Moreover, it takes longer until the reconfiguration is completed and the hardware is ready to execute. Therefore, in order to reduce the *hardware pressure* those functional blocks are decoupled that may be processed independent of rest of the encoding process. Decoupling of these functional blocks is performed with the surety that the encoding process does not deviate from the standard specification and a standard compliant bitstream is generated. Motion Estimation (ME) and Rate Distortion Optimized Mode Decision (RDO-MD) are decoupled as they are non-normative and standard does not fix their implementation. However, this decoupling of functional blocks affects the data flow of application (discussed in detail in Sect. 4.1.4).

As ME does not depend upon the reconstructed path of encoder, ME can be processed independently on the whole frame. Therefore, it is taken out of the MB Encoding Loop (as shown in Fig. 4.5) which will decouple the hardware for both Integer- and Fractional-pixel ME. Moreover, it is also worthy to note that some accelerating Data Paths of SATD (i.e QSub, Repack, Transform) are shared by (I) DCT, (I)HT_4×4, and (I)HT_2×2 Custom Instructions (see Table 4.1 in Sect. 4.2). Therefore, after the ME is completed for one frame and the subsequent MB Encoding Loop is started, these reusable Data Paths are already available which reduces

Table 4.1 Custom instructions and data paths for the H.264 video encoder

Functional component	Custom instruction	Description of custom instructions	Accelerating data paths
Motion Estimation (ME)	SAD16×16	Sum of Absolute Differences of a 16×16 Macroblock	SADrow
	SATD4×4	Sum of Absolute (Hadamard-) Transformed Differences of a 4×4 sub-block	QuadSub, Transform, Repack, SAV
Motion Compensation (MC)	MC_Hz_4	Motion Compensated Interpolation for Horizontal case for 4 Pixels	PointFilter, Repack, Clip3
Intra Prediction (IPred)	IPred_HDC	16×16 Intra Prediction for Horizontal and DC	CollapseAdd, Repack
	IPred_VDC	16×16 Intra Prediction for Vertical and DC	CollapseAdd, Repack
(Inverse) Transform	(I)DCT4×4	Residue calculation and (Inverse) Discrete Cosine Transform for 4×4 sub-block	Transform, Repack, (QuadSub)
	(I)HT_2×2	2×2 (Inverse) Hadamard Transform of Chroma DC coefficients	Transform
	(I)HT_4×4	4×4 (Inverse) Hadamard Transform of Intra DC coefficients	Transform, Repack
In-loop Deblocking Filter (LF)	LF_BS4	4-Pixel Edge Filtering for in-loop Deblocking Filter with Boundary Strength 4	Cond, LF_4

the reconfiguration overhead (latency and energy). As motion vectors are already stored in a full-frame based memory data structure, no additional memory is required when ME is decoupled from the MB Encoding Loop. Decoupling ME will also improve the instruction cache usage as same instructions are now processed for long time in one loop. A much better data-arrangement (depending upon the search patterns) can be performed to improve the data cache usage (i.e., reduced number of misses) when processing ME on frame-level due to the increased chances of availability of data in the cache. However, when ME executes inside the MB Encoding Loop these data-arrangement techniques may not help. This is because subsequent functional blocks (MC, DCT, CAVLC etc.) typically replace the data that might be beneficial for the next execution of ME.

The RDO-MD controls the encoded quality by deciding about the type of an MB (I-MB or P-MB). Furthermore, the I-MB/P-MB Mode Decision is also attached with this as an additional RD decision layer. The H.264 Benchmark Application employs an *exhaustive RDO-MD* scheme that computes both I- and P-MB encoding flows with all possible modes and then chooses the one with the best tradeoff between the required bits to encode the MB and the distortion (i.e., video quality) using a Lagrange Scheme, according to an adjustable optimization goal (see details in Sects. 2.2.3 and 4.4). The RDO-MD is additionally taken out of the MB Encoding Loop (see Fig. 4.5) to perform an early decision on MB type (I or P) and Mode (for I or P). This will further reduce the *hardware pressure* in the MB Encoding Loop and the total processing load (either I or P computation instead of both). Shifting RD is

less efficient in terms of bits/MB as compared to the *exhaustive RDO-MD* scheme as the latter checks all possible combinations to make a decision. However, RD outside the MB Encoding Loop is capable to utilize intelligent schemes to achieve a near-optimal solution, for instance, Inter-Modes can be predicted using homogeneous regions and edge map (see details in Sect. 4.4).

The H.264 encoder application architecture with reduced *hardware pressure* provides a good arrangement of processing functions that facilitates an efficient data flow. For multimedia applications, data format/structure and data flow are very important as they greatly influence the resulting performance. Therefore, the complete data flow of the encoder will be discussed along with the impact of the proposed application architectural adaptations.

4.1.4 Data Flow of the H.264 Encoder Application Architecture with Reduced Hardware Pressure

Figure 4.6 shows the data flow diagram of the H.264 application architecture with reduced *hardware pressure*. The boxes show the process (i.e., the processing function of the encoder) and arrows represent the direction of the flow of data structure (i.e., text on these arrows). D1 and D2 are two data stores that contain the data structures for current and previous frames. E1 and E2 are two external entities to store the coding configuration and encoded bitstream, respectively. The format of these data structures is shown in Fig. 4.7 along with a short description.

Motion Estimation (1.0, 1.1) takes Quantization Parameter from the Rate Controller (11.0) and Luma components of current and previous frames (CurrY, PrevY) from the two data stores D1 and D2 as input. It forwards the result (i.e., MV and SAD arrays) to the RDO-MD process (1.2) that selects the type of an MB and its expected coding mode. If the selected type of MB is Intra then the mode information (IMode) is forwarded to the Intra Prediction (2.1) block that computes the prediction using the reconstructed pixels of the neighboring MBs in the current frame (CurrYUV), otherwise PMode is forwarded to the Motion Compensation (2.0) that computes the prediction using previous frame (PrevYUV) and MV. The three transform processes (3.0–3.2) calculate the residue from using Luma and Chroma prediction results (PredYUV) and current frame data (CurrYUV) that is then transformed using 4×4 DCT. In case of Intra Luma 16×16 the 16 DC coefficients (TYDC Coeff) are further transformed using 4×4 Hadamard Transform (6.0) while in case of Chroma 4 DC coefficients (TUVDC Coeff) are passed to 2×2 Hadamard Transform process (5.0). All the transformed coefficients (TCoeff, HTUVDC Coeff, HTYDC Coeff) are then quantized (4.0, 5.1, 6.1). The quantized result (QCoeff, QYDC Coeff, QUVDC Coeff) is forwarded to CAVLC (9.0) and to the reconstructed/backward path, i.e., inverse quantization (6.3, 5.2, 4.1), inverse transform (6.2, 5.3, 7.0–7.2), and reconstruction (8.0). The reconstructed frame is then processed with in-loop Deblocking Filter (10.0) while the output of CAVLC (i.e., bitstream) is stored in the Bitstream Storage (E1). Depending upon the achieved

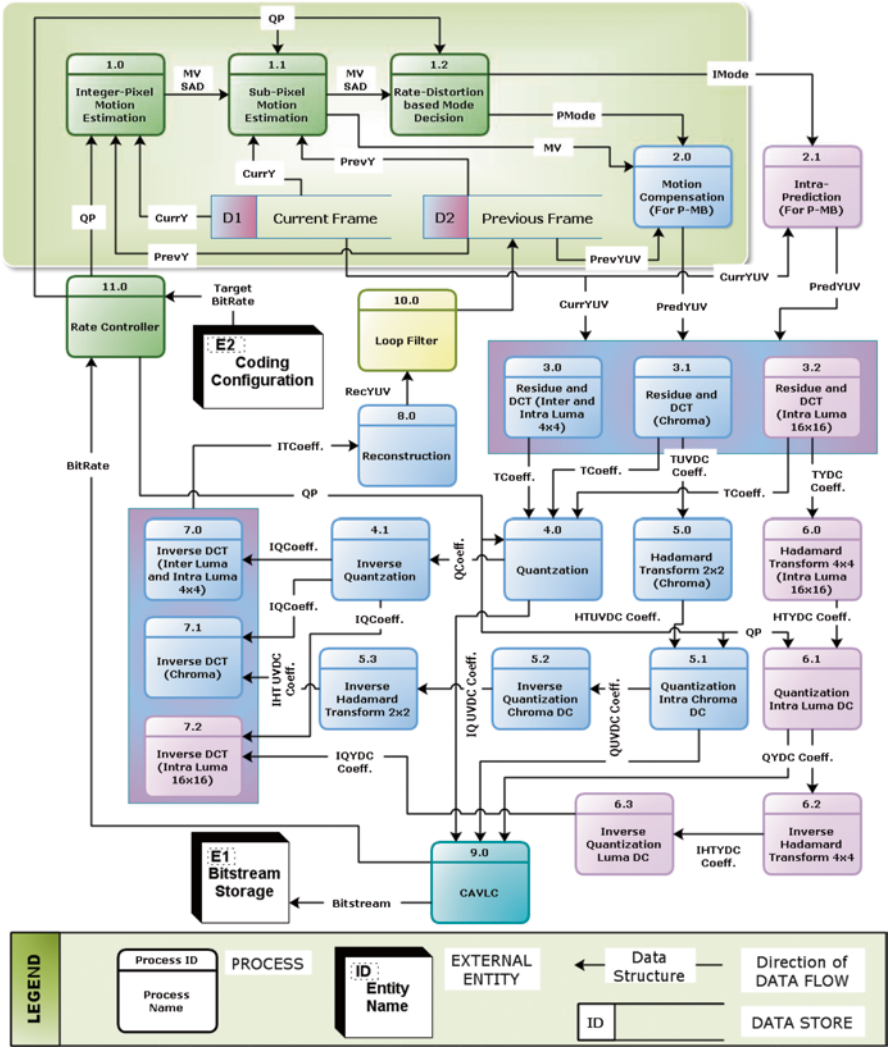
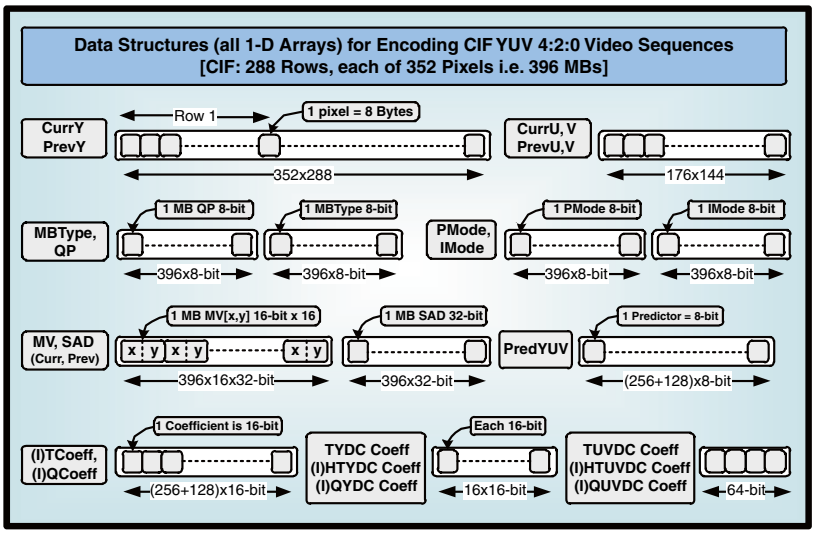


Fig. 4.6 Data flow diagram of the H.264 encoder application architecture with reduced hardware pressure

bit rate and coding configuration (E2) the Rate Controller (11.0) decides about the Quantization Parameter.

After the proposed adaptation, the size of data structure for interpolation result is much smaller than before adaptation. The new PredYUV (Fig. 4.7) data structure requires only *384 bytes ((256+128)*8-bits)* for CIF videos, as the prediction result for only one MB is required to be stored. On the contrary, pre-computing all interpolated pixels up to quarter-pixel resolution instead requires a big data structure (*16*Frame_Size bytes*) storage after interpolation and loading for residual calculation. For Quarter Common Intermediate Format (QCIF, 176×144) and Com-



Description of Data Structures			
MBType	Type of Macroblock (I or P)		
MV, SAD	Motion Vector and SAD Arrays	QP	Quantization Parameter
CurrYUV, PrevYUV	Luma and Chroma Components of Current and Previous Frames	Pred YUV	Luma and Chroma Components of Prediction Data for 1 MB
PMode IMode	Mode of P-MB (e.g.16x16 or 8x8) Mode of I-MB (16x16 or 4x4)	(I)TCoeff, (I)QCoeff	(Inverse) Transformed Coefficients (Inverse) Quantized Coefficients
TYDC Coeff (I)HTYDC Coeff (I)QYDC Coeff	DCT Transformed DC Coefficients for Intra Luma 16x16, (Inverse) Hadamard 4x4 Transformed and (Inverse) Quantized	TUVDC Coeff (I)HTUVDC Coeff (I)QUVDC Coeff	DCT Transformed DC Coefficients for Chroma, (Inverse) Hadamard 2x2 Transformed and (Inverse) Quantized

Fig. 4.7 Description and organization of major data structures

mon Intermediate Format (CIF, 352×288) resolutions, this corresponds to a *1584 (176*144*16/256)* and *6336 (352*288*16/256)* times bigger memory consumption, respectively, compared to the proposed on-demand interpolation.

Pre-computing all interpolation cases results in non-contiguous memory accesses. The interpolated frame is stored in one big memory, i.e., interpolated pixels are placed in between the integer pixel location. Due to this reason, when a particular interpolation case is called for Motion Compensation, the access to the pixels corresponding to this interpolation case is in a non-contiguous fashion (i.e., one 32-bit load will only give one useful 8-bit pixel value). This will ultimately lead to data cache misses as the data cache will soon be filled with the interpolated frame, i.e., including those values that were not required. Contrarily, the on-demand interpolation stores the interpolated pixels in an intermediate temporary storage using a contiguous fashion such that, four interpolated pixels of a particular interpolation case are stored contiguously in one 32-bit register. This improves the overall memory access behavior. Adaptations in the application architecture change the data flow from one processing function to the other. As the looping mechanism is changed, the data flow is changed. On the one hand, performing on-demand interpolation increases the probability of instruction cache miss. On the other hand, it improves

the data cache by offering a smooth data flow between prediction calculation and transform process, i.e., it improves the data flow as it directly forwards the interpolated result for residual calculation and then to DCT. Pre-computation is beneficial in-terms of instruction cache as it processes a similar set of instructions in one loop over all MBs. Conversely, on-demand interpolation is beneficial in-terms of data-cache which is more critical for data intensive applications (e.g., video encoders).

For reduced *hardware pressure* optimization, the Motion Estimation process is decoupled from the main MB Encoding Loop. Since now Motion Estimation executes in a one big loop, the instruction cache behavior is improved. The rectangular region in Fig. 4.6 shows the surrounded data structures whose flow is affected by this optimization of reduced *hardware pressure*. Before optimizing for reduced *hardware pressure*, Motion Estimation was processed on MB-level, therefore MV and SAD arrays were passed to the Motion Compensation process in each loop iteration. Since the encoder uses MVs of the spatially neighboring MBs for Motion Estimation, the data structure provides the storage for MVs of complete video frame (e.g., *396*32-bits* for a CIF frame). After optimizing for reduced *hardware pressure*, there is no change in the size of MV and SAD data structures. The MV and SAD arrays of the complete video frame are forwarded just once to the Motion Compensation process.

Additionally, now RDO-MD can be performed by analyzing the neighboring MVs and SADs. The type of MB and its prediction mode is stored at frame-level and is passed to the prediction processes. Without the proposed adaptations (i.e., when processing Motion Estimation and RDO-MD at MB-level), fast Mode Decision schemes cannot use the information of MVs and SADs of the spatially next MBs. On the contrary, the proposed application architecture facilitates much intelligent RDO-MD schemes where modes can be predicted using the motion properties of spatially next MBs, too (see Sect. 4.4).

Summarizing: the proposed application architectural adaptations not only save the excessive computations (thus energy) using on-demand interpolation for Motion Compensation and relax the *hardware pressure* in case of reconfigurable processors by decoupling the Motion Estimation and RDO-MD processes but also improves the data flows and instruction cache behavior.

4.2 Designing Low-Power Data Paths and Custom Instructions

For accelerating the hot spots of H.264 encoder application, various *modular* Custom Instructions (CIs) were designed and implemented. Table 4.1 gives the description of the implemented CIs of the H.264 video encoder. It is noticeable that some Data Paths (especially Repack and Transform) are used to implement different CI types. The measured and estimated power values for these CIs and the Data Paths are provided in Sect. 6.2.3.

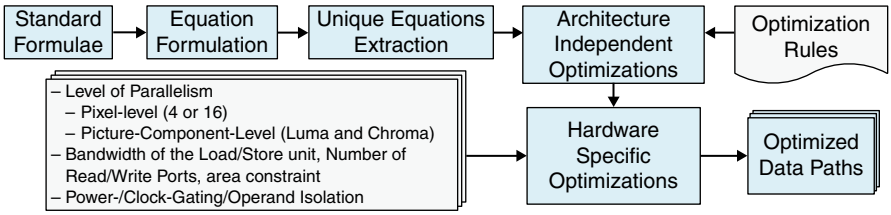


Fig. 4.8 Steps to create optimized data paths from the standard formulae

To design CIs, the constraints imposed by the architecture need to be considered (as discussed in Sects. 2.3.3 and 2.3.5). A typical Data Path has two 32-bit input, two 32-bit output, an 8-bit control signal, a clock, and a reset signal. Moreover the size of an Data Path is limited to approximately 500 Slices to ensure that it will fit in the so-called Data Path Containers (DPC). Additionally, there are special Data Paths containing four inputs and four outputs which are reserved for most commonly used functions (e.g., packing, adding values, re-arranging data). To receive the data needed from memory, two 128-bit Load/Store Units are available.

Different optimizations are performed to reduce the number of operations in a Data Path that directly results in area and power reduction. Figure 4.8 presents the steps to create optimized Data Paths from the formulae specified in H.264 standard [ITU05]. First, the standard formulae are transformed into pixel processing equations that are then processed for architecture-independent optimizations under a given set of optimization rules and constraints. A set of unique equations is extracted followed by optimizations at multiple levels to enhance the level of operation reusability. A set of architectural constraints (as discussed above) is considered to perform hardware level optimizations resulting in a low power Data Path. In the following sections, the design of important CIs from H.264 encoder and their composing Data Paths is described. The design for the Deblocking Filter CI in detail along with the proposed optimizations.

4.2.1 Designing the Custom Instruction for In-Loop Deblocking Filter

The H.264 codec employs an in-loop adaptive Deblocking Filter (after the reconstruction stage) for removing the blocking artifacts at 4×4 sub-block boundaries. The filtered image is used for motion-compensated prediction of future frames. Each boundary of a 4×4 sub-block is called one 4-pixel edge onwards as shown in Fig. 4.9. Each Macroblock (MB) has 48 (32 for Luma and 16 for Chroma) 4-pixel edges. The standard specific details of the filtering operation can be found in [ITU05].

Algorithm 4.1 shows the filtering conditions and filtering equations for Boundary Strength=4 (as specified in [ITU05]) where p_i and q_i (i=0, 1, 2, 3) are the pixel values across the block horizontal or vertical boundary as shown in Fig. 4.10.

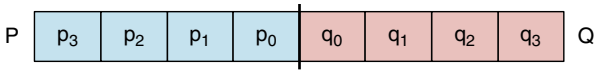


Fig. 4.9 4-Pixel edges in one macroblock

Compute Filtering Conditions and Filtered Pixels for Boundary Strength=4

```
IF (abs(q0–p0) < α) THEN
    IF (abs(q0-q1) < β) & (abs(p0-p1) < β) THEN
        IF (chromaEdgeFlag==0) THEN
            aq = abs(q0-q2) < β; ap = abs(p0-p2) < β;
        END IF
        IF (Boundary_Strength==4) THEN
            IF (chromaEdgeFlag==0)&(ap < β && Abs(p0 – q0) < ((α >> 2) + 2)) THEN
                p'0 = (p2 + 2*p1 + 2*p0 + 2*q0 + q1 + 4) >> 3;
                p'1 = (p2 + p1 + p0 + q0 + 2) >> 2;
                p'2 = (2*p3 + 3*p2 + p1 + p0 + q0 + 4) >> 3;
            ELSE
                p'0 = (2*p1 + p0 + q1 + 2) >> 2;    p'1 = p1;    p'2 = p2;
            END IF
            IF (chromaEdgeFlag==0)&(aq < β && Abs(p0 – q0) < ((α >> 2) + 2)) THEN
                q'0 = (p1 + 2*p0 + 2*q0 + 2*q1 + q2 + 4) >> 3;
                q'1 = (p0 + q0 + q1 + q2 + 2) >> 2;
                q'2 = (2*q3 + 3*q2 + q1 + q0 + p0 + 4) >> 3;
            ELSE
                q'0 = (2*q1 + q0 + p1 + 2) >> 2;    q'1 = q1;    q'2 = q2;
            END IF
        END IF
    END IF
END IF
```

Algorithm 4.1 The Filtering Process for Boundary Strength=4

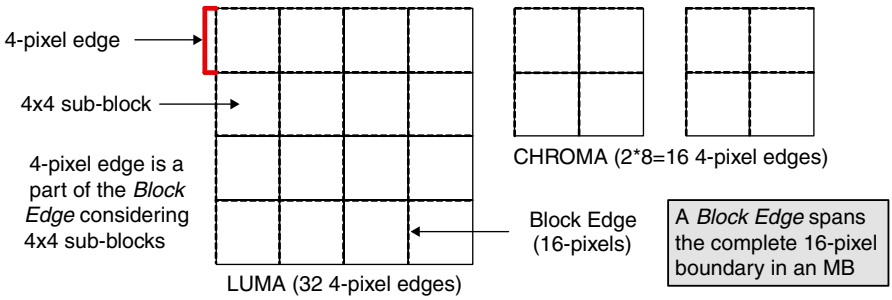


Fig. 4.10 Pixel samples across a 4 × 4 block horizontal or vertical boundary

Figure 4.11a shows the Deblocking Filter CI (named *LF_BS4*, Table 4.1) that targets the processing flow of Algorithm 4.1. This *LF_BS4* CI filters one four-pixel edge, which corresponds to the filtering of four rows each with 8 pixels. The *LF_BS4*

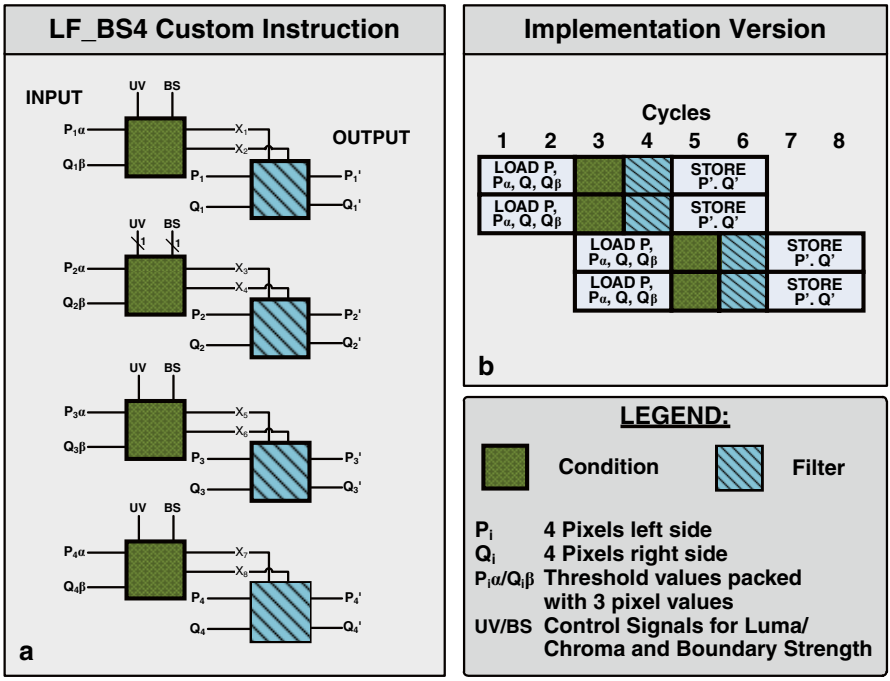


Fig. 4.11 Custom instruction for in-loop deblocking filter with example schedule

CI constitutes two types of Data Paths: the *condition* Data Path (Cond) computes all the conditions (Fig. 4.12a) and the *filter* Data Path (LF_4) performs the actual filtering operation (Fig 4.12b). The *LF_BS4* CI requires 4 Data Paths of each type to filter four rows of an edge. Threshold values α and β are packed with *P* (4-pixel group on left side of the edge; see Fig. 4.10) and *Q* (4-pixel group on right side of the edge) type pixels and passed as input to the control Data Path. The *UV* and *BS* act as control signals to determine the case of Luma-Chroma and Boundary Strength, respectively. The *condition* Data Path outputs two 1-bit flags *X1* (for filtering P-type, i.e., p_i pixels) and *X2* (for filtering Q-type, i.e., q_i pixels) that act as the control signals of the *filter* Data Path. The two sets of pixels (*P* and *Q* type) are passed as input to this Data Path and appropriate filtered pixels are chosen depending upon the two control signals.

Figure 4.11b shows the processing schedule of an Implementation Version of the *LF_BS4* CI with two instances of each of the *condition* and *filter* Data Paths. In first two cycles, two rows are loaded (*P* and *Q* of one row are loaded together due to the availability of 128-bit memory access[1]). In cycle 3, two *condition* Data Paths are executed in parallel followed by two parallel *filter* Data Paths in the cycle 4 to get the filtered pixels for 1st and 2nd row of the edge. In the mean time, next two loads

[1] It demonstrates how the available memory bandwidth can affect the design of a Custom Instruction (CI). This schedule highly depends upon the two 128-bit ports. In case only one port is available, only two pairs of condition-filter Data Paths would be sufficient to exploit the available memory bandwidth.

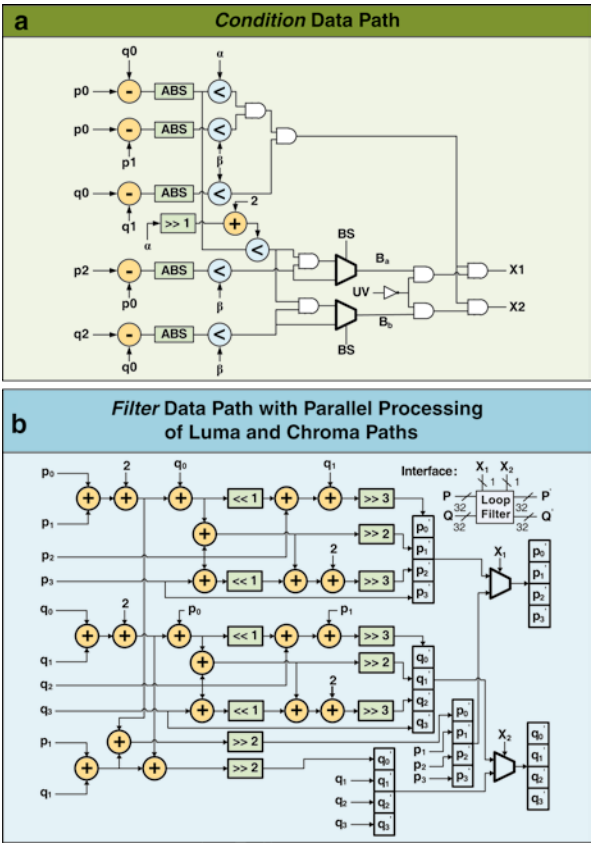


Fig. 4.12 The data paths for filtering conditions and filtering operation constituting the for custom instruction for in-loop deblocking filter

are executed. In cycle 5 and 6, the filtered pixels of 1st and 2nd rows are stored while *condition* and *filter* Data Paths are processed in parallel for 3rd and 4th rows. In cycle 7 and 8, the filtered pixels of 3rd and 4th rows are stored.

Now the two Data Paths are discussed. All of the *if-else* equations are collapsed in one *condition* Data Path that calculates two outputs to determine the final filtered values for the pixel edge. In hardware, all the conditions are processed in parallel and the proposed hardware implementation is 130× faster than the software implementation (i.e., running on GPP). It is noticed that the *condition* Data Path contains sub-byte and bit-level computations which are amenable to fine-grained reconfigurable fabric.

Figure 4.12b shows the optimized Data Path to compute the filtered pixels for Luma and Chroma and selects the appropriate filtered values depending upon *X1* and *X2* flags. This Data Path needs fewer operations to filter the pixels on block boundaries as compared to the standard equations. This Data Path is designed considering the steps shown in Fig. 4.8. It exploits the redundancies in the operation sequence, re-arranges the operation pattern, and reuses the intermediate results as much as possible. The shift operations are realized as bit-level rewiring. Note that the *filter* Data Path is made more reusable using multiplexers, thus both paths are processed

in parallel and the output of one part is selected depending upon which condition is chosen at run time. It is used to process two cases of Luma and one case of Chroma filtering depending upon the filtering conditions. The filtering of a four-pixel edge in software (i.e., running on GPP) takes 960 cycles for Boundary Strength = 4 case. The proposed CI (Fig. 4.11a) using these optimized Data Paths (Fig. 4.12a and b) requires only 8 cycles (Fig. 4.11b), i.e., a speedup of 120×. The measured power results the Data Paths and the estimated power of *LF_BS4* CI are presented in Sect. 6.2.3.

4.2.2 Designing the Custom Instructions for Motion Estimation

As discussed in Chap. 2, Sum of Absolute Differences (SAD) is used for Integer-Pixel Motion Estimation (IME) and Sum of Absolute Transformed Differences (SATD) is used for the Fractional-Pixel ME (FME). The CI *SAD16 × 16* computes the SAD of a complete MB that requires 256 subtractions, 256 absolute operations, 255 additions along with loading of 256 current and 256 reference MB pixels from memory. The *SAD16 × 16* CI constitutes two instances of the SADrow Data Path (Table 4.1) that computes SAD of 4 pixels of current MB w.r.t. 4 pixels of reference MB.

The *SATD4 × 4* CI (Fig. 4.13) uses four types of Data Paths to perform a complete 4×4 SATD operation.

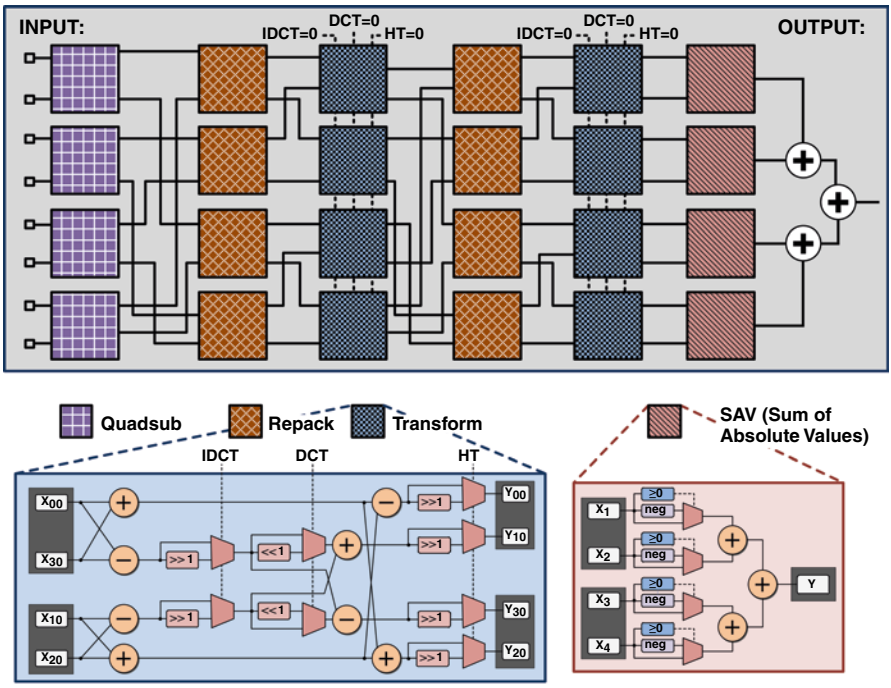


Fig. 4.13 Custom instruction for SATD4×4 showing the transform and SAV data paths

- **QuadSub** performs 4 subtractions; it takes eight unsigned 8-bit pixels P_i, Q_i, $i=0...3$ and returns four 16-bit signed residue outputs, i.e., $R_i=P_i-Q_i$; for $i=0...3$.
- **Repack** rearranges the 16-bit half-words of its 32-bit inputs by packing two 16-bit LSBs and two 16-bit MSBs in two 32-bit outputs. If $input_1=X_1 \circ X_2$ and $input_2=X_3 \circ X_4$, then $output_1=X_1 \circ X_3$ and $output_2=X_2 \circ X_4$.
- **Transform** (Fig. 4.13) performs a 4-point butterfly of (Inverse) Discrete Cosine Transform or (Inverse) Hadamard Transform. Four Transform Data Paths are used to perform a Hadamard Transform along one axis using only additions and subtractions. The second stage of this operation performs an additional arithmetical right-shift on the four results.
- **SAV** (Fig. 4.13) computes the absolute values of its four 16-bit inputs and returns their sum. After the SAV Data Path, the four results are accumulated with three additions to complete the *SATD4 × 4* CI.

4.2.3 Designing the Custom Instruction for Motion Compensation

As discussed in Sect. 2.2.1, Inter Prediction uses block-based MoHi Btion Compensation (MC) that employs a six tap Finite Impulse Response (FIR) filter with weights [*1/32, –5/32, 20/32, 20/32, –5/32, 1/32*] to generate the samples at half-pixel location for the Luma component of the reference frame.

The *MC_Hz_4* CI (Fig. 4.14) computes the half-pixel interpolated values. It takes two 32-bit input values containing eight pixels and applies a six-tap filter.

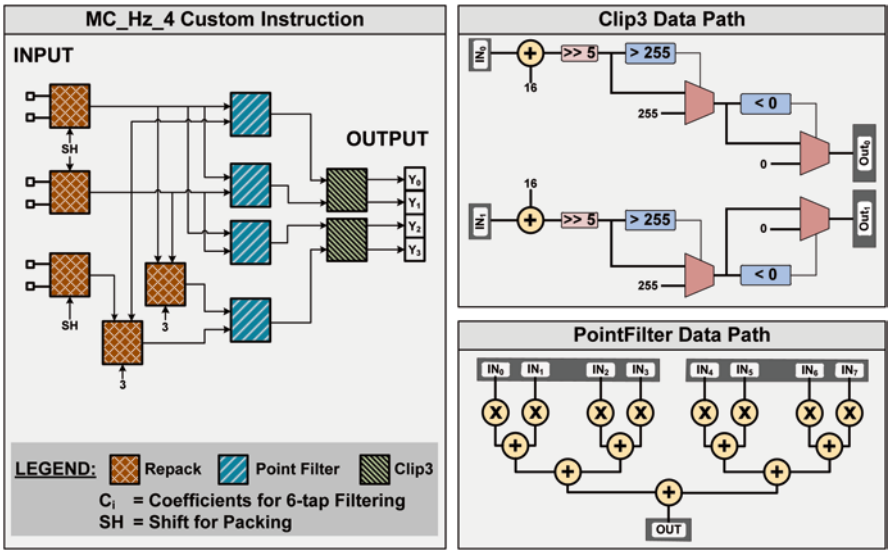


Fig. 4.14 Custom instruction for motion compensation showing different data paths

Table 4.2 Implementation results for various data paths of the H.264 video encoder

	Characteristics			
	# Slices	# LUTs	Latency (ns)	Reconfiguration time[a] (ms)
Clip3	252	413	9.8	0.91
PointFilter	184	300	15.1	0.86
LF_4	144	236	11.6	0.80
Cond	82	132	8.1	0.78
CollapseAdd	36	58	7.4	0.70
SADrow	104	185	13.0	0.79
SAV	58	93	8.4	0.78
Transform	124	217	7.5	0.82
QuadSub	20	32	3.2	0.70

[a] Using 36 MB/s reconfiguration bandwidth

In case of an aligned memory access, Repack rearranges the data for the filtering operation. Then the PointFilter Data Path (Fig. 4.14) performs the actual six-tap filtering operation. Afterwards, Clip3 Data Path (Fig. 4.14) performs the rounding and shift operation followed by a clipping between *0* and *255*.

4.2.4 Area Results for the Custom Instructions of H.264 Encoder

Table 4.2 shows the implementation results for various Data Paths of H.264 encoder synthesized for the Xilinx Virtex-II FPGA. The power results and the power measurement procedure will be discussed in Sect. 6.2.3. The critical path ranges from 3.2 ns to 15.1 ns, while the reconfiguration time (T_{reconf}) ranges from 0.70 ms to 0.91 ms.

4.3 Spatial and Temporal Analysis of Videos Considering Human Visual System

Although the digital image and video processing fields are built on a foundation of mathematical and probabilistic formulations, human intuition and analysis play a central role in the choice of one technique versus another, and this choice often is made based on subjective, visual judgments [GW02]. Therefore, important properties of the Human Visual System (HVS) are considered in this scope of this monograph to account for subjective (visual) quality. The luminance samples are represented with the help of 8-bit pixels, where ‘0’ represents the darkest pixel (i.e., black) and ‘255’ represents the brightest pixel (i.e., white). Some important properties of HVS that are important for image and video compression (as inspired from [GW02; Pra01; WOZ02]) are as follows (see [GW02; Pra01; WOZ02] for details):

1. Human eye is more sensitive to brightness compared to color, therefore, the spatial and temporal analysis is performed on the luminance component and the observations can be extrapolated for color components.

   a. When the eye is properly focused, light from an object outside the eye is imaged on the Retina. Pattern vision is afforded by the distribution of discrete light receptors over the surface of the Retina. There are two classes of receptors: Cones and Rods.
   b. The *Cones* function under bright light and can perceive the color tone; therefore, at high levels (showing better discrimination) vision is the function of Cones.
   c. The *Rods* work under low ambient light and can only extract the luminance information; therefore, at low levels of illumination vision is carried out by activity of the Rods. Rods serve to give a general, overall picture of the field of view and they are not involved in color vision.
   d. Therefore, at low ambient light, color has less importance compared to the luminance.

2. Perceived color of an illuminating light source depends upon the wavelength range in which it emits energy. Green wavelength contributes most to the perceived brightness. There exists a secondary processing stage in the HVS, which converts the three color values obtained by the Cones into one value that is proportional to the luminance and two other values that are responsible for the perception of chrominance, such that $Y=\int C(\lambda)a_y(\lambda)d\lambda$. C is the radiant intensity distribution of a light, λ is the wavelength, and $a_y(\lambda)$ is the relative luminous efficiency function.
3. The subjective brightness (intensity as perceived by the HVS) is a logarithmic function of the light intensity incident on the eye. Since digital images are displayed as a discrete set of intensities, the eye's ability to discriminate between different intensity levels is an important consideration.

   a. The *perceived brightness* is a function of contrast and light intensity. Visual system tends to overshoot and undershoot at the boundary of regions of different intensities as demonstrated by *Match bands* phenomenon. Another phenomenon is *simultaneous contrast*, where objects appear to the eye to become darker as the background gets lighter.
   b. *Brightness Adaptation:* The total range of distinct intensity levels that an eye can discriminate simultaneously is rather small when compared with the total adaptation range. Below that level, all stimuli are perceived as indistinguishable blacks.
   c. *Weber Ratio* (ΔI_C /I): where ΔI_C is the increment of illumination discriminable 50% of the time with background illumination I. The ability of the eye to discriminate between changes in light intensity at any specific adaptation level is also of considerable interest.
   d. The difference of luminance in a restricted area enhances the subjective importance compared to constant intensity regions.

4. Moving objects capture more attention of the eye compared to the stationary objects.

Considering the above HVS properties, an extensive investigation of several video sequences [Ari08; Xip10] was carried out to subjectively learn the HVS response

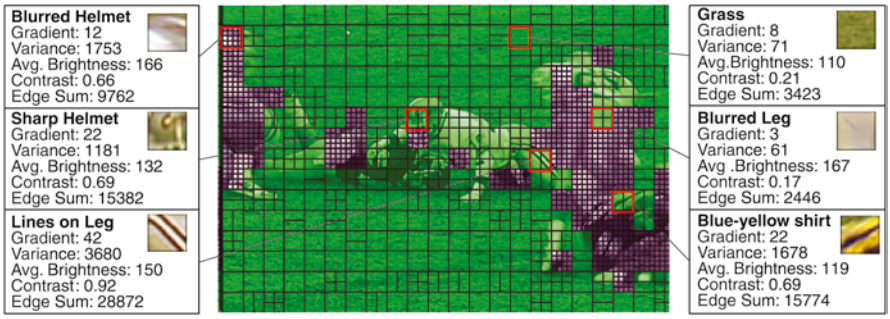


Fig. 4.15 Mode distribution and video statistics in the 7th frame of American Football

to different statistics of video frames and their corresponding coding modes. Figure 4.15 shows the coding mode distribution (P-MBs and I-MBs) for the 7th frame of 'American Football' sequence encoded with the *exhaustive RDO-MD* using JM13.2 software of H.264 video encoder [JVT10]. The MBs with highlighted border show the important MBs in the video frame along with their image statistics in the boxes. The players (helmet and sharp moving body parts, e.g., legs) are the regions of interest. These areas require better coding mode compared to other background objects (e.g., grass). Although the background grass is also thin textured, it is relatively less eye-catching. This grass can be characterized by low gradient and low variance and it changes only minimal from frame to frame. Therefore, it is highly probable to be encoded as P-MB using bigger block sizes, i.e., $P16 \times 16$, $P16 \times 8$, $P8 \times 16$, and $P8 \times 8$ (see Fig. 4.15). Moreover, the changes in brightness (measured by contrast) are also categorized as the region of interest. The higher the contrast is, the bigger the difference of occurring brightness values is. The helmets, legs, and shirts are indicated by high contrast value (compared to the background grass that exhibits low contrast) and thus encoded using $I4 \times 4$ or $P8 \times 8$ and below (see Fig. 4.15). A further measurement for rapid brightness changes is edge detection to identify the strength of an edge and the angle of an edge (Fig. 4.17). Body parts of the players contains significantly prominent edges compared to the stationary grass area.

This analysis revealed that MBs with high texture and fast motion (e.g., fast moving players) are more probable to be encoded as $I4 \times 4$, $P8 \times 8$, $P8 \times 4$, $P4 \times 8$, or $P4 \times 4$ coding mode. On the contrary, homogeneous or low-textured MBs with slow motion (e.g., grassy area) are more probable to be encoded as *SKIP*, $P16 \times 16$, $P16 \times 8$, or $P8 \times 16$ because the Motion Estimation (ME) has high probability to find a good match. Similar behavior was found in various other video sequences leading to the conclusion that majority of coding modes of a video frame can be predicted correctly (with high-probability) using spatial and temporal statistics of the current and previous video frames.

Figure 4.16 shows the percentage distribution of the optimal coding in Rafting and American Football sequences (i.e., fast motion sequences) at different Quantization Parameter (QP) values. It can be noticed in Fig. 4.16 that at higher QP values more than 60% modes are either *SKIP* or $P16 \times 16$. Considering a near-optimal coding mode can be predicted from the spatial and temporal properties of a video sequence, significant complexity and energy reduction may be achieved.

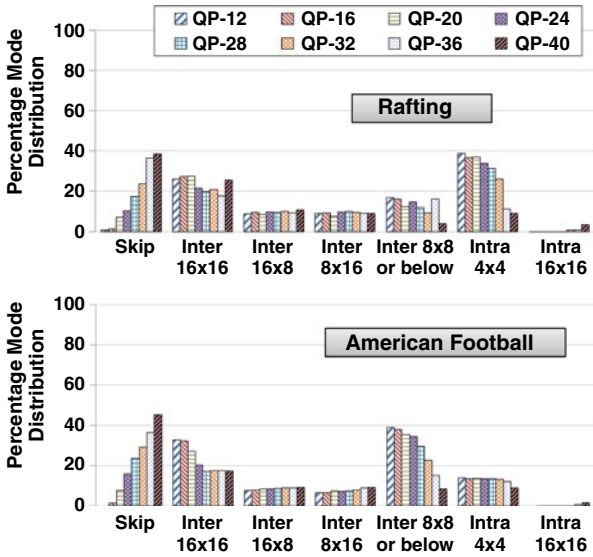


Fig. 4.16 Optimal coding mode distribution in rafting and American Football sequences at different Quantization Parameter (QP) values

Above-discussed analysis revealed that five primitive characteristics of a video frame are sufficient to categorize an MB, thus to predict a probably-correct coding mode. The decision of which video frame property to choose can be made considering the tradeoff between computational overhead and the provided precision in the early mode prediction.

Average Brightness (μ_{MB}) is used to categorize an MB as *dark* or *bright*. It is the average of luminance values $I(i,j)$ of an MB (Eq. 4.1).

$$\mu_{MB} = \left(\sum_{i=0}^{15} \sum_{j=0}^{15} (I(i,j)) + 128 \right) \gg 8 \tag{4.1}$$

Contrast (C_{MB}) is the difference in visual properties that makes an object distinguishable from the background and other objects. In this monograph—due to its simplicity—a modified version of Michelson Contrast [Mic27] is used as shown in Eq. 4.2.

$$C_{MB} = \left[\max_{0<(i,j)<16} I(i,j) - \min_{0<(i,j)<16} I(i,j) \right] \gg 8 \tag{4.2}$$

Variance (σ^2_{MB}) is a measurement for statistical dispersion (Eq. 4.3), thus it is used as descriptor of smoothness or measurement of texture. If all samples have the same brightness, then it is a flat/smooth area and the corresponding Variance is zero.

$$\sigma^2_{MB} = \sum_{i=0}^{15} \sum_{j=0}^{15} (I(i,j) - \mu_{MB})^2 \tag{4.3}$$

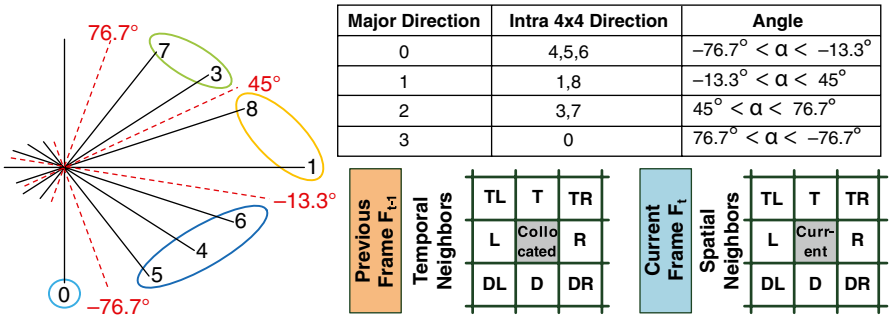
Gradient (G_{MB}) is defined as the rate of change of luminance. In this case, it measures the average rate of change of luminance over a whole 16×16 MB, vertically

(G_x) and horizontally (G_y). Therefore, it is regarded as an approximation of texture. The *first order Gradient* (G_{MB}) along a particular direction is approximated by using the difference between two pixel along that direction (Eq. 4.4).

$$G_x = \left(\sum_{i=0}^{15} \sum_{j=0}^{15} \left| \partial f / \partial x \right| + 128 \right) \gg 8, \quad \partial f / \partial x = I(i,j) - I(i-1,j)$$
$$G_y = \left(\sum_{i=0}^{15} \sum_{j=0}^{15} \left| \partial f / \partial y \right| + 128 \right) \gg 8, \quad \partial f / \partial y = I(i,j) - I(i,j-1)$$
$$G_{MB} = (|G_x| + |G_y| + 1)/2 \tag{4.4}$$

Texture and Edges: In addition to Gradient, a more precise edge detection—operating on a finer granularity—is required to predict the smaller coding modes more precisely. A Sobel Edge Filter is applied to obtain the magnitude and the direction of edges for every 4×4 sub-block. The Sobel Edge Filter has the advantage of providing both differencing and smoothing effect. The total edge values for a 4×4 sub-block, 8×8 block, and 16×16 MB are computed using Eq. 4.5. The direction angle (in degrees) with respect to the x-axis is calculated as $\alpha_{4\times4} = (180°/\pi) * tan^{-1}(G_y/G_x)$. It is used to classify an edge into one of the following four directional groups (Fig. 4.17).

$$S_x = \left(\sum_{i=0}^{3} \sum_{j=0}^{3} \begin{pmatrix} I(i+1,j-1) + 2I(i+1,j) + I(i+1,j+1) \\ -I(i-1,j-1) - 2I(i-1,j) - I(i-1,j+1) \end{pmatrix} \right)$$
$$S_y = \left(\sum_{i=0}^{3} \sum_{j=0}^{3} \begin{pmatrix} I(i-1,j+1) + 2I(i,j+1) + I(i+1,j+1) \\ -I(i-1,j-1) - 2I(i,j-1) - I(i+1,j-1) \end{pmatrix} \right)$$
$$S_{4\times4} = |S_x| + |S_y|; S_{8\times8} = \sum_{k=0}^{3} S_{4\times4}^k; S_{16\times16} = \sum_{k=0}^{3} S_{8\times8}^k \tag{4.5}$$



Major Direction	Intra 4x4 Direction	Angle
0	4,5,6	$-76.7° < \alpha < -13.3°$
1	1,8	$-13.3° < \alpha < 45°$
2	3,7	$45° < \alpha < 76.7°$
3	0	$76.7° < \alpha < -76.7°$

Fig. 4.17 Directional groups with respect to the edge direction angle and notion of spatial and temporal neighboring macroblocks

The experiments revealed that solely image statistics are not sufficient to form a good prediction of possible coding mode. A very high textured MB is well captured by ME if it is stationary or exhibit small translational motion. In fact, the best coding mode may even be a *SKIP* mode if a textured MB is stationary. A prediction purely based on image statistics would possibly tend to an Intra mode choice, thus wasting a noticeable amount of bits. Therefore, in addition to the spatial properties of video sequences, the temporal properties (i.e., motion-field and mode statistics of the previously encoded spatial and temporal neighbors) are also evaluated to corroborate the early prediction decision. The following temporal properties are considered (considering the notion of neighboring MB as shown in Fig. 4.17):

SAD and MV of the Collocated MB: A high SAD value and long MVs of the collocated MB points to the fact that ME could not find a good match, as the MB probably exhibits a hectic motion or it is the part of a suddenly revealed/hidden object. In this case I-MB may be a good choice as a coding mode, while a short MV and a small SAD value indicate an Inter mode candidate. If the collocated MB was predicted to be an I-MB and all P-MB modes were excluded, no ME was executed and therefore no SAD value and MV is available. In this case, only weighted SAD combinations of spatially neighboring MBs are exploited.

SAD and MV of the Spatial and Temporal Neighbors: Similarly, if the SAD value for the neighboring MBs is small with a short MV, the current MB tends to occupy only medium-to-slow motion and it is probably part of an object with similar characteristics or background. High SAD values point to the significant variations in this region, thus I-MB or P-MB with smaller block partitions are the probable coding modes.

Coding Modes of the Neighboring MBs: Another parameter, considered is the correlation of neighboring MB coding modes. If several spatial and/or temporal neighboring MB are encoded as I-MB, the current MB probably belongs to a fast-moving object. Therefore, the probable coding mode for this is also I-MB. On the contrary, if all neighboring MBs are coded with P-MB modes, the current MB is likely to be coded with a P-MB mode.

In conclusion, investigating the spatial and temporal properties of MBs reveals very useful information about the more-probably coding mode.

4.3.1 HVS-based Macroblock Categorization

The spatial and temporal properties (as discussion in Sect. 4.3) are used to categorize Macroblocks (MBs) in the following categories which will hint towards the probable coding mode for these MBs.

Video Frame Statistics based Categorization: Depending upon their spatial statistics, MBs can fall in one or many of the following categories:

Average Brightness ($\mu_M B$)	Very dark (μ_{VD}), dark (μ_D), bright (μ_B), very bright (μ_{VB})
Contrast ($C_M B$)	Low (C_L), high (C_H) contrast
Variance ($\sigma^2_M B$)	Very low (V_{VL}), low (V_L), high (V_H) variance
Gradient ($G_M B$)	Very low (G_{VL}), low (G_L), high (G_H) gradient
Edge (S_{MB})	Low (S_L), highly (S_H) edged

Combinations of the above-defined categories are used to predict the MB content characteristics (Eq. 4.6). A low gradient and a low variance value are very good indicators for smooth and flat regions. If such MBs exhibit slow motion, P16×16 mode is the more probable coding mode. Similarly, smooth steady regions are captured by ME using block sizes above *P8×8*.

$$\begin{aligned} MB_{HighTextured} &= (S_H \ \& \ V_H)||(S_H \ \& \ G_H)||(G_H \ \& \ V_H) \\ S_{MB_StrongThick} &= !V_H \ \& \ S_H \ \& \ \mu_B \ \& \ G_H \\ S_{MB_StrongThin} &= !\mu_B \ \& \ G_H \ \& \ V_H \ \& \ (!\mu_D) \\ S_{MB_ManyThin} &= S_H \ \& \ \mu_B \ \& \ G_H \ \& \ V_H \end{aligned} \tag{4.6}$$

Directional Statistics: An edge direction is called dominant if the edge sum belonging to an edge direction group '*i*' (see Fig. 4.17) significantly contributes to the total edge sum of this MB.

$$EDir_{MB_Dominant} = \begin{cases} 1, & S_i > \varepsilon * S_{MB}; i \in \{0,1,2,3\} \\ 0, & Otherwise \end{cases}$$

$$EDir_{MB_Vt} = S_3 > 0.5 * S_0 \qquad EDir_{MB_Hz} = S_1 > 0.5 * S_0 \tag{4.7}$$

Motion-Field Statistics are obtained using the motion characteristics of the neighboring MBs as follows:

$$\begin{aligned} SAD_{MB_Spatial} &= (SAD_L + SAD_{TL} + SAD_T)/3 \\ SAD_{MB_Neighbors} &= (SAD_L + SAD_T + SAD_{TR} + SAD_{MB_Collocated})/4 \end{aligned} \tag{4.8}$$

Coding-Mode-FieldTotal Statistics are obtained considering the coding modes of the spatial (in the current frame F_t) and temporal (in the previous frame F_{t-1}) neighboring MBs encoded as an I-MB.

$$\begin{aligned} INb_{Spatial} &= isI(MB_{Ft_L}, MB_{Ft_T}, MB_{Ft_TL}, MB_{Ft_TR}) \\ INb_{Temporal} &= isI(MB_{Ft-1_R}, MB_{Ft-1_DR}, MB_{Ft-1_D}, MB_{Ft-1_DL}) \\ INb_{TemporalTotal} &= INb_{Temporal} + isI(MB_{Ft-1_Collocated}) \\ &\quad + isI(MB_{Ft-1_L}, MB_{Ft-1_T}, MB_{Ft-1_TL}, MB_{Ft-1_TR}) \\ INb_{Total} &= INb_{Spatial} + INb_{Temporal} + isI(MB_{Ft-1_Collocated}) \end{aligned} \tag{4.9}$$

4.3.2 QP-based Thresholding

QP-based thresholds are used for the above-discussed MB categorization (Sect. 4.3.1) and predicting the probable coding mode of MBs considering the

Fig. 4.18 Mode distribution of frame 4 in Rafting sequence using the exhaustive RDO-MD for two different QP values: *Left*: QP = 16 and *Right*: QP = 38

above-discussed analysis. For higher QP values, the effect of texture and motion becomes blurry due to the increased number of zero coefficients. It follows the fact that finding a good prediction is easier for ME, thus the number of injected I-MBs decreases. Therefore, with changing QP values, the thresholds (related to the decisions operating on the referenced frames) need to be adapted. This observation is illustrated in Fig. 4.18. Frame encoded using QP = 16 has much higher number of I-MBs compared to the same frame encoded using QP = 38.

Extensive experimentation was performed using different QPs (12–40) and several video sequences (only a small subset of all sequences used for validation in Sect. 4.4.4) to evaluate these thresholds. Polynomial curve fitting (using MATLAB) was performed to obtain threshold equations as a function of QP, see Eq. 4.10. Experiments revealed that only the thresholds for SAD, edge sum and motion vector (thus the major characteristics for motion and texture detection) that operate on the reconstructed video frame react to the changing QPs. Table 4.3 presents the

Table 4.3 Thresholds and multiplying factors used in ACCoReS

Thresholds								
Brightness	μ_{VD}	70	*Variance*	V_{VL}	0.5	*Texture edge*	Th_{Dir}	1000
	μ_D	85		V_L	1.25		$Th_{S\text{-}Fast}$	5000
	μ_B	135		V_H	2		$Th_{S\text{-}Slow}$	1350
	μ_{VB}	175	*Gradient*	G_{VL}	5		$Th_{S\text{-}P16x16}$	500
Contrast	C_L	0.2		G_L	10		$Th_{S\text{-}P8x8}$	1000
	C_H	0.7		G_H	15		Th_{Edge}	200
Intra neighbors	Th_{I1}	6	*Intra neighbors*	Th_{I4}	1	*SKIP*	$Th_{MV\text{-}Skip}$	3
	Th_{I2}	4		Th_{I5}	2		$Th_{SAD\text{-}Skip}$	323
	Th_{I3}	5	*Motion*	Th_{AvgSAD}	2500		$Th_{S\text{-}Skip}$	4096
Multiplying factors								
Motion	δ_1	0.4	*Motion*	δ_4	0.6	*Texture edge*	Ψ	2.5
	δ_2	0.6		δ_5	1.4		ε	0.7
	δ_3	0.5		δ_6	1			

remaining thresholds (which are not affected by changing QPs) and other multiplying factors. These QP-based thresholds and the MB categories (Sect. 4.3.1) based on the analysis of spatial and temporal properties of the input video are used by the Adaptive Computational Complexity Reduction Scheme (ACCoReS, Sect. 4.4) to predict the probable coding mode of MBs.

$$TH_{SAD} = \begin{cases} 2500, QP_{prev} < 20 \\ 9000, QP_{prev} \geq 40 \\ -0.3QP_{prev}^3 + 38.3QP_{prev}^2 - 115.9QP_{prev} + 11897, Otherwise \end{cases}$$

$$TH_{E_High} = \begin{cases} 10000, QP_{prev} < 20 \\ 13000, QP_{prev} \geq 28 \\ -31.25QP_{prev}^2 + 1875QP_{prev} - 15000, Otherwise \end{cases}$$

$$TH_{E_Low} = \begin{cases} 8000, QP_{prev} < 20 \\ 10000, QP_{prev} \geq 24 \\ 500QP_{prev} - 200, Otherwise \end{cases}$$

$$TH_{(MV1,MV2,MV3)} = \begin{cases} (20, 45, 30), QP_{prev} \leq 28 \\ (30, 55, 40), QP_{prev} \geq 36 \\ 1.25QP_{prev} - (15, 10, 5), Otherwise \end{cases} \quad (4.10)$$

4.3.3 Summary of Spatial and Temporal Analysis of Videos Considering Human Visual System

This section illustrated the analysis of spatial and temporal video properties and the relationship of different video properties and the optimal coding mode are discussed. Based on this analysis a detailed Macroblock categorization is performed, while considering the properties of the Human Visual System. In order to react to the run-time varying coding conditions (e.g., bit rates), the thresholds are formulated as a function of Quantization Parameter. This analysis is used by the adaptive complexity reduction scheme, energy-aware Motion Estimation scheme, and multi-level rate control.

4.4 An HVS-Based Adaptive Complexity Reduction Scheme

The proposed Adaptive Computational Complexity Reduction Scheme (ACCoReS, Fig. 4.19) for H.264 encoder predicts the expected Macroblock (MB) type and its coding mode even before processing the actual RDO-MD. It uses the spatial and temporal properties of the input video sequence, i.e., image statistics, motion field properties, and history-based information of the coding modes. The step-by-step procedure is given as follows.

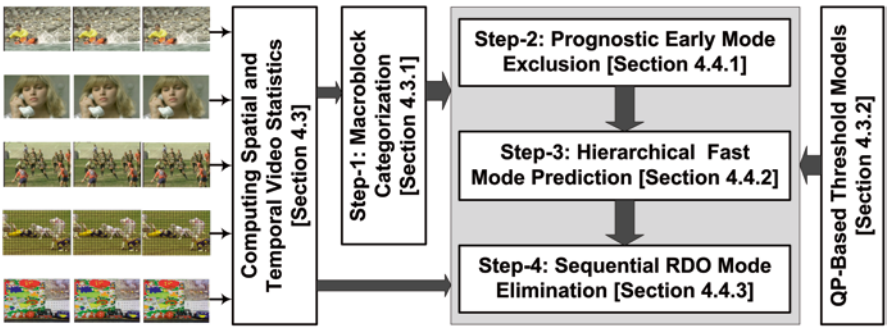


Fig. 4.19 Overview of the adaptive computational complexity reduction scheme (ACCoReS) showing different processing steps and MB categorizations

Step-1: First, the HVS-based categorization of MBs (Sect. 4.3.1) is performed using the spatial and temporal video statistics and the QP-based thresholds (Sect. 4.3.2).

Step-2: Afterwards, a Prognostic Early Mode Exclusion for I-MB and P-MB coding modes is incorporated that excludes the highly unlikely modes. It exploits different image statistics, motion-field properties, and previously computed distortion data (e.g., based on correlation of the modes of previously encoded neighboring MBs) to exclude as many I-MB and P-MB coding modes as possible before the actual RDO-MD and Motion Estimation while keeping the bit rate and distortion loss within an imperceptible range (see Sect. 4.4.4). In many cases the curtailed set of modes is left with either I-MB or P-MB modes, especially for low-motion sequences. As a result it provides a significant complexity reduction (thus processing improvement and reduced energy consumption) at the cost of an insignificant overhead due to the image statistics calculation.

Step-3: A second level Hierarchical Fast Mode Prediction analyzes this curtailed set of modes and provides a set of candidate coding modes, which are then processed for RDO-MD.

Step-4: In the last step, Sequential RDO Mode Elimination is done. It processes the candidate coding modes one-by-one starting from the bigger partitions. After a mode is processed, it is evaluated for the termination condition or to exclude further irrelevant modes.

In the best case, exactly one MB type and only one coding mode corresponding to this MB type (out of 20 for P and 592 for I) is processed. The principal distinctions of the proposed ACCoReS compared to the state-of-the-art approaches are the Prognostic Early Mode Exclusion and the Hierarchical Fast Mode Prediction that exclude more than 70% of the possible coding modes even before starting the fast RDO-MD and ME while keeping the bit rate and distortion loss imperceptible (see Sect. 4.4.4). Now, the different processing stages of ACCoReS will be presented in detail.

4.4.1 Prognostic Early Mode Exclusion

The Prognostic Early Mode Exclusion scheme starts with a classification of MBs into the following two distinct groups using Eq. 4.11:

- Group-A: High-textured MB containing medium to fast motion
- Group B: Flat, homogenous regions with slow motion

Algorithms 4.2 and 4.3 present the pseudo-codes of Prognostic Early Mode Exclusion for both Group-A and Group-B, respectively. In case of Group-A, I16×16 is excluded (line 3) due to high texture and the best choice would most probably be P8×8 or I4×4. However, exclusion of P16x16 at this point is critical as a wrong exclusion may result in a significantly increased bit rate. Therefore, the exclusion decision of P16×16 is performed in the Hierarchical Fast Mode Prediction step. Lines 4–7 and 8–11 check for slow motion using the motion statistics of the spatial neighboring MBs and exclude the smaller block partitions and I4×4 (lines 5, 9). Lines 12–15 detect a high texture and hectic motion region. In this case, I4×4 coding mode is selected and all other modes are excluded.

$$Group_{MB} = \begin{cases} A, (MB_{HighTextured} \ \& \ (\mu_B || C_H)) || V_H || EDir_{MB_Dominant} \\ \quad || (S_{MB_StrongThick} || S_{MB_StrongThin} || S_{MB_ManyThin}) \\ \quad || (MB_{HighTextured} \ \& \ SAD_{MB_Collocated} > Th_{SAD}) \\ \quad || (\mu_{VB} || (INb_{Total} > Th_{I4})) \ \& \ (!G_L) \& (!V_L)) \\ \quad || ((INb_{Spatial} > Th_{I4}) \ \& \ (S_{16x16} > Th_{Edge})) \\ B, Otherwise \end{cases} \tag{4.11}$$

In case of Group-B, a more sophisticated scheme systematically excludes the most unlikely modes. Lines 3–5, 6–9, 13–27 check for slow motion, flat and homogenous

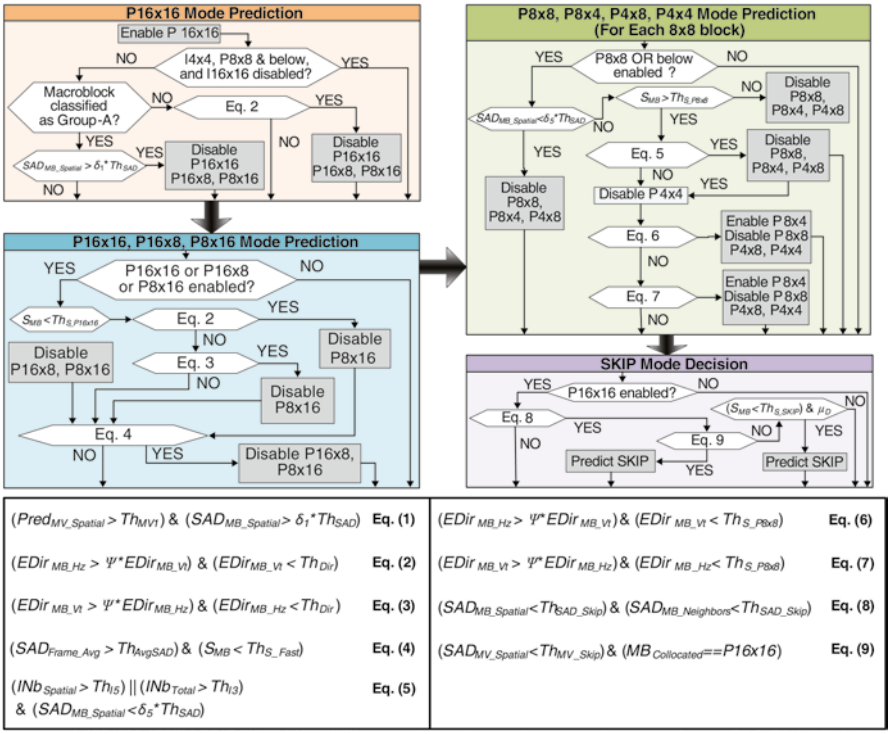
1. **GROUP-A: High-textured MB containing medium-to-fast motion**
2. M = {P16x16, P16x8, P8x16, P8x8, P8x4, P4x8, P4x4, I16x16, I4x4} *// Initialize the possible coding modes with all modes*
3. M ← M \ {I16x16}; *// Exclude I16x16*
4. If ($SAD_{MB_Spatial} < \delta_3^* Th_{SAD}$) {
5. M ← M \ {P8x8, P8x4, P4x8, P4x4, I4x4}; *// Exclude I4x4, P8x8 and below*
6. return; *// Go to Step-3 (Section 4.4.2)*
7. }
8. If (($Pred_{MV_Spatial} < Th_{MV1}$) & ($SAD_{MB_Spatial} < \delta_4^* Th_{SAD}$)) {
9. M ← M \ {I4x4}; *// Exclude I4x4*
10. return; // Go to Step-3
11. }
12. If ((($INb_{TemporalTotal} > Th_{I1}$) & ($INb_{Spatial} > Th_{I2}$)) || (($Pred_{MV_Spatial} > Th_{MV2}$) & (($SAD_{MB_Collocated} > Th_{SAD}$) || ($INb_{Total} > Th_{I3}$)))) {
13. M ← M \ {P8x8, P8x4, P4x8, P4x4}; *// Exclude P8x8 and below*
14. return; *// Go to Step-3*
15. }
16. return; *// Go to Step-3*

Algorithm 4.2 Pseudo-code of group-A for prognostic early mode exclusion

```
1.  GROUP-B: Flat, homogenous regions with slow-to-medium motion
2.  M = {P16x16, P16x8, P8x16, P8x8, P8x4, P4x8, P4x4, I16x16, I4x4} // Initialize the pos-
    sible coding modes with all modes
3.  If (SAD_MB_Spatial <= δ1*Th_SAD) {
4.      M ← M \ {P8x8, P8x4, P4x8, P4x4, I4x4}; // Exclude I4x4, P8x8 and below
5.  }
6.  If (V_L & G_VL & (!S_MB_StrongThick) & (!S_MB_StrongThin) & (!S_MB_ManyThin)) {
7.      If ((SAD_MB_Collocated < δ3*Th_SAD) & (SAD_MB_Spatial < δ2*Th_SAD)) {
8.              M ← M \ {I16x16}; // Exclude I16x16
9.      }
10.     M ← M \ {P8x8, P8x4, P4x8, P4x4, I4x4}; // Exclude I4x4, P8x8 and below
11.     return;     // Go to Step-3
12. }
13. If (V_L & G_L & S_L) {
14.     M ← M \ {P8x8, P8x4, P4x8, P4x4, I16x16}; // Exclude I16x16, P8x8 and below
15.     If ((μ_D || C_L) & (!MB_HighTextured)) {
16.             M ← M \ {I4x4};     // Exclude I4x4
17.             return;     // Go to Step-3
18.     }
19.     If ((Pred_MV_Spatial < Th_MV1) & (SAD_MB_Spatial < δ2*Th_SAD)) {
20.             M ← M \ {I4x4};     // Exclude I4x4
21.     }
22. If ((SAD_MB_Spatial < δ5*Th_SAD) & ((SAD_MB_Neighbors < δ5*Th_SAD) & (!MB_HighTextured)
    & (INb_Spatial > Th_I4)) {
23.             M ← M \ {I4x4};     // Exclude I4x4
24.             return;     // Go to Step-3
25.     }
26.     return;     // Go to Step-3
27. Else {
28.     If (μ_D & G_L & (!MB_HighTextured)) {
29.             M ← M \ {I4x4};     // Exclude I4x4
30.             return;     // Go to Step-3
31.     }
32.     Exclude I16x16 and Re-enable I4x4
33.     If ((SAD_MB_Spatial < δ5*Th_SAD) & ((SAD_MB_Neighbors < δ5*Th_SAD) & (!MB_HighTextured)
        & (INb_Spatial > Th_I4)
            || ((Pred_MV_Spatial < Th_MV1) & (isI(MB_Ft-1_Collocated))) {
34.             M ← M \ {I4x4};     // Exclude I4x4
35.     }
36.     If (Pred_MV_Spatial > Th_MV3) {
37.             M ← M \ {P8x8, P8x4, P4x8, P4x4 };      // Exclude P8x8 and below
38.             return;     // Go to Step-3
39.     }
40.     return;     // Go to Step-3
41. }
```

Algorithm 4.3 Pseudo-Code of Group-B for Prognostic Early Mode Exclusion

region, respectively. In these cases, I4×4, P8×8 and smaller partition modes are excluded. If a homogenous MB is stationary, P16×16 is predicted to be the most probable coding mode; otherwise, I16×16 is additionally processed (line 8). Lines 15–18, 19–25, 28–31 detect low motion and dark low-to-medium texture to exclude



$(Pred_{MV_Spatial} > Th_{MV1})$ & $(SAD_{MB_Spatial} > \delta_1 * Th_{SAD})$ **Eq. (1)**	$(EDir_{MB_Hz} > \Psi * EDir_{MB_Vt})$ & $(EDir_{MB_Vt} < Th_{S_P8x8})$ **Eq. (6)**
$(EDir_{MB_Hz} > \Psi * EDir_{MB_Vt})$ & $(EDir_{MB_Vt} < Th_{Dir})$ **Eq. (2)**	$(EDir_{MB_Vt} > \Psi * EDir_{MB_Hz})$ & $(EDir_{MB_Hz} < Th_{S_P8x8})$ **Eq. (7)**
$(EDir_{MB_Vt} > \Psi * EDir_{MB_Hz})$ & $(EDir_{MB_Hz} < Th_{Dir})$ **Eq. (3)**	$(SAD_{MB_Spatial} < Th_{SAD_Skip})$ & $(SAD_{MB_Neighbors} < Th_{SAD_Skip})$ **Eq. (8)**
$(SAD_{Frame_Avg} > Th_{AvgSAD})$ & $(S_{MB} < Th_{S_Fast})$ **Eq. (4)**	$(SAD_{MV_Spatial} < Th_{MV_Skip})$ & $(MB_{Collocated} == P16x16)$ **Eq. (9)**
$(INb_{Spatial} > Th_{I5}) \,\|\|\, (INb_{Total} > Th_{I3})$ & $(SAD_{MB_Spatial} < \delta_5 * Th_{SAD})$ **Eq. (5)**	

Fig. 4.20 Processing flow of the hierarchical fast mode prediction

I4×4 mode; otherwise, I4×4 mode is re-enabled to avoid significant visual quality loss. Lines 33–39 assure that modes with smaller block partitions are only excluded if low motion and/or low textured are detected.

4.4.2 Hierarchical Fast Mode Prediction

The Hierarchical Fast Mode Prediction (Fig. 4.20) performs a more refined second-level mode exclusion to obtain a set of candidate coding modes, which is later evaluated by the RDO-MD process with an integrated Sequential RDO Mode Elimination mechanism.

P16×16 Mode Prediction: If all modes except *P16×16* are already excluded, then *P16×16* is processed unless *SKIP* mode is detected in the last step of Fig. 4.20. On the contrary, *P16×16* is excluded if the MB has fast motion and high texture.

P16×16, P16×8, P8×16 and P8×8 Mode Prediction: Based on the assumption "*the pixels along the direction of local edge exhibit high correlation, and a good prediction could be achieved using those neighboring pixels that are in the same direction of the edge*", the main edge direction is investigated to split the MB

accordingly. Hence, if the main edge direction is determined to be horizontal or vertical, $P16 \times 8$ or $P8 \times 16$ block type is chosen, respectively. A very small edge sum points out the presence of a homogeneous region, so only the $P16 \times 16$ is processed.

Sub-P8×8 Mode Prediction: In case the SAD of the neighboring MBs is too high, P4×4 mode is predicted. In case the dominating horizontal or vertical edge direction is detected, P8×4 or P4×8 partition is selected, respectively.

Skip Mode Prediction: If SAD of an MB in $P16 \times 16$ mode is significantly low, a perfect match could be very well predicted by ME. Such MBs are highly probable to be *SKIP*, thus saving complete ME computational load. Similarly, if the collocated MB is highly correlated with the current MB, then the probability of *SKIP* is very high, e.g., the complete region is homogeneous. Moreover, if the MB lies in a dark region, the human eye cannot perceive small brightness variations. Thus, the insignificant distortion introduced by a forceful *SKIP* is tolerable here.

4.4.3 Sequential RDO Mode Elimination

An integrated Sequential RDO Mode Elimination mechanism re-evaluates the candidate coding modes for sequential elimination, i.e., after P16×16 is processed, P16×8, P8×16, P8×8, and below are re-evaluated for elimination as specified in Fig. 4.20. However, for Sequential RDO Mode Elimination, the spatial SAD and MV values are replaced by the actual SAD and MV of the previously evaluated mode.

4.4.4 Evaluation of the Complexity Reduction Scheme

Table 4.4 provides the comparison (average and maximum) of ACCoReS with the *exhaustive RDO-MD* for distortion, bit rate (a positive ΔBit Rate shows the bit rate saving) and speedup. Each result for a sequence is the summary of 8 encodings using different QP values. The average PSNR loss is approximately 3%, which is visually imperceptible. However, ACCoReS provides a significant reduction in the computational complexity, i.e., performance improvement of up to 19× (average 10×) compared to the *exhaustive RDO-MD*. The major speedup comes from slow motion sequences (Susie, Hall, Akiyo, Container, etc.) as smaller block partitions and I-MB coding modes are excluded in the Prognostic Early Mode Exclusion stage.

Figure 4.21 presents the percentage mode exclusions with respect to the total possible mode combinations for a large set of diverse sequences (averaged results for QPs ranging from 12 to 40). In the best case, up to 73% (average >50%) coding modes are excluded. Figure 4.21 also shows that the large number of modes are excluded in case of slow motion sequences (Susie, Hall, Akiyo, Container, etc.) due to the early exclusion of smaller block partitions and I-MB coding modes. Figure 4.22 shows the breakdown of different modes used in encoding of various sequences. In case of slow-motion sequences (Akiyo, Susie, and Carphone) more modes are excluded because of the correct identification of homogeneous regions. In this case more P8×8 and I4×4

Table 4.4 Summary of PSNR, bit rate, and speedup comparison for various video sequences (each encoded using eight different QPs)

	Sequence	Average			Maximum		
		ΔPSNR (%)	ΔBit rate (%)	Speedup (x)	ΔPSNR (%)	ΔBit rate (%)	Speedup (x)
CIF	Bus	3.35	6.69	9.07	4.63	12.00	11.56
	Susie	1.87	1.64	11.91	2.47	12.37	14.59
	Football	4.91	2.74	9.65	5.66	3.37	13.05
	Foreman	2.02	4.44	9.97	3.31	16.73	12.70
	Tempete	3.42	10.22	8.47	4.78	14.53	10.75
	Hall	1.82	6.79	12.33	4.34	29.92	14.81
	Rafting	4.29	4.51	9.72	4.84	5.67	12.62
	Mobile	3.38	6.42	8.52	5.05	11.61	10.99
	Am. Football	3.91	7.81	8.76	5.41	10.52	11.61
QCIF	Akiyo	0.61	−3.41	12.75	1.24	1.75	17.27
	Carphone	2.44	6.39	10.20	3.19	11.51	12.86
	Coastguard	2.53	4.58	9.35	4.04	11.32	12.53
	Container	1.06	−7.15	13.00	1.57	4.01	19.13
	Husky	4.83	5.73	7.71	6.18	7.44	10.31
	Miss America	0.73	−8.86	12.05	1.72	14.25	14.72
	News	1.77	−3.64	12.21	2.12	0.37	16.71

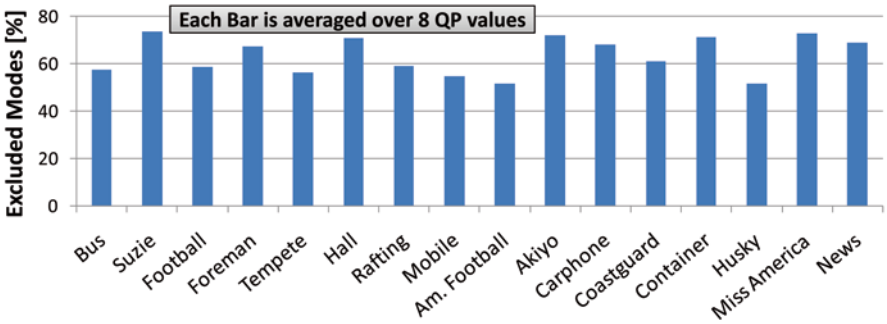


Fig. 4.21 Percentage mode excluded in ACCoReS for various video sequences

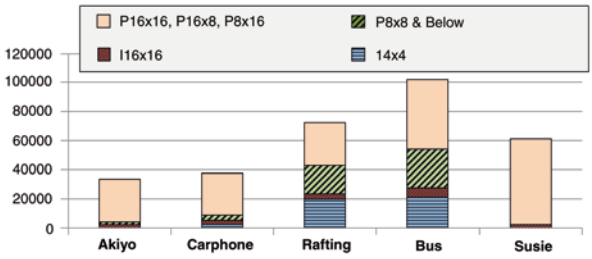


Fig. 4.22 Distribution of mode processing for QP=28

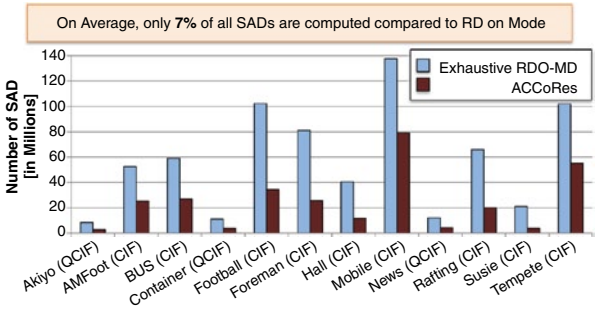


Fig. 4.23 Comparison of total SAD computations for various video sequences

are excluded with an insignificant loss in rate and distortion (see Table 4.4). On the contrary, more I4×4 modes are processed for Rafting and Bus.

Figure 4.23 shows the number of SAD computed using the ACCoReS scheme and the *exhaustive RDO-MD* scheme for various video sequences. ACCoReS computes on average 27% of the SADs computed by the *exhaustive RDO-MD* scheme. The major SAD savings come in case of fast -motion (Football, Foreman, and Rafting) and highly textured sequences (Tempete and Mobile) as bigger block partitions are excluded in the Prognostic Early Mode Exclusion stage.

4.4.4.1 In-Depth Comparison with the Exhaustive RDO-MD

Figure 4.24 shows the in-depth comparison of ACCoReS with the *exhaustive RDO-MD* for Susie sequence. It shows that ACCoReS suffers from an average PSNR loss of 0.8 dB (max: 1.4 dB, min: 0.19 dB), which is visually imperceptible (above 40 dB). However, ACCoReS achieves a significant reduction in the computational complexity, i.e., ACCoReS processes only 17% of SADs (reduced ME load which is the most compute-intensive functional block) compared to the *exhaustive RDO-MD*. The circles in Fig. 4.24 show the region of sudden motion that causes disturbance in the temporal-field statistics. As a result, ACCoReS suffers from a higher PSNR loss but also provides high SAD savings. Moreover, ACCoReS maintains a smooth SAD computation curve, which is critical for embedded systems, while the *exhaustive RDO-MD* suffers from excessive SADs. The PSNR curve shows that after frame 70, the mode prediction quality of ACCoReS improves due to the stability in the temporal-field statistics.

Figure 4.25 shows the frame-wise distribution of correct mode selection by ACCoReS for Susie sequence at QP=28. On average 74% of MBs are encoded with the

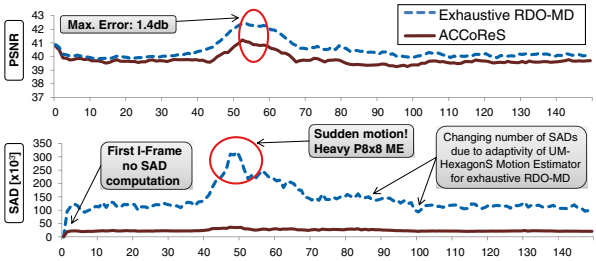


Fig. 4.24 Frame-level in-depth comparison for Susie sequence

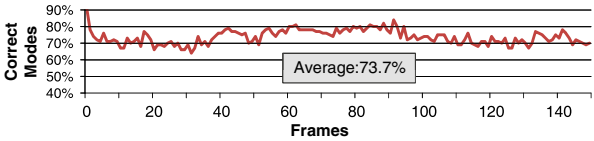


Fig. 4.25 Frame-level in-depth evaluation of correct mode prediction

correct mode (MB Type and the corresponding block size), i.e., as selected by the *exhaustive RDO-MD*. The correct modes predicted by ACCoReS range from 63 to 83%.

Figure 4.26 illustrates the visual comparison of coded modes using ACCoReS (top) and the *exhaustive RDO-MD* (bottom) for the 17th video frame of the American Football sequence (QP=28). The grassy region is almost correctly predicted (i.e., mostly P16×16 and partitions above P8×8 are used) and no I4×4 are false predicted. ACCoReS predicts nearly in all cases P16×16 for the grassy region, because it is quite homogeneous and the motion is low. In overall, the prediction complies with the best mode (as predicted by the *exhaustive RDO-MD*) in most of the cases. Similar observations hold for the players at the left and right border side. The main different point is encircled, where the *exhaustive RDO-MD* used I4×4 while ACCoReS failed to predict. The movement is slightly below the motion threshold, thus is not detected as fast-moving region. Additionally, this area is considered flat

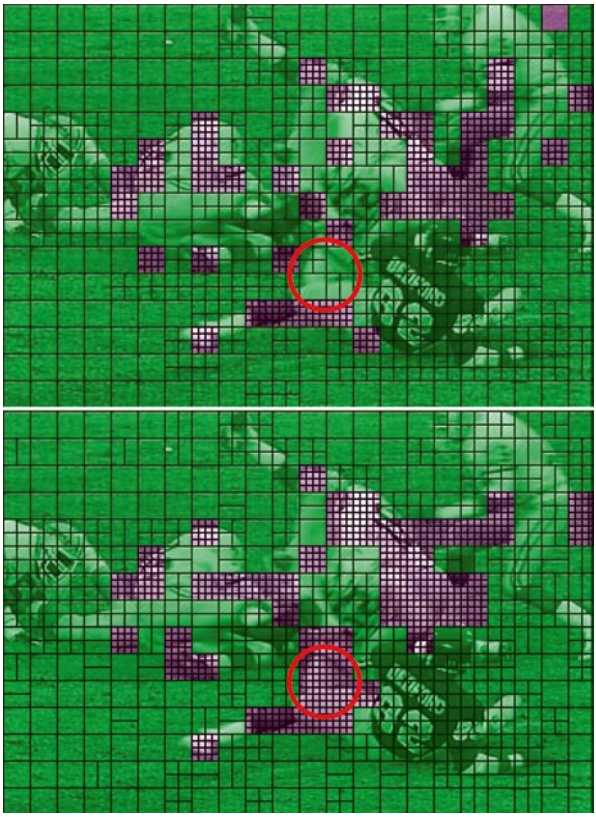


Fig. 4.26 MB-level mode comparison with the exhaustive RDO-MD: frame 17 of American Football. *Top*: ACCoReS [PSNR=33.28 dB], *Bottom*: Exhaustive RDO-MD [PSNR=34.52 dB]

as it is blurred and exhibit less texture details. Consequently, these MBs are encoded with P8x8 and above modes. On average, wrong decisions in this frame generate a PSNR loss of 1.2 dB (33.28 dB vs. 34.52 dB), while both frames required a similar amount of bits (76672 vs. 77760 bits). On overall, the proposed ACCoReS predicts more than 70% of the total modes similar to the *exhaustive RDO-MD*.

4.4.4.2 Overhead of Computing Video Sequence Statistics

The performance gain of ACCoReS comes at the cost of additional computation of spatial and temporal video statistics. Experiments demonstrate that the PC-based software implementation of these statistics computations are 4.6% of the total encoding time using ACCoReS, which is already up to 19× smaller than the encoding time with the *exhaustive RDO-MD*. Compared to the performance savings of ACCoReS, this overhead is negligible. The additional memory requirements are *(#statistics)*#MBs*16bits*, where *#spatial + #temporal statistics = 5 + 2*.

4.4.4.3 Summary of the HVS-based Adaptive Complexity Reduction Scheme

This section presented the adaptive computational complexity reduction scheme that excludes the improbable coding modes even before the actual RDO-MD and Motion Estimation processes. This scheme uses the HVS-based MB categorization. First the imprabable modes are excluded from the candidate list in a relaxed prognostic early mode exclusion step. Afterwards, a more aggressive exclusion is curtailing of the candidate coding mode set is performed in a hierarchical fast mode prediction step. The output of this step is processed using an RDO-MD with consideration of sequential mode exclusion, i.e., depending upon the output of an evaluated mode, further modes are excluded from the candidate set.

4.5 Energy-Aware Motion Estimation with an Integrated Energy-Budgeting Scheme

As discussed in Sect. 3.1, Motion Estimation (ME) is the most compute-intensive and energy demanding functional blocks of an H.264 encoder. Figure 3.3 in Sect. 3.1.2 illustrated that ME may consume up to 65% of the total encoding energy, where the ME energy consumption is directly proportional to the number of computed SADs to determine the best match (i.e., the MB with the minimum distortion). The available energy budgets may change according to various application scenarios on mobile devices. Varying motion types and changing status of available energy budgets stimulate the need for a run-time adaptive energy-aware Motion Estimation scheme while exhibiting minimal loss in video quality (PSNR). The energy-aware Motion Estimation needs to consider the following run-time varying scenarios while keeping a good video quality (PSNR). These scenarios are:

- available energy (due to a changing battery levels or allocated energy in a multi-tasking system)
- video sequence characteristics (motion type, scene cuts, etc.)
- user-defined coding conditions (duration, quality level, etc.)

The challenge that arises here is: how much energy budget should be allocated to the ME of one video frame or even one MB when considering run-time varying scenarios (as argued above). The allocated energy-budget to an MB or video frame will determine the number of computed SADs. For a fast moving MB more ME effort is required while for a stationary MB less effort is required (i.e., reduced number of SADs). A less ME effort for a textured MB with high motion may result in significant PSNR loss. Therefore, in order to efficiently exploit the available energy, carefully allocating the energy budget to different frames and MBs is crucial. It is obviously not trivial to decide under which circumstances the allocated energy budget will be sufficient enough to keep the PSNR loss insignificantly low (compared to Full Search ME) when considering run-time varying scenarios. Hence, a run-time adaptive energy-budgeting scheme for energy-aware Motion Estimation is desirable.

This section introduces a novel run-time energy-aware Motion Estimation scheme for H.264 that adapts at run time according to the available energy level. It consists of different processing stages. The Motion Estimator is integrated with a predictive energy-budgeting (enBudget) scheme that predicts the energy budget for different video frames and different Macroblocks (MBs) in an adaptive manner considering the run-time changing scenarios of available energy, video frame characteristics, and user-defined coding constraints while keeping a good video quality. This is achieved by so-called *Energy-Quality (EQ) Classes* that the enBudget scheme assigns to different video frames and fine-tunes at MB level depending upon the predictive energy quota. Each EQ-Class represents a different ME configuration. Therefore, these EQ-Classes differ in term of their energy requirements and the resulting video quality. The enBudget scheme does not waste energy budget for homogeneous or slow moving parts of a video sequence that do not require high ME effort (i.e., more SAD computations). Instead, the saved energy budget in case of slow motion sequences is allocated to the high motion sequences. This enables the enBudget scheme to dynamically move in the energy-quality design space at run time using the concept of EQ-Classes.

The enBudget scheme requires an adaptive Motion Estimator with multiple processing stages in order to realize EQ-Classes. A novel Motion Estimator is proposed in the scope of this work to facilitate the design of EQ-Classes considering different processing stages.

4.5.1 Adaptive Motion Estimator with Multiple Processing Stages

Now the constituting processing stages of the proposed adaptive Motion Estimator will be explained.

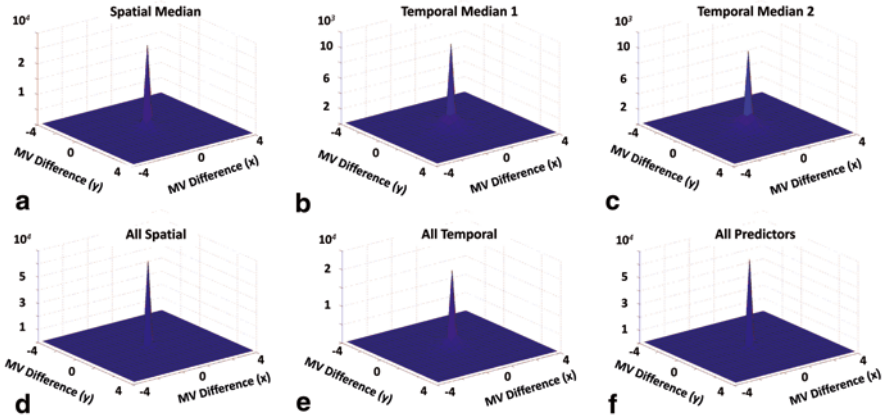


Fig. 4.27 Motion vector difference distribution in Foreman sequence (256 kbps) for various predictors compared to the optimal motion vector (obtained using the full search algorithm)

1. **Initial Search Point Prediction:** An adaptive Motion Estimator starts with an *Initial Search Point Prediction* stage that provides a good guess of the *vicinity* where the best match has a high probability to be found. A good predictor provides a good starting point to converge quickly to the best match (i.e., near-optimal Motion Vector, MV). Based on the assumption, *the blocks of an object move together in the direction of the object motion*, spatial and temporal neighbors are considered as good predictor candidates. Therefore, MV of the current MB is highly correlated with MVs of the spatially and temporally adjacent MBs that have been previously calculated. A set of predictors is selected by analyzing the MV difference distribution between various predictors (see Eq. 4.12 and the predictor set below) and the optimal MV (i.e., obtained by using the *Full Search* algorithm). Figure 4.27a–c shows that spatial median predictor ($Median_{Spatial}$, Eq. 4.12) has a higher correlation with the optimal MV compared to the temporal median predictors ($Median_{Temporal1}$, $Median_{Temporal2}$, Eq. 4.12). This implies that $Median_{Spatial}$ needs to be examined first as it has high probability to be the *True Predictor*[2] (i.e., to find a near-optimal MV). It is noticed that $Median_{Temporal1}$ and $Median_{Temporal2}$ are also highly probable to be the *True Predictors* especially when the MB is moving vertically or horizontally with a constant velocity.
   Figure 4.27d, e illustrates that the spatial predictors exhibit a higher correlation with the optimal MV compared to the temporal predictors. On overall, when considering all of the predictors the probability of finding a near-optimal MV is very high and refinement search stage will provide the best MV (MV_{Best}, close to or similar to that of the *Full Search* algorithm). The final selected predictor set is:

$$Predictors_{Spatial} = \{MV_{Zero}, MV_{Left}, MV_{Top}, MV_{Top-Left}, MV_{Top-Right}, Median_{Spatial}\}$$

[2] *True Predictor* represents the displacement close to the optimal MV obtained by the *Full Search* algorithm.

$$Predictors_{Temporal} = \{MV_{Collocated}, Median_{Temporal1}, Median_{Temporal2}\}$$

$MV_{Collocated}$ is the MV of the collocated MB in the previous frame (F_{t-1}).

$$\begin{aligned} Median_{Spatial} &= median(MV_{Left}, MV_{Top}, MV_{Top-Right})_{F_t} \\ Median_{Temporal1} &= median(MV_{Left}, MV_{Top}, MV_{Top-Right})_{F_{t-1}} \\ Median_{Temporal2} &= median(MV_{Right}, MV_{Down}, MV_{Down-Right})_{F_{t-1}} \end{aligned} \tag{4.12}$$

After analyzing the predictor correlation in Fig. 4.27, a set of conditions is formulated for the early termination of the ME process. Figure 4.28 shows the conditions for the predictor set for early termination to save energy depending upon the characteristics of motion field. Motion field changes with the properties of input video sequence thus results in adaptation at run time. Selected predictors are processed for SAD. The predictor with minimum SAD is compared against $Threshold_{pred}$ (see Eq. 4.13) for early termination. If early termination is not detected then this predictor serves as the search center for the next ME stage.

2. **Search Patterns:** The *Initial Search Point Prediction* stage is followed by *Traversing the Search Pattern* stage, which takes the best predictor as the search center and evaluates different candidate points on the search pattern. The proposed adaptive Motion Estimator incorporates the following four different search patterns:

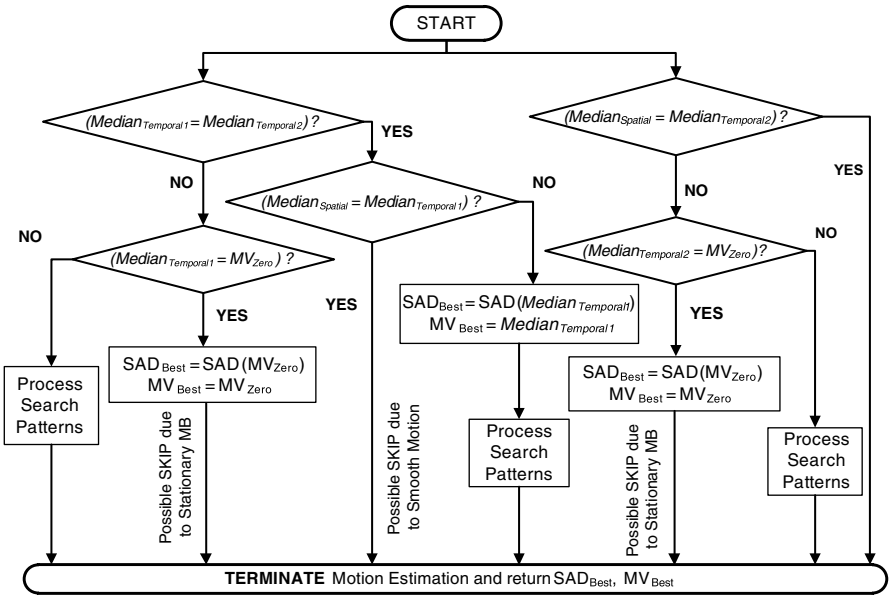


Fig. 4.28 Predictor conditions for motion-dependent early termination

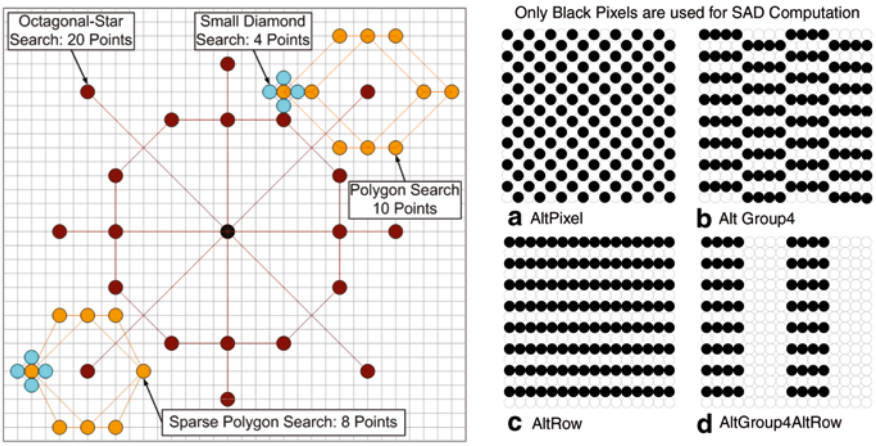


Fig. 4.29 Four search patterns used in the adaptive motion estimator and the pixel-decimation patterns for SAD computation

a. ***Octagonal-Star Search Pattern:*** This pattern consists of 20 search points and handles large irregular motion cases. Figure 4.29 shows an Octagonal-Star search pattern executing at a distance of 8 pixels from the search center. For small-medium motion and CIF/QCIF video sequences only one Octagonal-Star search pattern is processed. For heavier motions/high resolutions (D-1, HD), multiple Octagonal-Star search patterns may be executed (each extended by a pixel distance of 8). The MV with minimum SAD in this step will be chosen as the search center of the next search pattern.
b. ***Polygon and Sparse Polygon Search Patterns:*** A Polygon search pattern consists of 10 search points and narrows the search space after processing of Octagonal-Star search pattern. Figure 4.29 shows both Polygon and Sparse Polygon search patterns. This search pattern favors horizontal motion over vertical motion, because in typical video scenes horizontal motion is dominant as compared to the vertical motion. The MV with minimum SAD after this processing stage serves as the center for the next search pattern.
c. ***Small Diamond Search Pattern:*** At the end, a 4-point Diamond search pattern is applied to refine the motion search.

If the calculated SAD of a candidate point in a search pattern is less than the current SAD_{Best}, then SAD_{Best} and MV_{Best} are replaced by the calculated SAD and the candidate point. After processing all candidates points, the SAD_{Best} is checked against a threshold to check the termination criterion. If not terminated, MV_{Best} is set as the center for the next search step.

3. **Stopping Criteria:** Early termination is integrated in patterns to stop the search in case the current SAD_{Best} is smaller than a threshold. Early termination results in energy saving but care should be taken in consideration to avoid false termination. Two different thresholds are used for early termination: $Threshold_{pred}$ in the *Initial Search Point Prediction* stage and $Threshold_{pattern}$ in the *Search Patterns* stage (Eq. 4.13).

$$\begin{aligned} Threshold_{pred} &= SAD_{stationaryMB} * (1+\delta) * \alpha_{power} \\ Threshold_{pattern} &= SAD_{stationaryMB} * (1+\gamma) * \alpha_{power} \end{aligned} \tag{4.13}$$

$SAD_{stationaryMB}$ thereby is the SAD for a static MB (i.e., with zero motion). α_{power} (Eq. 4.13) is the power-scaling factor that provides a tradeoff between reconstructed video quality and energy consumption. The value of α_{power} is determined dynamically (see Eq. 4.14). E_{C1} and E_{C1} are the normalized energy values of two consecutive EQ-Classes values.

$$\alpha_{power} = \begin{cases} 1.0; \text{For the highest Energy EQ-Class} \\ \quad \alpha_{power} + (E_{C1} - E_{C2}); \text{Else} \end{cases} \tag{4.14}$$

δ and γ are modulation factors to provide a tradeoff between reconstructed video quality and search speed. Initial values of δ and γ are determined by Quantization Parameter (QP; see Eq. 4.15). The larger the value of QP is, the larger are the values of δ and γ. This is due to the following reason: when the quantization step is larger, the quantization noise is also larger and most of the details of the reconstructed image are lost and in this case, the difference between the best matching block and the sub-optimal matching block becomes blurry. c_1 and c_2 are user-defined weights to control the effect of change in QP value. If ME fails to achieve the time line, i.e., targeted frame rate ($FrameRate_{Target}$) encoding, then the values of δ and γ are increased at the cost of loss in PSNR (Eq. 4.15).

$$\begin{aligned} &\delta = c_1 * (QP > 20?QP - 20 : 0) \\ &\delta += (FrameRate_{Target} - FrameRate_{Achieved})/(2 * FrameRate_{Target}) \\ &\gamma = c_2 * (QP > 20?QP - 20 : 0) \\ &\gamma += (FrameRate_{Target} - FrameRate_{Achieved})/FrameRate_{Target} \end{aligned} \tag{4.15}$$

A technique to reduce the energy of ME is block matching with pixel decimation. Therefore, for reducing the energy of one SAD computation, several pixel decimation patterns are proposed that are discussed in the following.

Matching Criterion (SAD) Decimation Patterns: For block matching, the matching criterion (SAD) is evaluated using every pixel of the MB. For one SAD computation, 256 subtractions, 256 absolute operations, 255 additions are required

Table 4.5 Comparing the video quality of different SAD decimation patterns for encoding of Susie CIF video sequence (30fps@256 kbps)

Motion estimator	Video quality of different SAD decimation patterns (PSNR [dB])				
	Original	Alt-Pixel	Alt-Group4	Alt-Row	AltGroup4-AltRow
Full Search	40.31	40.26	40.26	40.25	40.09
UMHexagonS	40.24	40.16	40.14	40.16	39.90
UMHexagonS Simple	40.18	40.06	40.04	40.05	39.80
EPZS	40.29	40.24	40.25	40.24	40.09

along with loading of 256 current and 256 reference MB pixels from memory. In order to save energy for one SAD computation (reducing memory transfers and computation) some pixels from SAD computations may be excluded when the available energy is low. Since the block-based ME is based on the assumption that *all the pixels in an MB move by the same amount*, therefore, a good estimation of motion could be obtained by using only a fraction of the pixels in an MB. An aggressive decimation will result in an inaccurate ME if the videos contain small objects or high texture information. Therefore, the main issue is to find such a scheme for matching pixel decimation that will not cause much degradation in visual quality. Figure 4.29 shows four decimation patterns considered for evaluation.

AltPixel, *AltGroup4*, *AltRow* patterns (Fig. 4.29) reduce the number of pixels for SAD computation by 2, while *AltGroup4AltRow* reduces by 4 that directly corresponds to an energy reduction (due to reduced memory transfers and computations) and still provides an insignificant PSNR loss. An analysis to explore the quality impact (PSNR in dB) of these patterns on four benchmark Motion Estimators (see details in Sect. 2.2.3) is shown in Table 4.5. *AltGroup4AltRow* gives a PSNR loss of 0.2 dB and 0.34 dB for *EPZS* [Tou02] and *UMHexagonS* [CZH02], respectively.

Although *AltPixel* and *AltGroup4* reduce the energy, these are not very beneficial for cache-based architectures because the data is already in the cache. On the other hand, *AltRow* and *AltGroup4AltRow* are beneficial for cache-based architectures as they skip row by row. Skipping the complete row is not advantageous for heavy-textured videos with small objects. Therefore, only *AltGroup4AltRow* (obtain a significant energy reduction even for cache-based architectures) and *AltGroup4* (counter the issue of heavy-textured videos) are used for designing EQ-Classes as they provide good tradeoff between energy saving and PSNR loss. These patterns scale down accordingly for different block modes in H.264.

Now the enBudget scheme will be presented. It uses the above-mentioned adaptive Motion Estimator for designing the EQ-Classes, where each EQ-Class represent a certain ME configuration in terms of different ME stages, i.e., a certain combination of settings of *Initial Search Point Predictors*, *Search Patterns*, and *SAD Decimation Patterns*.

4.5.2 enBudget: The Adaptive Predictive Energy-budgeting Scheme

Figure 4.30 shows the overview of the enBudget scheme. The proposed scheme has three major phases:

- Group of Pictures (GOP)-level allocated energy quota computation
- Frame-level energy budget prediction and *Base EQ-Class* selection.
- *MB-level EQ-Class* refinements and upgrading/downgrading of *Base EQ-Class* to determine the final EQ-Class for each MB.

The input to the enBudget scheme is available energy (battery status), user constraints (e.g., quality level, desired duration of encoding, etc.), compile-time analysis of ME (i.e., average case energy distribution in a video encoder, see Fig. 3.3 in Sect. 3.1.2), encoder configuration (e.g., encoding frame rate and target bit rate), and video frame properties (Brightness, Texture, SAD, and MV). A set of compile-time designed EQ-Classes with average and minimum energy requirements is provided to the enBudget scheme. The average energy is estimated through extensive experiments using a wide-range of video sequences with diverse properties. However, at run time the average energy is updated considering the actual energy consumption of the EQ-Class (that depends upon the currently coded video sequence) using a weighted error mechanism. The step-by-step flow of the enBudget scheme is as follows:

GOP-Level: It may happen that the early GOPs in the video sequence may consume a major portion of the total available energy and the later GOPs are left with too less energy budget. This may harm the overall PSNR of the video sequence. Therefore, to avoid such scenarios, each GOP is allocated a separate energy quota.

Frame-Level: Available energy status, user-defined constraints, compile-time analysis of ME, and encoder configuration are used for computing the allocated energy quota (E_{Quota}) which is same for all frames in a GOP. It may happen that the E_{Quota} is more than the actual energy requirements of one frame ME. Examples

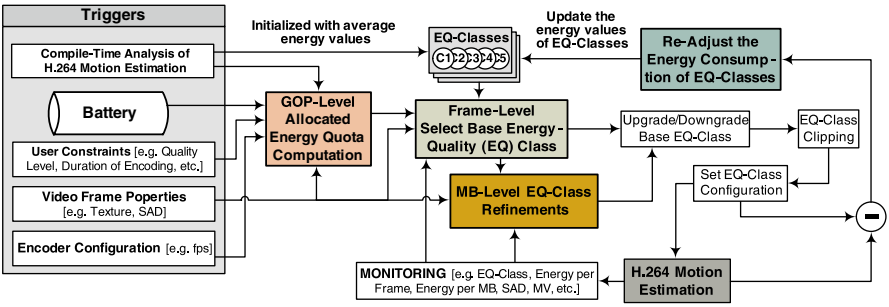


Fig. 4.30 Flow of the enBudget scheme for energy-aware motion estimation

of such scenarios are (a) the battery level is full and user wants a short duration encoding, (b) the frame is homogenous and stationary or it exhibits low-to-medium motion, (c) the motion properties are amenable to the search pattern of the ME. In this case, the over-estimated E_{Quota} is adjusted for computing the predictive energy budget (E_{pred}) of a frame, such that the energy wastage due to the unnecessary SAD computations (as possible in above-mentioned a-c cases) is avoided. Depending upon the E_{pred} value a frame-level *Base EQ-Class* is determined. After each frame is encoded, E_{pred} of the next video frame is readjusted in a feedback loop considering that consecutive video frames exhibit high correlation (except scene cuts).

Macroblock-Level: Since different MBs of a frame may exhibit diverse texture and motion properties, an energy distribution approach is incorporated that gives more energy to the complex MBs (i.e., high texture, high motion) and less energy to homogenous or slow-moving MBs. In order to provide a consistent control at frame-level, the *Base EQ-Class* is kept same for all MBs in the frame and refinements are computed for each MB. A refinement may be defined as the upgrade or downgrade step to the *Base EQ-Class* that determines a higher energy class or lower energy class with respect to the *Base EQ-Class*. To avoid the violation of the E_{pred}, a clipping mechanism is integrated. For an MB, a final *MB-level EQ-Class* is then determined and the ME configuration (e.g., Search Pattern, SAD Decimation Pattern) for the corresponding EQ-Class is set. Afterwards, the ME is performed for an MB and the actual energy consumption ($E_{consumed}$), SAD, and MV are monitored.

After encoding all of the MBs in a video frame, the difference between E_{Quota} and $E_{consumed}$ is computed and the E_{pred} for the next frame is updated in a feedback loop using this error. Moreover, depending upon the error between the energy of *Base EQ-Class* ($E_{BaseClass}$) before ME and $E_{consumed}$, the energy of all EQ-Classes is re-adjusted (see Fig. 4.30).

Before moving to the run-time algorithm of the enBudget scheme, the design of EQ-Classes is discussed that serve as the foundation to the enBudget scheme and enables it to move in the energy-quality design space at run-time.

4.5.2.1 Designing Energy-Quality (EQ) Classes

The enBudget scheme supports the run-time tradeoff between the allocated energy to the ME and the resulting visual quality (PSNR) for a given bit rate using the design-time prepared *Energy-Quality (EQ) Classes*. More SAD computations will provide better match (i.e., better PSNR) but at the cost of higher energy consumption. These EQ-Classes are designed using various combinations of the Initial Search Point Prediction, Search Patterns, and SAD Decimation Pattern of the adaptive ME (as discussed in Sect. 4.5.1). Each EQ-Class provides an energy saving (as it differs in its ME configuration) and suffers from a certain PSNR loss. Ideally for computing the energy saving and PSNR loss, EQ-Classes should be benchmarked against the *Full Search* ME as it provides the optimal match. However, as discussed in Sect. 2.2.3, the *Full Search* ME demands huge amount of energy and it is impracticable in real-world applications. Therefore, *UMHexagonS* (a fast adaptive ME

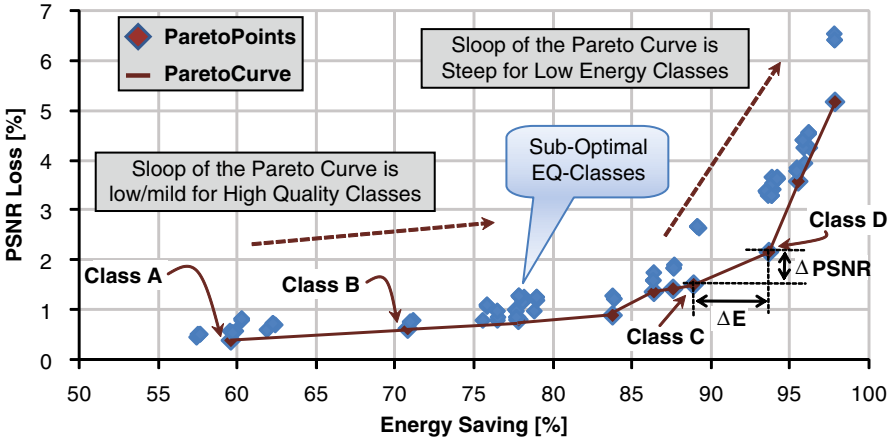


Fig. 4.31 Energy-Quality (EQ) classes: energy-quality design space exploration showing various pareto points and the pareto curve

provides almost similar PSNR compared to the *Full Search* ME while providing huge computation reduction [CZH02]) is used as a benchmark for computing the energy savings and PSNR loss of different EQ-Classes.

Figure 4.31 shows the EQ-Class energy-quality design space for Foreman video sequence (CIF@30fps, 256 kbps). Since the ME configurations form a discrete set of EQ-Classes, the problem of optimal ME configuration selection (i.e., EQ-Class selection) can be solved by *Pareto analysis* [Das99]. In the experiments of Fig. 4.31, 8 Search Pattern combinations, 4 sets of Initial Search Point Prediction, and 3 SAD Decimation Patterns (i.e., altogether 96 EQ-Classes) are used. The optimum EQ-Classes are the points in the energy-quality design space that form the Pareto Curve (as shown by the line in Fig. 4.31). All EQ-Classes that lie above the Pareto Curve are sub-optimal. It is worthy to note that this Pareto Curve provides optimal EQ-Classes for a certain video sequence under certain coding settings. Due to the diverse and unpredictable nature of video sequences and unpredictable demands of end-users (i.e., varying bit rates), it is impossible to determine a set of EQ-Classes at design time which provides the optimal ME configuration for all possible combinations of diverse video properties and coding configurations. Therefore, an extensive energy-quality design space exploration is performed for various video sequences. From this analysis, a set of EQ-Classes is carefully selected considering the similarities in the Pareto Curves.

Figure 4.31 shows cases where some EQ-Classes are close to each other on the Pareto Curve, i.e., they exhibit only minimal difference in their energy reduction and the corresponding PSNR loss. As each EQ-Class brings a certain PSNR variation, more EQ-Classes will cause frequent changes in the visual quality, thus visually uncomfortable for the user. Moreover, oscillation will result in a random motion field, which will disturb the behavior of motion-dependent terminations in the adaptive Motion Estimator. Therefore, a subset of EQ-Classes is selected (as shown in Table 4.6 with their corresponding configuration), such that at run time

Table 4.6 Configuration and energy consumption for the chosen Energy-Quality (EQ) classes

Classes	Pattern set	SAD decimation pattern	Avg. energy (μWs)[a]	Min energy (μWs)[a]
C_1	Oct + SPoly + Diamond	Full SAD	85.72	5.25
C_2	Poly + Diamond	Full SAD	59.47	4.97
C_3	Diamond	Full SAD	26.93	2.91
C_4	Diamond	AltGroup4	14.52	1.43
C_5	SPoly + Diamond	AltGroup4AltRow	10.66	0.67
C_6	Diamond	AltGroup4AltRow	5.81	0.59

[a] Averaged over various test video sequences for 90 nm ASIC.

the switching between two EQ-Classes provides a significant energy reduction. Moreover, less number of EQ-Classes will also reduce the execution time of the enBudget scheme.

All EQ-Classes use the complete set of predictors because it is the most crucial ME stage. To demonstrate this fact, the search efficiency of each ME stage (Sect. 4.5.1) is investigated for several exemplary MBs of Foreman video sequence (see Fig. 4.32). The search efficiency of each ME stage is defined by the decrease in SAD that it brings at the cost of certain energy consumption. The search efficiency of these ME stages may change for different MBs in a video frame. Figure 4.32 shows that among all ME stages, the search efficiency of Initial Search Point Prediction is the highest. Therefore, all EQ-Classes use the complete set of Initial Search Point Predictors. The gradient of Pareto Curve is defined as ΔPSNR/ΔE. Figure 4.31 shows that the gradient for high energy EQ-Classes is low, while the gradient for low energy EQ-Classes is quite high, i.e., $\Delta PSNR_{AB}/\Delta E_{AB} << \Delta PSNR_{CD}/\Delta E_{CD}$. Therefore, care needs to be taken when downgrading a high-energy EQ-Class to a low-energy EQ-Class.

Each EQ-Class is initialized with an average-case energy consumption value. However, due to the varying video sequence properties and adaptive early termina-

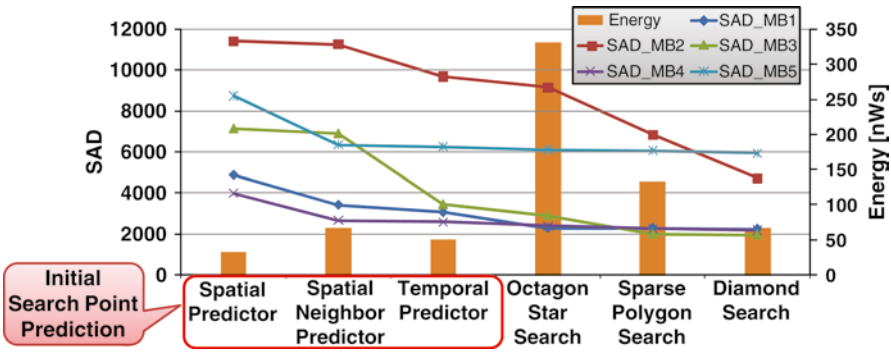


Fig. 4.32 SAD vs. energy consumption comparison of different motion estimation stages for Foreman sequence

tion (Sect. 4.5.1), each EQ-Class may provide different energy saving for different video sequences. Therefore, the energy of an EQ-Class is updated at run-time depending upon the actual energy consumption for the given video sequence properties (as discussed later in this section).

4.5.2.2 Run-time Algorithm of the enBudget Scheme

Algorithm 4.4 shows the pseudo-code of the enBudget scheme. Available energy status (i.e., current battery level), user-defined constraints, encoder configuration, compile-time ME analysis, video frame properties, and a set of EQ-Classes (Table 4.6) are passed as input to the enBudget scheme (Fig. 4.30). The flow of algorithm is systematically discussed as follows:

Step-1: GOP-Level (Lines 4–15): First, Quality Level (Q_L) as specified by user-defined constraints is readjusted depending upon the current battery level (B_L) to ensure successful encoding in the given B_L (line 6). If *useQualityLevel* is set then, in lines 9–13 the GOP-level energy quota (E_{Quota}) is computed depending upon the Q_L, otherwise the E_{Quota} is computed using the B_L and the encoding duration required by the user (line 14). This E_{Quota} is then used for predicting the energy budget for all frames in the GOP.

Step-2: Frame-Level (Lines 16–27): The energy budget (E_{pred}) for one frame ME is predicted using the E_{Quota} and video frame properties. The energy error from the previous frame is back propagated (using a weighting factor ξ_1, which controls the strength of back propagation) for the E_{pred} calculation of the next frame (line 17). Since different video frames may have different spatial and temporal properties, E_{pred} calculation needs to consider this fact. For example, a scene cut may require more energy (as it will be shown in Sect. 4.5.3) due to a sudden disturbance in the temporal properties of a video sequence. Therefore, in order to cope with the unpredictable nature of video data, the E_{pred} is scaled using the amount of texture difference ($TDiff_{AVG}$: computed using the Sobel Operator) between two consecutive video frames (lines 18–20). The scaled E_{pred} is used to select the frame-level *Base EQ-Class*. The frame-level *Base EQ-Class* is readjusted depending upon the brightness of the current video frame and the average motion of the previous video frame (lines 23–25). As discussed above using Fig. 4.31, the gradient for high energy EQ-Classes is much less than that of the low energy EQ-Classes, thus care needs to be taken when downgrading a high-energy EQ-Class to a low-energy EQ-Class. Therefore, the C_{Delta} is clipped between ±2 in line 26. The frame-level *Base EQ-Class* and the corresponding energy are then passed to the *MB-level EQ-Class* selection stage.

Step-3: MB-Level (Lines 28–42): Since, different MBs of a video frame may have changing texture and motion properties, therefore—at MB-level—the goal of the enBudget scheme is to refine the frame-level *Base EQ-Class* for each MB of the frame. It computes the EQ-Class refinement depending upon the MB properties and upgrades or downgrades the *Base EQ-Class* accordingly. Dark homogeneous

```
1.  Function enBudget ( )     // For Each Video Frame
2.  // Input: Image and Motion Statistics: Brightness (B), Texture (T, Tdiff_Avg),
    N_darkMBs = Σ_{i=1}^{#MBs} (B_iAvg < Th_B), MV, SAD, Battery Level (B_L), User Constraints (Duration: D_E,
    Quality Level: Q_L), Encoder Configuration (e.g., fps, Target Bit Rate: TBR), Energy-wise
    sorted list of EQ-Classes C=(C_1, ..., C_n) (Table 4.6), where C_1 is the max energy class and C_n
    is the min energy class, ME_ratio: ratio of ME energy to the encoding energy
3.  BEGIN
4.  // Step-1: GOP-Level Allocated Energy Quota: Compute once per GOP, this quota is
    same for all frames in the GOP
5.  If (first_Frame_of_GOP) {
6.  If (B_L < β*B_Total) Q_L←Low;    // readjust the quality level depending upon the current
      battery level
7.  C_Base←Ø;        C_MB←Ø;      // initializes the Frame- and MB-Level EQ-Classes
8.    E_Quota=0;     E_consumed=0;  Error_Class=0;   Error_Quota=0;   C_Delta=0;
9.    If (useQualityLevel) {
10.      If (Q_L == High)          E_Quota=getEnergy(C_1);
11.      Else If (Q_L == Medium)   E_Quota=(getEnergy(C_1)+getEnergy(C_n)+1)/2;
12.      Else If (Q_L == Low)      E_Quota=getEnergy(C_n);
13.   }
14.   Else     E_Quota = min(max(ME_ratio * B_L/D_E, getEnergy(C_n)), getEnergy(C_1));
15. }
16. // Step-2: Frame-Level: Determine Base EQ-Class
17. E_pred = E_Quota + ξ_1 * Error_Quota
18. If (Tdiff_AVG > τ_1) ε = max(min(Tdiff_AVG / τ_1, τ_2), τ_3);
19. Else              ε = 1;
20. E_pred = min(max(ε * E_pred, getEnergy(C_n)), getEnergy(C_1));
21. C_Base = getClass(E_pred);    // get the closest Frame-Level EQ-Class
22. // Image-/Motion-Based EQ-Class Adjustments
23. If ((N_darkMBs > N_dark1) & (SAD_Avg < Th_SAD1))     C_Delta++;
24. Else If (SAD_Avg > Th_SAD1)  C_Delta– –; // upgrade for high motion
25. Else C_Delta += ((N_darkMBs > N_dark1) + (N_darkMBs > 2*N_dark1) + (SAD_Avg < Th_SAD2));
26. C_Delta = min(max(C_Delta, –2), 2) + (T < τ_4);
27. C_Base = min(max(C_Base+C_Delta, C_n), C_1);          E_Base = getEnergy(C_Base);
28. // Step-3: MB-Level EQ-Class Refinements
29. For all Macroblocks {
30.   C_Delta += ((N_darkMBs > N_dark1) & (SAD_Avg < Th_SAD1) & (B_MB < Th_B)); // Dark MB
31.   C_Delta += ((N_darkMBs > N_dark1 || SAD_Avg < Th_SAD2) & (B_MB < Th_B));
32.   If (B_MB ≥ Th_B) {
33.     C_Delta += ((S_MB < δ_1*Th_S || SAD_MB_Collocated < δ_1*Th_SAD3) + ((S_MB < δ_2*Th_S || SAD_MB_Collocated
        < δ_2*Th_SAD3));
34.     C_Delta – = ((S_MB > (δ_1+δ_2)*Th_S || SAD_MB_Collocated > (δ_1+δ_2)*Th_SAD3);
35.   }
36.   C_Delta = min(max(C_Delta + (SAD_Avg < Th_SAD2) + (SAD_Avg < Th_SAD4), –2), 2);
37.   If (MB == 0 & SAD_MB_Collocated < Th_SAD5) // Stationary MB
38.     C_Delta C_Delta += ((SAD_Avg < δ_3*TBR) – 2*(SAD_MB_Collocated > δ_4*TBR);
39.   C_MB = min(max(C_Base + min(max(C_Delta, –2), 2)), C_n), C_1);
40.   // Perform Energy-Aware Motion Estimation
41.   E_consumed = Motion Estimation (C_MB)          // (see Class Configuration in Table 4.6)
42. }
43. // Step-4: MB-Level EQ-Class Refinements
44. Error_Class=E_consumed – E_Base;                  Error_Quota=E_Quota=0 – E_consumed;
45. For all EQ-Classes{
46.   Energy[C_i] = max(getEnergy(C_i) + ξ_2*Error_Class, getMinEnergy(C_i));
47. }
48. END
```

Algorithm 4.4 Pseudo code of the Run-Time Adaptive Predictive Energy-Budgeting Scheme

MBs with slow-medium motion (lines 30–33) or stationary MBs (lines 37–38) require less ME effort, therefore the refinement is computed for downgrading. This downgrade results in significant energy savings without PSNR loss for low-textured MBs with slow motion. Alternatively, for MBs with high texture or high motion, the refinement is computed for upgrading (line 34, 38). Clipping in lines 36 and 39 is performed to avoid excessive downgrading or upgrading that may result in severe PSNR loss or excessive energy consumption, respectively. *MB-level EQ-Class* is computed (line 39) and the corresponding configuration is forwarded to the ME (as specified in Table 4.6).

Step-4: Error Computation and Readjustments (Lines 43–47): After the ME is completed for all MBs of a frame, $E_{consumed}$ is used to compute the error between $E_{consumed}$ & *Base EQ-Class* energy (E_{Base}) and $E_{consumed}$ & E_{Quota} (line 44). $Error_{Class}$ is used to readjust the average-case energy of all EQ-Classes in a weighted manner (lines 45–47) to adapt considering the properties of currently coded video frames. E_{Quota} is back propagated to update the E_{pred} of the next video frame (line 17). ξ_1 and ξ_2 are two weighting factors that control the strength of error back propagation.

4.5.3 Evaluation of Energy-Aware Motion Estimation with an Integrated Energy-Budgeting Scheme

Now the enBudget scheme will be integrated in two different Motion Estimators: (a) The adaptive Motion Estimator as proposed in Sect. 4.5.1, (b) *UMHexagonS* [CZH02]. The energy and video quality (PSNR) comparison will be performed for the adaptive Motion Estimator with and without the enBudget scheme. For energy estimation, the proposed power-model (see details in Sect. 3.4) is used. This section also provides the frame-level and MB-level analysis of energy consumption for the energy-aware Motion Estimation (i.e., the adaptive Motion Estimator of Sect. 4.5.1 and the enBudget scheme of Sect. 4.5.2). At the end, the enBudget energy consumption will be compared for various fabrication technologies. The experimental setup is: search range = 16, bit rate = 256 kbps, frame rate = 30 fps, Group of Pictures = IPPP. Table 4.7 shows the coefficients and thresholds used in the algorithm of the enBudget scheme (Algorithm 4.4). These coefficients and thresholds use the similar methodology as discussed in Sect. 4.3.1 and 4.3.2. Note, all results include the leakage and dynamic energy consumption considering the fact that ME hardware is power-gated after the completion of one frame ME.

Table 4.7 Coefficients and thresholds used by the algorithm of enBudget in Algorithm 4.4

Attribute	Value	Attribute	Value	Attribute	Value	Attribute	Value
τ_1	3*#MBs	δ_1	0.9	Th_{SAD1}	900	δ_1	0.5
τ_2	2	δ_2	0.7	Th_{SAD2}	500	δ_2	0.5
τ_3	1.1	δ_3	1300	Th_{SAD3}	2500	δ_3	0.15
τ_4	3000	δ_4	2000	Th_{SAD4}	400	δ_4	#MBs/2
Th_S	13000	Th_B	85	Th_{SAD5}	256	Th_B	

4.5.4 Comparing Adaptive Motion Estimator with and Without the enBudget Scheme

Figure 4.33 illustrates that compared to the original adaptive Motion Estimator (as proposed in Sect. 4.5.1), the adaptive Motion Estimator with enBudget achieves an energy saving of up to 72% (avg. 60%) with an insignificant PSNR loss of 0.08 dB. This shows that the benefit of incorporating the enBudget scheme in an adaptive Motion Estimator to transform it into an energy-aware Motion Estimation scheme. In some cases (*Clair*, *Mobile*, *Hall*), the video quality is even slightly better compared to the original adaptive Motion Estimator. It is due to the fact that energy from smooth MBs is saved and more energy is provided to the textured MBs which results in a quality improvement in certain regions. This contributes to the overall video quality.

4.5.5 Comparing UMHexagonS with and Without the enBudget Scheme

In order to validate the benefit and applicability of the enBudget scheme to other fast adaptive ME schemes that have multiple ME stages, the enBudget scheme is additionally integrated with *UMHexagonS* [CZH02]. Figure 4.34 illustrates that compared to

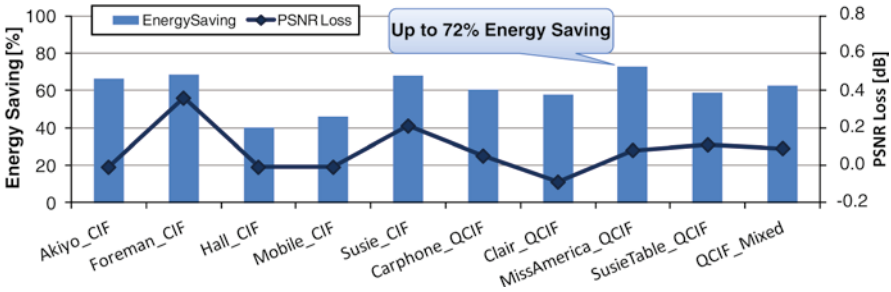


Fig. 4.33 Energy and quality comparison for the adaptive motion estimator with and without the enBudget for various video sequences

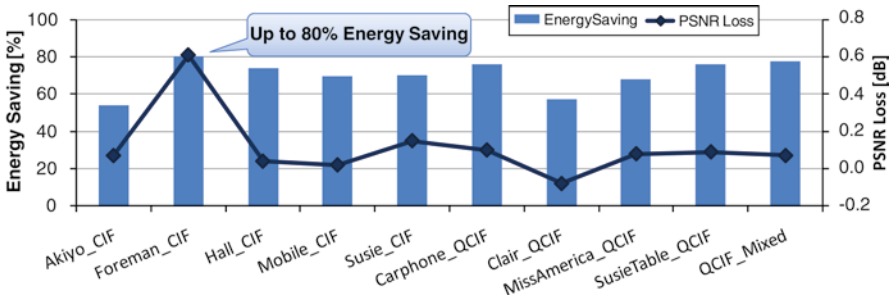


Fig. 4.34 Energy and quality comparison for the UMHexagonS [CZH02] with and without the enBudget for various video sequences

the original *UMHexagonS*, *UMHexagonS with enBudget* achieves an energy saving of up to 80% (avg. 70%) with a slight PSNR loss of 0.11 dB. This shows that the enBudget scheme is equally beneficial for other state-of-the-art fast adaptive MEs as well.

4.5.5.1 Frame-level and MB-Level Analysis

Figure 4.35 shows the frame-wise energy consumption (for a 90 nm technology) for three QCIF video sequences when using the proposed energy-aware Motion Estimation with the enBudget scheme. Label-A points to the fact that, for a slow motion sequence (*Clair*), the energy consumption line is smooth as the consecutive frames have high correlation, homogeneous background, and low motion. For *Clair* sequence the energy-aware Motion Estimation converges to the EQ-Classes C_5 and C_6 (see Table 4.6) depending upon the type of MBs (i.e., moving or stationary). For such sequences, the energy-aware Motion Estimation provides significant energy savings (see Fig. 4.33).

Label-B points to a more interesting scenario. For validating the robustness, the proposed energy-aware Motion Estimation scheme is tested for some mixed video sequences (e.g., *SusieTable*, alternate 50 frames of *Susie* and *Table* sequences are merged). Scene cuts and sudden changes in video frame properties can be realized in such sequences. Label-B in Fig. 4.35 points to the sudden energy consumption peaks, which are mainly due to the scene cuts or disturbance in the temporal properties of video frames. In such cases, the energy-aware Motion Estimation scheme selects EQ-Classes C_1 and C_2 for MBs in the scene cuts or MBs with high motion. The detailed MB-level energy map for the scene cut corresponding to Label-B is shown Fig. 4.36. Due to the scene cut there is a texture difference in two consecutive video frames and the motion field is disturbed, as the objects of next frames no longer exist in the previous frame. As in this case, the video frame at scene cut is not encoded as Intra picture, ME requires high effort to find matches. This fact is visible from the 0.3–0.5 µWs regions in Frame#100, where MBs required more energy for ME. Similar effect is visible in many frames (varying peaks in Fig. 4.35).

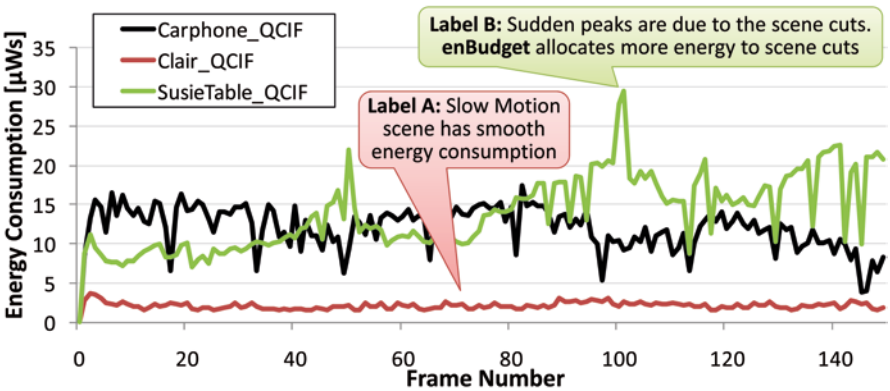


Fig. 4.35 Frame-wise energy consumption of the energy-aware motion estimation

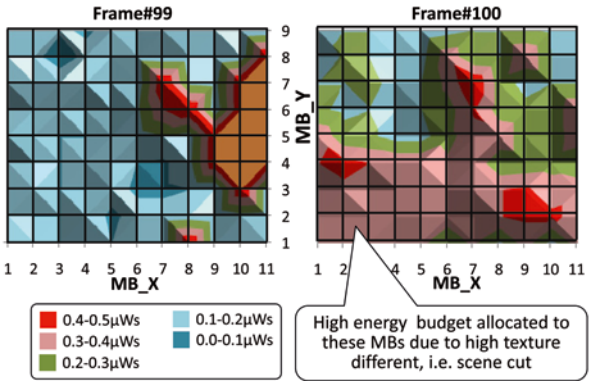


Fig. 4.36 Macroblock-wise energy consumption map of two exemplary frames in the SusieTableMix_QCIF sequence for a 90 nm technology

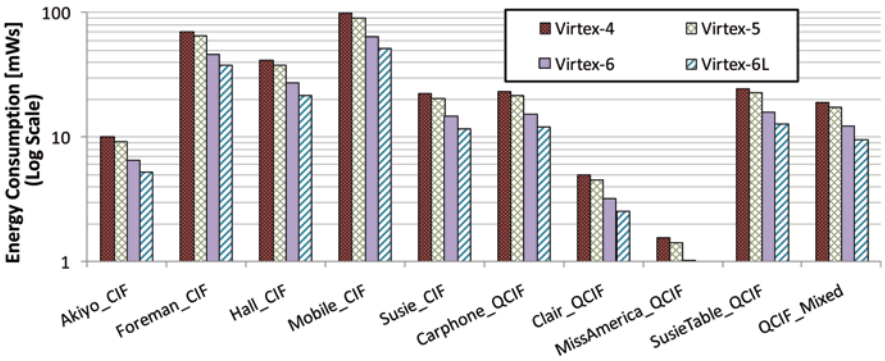


Fig. 4.37 Energy consumption of the energy-aware motion estimation for various FPGA fabrication technologies for various video sequences

4.5.5.2 Energy Comparison for Different Fabrication Technologies

Figure 4.37 shows the energy consumption of various video sequences for different FPGA fabrication technologies. Due to its low-power improvements [Kle10], Virtex-6/-6L-based implementations have less energy consumption compared to other FPGAs.

4.5.5.3 Overhead of enBudget and Hardware Design

Table 4.8 shows the performance, area, and energy overhead of the enBudget scheme for Xilinx Virtex-4-vlx160 (90 nm). The complete hardware implementation is in integer arithmetic. The overall energy overhead is insignificant as it is 10^6 times smaller than the energy benefit of the enBudget scheme. The memory overhead for storing Texture and Brightness is *2*#MBs*16-bits*. Texture and Brightness computation for one MB requires 160 and 4 cycles at the cost of 129 and 31 slices.

Table 4.8 Performance, area, and energy overhead of enBudget

		Virtex-4-vlx160 FF1148 [90 nm]				
		Latency (Cycles)	Area		Energy [nWs]	
			Slices	(GE)	Leakage	Dynamic
Group of Pictures (GOP)-Level		51	1,028	24,967	22.44	21.42
Frame-level		75	1,001	24,516	31.50	34.50
Macroblock (MB)-level		4	597	14,252	1.04	1.36
Total [for 1 frame]	QCIF	472	2,626	63,735	575.24	528.09
	CIF	1660	2,626	63,735	2024.60	1858.65

4.5.5.4 Summary of the Energy-Aware Motion Estimation and Energy-budgeting Scheme

This section presented the energy-aware Motion Estimation scheme that employs the concept of *Energy-Quality Classes*, which enables it to move in the energy-quality design space at run time. First, an adaptive Motion Estimator with multiple processing stages is presented that provides a foundation for designing the *Energy-Quality Classes*. The design of these *Energy-Quality Classes* is explained using a fast motion sequence, while highlighting the importance of different processing stages. The Motion Estimator is integrated with an adaptive energy-budgeting scheme that predicts the energy budget for different video frames and different MBs considering the run-time changing scenarios of available energy, video frame characteristics, and user-defined coding constraints while keeping a good video quality. Such an energy-aware Motion Estimation scheme is crucial for advanced video encoders when targeting battery-powered embedded multimedia systems. Especially, it is beneficial for low-cost battery-powered mobile devices where available energy status is changing erratically and energy-aware algorithms decide the life-time of the device.

4.6 Summary of Low-power Application Architecture

In order to achieve high energy savings, there is a need to redesign an application considering the potential of the underlying hardware platform. Therefore, first the H.264 video encoder application architecture is redesigned targeting reconfigurable processors. Several optimizations were performed to reduce the hardware pressure, i.e., the fabric requirements of a given computational hot spots. The data flow and data structures are discussed in detail along with their impact on the instruction and data caches. Afterwards, the design of low-power Custom Instructions (CIs) and Data paths was discussed. It was explained that operation reduction is required to reduce the dynamic power the Data Paths. A case was explained in detail using the In-Loop Deblocking Filter of the H.264 codec.

A detailed analysis of the spatial and temporal video properties was presented in Sect. 4.3. Different properties of a Human Visual System (HVS) were discussed.

Considering this discussion, different relationship between the optimal coding mode and the video properties were analyzed. Afterwards, important spatial and temporal video properties were selected. Using these video and HVS properties, rules for Macroblock categorizations were formulated. These rules facilitate the design of adaptive complexity reduction and energy-aware Motion Estimation schemes. To support various bit rates, Quantization Parameter based thresholding is employed for the Macroblock categorization.

The HVS-based Macroblock categorization is used by the adaptive computational complexity reduction scheme (see Sect. 4.4) which operates in three main steps. First the improbable coding modes are excluded using a relaxed prognostic early mode exclusion. Afterwards, a more aggressive exclusion is performed using a hierarchical fast mode prediction. In the third step, Mode Decision process is performed, where the candidate modes are processed one by one and the depending upon the output of a candidate mode, further improbable modes are excluded. The evaluation of the adaptive complexity reduction scheme is provided in Sect. 4.4.4 that demonstrates that 70% improbable modes are excluded with a minimal quality loss. Distribution of different evaluated modes and a frame-wise analysis of the correctly predicted modes is presented. Furthermore, a subjective comparison of the predicted modes and the optimal modes is performed to highlight the regions of misprediction.

For each candidate coding mode, an energy budget is computed using a predictive energy-budgeting scheme. This scheme is integrated in an adaptive Motion Estimator (see Sect. 4.5.1) to realize an energy-aware Motion Estimation scheme. To provide a run-time adaptivity for varying scenarios of available energy, changing user constraints and video properties, different *Energy-Quality Classes* are proposed. Each *Energy-Quality Class* provides a certain video quality at the cost of a certain energy consumption. It thereby enables the Motion Estimation to move in the energy-quality design space at run time in order to react to the unpredictable scenarios. The Motion Estimator is evaluated with and without adaptive energy-budgeting scheme in order to demonstrate the benefit of budgeting and *Energy-Quality Classes*. Moreover, a frame-level energy consumption analysis is provided to show that the proposed budgeting scheme allocates less energy to the homogeneous Macroblocks with slow to medium motion, and more energy to the textured Macroblocks with fast motion.

Chapter 5
Adaptive Low-power Reconfigurable Processor Architecture

This chapter presents the novel adaptive low-power reconfigurable processor architecture with a run-time adaptive energy management scheme. It exploits the novel concept of *Selective Instruction Set Muting* with multiple muting modes. The first section analyzes different scenarios, while motivating the need for run-time energy management. Afterwards, the adaptive energy management scheme with the novel concept of Custom Instruction (CI) Set Muting is discussed in Sect. 5.2. In this section different CI muting modes are explained along with the corresponding configuration of sleep transistors for different parts of the reconfigurable fabric. Afterwards, the required power-shutdown infrastructure is discussed. In Sect. 5.2.3 an overview of the energy management scheme is provided highlighting different requirements and steps considered at design-, compile-, and run-time.

The energy management scheme operates in two major steps. First it determines the energy minimizing instruction set considering the tradeoff related to leakage, dynamic, and run-time energy under run-time varying constraints of performance and reconfigurable fabric area (see Sect. 5.3). Afterwards, it determines the temporarily unused set of CIs and determines an appropriate muting mode for each CI considering the requirements of the currently executing and the upcoming computational hot spots (see Sect. 5.4). Section 5.4.3 presents how the energy benefit of a muting candidate is computed. Section 5.4.4 discusses that how the requirements of the upcoming hot spot are predicted and how the weighting factors for different CIs of the upcoming hot spot are computed.

5.1 Motivational Scenario and Problem Identification

Besides dynamic and leakage power, reconfigurable processors suffer from the power consumed when reconfiguring the instruction set. As discussed earlier in Sect. 3.4, the energy consumption of a reconfigurable processor (e.g., RISPP [Bau09], see Sect. 2.3.5) consists of the following components:

$$E_{ReconfProc} = E_{cISA_dyn} + E_{cISA_leak} + E_{FPGA_dyn} + E_{FPGA_leak} + E_{FPGA_reconf} \tag{5.1}$$

M. Shafique, J. Henkel, *Hardware/Software Architectures for Low-Power Embedded Multimedia Systems,*
DOI 10.1007/978-1-4419-9692-3_5, © Springer Science+Business Media LLC 2011

'cISA' and 'FPGA' denote the Core Instruction Set Architecture (i.e., the core processor) and the run-time reconfigurable FPGA fabric (for Data Paths and CIs), respectively. Note that the energy for performing a reconfiguration E_{FPGA_reconf} is actually part of the dynamic energy consumption but for clarity of subsequent discussions, it is listed separately. The process of reconfiguration causes the switching of configuration bits of the reconfigurable logic (CLBs: Configurable Logic Blocks) and connections (switching matrix) in order to realize different Data Paths (i.e., hardware accelerators) within the reconfigurable fabric. Therefore, E_{FPGA_reconf} may impose a non-negligible limitation on energy efficiency in reconfigurable processors [Te06]. For instance, executing a specific CI_i using a reconfigurable fabric typically leads to a reduced dynamic energy consumption in comparison to executing that CI using CISA ($E_{FPGA_dyn}(CI_i) < E_{cISA_dyn}(CI_i)$) due to faster CI execution (achieved by exploiting the inherent data-level parallelism). However, providing CI_i in the reconfigurable fabric introduces an initial overhead $E_{FPGA_reconf}(CI_i)$. The total number of executions of CI_i is therefore important to determine whether or not it is beneficial to execute CI_i using the reconfigurable fabric.

Let us have a deeper look at the problem using the H.264 video encoder application with three major hot spots namely Motion Estimation (ME), Encoding Engine (EE), and Loop Filter (LF) that execute subsequently for each video frame and require different sets of CIs (see Sect. 4.2, p. 80 for details). Figure 5.1a shows a simplified time scale of the execution of these hot spots and their related reconfigurations, where E_{FPGA_reconf} represents the major energy component. Figure 5.1b shows a different scenario for the same application where lesser reconfigurations are used (i.e., E_{FPGA_reconf} is smaller) to save energy (compared to Fig. 5.1a) at the cost of a slower frame encoding time.

The situation takes another shift when exploring this scenario for 90 nm (and below) technology nodes where leakage power may be more dominant, thus becoming imperative in the energy-aware design of reconfigurable processors [Ge04]. Hardware shutdown may be performed to reduce the leakage power of reconfigurable processors by switching-off the power supply to the reconfigurable regions with the help of high-V_t mid-oxide sleep transistors. The following components of a reconfigurable fabric can be individually shut down:

- **Logic:** Configurable Logic Blocks (CLBs) and programmable interconnect switch matrices (i.e., the routing resources that connect various CLBs)
- **Configuration SRAM:** The SRAM[1] cells that store the control bits, which define the configuration of the Logic

Note, shutting down the configuration SRAM of a reconfigurable region results in loss of its configuration data (as it is volatile). Therefore, it must be reconfigured again after powering-on, potentially requiring the overhead of an additional reconfiguration.

Then, **the challenging question arises**: whether to better shut down regions of the reconfigurable fabric (and execute the CIs using the CISA instead) to reduce

[1] In Xilinx FPGAs, 38% of the leakage power is consumed by the configuration SRAMs [TL03].

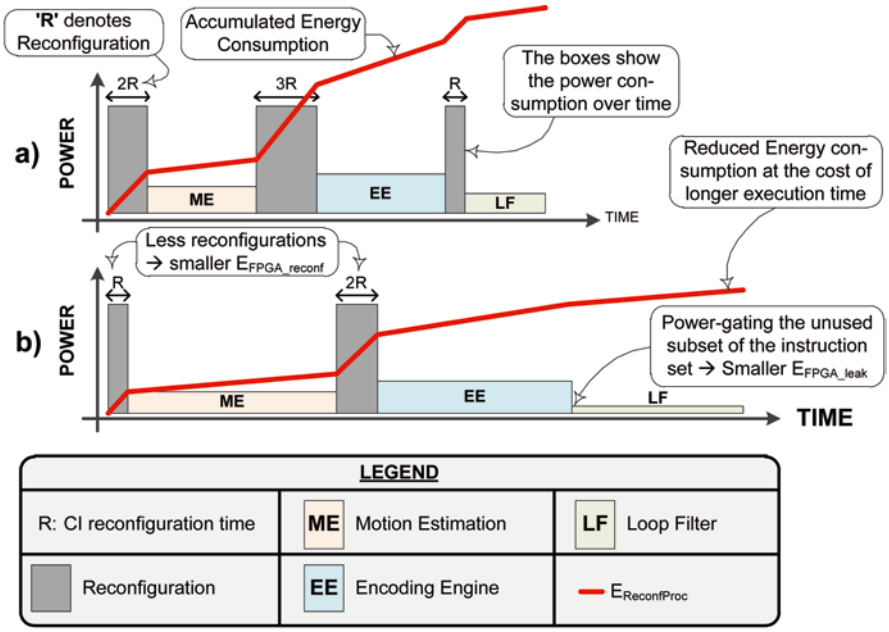


Fig. 5.1 Simplified comparison of energy consumption, highlighting the effects of different reconfiguration decisions

E_{FPGA_leak} or using a larger share of the reconfigurable fabric to decrease the application execution time at the cost of a higher E_{FPGA_reconf}. In Fig. 5.1b lesser CIs are executed on the reconfigurable fabric (smaller E_{FPGA_dyn}) and a bigger portion of the reconfigurable fabric can be shut down[2] (indicated by the lower heights of the boxes, e.g., ME). However, due to a longer execution time of the hot spot, E_{FPGA_leak} in Fig. 5.1b is not significantly reduced and $E_{cISA_leak}+E_{cISA_dyn}$ grow larger as more CIs are now executed using the CISA. Similar scenarios could be drawn for other applications alike, especially when considering multi-tasking systems where it cannot be predicted at compile/design time:

a. which task will obtain which share of the reconfigurable fabric
b. what is the task priority (may change at run time)
c. which task will run under which performance constraint, e.g., due to changing user preferences (e.g., desired frames per second in case of the H.264 application)

It is obviously not trivial to decide under which circumstances the execution using a reconfigurable fabric is energy-efficient or not especially when the application exhibits characteristics that cannot be predicted at design/compile time.

[2] At 150 nm, shutting down the currently unused portions of the reconfigurable fabric may not lead to noticeable savings and thus E_{FPGA_reconf} may dominate whereas in case of 65 nm it may be vice versa.

The problem is that under scenarios of run-time changing performance and/or area budgets, it can hardly be predicted at design/compile time which set of CI Implementation Versions will minimize the energy consumption when considering leakage, reconfiguration, and dynamic energy. At some point in time leakage may dominate, while at some other points in time (e.g., due to changed system constraints), reconfiguration energy may dominate. Decisions made solely at design/compile time will therefore with high certainty lead to energy-inefficient scenarios. Hence, a technology-independent run-time adaptive energy management scheme for reconfigurable processors is desirable.

5.1.1 Summary of the Motivational Scenario and Problem Identification

This section illustrated the need for run-time adaptive energy management with the help of different scenarios for H.264 video encoder. It was discussed that in which scenario leakage energy is more critical and in which scenario reconfiguration energy is more critical. It is also discussed why there is a need for joint consideration of leakage, dynamic, and reconfiguration energy in order to minimize the overall energy in dynamically reconfigurable processors. This section also discussed that why this problem cannot be solved at compile-time and why there is a need for run-time adaptive energy management.

5.2 Run-time Adaptive Energy Management with the Novel Concept of Custom Instruction Set Muting

Section 5.1 provided the motivational scenarios for identifying the energy problem in reconfigurable processors highlighting the issues related to leakage and reconfiguration energy under run-time varying scenarios. This section will introduce the novel concept of instruction set oriented shutdown (Sect. 5.2.1) that enables a far higher potential for leakage energy savings. The proposed concept of instruction set muting requires a power-shutdown infrastructure which is described in Sect. 5.2.2. Section 5.2.3 illustrates the decisions taken at design-, compile- and run time along with the *Run-Time Adaptive Energy Management Scheme*. The energy management scheme dynamically determines a set of energy-minimizing CI Implementation Versions for each hot spot considering leakage, dynamic, and reconfiguration energy such that these CIs fulfill the given performance and reconfigurable fabric area constraints. Afterwards, it decides which subset of CIs shall be muted at what time and in which mode in order to minimize the overall energy (considering leakage, dynamic, and reconfiguration energy). The details of determining the *Energy-Minimizing Instruction Set* and the *Instruction Set Muting* including the formal problem description and the algorithms will be discussed in the subsequent sections.

5.2.1 Concept of Muting the Custom Instructions

Before proceeding to the run-time adaptive energy management, this section introduces the concept of Instruction Set Muting which provides the foundation for the run-time adaptive energy management.

As discussed in Sect. 2.4, state-of-the-art low-power approaches in ASICs and FPGAs, deploy shutdown schemes that statically determine the parts of a reconfigurable fabric (Logic or Logic+Configuration SRAM) that can be powered-off [Ge04; MM05]. These approaches monitor the usage/state of a particular hardware and issue the shutdown signal to the hardware, e.g., after the hardware is idle for a certain threshold time (e.g., [Ge04]). These approaches mainly focus on hardware-oriented shutdown of the reconfigurable fabric irrespective of the application context (e.g., control flow, application priority etc.) and execution length of hot spots. Therefore, idle periods of Custom Instructions (CIs) cannot be exploited for the purpose of energy savings. When targeting reconfigurable processors, it is no longer efficient to employ the above-mentioned approaches, as it cannot be determined at compile time which CIs will be reconfigured on which part of the reconfigurable fabric. As a result, these hardware-oriented shutdown schemes suffer from the limitation of inflexibility and are highly dependent upon the underlying shutdown policy (see Chaps. 2 and 3).

A novel technique is proposed in this monograph, that shuns the leakage energy at the abstraction level of CIs (i.e., an instruction set oriented shutdown). This concept is named as ***selectively muting the CIs***. The proposed technique uses a power-shutdown infrastructure (see Sect. 5.2.2) in order to define the so-called CI muting modes (see Table 5.1) each leading to particular leakage energy savings. The proposed concept relates leakage energy to the execution context of an application, thus enabling a far higher potential for leakage energy savings. The run-time adaptive energy management in Sect. 5.2.3 aims at exploiting this potential. It decides which parts of the CI set shall be muted at what time and in which mode in order to minimize the overall energy (considering leakage, dynamic, and reconfiguration energy), as discussed in Sect. 5.2.3.

A CI may be muted through one of the following muting modes (see Table 5.1):

Mode I: Non-Muted CI (NM-CI): CI is active and operational.

Mode II: Virtually-Muted CI (VM-CI): CI cannot be executed due to the powered-off Logic. No reconfiguration is required in order to deploy this CI as its Configuration SRAM is kept powered-on. Hence, the otherwise necessary reconfiguration energy is not consumed. Therefore, the reduction in leakage energy is lower compared to Mode III (below). Mode II is beneficial when a subset of CIs is not demanded for a rather short period.

Mode II: Fully-Muted CI (FM-CI): CI is not operational, as both Logic and Configuration SRAM are powered-off. This significantly reduces the leakage energy. However, in order to deploy this CI, a reconfiguration is required which costs recon-

Table 5.1 Various custom instruction (CI) muting modes

Logic	Configuration SRAM	CI muting	Use-case for the CI
ON	ON	I. Non-Muted (NM-CI)	CI is demanded or it is scheduled to be reconfigured soon
ON	OFF	N/A	N/A (turning the Logic on but the configuration off may lead to undesired system behavior)
OFF	ON	II. Virtually-Muted (VM-CI)	CI is not demanded, but expected to be demanded soon
OFF	OFF	III. Fully-Muted (FM-CI)	CI is not demanded and it is not scheduled to be reconfigured soon

figuration energy and latency. Mode III is beneficial when a subset of CIs is not demanded for a rather long period.

The challenge is to determine which muting mode of Table 5.1 is beneficial for which set of CIs under run-time varying application contexts, i.e., which muting modes for CIs will bring more energy reduction while jointly considering the leakage, dynamic, and reconfiguration energy. This decision depends upon the execution length of the computational hot spots during which different CIs are used for the application acceleration. Moreover, this decision also depends upon the requirements of upcoming hot spot executions and the performance constraints (i.e., more or less reconfigurable fabric is required to accelerate hot spots). This challenge will be addressed by the proposed **Selective Instruction Set Muting** technique, which will be discussed in detail in Sect. 5.4.

To realize these muting modes, a power-shutdown infrastructure is required, as discussed below.

5.2.2 Power-shutdown Infrastructure for the Muted Custom Instructions

Figure 5.2 provides an overview of the infrastructure needed to apply the CI muting technique for reconfigurable processors. Multiple Data Path Containers (DPCs) are connected to a core pipeline. Each DPC is composed of multiple reconfigurable tiles and each tile contains Configurable Logic Blocks (CLBs) and programmable interconnect switch matrices (i.e., the routing resources that connect different CLBs). Control bits define the configuration of logic and routing resources and are stored in local Configuration SRAM, as shown in Fig. 5.2.

In order to realize different CI muting modes (as shown in Table 5.1), the power supply of each DPC is connected to two independent sleep transistors, one for the Logic and the other for the Configuration SRAM. Note that these two sleep transistors are used for all tiles of a particular DPC, whereas two different DPCs use different sleep transistors. The control signal for these sleep transistors for a given muting mode is specified in Table 5.1.

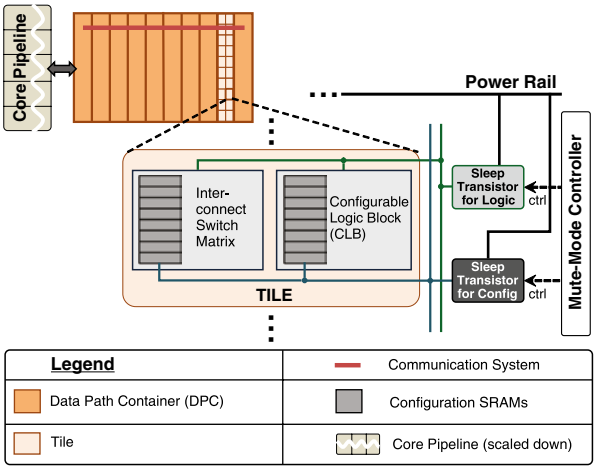


Fig. 5.2 Infrastructure necessary to exert the proposed CI muting technique

Figure 5.3 shows an example of muting the temporary unused set of CIs. After determining the energy-minimizing set of CIs for the current hot spot, the energy management scheme decides the muting mode of CIs. In order to set a particular muting mode for a CI, the control signal (as specified in Table 5.1) for the sleep transistors are issued to all DPCs of this CI. In Fig. 5.3, CI-A is Virtually-Muted (i.e., only the logic of DPC-1 and DPC-2 is power-gated) and CI-C is Fully-Muted (i.e., the logic and configuration SRAM of the DPC-6, DPC-7, and DPC-8 are power-gated). CI-B is kept in the Non-Muted mode (i.e., the logic and configuration

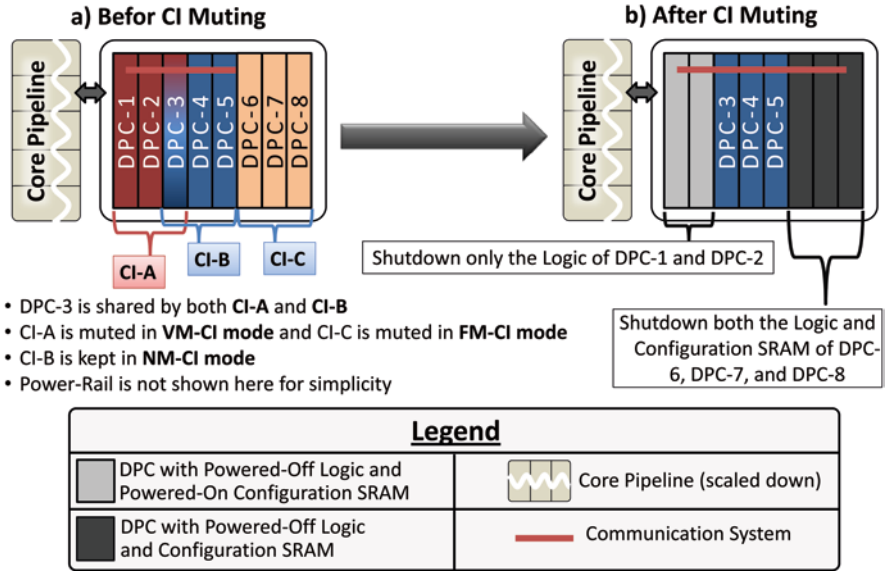


Fig. 5.3 Muting the temporarily unused instruction set

SRAM of the DPC-3, DPC-4, and DPC-5 are kept powered-on) as it is used by the current hot spot.

This power-shutdown infrastructure is currently not available in today's commercial FPGAs. Therefore, previous work in reconfigurable processors has not explored such a leakage energy reduction technique at the instruction set level. It is envisioned that that if FPGA vendors would provide this simple infrastructure, there would be a great opportunity to exert the proposed CI muting technique. It is especially beneficial for highly flexible Custom Instruction set architectures like RISPP [Bau09]. Consequently, reconfigurable processors would be far more energy efficient.

5.2.3 Run-time Adaptive Energy Management

Now the run-time adaptive energy management in reconfigurable processors will be explained in detail. Figure 5.4 presents an overview of the steps to be done at design, compile, and run time while highlighting the proposed run-time adaptive energy management scheme, its main tasks, and its connection to the system. At design time, the size of the reconfigurable fabric (i.e., how many DPCs are provided for loading Data Paths) and the core processor are fixed for a certain fabrication technology node (that determines their corresponding power properties). At compile time, the Data Paths are designed and their configuration bitstreams are generated.

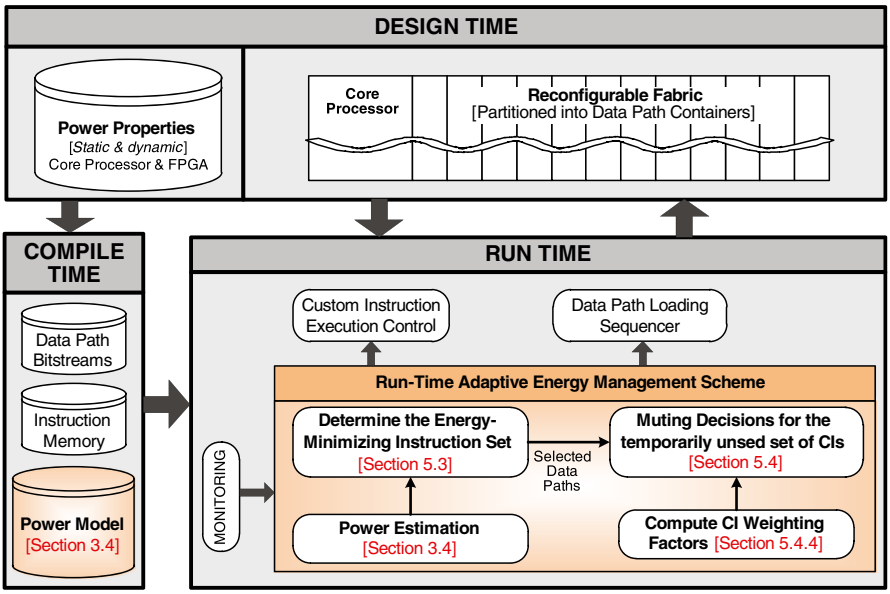


Fig. 5.4 Overview of the proposed adaptive low-power reconfigurable processor with run-time adaptive energy management along with the design-, compile-, and run-time steps

Additionally, the configuration for various Implementation Versions is generated at compile time (using the in-house developed automatic tool chain) considering different resource constraints (i.e., different types of Data Paths in varying quantities). The bitstreams of Data Paths and the Custom Instruction (CI) Implementation Versions were used to build the power model of dynamically reconfigurable processors (as discussed in Sect. 3.4, p. 63). This power model is then used to estimate the power at run time.

At run time, the key tasks of the energy management scheme are:

a. to dynamically determines a set of energy-minimizing CI Implementation Versions for each hot spot considering leakage, dynamic, and reconfiguration energy such that these CIs fulfill the given performance and reconfigurable fabric area constraints.
b. to determine the muting decisions for the temporarily unused subset of the CIs.

These decisions may depend upon the number of CI executions that may vary at run time due to the application level adaptivity (as discussed in Chap. 4), changing input data, performance constraints, and the execution length of the hot spot. The **online-monitoring** and the **prediction** scheme (as discussed in Sect. 2.3.5) are used to track and dynamically update the CI execution frequencies (i.e., which CI has executed how often for a certain hot spot). This is used as an input to the energy management scheme for choosing the energy-minimizing set of CI Implementation Versions.

The **power consumption** of different CI Implementation Versions is estimated using the proposed power model for dynamically reconfigurable processors considering the power used by computations (Data Paths), communication (buses), and local memory (as presented in Sect. 3.4, p. 63). The estimated power of CI Implementation Versions is forwarded to the energy management scheme. Since the placement of a Data Path on the reconfigurable fabric is unknown at the time the energy-minimizing set of CIs is determined, an average-case number of bus segments for the communication (see Sect. 3.4.1) are considered in the power estimation.

The estimated power consumption of the CI Implementation Versions and the predicted CI execution frequencies are forwarded to the energy management scheme for **choosing an energy-minimizing set** of CI Implementation Versions under varying constraints (details are explained in Sect. 5.3). Although each fabrication technology exhibits distinct leakage and dynamic power properties, the goal is to minimize the overall energy consumption (i.e., jointly considering leakage, dynamic, and reconfiguration energy) such that the chosen set of CI Implementation Versions fulfill the given performance and reconfigurable fabric area constraints. Therefore, the energy management scheme is beneficial for various fabrication technologies and different reconfigurable architectures. It is noted that the performance constraint and the amount of available reconfigurable area (i.e., the number of available DPCs) may change at run time due to, for example, user requirements, input data properties, changing number of tasks and their priorities (see Sect. 5.1).

Depending upon the chosen set of CI Implementation Versions, the energy management scheme determines the **muting decisions** of the temporarily unused set of CIs (details are explained in Sect. 5.4). The proposed technique uses various muting modes that enable leakage energy reduction at the abstraction level of CIs. The energy management scheme determines at run time which subset of CIs should be put into which muting mode (Table 5.1) at which time by evaluating at run time the possible associated energy benefit (a joint function of leakage, dynamic, and reconfiguration energy). Besides the requirements of the current and the upcoming hot spots, the **weighting factors** (see details in Sect. 5.4.4) of different CIs in a hot spot are given as the input to compute the benefit of a particular muting mode. The weighting factor of a CI represents the relative contribution of a CI (compared to other CIs) for the accelerated execution of a hot spot. The weighting factor of a CI in a hot spot is determined by considering the expected execution frequency of CIs, the time from the start of a hot spot until their first execution, and the average time between two executions of the same CI (details are explained in Sect. 5.4.4). The energy minimizing set of CIs and CI-level muting enables the energy management scheme to dynamically move in the *energy-performance design space* at run time depending upon the varying area and performance constraints.

Depending on the chosen set of CI Implementation Versions and the CI muting decision, certain Data Paths need to be reconfigured in the powered-on DPCs. Dynamically reconfigurable processors employ a *Data Path Loading Sequencer* to schedule the reconfigurations of the Data Paths required by the current hot spot [BSKH08]. In case there is no empty DPC, it also determines which Data Path shall be replaced to load the required Data Path [BSH09b].

5.2.4 Summary of the Run-time Adaptive Energy Management and CI Muting

In this section, the novel concept of CI muting was introduced that raises the abstraction level of shutdown to the instruction set level and provides the foundation for the run-time adaptive energy management. An overview of the proposed adaptive low-power reconfigurable processor along with the design-, compile-, and run-time steps was discussed in this section. Additionally different components for run-time adaptive energy management were introduced. Without these components and various CI muting modes (that enable to dynamically move in the *energy-performance design space* at run time depending upon the varying area and performance constraints), overall energy reduction (considering leakage, dynamic, and reconfiguration energy) could not be efficiently achieved in an adaptive manner. In the following, the components for run-time adaptive energy management will be presented in detail along with the formal problem description, analysis, developed solutions and algorithms, and their implementation results.

5.3 Determining an Energy-minimizing Instruction Set

The previous section presented the overview of different components of the proposed *Run-time Adaptive Energy Management Scheme* which is the key to realize an adaptive low-power reconfigurable processor architecture. As discussed in the overview, the energy management scheme considers leakage, dynamic, and reconfiguration energy to determine/choose an energy-minimizing set of Custom Instruction (CI) Implementation Versions that fulfill the reconfigurable fabric area and performance constraints. These constraints may vary at run time due to changing user requirements, tasks and their priorities, input data etc. For choosing the energy-minimizing set of CI Implementation Versions, the following information is required as input: (a) CIs that are expected to be executed in the current hot spot, (b) the predicted CI execution frequencies (Sect. 2.3.5, p. 35), and (c) the estimated power consumption of different CI Implementation Versions (Sect. 3.4, p. 63). The output is exactly one Implementation Version for each of the expected CIs.

5.3.1 Formal Problem Modeling and Energy Benefit Function

One of the basic tasks of the energy management scheme is to choose a set '*C*' of CI Implementation Versions to implement the demanded CIs for an upcoming hot spot (as shown in Fig. 5.4). The inputs to the algorithm for determining '*C*' are:

- the area constraint of the reconfigurable fabric N_{DPC_avail},
- the performance constraint $L_{HS_constraint}$, and
- a set of CIs expected to be executed in the current hot spot
- the predicted execution frequency of the expected CIs $F[CI_i]$ (i.e., the number of the expected CI executions which is obtained by an online-monitoring and prediction of the CI execution within a hot spot, Sect. 2.3.5).

As discussed earlier in Sect. 5.2, all of these parameters may change at run time. The following three constraints need to be fulfilled as the fundamental requirements:

Area Constraint: The chosen CI Implementation Versions can be implemented with the given amount of DPCs (N_{DPC_avail}, see Eq. 5.2), i.e., the number of Data Paths required to implement the chosen set of Implementation Versions should not exceed N_{DPC_avail}.

$$\left|\bigcup_{\vec{c}\in C} \vec{c}\right| \leq N_{DPC_avail} \tag{5.2}$$

Performance Constraint: a given performance constraint while minimizing the energy consumption. In case of the H.264 video encoder (see Sect. 5.2), the performance constraint is given as the targeted frame rate and the relative performance constraints (in percent) of the three major hot spots, resulting in $L_{HS_constraint}$ in cycles, i.e., the performance constraint of a specific hot spot. Furthermore, the

expected execution time $L_{cISA_HS_expected}$ of all non-CI instructions (i.e., the instructions that are executed using the cISA) is given. $L_{cISA_HS_expected}$ is independent from the chosen Implementation Versions but it needs to be considered to determine the overall performance. Note, reconfiguration latency is not considered in Eq. 5.3 as it depends upon the currently available Data Paths (the term $\vec{a}$ in Eq. 5.5 corresponds to the currently available Data Paths) and the *Reconfiguration Prefetching* (see Sect. 2.3.5). It may happen that some of the Data Paths required by the early-executing CIs may already be available. In this case, consideration of the reconfiguration latency in the calculation of $L_{HS_required}$ may violate the $L_{HS_constraint}$, thus may lead to a sub-optimal solution. Moreover, the algorithms of the *Data Path Loading Sequencer* (see [Bau09; BSKH08] for further details) also result in a performance improvement. Altogether, the currently available Data Paths, the *Reconfiguration Prefetching,* and the performance improvement due to the *Data Path Loading Sequencer* may already hide the reconfiguration latency. For a chosen set of Implementation Versions 'C' the performance constraint is evaluated by Eq. 5.3. If the performance constraint cannot be fulfilled then the fastest achievable performance is targeted.

$$L_{HS_required} := L_{cISA_HS_expected} + \sum_{\vec{c} \in C} (F[\vec{c}.getCI()] * \vec{c}.getLatency()) \leq L_{HS_constraint} \tag{5.3}$$

One Implementation Version per CI: for each demanded CI one Implementation Version (potentially also the cISA implementation) is chosen (Eq. 5.4). Note, all CIs can be executed using cISA, i.e., without any Data Paths (see Sect. 2.3.5, p. 35).

$$\forall i : |C \cap CI_i| = 1 \tag{5.4}$$

When multiple combinations of Implementation Versions fulfill the above three constraints (i.e., Eqs. 5.2, 5.3, and 5.4) then the goal is to minimize the overall energy consumption of the hot spot considering leakage, dynamic, and reconfiguration energy, i.e., minimize Eq. 5.5. There might exist a very low-energy implementation of a certain CI. However, in order to minimize the overall energy of a hot spot, all CIs executing in this hot spot needs to be considered along with their expected execution frequency (that may change at run time). Thus, various CI Implementation Versions jointly contribute towards minimizing the overall energy of a hot spot for a given amount of DPCs (N_{DPC_avail}) and performance constraint ($L_{HS_constraint}$).

$$\begin{aligned} E_{CI_HS_Total} = & \sum_{\vec{c} \in C} (F[\vec{c}.getCI()] * \vec{c}.getLatency() * P_{CI_dyn}(\vec{c})) \\ & + P_{DPC_leak} * \left| \bigcup_{\forall \vec{c} \in C} \vec{c} \right| * L_{HS_required} + E_{DPC_reconf} * \left| \vec{a} \triangleright \bigcup_{\forall \vec{c} \in C} \vec{c} \right| \end{aligned} \tag{5.5}$$

The first summand in Eq. 5.5[3] denotes the total dynamic energy of the chosen Implementation Versions. Some Implementation Versions use the reconfigurable fabric and the others use cISA for execution (see Sect. 2.3.5). The predicted execution frequency of the CIs (which is independent of a particular Implementation Version of the CI) is used to determine the total dynamic energy. For a given amount of DPCs, an Implementation Version of a CI_A with a much higher execution frequency compared to another CI_B would consume more dynamic energy compared to an Implementation Version of CI_B. When using the reconfigurable fabric, the dynamic power consumption—among others—depends on the number of used bus segments (see Eq. 3.2, Sect. 3.4.2), which itself depends on the Data Path positioning (see Sect. 3.4, p. 63), i.e., the relative position of the communicating Data Paths on the reconfigurable fabric. Although the actual Data Path positioning is considered for the power model (Chap. 6) and for simulating the energy consumption for results (Sect. 5.3.3 and 5.4.5, and Chap. 7), an averaged value is used for $P_{CI_dyn}(\vec{c})$ (that abstracts from the Data Path positioning) in order to be able to choose Implementation Versions dynamically at run time (i.e., before their actual Data Path positioning is known).

The second summand in Eq. 5.5 stands for the leakage energy of the required DPCs and for the hot spot execution time ($L_{HS_required}$ see Eq. 5.3). Bigger Implementation Versions may result in reduced dynamic energy due to faster execution (by exploiting more parallelism), but they require more Data Paths. The increased number of Data Paths to realize bigger Implementation Versions also results in increased leakage power. On overall, the total leakage energy also depends upon the $L_{HS_required}$. Therefore, in some cases a bigger Implementation Versions may result in relatively more leakage energy (compared to a smaller Implementation Versions) due to increased leakage power of more Data Paths. In some other cases a bigger Implementation Versions may result in relatively less leakage energy due to the faster execution, i.e., $L_{HS_required}$.

The third summand denotes the energy for reconfiguring the currently unavailable Data Paths (the term $\vec{a}$ in Eq. 5.5 corresponds to the currently available Data Paths). Depending upon the CIs used in the previous hot spot, some of the Data Paths required to realize the CI Implementation Versions for the current hot spot might already be available. Therefore, in Eq. 5.5, the reconfiguration energy for only the additionally required (i.e., currently unavailable) Data Paths is considered.

5.3.2 Algorithm for Choosing CI Implementation Versions

When determining the energy minimizing CI set, the run-time nature of the energy management scheme needs to be considered. Therefore, the following three means are applied for acceleration:

[3] The reactivation energy for one DPC is 3.5 pWs [Te06] while the energy of a hot spot is typically in multiples of mWs, i.e., approximately 10^9 times bigger (see Sect. 5.3.3). Therefore, the DPC reactivation energy overhead is not included in Eq. 5.5, as it does not affect the selection decision at the abstraction level of computational hot spots.

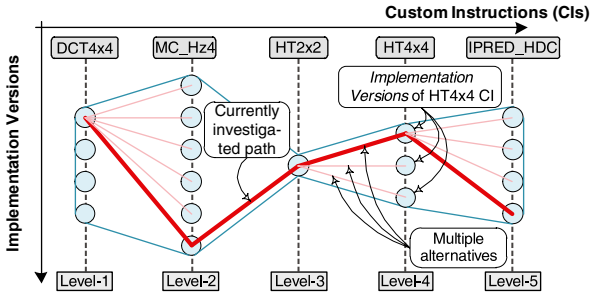


Fig. 5.5 Search space of five CIs with their implementation versions at the corresponding levels and the path of the energy-minimizing instruction set

a. Efficiently traversing the search space
b. Simplifying the cost function and incrementally updating the total cost
c. Early pruning of the search space

Traversing the search space: Figure 5.5 shows five CIs (x-axis) from a hot spot of the H.264 encoder and their corresponding Implementation Versions (y-axis). To distinguish between the CIs and the traversing sequence, the term 'levels' is used when analyzing the search space. To comply with Eq. 5.4, exactly one Implementation Version must be chosen at each level. The thick line (in Fig. 5.5) indicates a path through the levels that fulfills Eqs. 5.2, 5.3, and 5.4. The thin lines indicate the various alternatives at a certain level. When pruning the design space, it is important to determine invalid or suboptimal solutions as early as possible. Therefore, the CIs are sorted in the sequence in which the search space is traversed (x-axis) according to their importance $imp(CI_i)$, i.e., their expected latency improvement[4] compared to their respective cISA execution (averaged over all Implementation Versions). Compared to the opposite sorting (i.e., the CI with minimal $imp(CI_i)$ is traversed first) this reduces the average number of cost function calculations by 76.4× (from 36,766 down to 481) per video frame (in the example of an H.264). This reduction comes from pruning rather large parts of the search space (while obtaining the same result).

$$imp(CI_i) := F[CI_i] * \sum_{\forall \vec{c}_{ij} \in CI_i} (\vec{c}_{i_cISA}.getLatency() - \vec{c}_{ij}.getLatency()) / |CI_i| \tag{5.6}$$

Algorithm 5.1 shows the pseudo code of the algorithm for choosing the set of CI Implementation Versions that minimizes the overall energy consumption of a hot spot under given area and performance constraints. The pseudo code of the proposed algorithm is explained step-by-step in the following.

Calculating the Cost Function: The first step is to prepare arrays for the minimum energy consumption and the minimum latency of Implementation Versions for each level (lines 3–6). The energy consumption of a CI Implementation Version

[4] Experiments demonstrate that, in most of the cases, the CI *Implementation Version* with the fastest execution latency is also the one that provides the minimum dynamic energy due to its speedup, especially in case of tighter performance constraints. However, in terms of reconfiguration energy it might not always be the best choice.

```
1.  // Input: Area Constraint: N_DPC_avail; Performance Constraint: L_HS_constraint; sorted set of demanded CIs:
      CI[Level]; expected CI execution frequency: F[CI_i]; sum of the latencies of all non-CIs
      in the hot spot:
      L_cISA_HS_expected; DPC leakage power: P_DPC_leak; Energy for one reconfiguration: E_DPC_reconf
2.  // Output: Path of chosen CI Implementation Versions p_Best and its Energy E_Best
3.  ∀Levels l { // Prepare arrays for the min. energy consumption and the min. latency of
      Implementation Versions for each level
4.    L_CI_Level_Min[l] ← F[CI[l]]*MIN{c⃗_ij.getLatency()|c⃗_ij ∈ CI[l]};
5.    E_CI_Level_DynMin[l] ← F[CI[l]]*MIN{c⃗_ij.getEnergy()|c⃗_ij ∈ CI[l]};
6.  }
7.  L_HS_Min ← L_cISA_HS_expected + Σ_∀Level l L_CI_Level_Min[l];  // initialize total minimum latency
8.  E_HS_CI_DynMin ← Σ_∀Level l E_CI_Level_DynMin[l];              // initialize total minimum energy
9.  p_Best ← ∅;                                                    // initializes best so-far determined path
10. p_curr ← ∅;                                                    // initializes currently investigated path
11. Function ExploreLevel (Level l, p_curr, p_Best);               // starts from Level 1
12. BEGIN
13. ∀c⃗ ∈ CI[l] {      // For all Implementation Versions at the current Level
14.   N_DPC_required ← |⋃_{∀o⃗∈p_curr∪{c⃗}} o⃗|;      // compute the total number of DPCs required to realize
        c⃗ and all Implementation Versions in p_curr
15.   if (N_DPC_required > N_DPC_avail)          // Pruning Rule 1: Area Constraints
16.     continue;
17.   L_HS_Temp ← L_HS_Min − L_CI_Level_Min[l] + F[CI[l]]*c⃗.getLatency();
18.   if (L_HS_Temp > L_HS_constraint)           // Pruning Rule 2: Perf. Constraints
19.     continue;
20.   E_HS_CI_DynTemp ← E_HS_CI_DynMin − E_CI_Level_DynMin[l] + F[CI[l]]*c⃗.getEnergy();
21.   N_DPC_reconf ← |a⃗ ▷ ⋃_{∀o⃗∈p_curr∪{c⃗}} o⃗|;      // compute the number of DPCs that will be reconfigured
      to realize p_curr
22.   E_HS_DPC_reconf ← E_DPC_reconf*N_DPC_reconf;       // total reconfiguration energy of p_curr
23.   E_HS_DPC_leak ← P_DPC_leak*N_DPC_required*L_HS_Temp;  // total leakage energy of p_curr
24.   E_HS_CI_minTemp ← E_HS_DPC_DynTemp + E_HS_DPC_reconf + E_HS_DPC_leak
25.   if (p_Best ≠ ∅ ∧ E_Best < E_HS_CI_minTemp) // Pruning Rule 3
26.     continue;
27.   if (l = LastLevel) {   // valid solution found, i.e., an Implementation Version is success fully chosen
28.     p_Best ← p_curr;
29.     E_Best ← E_HS_CI_minTemp;
30.     return (p_Best, E_Best);
31.   }
32.   // Explore the next level
33.   L_HS_Min ← L_HS_Temp;                  // update the overall minimum latency
34.   E_HS_CI_DynMin ← E_HS_CI_DynTemp;      // update the overall minimum energy
35.   (p_Best, E_Best) ← ExploreLevel (l + 1, p_curr ∪ {c⃗}, p_Best)
36.   // Restore L and E values for further incremental updates
37.   L_HS_Min ← L_HS_Min + L_CI_Level_Min[l] − F[CI[l]]*c⃗.getLatency();
38.   E_HS_CI_DynMin ← E_HS_CI_DynMin + E_CI_Level_DynMin[l] − F[CI[l]]*c⃗.getEnergy();
39. }
40. return (p_Best, E_Best);
41. END
```

Algorithm 5.1 Pseudo code of Determining the Energy Minimizing Instruction Set

thereby corresponds to an offline-calculated average dynamic energy as discussed for Eq. 5.5. In the second step, the sums of the array entries are calculated to obtain the fastest possible execution time L_{HS_Min} and the minimum possible dynamic Implementation Version energy consumption $E_{HS_CI_DynMin}$ for the hot spot (see lines 7–8), irrespective of the area and performance constraints. Whenever a specific Implementation Version is chosen at a certain level, these two values are incrementally updated (see lines 17, 33 and 20, 34), i.e., for L_{HS_Min} the minimum-latency Implementation Version that was initially used to calculate the sum is replaced by the actually chosen Implementation Version at this level (same procedure for $E_{HS_CI_DynMin}$). Therefore, L_{HS_Min} is calculated without the need to iterate through all levels for one calculation. As it always represents the fastest possible hot spot execution time at the current level, it is used for pruning (similar for $E_{HS_CI_DynMin}$).

Pruning Rules: The following three pruning rules are incorporated to determine invalid or suboptimal solutions as early as possible:

a. ***Pruning Rule 1: Area Constraint:*** Parts of the search space that require more DPCs than what is available are discarded (lines 14–16).
b. ***Pruning Rule 2: Performance Constraint:*** The above-discussed iteratively-updated value L_{HS_Min} can directly be used to prune those parts of the search space that cannot fulfill the performance constraint (see lines 17–19), as for the not-yet traversed levels the fastest possible Implementation Version execution is assumed. Additionally, in line 7 the at-compile-time calculated execution time of the cISA is considered. If no valid solution exists within a given area constraint, the energy management scheme chooses the set of Implementation Versions that offers the fastest achievable performance.
c. ***Pruning Rule 3: Sub-optimal Energy Consumption:*** Whenever the algorithm finds a valid solution (i.e., successfully chooses an Implementation Version for the last level) the energy consumption for this solution is then stored for further comparison (E_{Best} in line 29). When searching afterwards for alternative valid solutions, their energy consumption is compared against E_{Best} (line 25). Therefore, the incrementally updated dynamic Implementation Version energy $E_{HS_CI_DynMin}$ and the leakage and reconfiguration energy of DPCs[5] need to be considered (see lines 21–24).

After choosing the energy-minimizing set of CI Implementation Versions, the energy management scheme determines the muting decision for the temporarily unused subset of the Custom Instruction Set (details and algorithm will be discussed in Sect. 5.4, p. 146). Note: the energy management scheme performs CI muting at the start of each hot spot to avoid frequent on-off switching of the sleep transistors (typically the length of a hot spot is several milliseconds[6], see Sect. 5.3.3).

[5] Due to the irreversible nature of the operator $\cup$ (used in line 21), they cannot be incrementally updated like $E_{HS_CI_DynMin}$.

[6] The length of a hot spot is predicted from the latency values of different CIs used in the hot spot and their corresponding expected execution frequencies (as predicted by the online-monitoring Sect. 2.3.5).

5.3.3 Evaluation and Results for Energy-Minimizing Instruction Set

Now the adaptive energy management scheme will be evaluated for various fabrication technology nodes using the H.264 video encoder application (as discussed in Chap. 4). The parameters and their corresponding values (for different fabrication technologies) that are used as the basic input in the following experiments are presented in Table 5.2 with their corresponding sources of information.

5.3.3.1 Evaluating the Adaptive Energy Management Scheme on Different Technologies

Figure 5.6 shows the energy-performance design spaces in which the *Adaptive Energy Management Scheme* moves at run time—for a certain technology—to achieve the overall minimum energy for a given performance and area constraint. In order to show the technology-independent nature of the energy management scheme (i.e., it is beneficial for various fabrication technologies), it is evaluated for four different technologies under various performance and area constraints. Since leakage energy is dominant in 65 nm and 90 nm, the energy management scheme may choose a different set of CI Implementation Versions for 65 nm and 90 nm (more variation in regions E-4 and E-7) compared to 150 nm (E-9). However, due to low-power optimizations in the new Virtex-6 (40 nm and 40 nmL, [Kle10]) the overall leakage energy is significantly reduced, therefore, the energy management scheme makes a different decision in choosing the energy minimizing instruction set for 40 nm and 40 nmL (as shown by different energy variations in E1-E3). 40 nmL is a version of Virtex-6 that operates at a lower voltage, therefore, the overall energy is further reduced compared to 40 nm (E3).

Figure 5.6a–d contain a flat region showing similar minimum energy points for different performance constraints. This is because of two reasons: (1) either the area (i.e., reconfigurable fabric) is insufficient or (2) the achieved performance is greater than the performance constraint in order to minimize the leakage energy that may increase due to the slow execution. Note: the performance improvement comes in discrete steps, as a set of CIs collectively results in a speed up of the hot spot. A similar behavior can be observed in E-11 for 150 nm.

There is an interesting scenario that shows the efficiency of the energy management scheme: In E-5, the overall energy is very high (as leakage is dominant) because of a longer execution time. As soon as more DPCs are available the energy management scheme decides to switch to a faster execution (although not required from a performance point of view) thus cutting down the leakage energy significantly (E-6). In short, it can indeed be beneficial to pay an additional reconfiguration to reduce the leakage energy. However, the scenario is changed for 40 nm and 40 nmL due to device-level leakage optimizations, Fig. 5.6 (E1, E3).

Table 5.2 Parameters and evaluation conditions with their corresponding reference sources

	Attributes	40 nmL** (low power)	40 nm**	65 nm	90 nm	150 nm	Source
	Voltage [V]	0.9	1.0	1.0	1.2	1.5	[Xil10a]
Reconfigurable Fabric	FPGA	Virtex 6 (−1L)	Virtex 6 (−1)	Virtex-5 xc5vlx85	Virtex-4 xc4vlx80	Virtex-II xc2v6000	
	Total size [CLBs]	–	–	6480*	8960	8448	[Xil10a]
	Total leakage power [W]	–	–	1.297	0.854	0.068	[Xil10b]
	Min. dynamic power [W]	–	–	0.492	0.48	1.2	[Xil10a]
	Size of 1 DPC [CLBs]	68*	68*	68*	96	96	[BSH08a]
	Leakage power of 1 DPC [mW]	6.99	9.46	13.51	9.15	0.77	[Xil10b]
	Dynamic power scaling factor	0.229	0.287	0.41	0.4	1	[Xil10a]
cISA	Size of Leon Core and Run-Time Management System [CLBs]	1980*	1980*	1980*	2816	2816	[BSH08a]
	FPGA-to-ASIC: Leakage [mW]	8.201	11.100	15.852	10.736	0.907	[KR07]
	Technology scaling factor	(0.34)	(0.34)	(0.445)	(0.583)	(0.763)	[BTM00]

*The Virtex-5/6 internal CLB Composition is Different Compared to Previous FPGAs
**The Power Values are Scaled from Virtex-5 according to [Kle10]

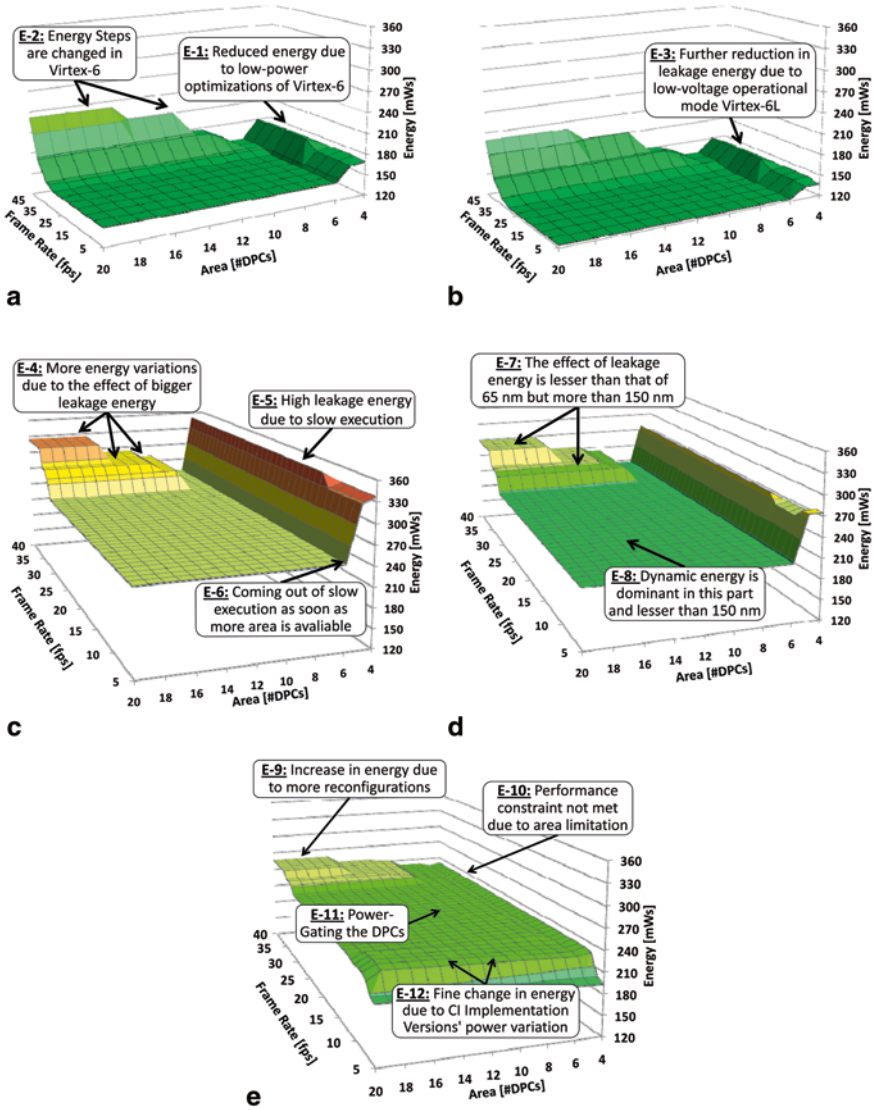


Fig. 5.6 Energy-performance design spaces: evaluation of the energy minimization space using the adaptive energy management scheme under various area and performance constraints for four fabrication technologies for an encoding of 40 QCIF (176×144) frames

Figure 5.7 presents the breakdown of energy consumption for the four technologies when encoding at 35 fps with a different amount of DPCs. Figure 5.7 shows that in case of 150 nm dynamic and reconfiguration energy make up for the major part of the total energy consumption. When moving from 4 to 20 DPCs the performance improvement comes at the cost of additional reconfiguration energy. For 65 nm and

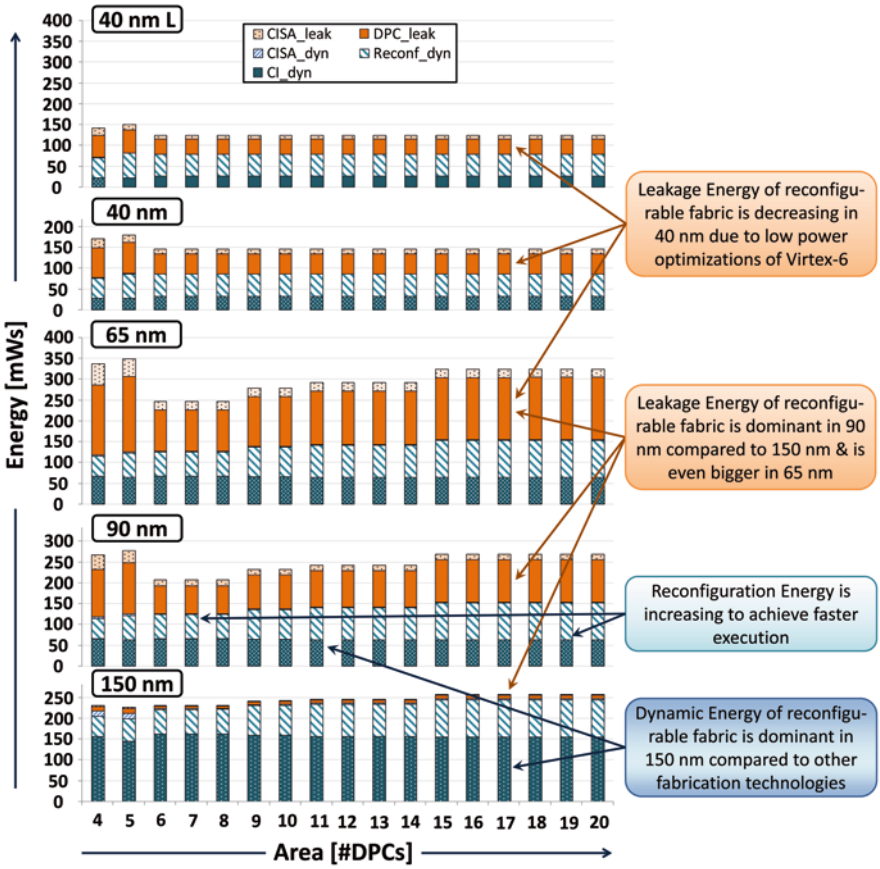


Fig. 5.7 Comparison of energy components in different fabrication technologies under various area constraints

90 nm, when moving from 5 to 6 DPCs there is a decrease in the total energy as for faster execution the leakage energy has been reduced significantly (as also shown in E-6 of Fig. 5.6). From 7–20 DPCs, the leakage energy is changing due to each additional powered-on DPC. However, in case of 40 nm and 40 nmL, the reconfiguration energy and leakage energy are comparable. Here due to reduction of leakage energy in 40 nm and 40 nmL, the energy management scheme chooses a different energy-minimizing set of Implementation Versions compared to the case of 65 nm.

5.3.3.2 Evaluating the Adaptive Energy Management Scheme for Encoding of different Resolutions

Figure 5.8 presents the energy-performance design spaces for encoding of two different resolutions at 40 nm technology. The encoding of different video resolutions

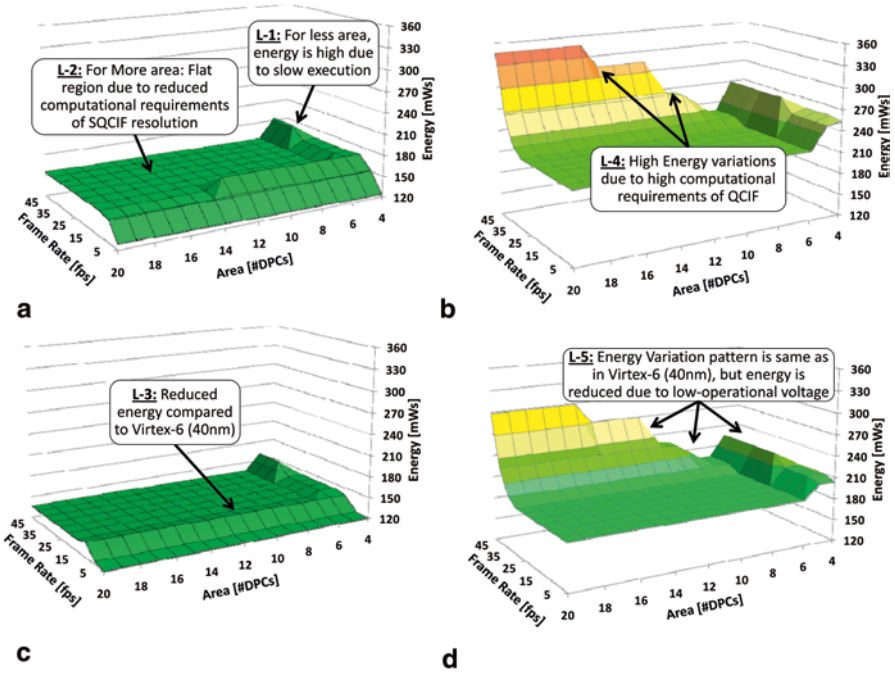


Fig. 5.8 Comparing energy-performance design spaces for different video resolutions when using the energy management scheme under various area and performance constraints for an encoding of 60 video frames

results in different computational requirements. QCIF resolution (176×144 pixels) has 2× more Macroblocks (MBs) to encode compared to SQCIF resolution (128×96 pixels). Therefore, in order to meet same frame per second performance, QCIF requires more DPCs compared to SQCIF, which directly corresponds to increased energy. The fact is notable in the Fig. 5.8 (L4). In case of SQCIF the performance constraint is met with relatively less number of DPCs compared to QCIF (see L2). When comparing 40 nm and 40 nmL, the energy-performance design spaces are almost similar, as the only difference is in the operational voltage that reduces both leakage and dynamic power.

Figure 5.9 shows the breakdown of energy at the time-frame level (each time-frame = 0.05 MCycles) along with the number of CI executions for QCIF@30 fps using 65 nm. It shows the contribution of different energy components at different time instances. At the start of the Motion Estimation (ME) hot spot there occur several reconfigurations. Thus, the reconfiguration energy is dominant (Label-A: until 0.2 MCycles). While reconfiguring the Data Paths for the CIs of ME, the number of CI executions per time frame is increasing gradually that also demonstrates a gradual acceleration of the ME hot spot. During the execution of the ME hot spot (Label-B: 0.2–1.15 MCycles) the leakage energy is dominant. In 1.15 MCycles the ME hot

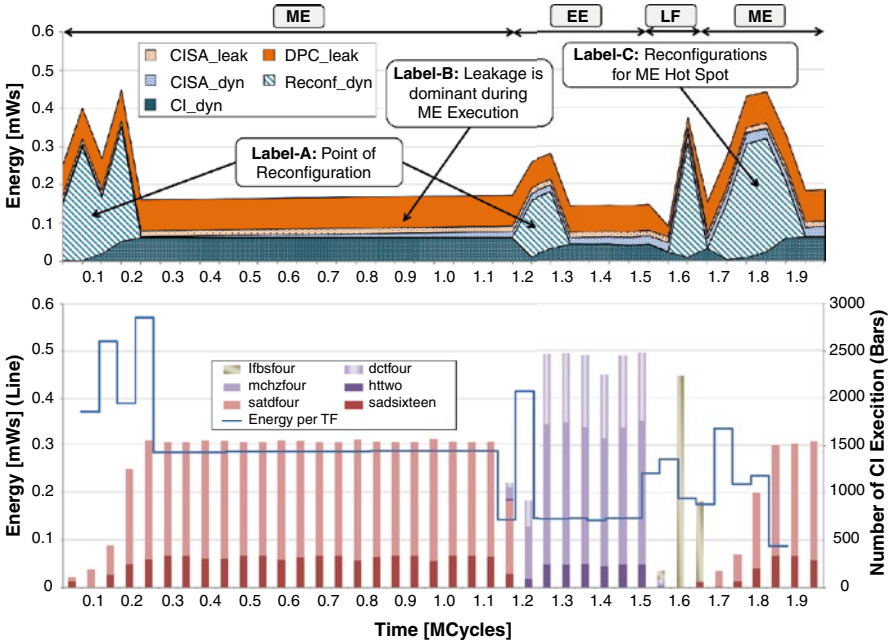


Fig. 5.9 CI Execution results for 30 fps on 65 nm showing a detailed breakdown of energy components highlighting the contribution of reconfiguration and leakage energy. The lower graph shows the detailed execution pattern of various CIs executing in different hot spots of the H.264 video encoder along with total energy consumption

spot has finished executing and the Data Paths for the EE (Encoding Engine) hot spot start reconfiguring. It is also notable in Fig. 5.9 that only few reconfigurations are performed for the EE hot spot. It is because of the reason that the CIs in EE hot spot share Data Paths from the previous ME hot spot. These shared Data Paths are already available in the DPCs and no additional reconfiguration is required.

Note: the leakage energy in LF hot spot is less compared to that in the ME and EE hot spots due to the muting of the temporarily unused set of CIs (details of CI muting will be discussed in Sect. 5.4). In this case, only two Data Paths are used and the remaining ones are power-gated to save leakage. Since the time between LF (Loop Filter) and next frame ME is small, the reconfiguration Energy is dominant (Label-C) within this region. A single frame encoding finishes in 1.65 MCycles.

5.3.3.3 Hardware Implementation

The adaptive energy management scheme is implemented on a Xilinx Virtex-II based hardware prototype (see Fig. 6.1 in Sect. 6.1) that was also used for the power measurements (see the hardware implementation results in Table 5.3). The recur-

Table 5.3 Hardware implementation results for the energy management scheme on the RISPP prototyping platform (see Fig. 6.1 in Sect. 6.1)

Implementation details	
#Slices	615
#LUTs	883
#MULT18×18	7
Gate Equivalents	39,794
Clock delay [ns]	10.371

sive call in the pseudo code (Algorithm 5.1) is thereby implemented in an iterative way and an array stores the currently explored Implementation Version for each level. The logic is implemented in form of a state machine with 17 states, where eight states are responsible for calculating the cost function and the pruning conditions. One complete calculation of cost function requires on average 12 cycles. The overall average performance overhead is 5,772 cycles per video frame (for 481 calls of cost function, as mentioned in Sect. 5.3), which is insignificant. The overall power overhead of the energy management scheme is 42.237 mW (41 mW dynamic + 1.237 mW leakage) for a Xilinx Virtex-II based hardware prototype. However, the hardware for the scheme is only used at the start of each hot spot for choosing the set of CI Implementation Versions and during the hot spot execution it is not used. Therefore, the overall energy overhead is insignificant compared to its energy benefit. Moreover, for the final system an ASIC-based implementation is foreseen which would result in much lesser power overhead.

5.3.4 Summary of Energy Minimizing Instruction Set

The adaptive energy management scheme chooses an energy-minimizing set of CI Implementation Versions for each computation hot spot such that this set fulfils the reconfigurable fabric area and performance constraints. The goal is to minimize the overall energy of the hot spot while considering leakage, dynamic, and reconfiguration energy along with the predicted CI execution frequency. In order to expedite the algorithm (especially when considering its run-time nature), three means are applied. These are (a) Efficiently traversing the search space, (b) Simplifying the cost function and incrementally updating the total cost, (c) Early pruning of the search space to determine invalid or suboptimal solutions as early as possible. Evaluation for various fabrication technologies showed that the proposed scheme moves in the *energy-performance design space* at run time and it is equally beneficial for various technologies, various performance constraints, and changing amount of available reconfigurable fabric area (i.e., available DPCs). After choosing the energy-minimizing set of CI Implementation Versions, the energy management scheme determines the muting decisions for the temporarily unused set of CIs, i.e., which muting mode is beneficial for which subset of CIs when considering leakage, dynamic, and reconfiguration energy.

5.4 Selective Instruction Set Muting

As discussed earlier in Sect. 5.2.1, the adaptive energy management scheme employs a *Selective Instruction Set Muting* technique that shuns the leakage energy at the abstraction level of Custom Instructions (CIs), i.e., an instruction set oriented shutdown. When targeting dynamically reconfigurable processors, it is hard to determine at compile time which CIs will be reconfigured on which part of the reconfigurable fabric (i.e., in which DPCs). As a result, the hardware-oriented shutdown schemes [Ge04; MM05]—that monitor the idle usage/state of a particular hardware to issue the shutdown signal—suffer from the limitation of inflexibility and are highly dependent upon the underlying shutdown policy. Contrarily, the instruction set oriented shutdown (i.e., CI-level muting) relates leakage energy to the execution context[7] of an application to enable a far higher potential for leakage energy savings. Considering the power-shutdown infrastructure (as discussed in Sect. 5.2.2), several *CI muting modes* (see Table 5.1 in Sect. 5.2.1) can be defined, each leading to particular leakage energy savings. These muting modes are:

a. **Non-Muted CI** (NM-CI): CI is active and operational.
b. **Virtually-Muted CI** (VM-CI): CI cannot be executed due to the powered-off Logic. No reconfiguration is required in order to deploy this CI as its Configuration SRAM is kept powered-on. It is beneficial when a subset of CIs is not demanded for a rather short period.
c. **Fully-Muted CI** (FM-CI): CI is not operational, as both Logic and Configuration SRAM are powered-off. A reconfiguration is required (that costs reconfiguration energy and latency) to deploy this CI. It is beneficial when a subset of CIs is not demanded for a rather long period.

After the energy-minimizing set of CI Implementation Versions is chosen (Sect. 5.3), the muting modes are determined for the temporarily unused subsets of CIs. Then **the challenging question arises,** which subset of the CI set shall be muted at what time and in which mode (VM-CI or FM-CI, see Table 5.1) under run-time varying application contexts in order to minimize the overall energy, considering the tradeoff between leakage energy saving and reconfiguration energy overhead. This decision depends upon the execution length and the requirements of the computational hot spots (during which different CIs are used for the application acceleration). These parameters may vary at run time depending upon the application execution properties, and the area and performance constraints (i.e., more or less reconfigurable fabric is required to accelerate hot spots), as it will be motivated in Sect. 5.4.1. Therefore, the CI muting decision cannot be determined at compile-time.

In the following, the importance of different muting modes (VM-CI and FM-CI) and their relationship to the muting duration (i.e., in which scenario which muting mode is beneficial) are discussed with the help of simple motivational scenarios in Fig. 5.10.

[7] Instead of idle hardware state monitoring, idle periods of CI usages (i.e., temporarily unused subset of CIs) are exploited for the purpose of energy savings.

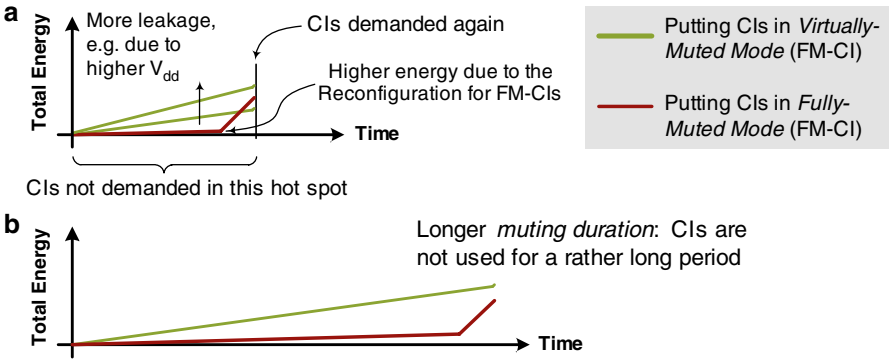


Fig. 5.10 Comparing the energy requirements of virtually- & fully-muted CIs for two scenarios

5.4.1 Problem Description and Motivational Scenarios

Various challenging questions arise given the three muting modes from above: For instance, whether VM-CIs or FM-CIs provide more energy reduction when both leakage energy and reconfiguration energy are considered. This is reflected by the following equation (Eq. 5.7) where the decision of a muting mode depends upon the execution length of a hot spot (i.e., L_{HS}):

$$P_{VM_CIs} * L_{HS} \overset{?}{<} P_{FM_CIs} * L_{HS} + \sum (P_{DPC_reconf} * T_{DPC_reconf}) \tag{5.7}$$

Additionally, it needs to be clarified whether this decision can be determined statically ([Ge04, MM05]) or whether it requires a dynamic decision. As it will be shown later on, this depends on a specific run-time scenario (i.e., application's execution context).

Figure 5.10 shows three different run-time scenarios and compares the energy requirements of VM- and FM-CIs. It is noticeable that the energy requirements of FM-CIs are significantly lower for most of the time. However, when a FM-CI is demanded again for executing a hot spot, its muting mode is switched to NM-CI. It increases the energy requirements significantly due to the demanded reconfiguration of these FM-CIs (see the 2nd addend on the right hand side of Eq. 5.7). This fact is noticeable in Fig. 5.10a where the muting duration of FM-CIs is too short to amortize the reconfiguration overhead $P_{DPC_reconf} * T_{DPC_reconf}$. An alternate scenario can also been seen in Fig. 5.10a where the leakage energy in VM-CIs is large compared to the reconfiguration energy of FM-CIs. Such a scenario may result due to a higher V_{dd} in order to support a higher clock frequency, which is required to fulfill the performance constraints.

Figure 5.10b shows the scenario, where the leakage energy of the VM-CIs is higher than that of FM-CIs as the muting duration is too long (e.g., due to relaxed user constraints). Therefore, the CIs in the *fully-muting* mode amortize the reconfiguration overhead. Actually, the muting duration for CIs varies due to the changing application contexts as a result of (1) varying control flow of the application,

(2) changing application priorities in multi-tasking systems (varying the amount of available reconfigurable fabric assigned to it), and (3) changing user preferences (e.g., performance constraints, thus changing hot spots' execution length). Hence, it is not possible to predict the energy requirements of VM-CIs at compile time.

The three scenarios in Fig. 5.10 demonstrate that it is typically not possible to decide Eq. 5.7 at compile time. Therefore, a run-time *Selective Instruction Set Muting* technique is desirable for adaptive low-power reconfigurable processors. It determines at run time which subset of CIs should be put into which muting mode (Table 5.1) at what time. For this, it evaluates the possible associated energy benefit (a joint function of leakage, dynamic, and reconfiguration energy) at run time.

5.4.2 Operational Flow for Selective Instruction Set Muting

The proposed technique evaluates a possible energy benefit (a function comprising leakage, dynamic, and reconfiguration energy, see Sect. 5.4.3) of different CIs to select an appropriate muting mode for the corresponding DPCs at run time[8]. Figure 5.11 presents a time line showing the execution sequence of previous, current, and upcoming hot spots along with the point of time where the CI muting mode is selected.

Figure 5.12 presents the flow of the CI muting technique. It is triggered ahead of a hot spot execution. The key inputs are:

- a list of Data Paths that are available from the previous hot spot ($\vec{p}$) and
- lists of Data Paths that are required by the current and the upcoming hot spots ($\vec{c}$ and $\vec{n}$, see Fig. 5.13).

The algorithm returns non-muted, virtually-muted, and fully-muted DPCs for the current hot spot (DPC_{NM}, DPC_{VM}, DPC_{FM}). The four major steps are:

Step 1: The DPCs to fulfill the Data Path requirements of the current hot spot (see Fig. 5.13) are kept in *non-muted* mode (DPC_{NM}, i.e., DPCs in active state). After-

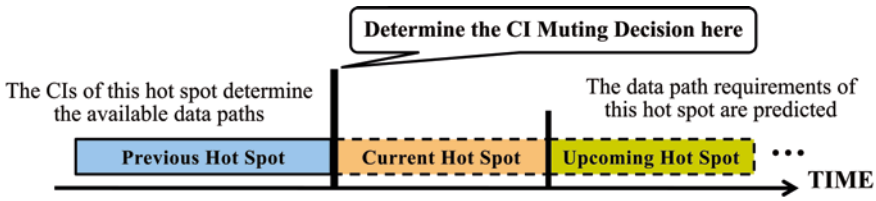


Fig. 5.11 Time-line showing the execution sequence of hot spots and the situation for a CI muting decision

[8] As mentioned in Sect. 5.2.2, in order to set a particular muting mode for a CI, the control signals (as specified in Table 5.1) for the sleep transistors (for Logic and the Configuration SRAM) are issued to all DPCs of this CI.

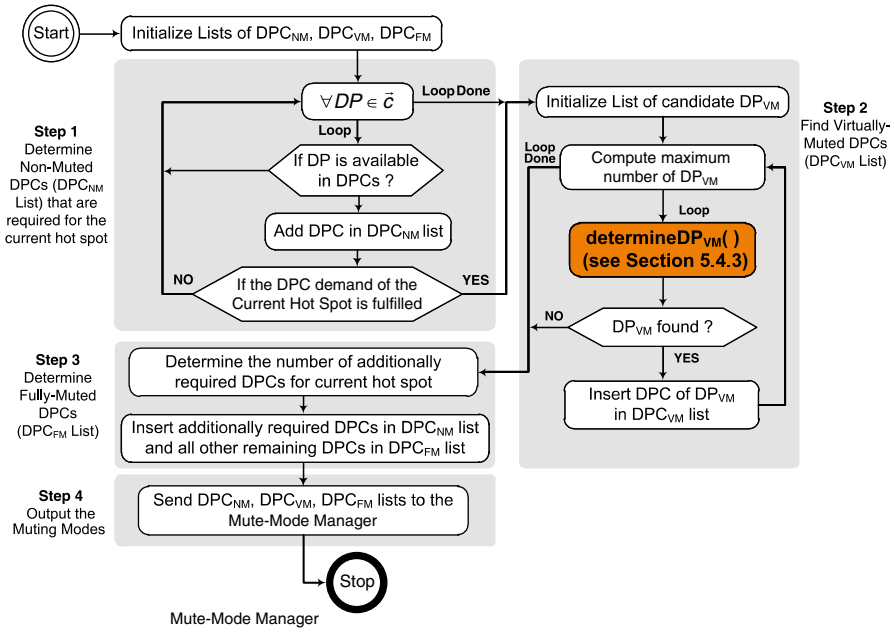


Fig. 5.12 Flow for selecting a muting mode for the custom instruction (CI) set

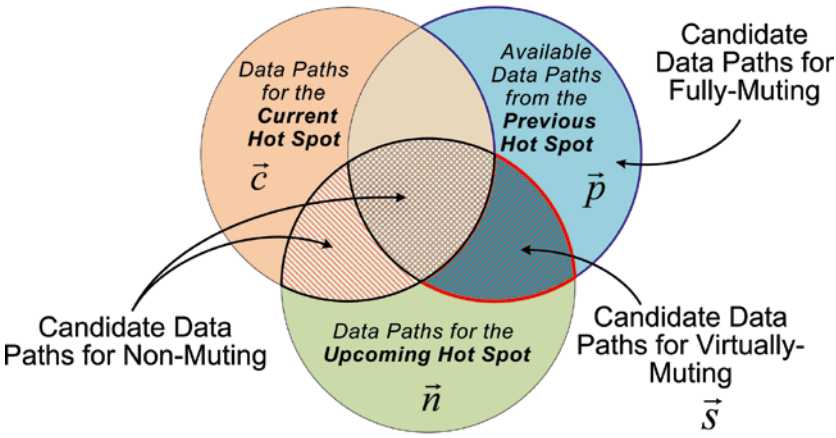


Fig. 5.13 Venn diagram showing the data path requirements of previous, current, upcoming hot spots

wards, the Data Paths required for $\vec{c}$ are checked if they are already available in DPCs (i.e., common Data Paths in $\vec{c}$ and $\vec{p}$). If yes, then these DPCs are added to the DPC_{NM} list.

Step 2: Afterwards, the requirements of the upcoming hot spot are predicted (see details in Sect. 5.4.4) and *virtually-muting* DPCs (DPC_{VM}) are determined. At the

start of the current hot spot, the Data Paths that are available from the previous hot spot are compared to the Data Paths required by the upcoming hot spot.

- If a Data Path is currently available, not needed for the current hot spot, but needed again for the upcoming hot spot, it is a candidate for the *virtually-muting* mode (Fig. 5.13) and it is added to the candidate list.
- Then, the maximum number of *virtually-muted* Data Paths (DP_{VM}) is computed by considering the requirements of the current and the upcoming hot spots and the total number of DPCs.
- However—depending upon the requirements of the current hot spot—the maximum number of DP_{VM} may be smaller than the total number of *virtually-muting* mode candidates as some of the DPCs may need to be reconfigured to fulfill the performance requirements of the hot spot. Therefore, the '**determineDP$_{VM}$**' function (details in Sect. 5.4.3) evaluates the energy benefit of all candidates for the *virtually-muting* mode. It then chooses the one that provides the highest energy benefit among all candidates.
- If the energy benefit (no additional reconfiguration required) overcomes the overhead (larger leakage) then the DPC of DP_{VM} are added into the DPC_{VM} list. Alternatively, no DPC is put into the DPC_{VM} list.
- The function '**determineDP$_{VM}$**' is iteratively executed until maximum number of DP_{VM} is zero.

Step 3: If some of the Data Paths required by the current hot spot are not available, then more DPCs are kept in *non-muted* mode as they will be reconfigured to fulfill the requirements of the current hot spot. These DPCs are added into the DPC_{NM} list. Those DPCs where the *non-muted* mode is not beneficial or which are not needed in the current and the upcoming hot spots are put into the fully-muted mode (i.e., added to the DPC_{FM} list), as they may be used rather late during the application execution flow.

Step 4: In the last step, DPC_{NM}, DPC_{VM}, DPC_{FM} lists are sent to the *Mute-Mode Controller* (Sect. 5.2.2) that issues the control signals (see Table 5.1) for the sleep transistors (for Logic and the Configuration SRAM).

Now, the '**determineDP$_{VM}$**' (from Fig. 5.12) function (which is used for computing the energy benefit of muting and for identifying a *virtually-muted* DPC) is discussed in the following.

5.4.3 Analyzing the Energy Benefit Function of Muting

Algorithm 5.2 shows the pseudo code for identifying one Data Path for virtually-muting (DP_{VM}) out of all virtually-muting candidates. The key inputs are: virtually-muting candidates ($\vec{s}$), Data Paths required for the upcoming hot spot ($\vec{n}$), set of CI Implementation Versions that are expected to be required for the upcoming hot spot (CI_{next}), expected execution time of the current hot spot ($tExec_{curr_HS}$), and a table of the CI weighting factors (ω_{CI}).

1. Function **determineDP$_{VM}$** $(\vec{t},\ \vec{s},\ \vec{n},\ CI_{next},\ \omega_{CI},\ tExec_{curr_HS})$
2. // **Input:** $\vec{t}$: temporary copy of currently required Data Paths, $\vec{s}$: virtually-muting candidates, $\vec{n}$: Data Paths required for the upcoming hot spot, CI_{next}: set of Implementation Versions for CIs that are expected to be required for the upcoming hot spot, $tExec_{curr_HS}$: expected execution time of the current hot spot, ω_{CI}: a table of the CI weighting factors (Figure 5.14)
3. // **Output:** DP$_{VM}$: *virtually-muted* Data Path
4. **BEGIN**
5. $bestE_{benefit} = 0$; // initialize the energy benefit
6. $DP_{VM} \leftarrow NULL$; // initialize virtually muted Data Paths
7. $\forall DP \in \vec{s}$ { // determine one Data Path from candidate list
8. $E_{ReconfBenefit} = P_{DPC_reconf} * T_{DPC_reconf}$;
9. $E_{LeakBenefit} = L_{Benefit}(\vec{t},\ DP) * (P_{DPC_leak} * |\vec{n}| + P_{cISA_leak})$;
10. $E_{LeakOverhead} = tExec_{curr_HS} * P_{VM_DPC_leak}$;
11. $dynE_{diff} = dynE_{diff}(\vec{t},\ DP)$; // see Eq. 5.9
12. $E_{Benefit} = E_{ReconfBenefit} + E_{LeakBenefit} - E_{LeakOverhead} + dynE_{diff}$;
13. $if(E_{benefit} > bestE_{benefit})$ {
14. $bestE_{benefit} = E_{benefit}$; $DP_{VM} = DP$;
15. }
16. }
17. return DP_{VM};
18. **END**

Algorithm 5.2 Pseudo Code for Finding a Data Path for Virtually-Muting Mode

Each Data Path in the candidate list $\vec{s}$ is evaluated for the energy benefit (lines 7–16). The Data Path that provides the highest energy benefit among all candidates $\vec{s}$ is then chosen as the one DP_{VM} (lines 13–15). There are four parts for the energy evaluation (line 12):

Reconfiguration Energy Benefit (E$_{RecBenefit}$, line 8): When a hot spot starts execution, its Data Paths are reconfigured into DPCs. In case a Data Path is still available after the execution of the current hot spot, one less reconfiguration is required for the upcoming hot spot. Thus, a DP_{VM} provides an energy benefit of one saved reconfiguration. Moreover, it also results in a latency improvement of one reconfiguration (approximately 0.63 ms) compared to the fully-muted DPC.

Leakage Energy Benefit (E$_{LeakBenefit}$, line 9): As the DP_{VM} will be available when the upcoming hot spot starts executing, the CIs of that hot spot may execute in a faster Implementation Version compared to the case when it is not available. This results in a performance improvement for the upcoming hot spot compared to the fully-muted Data Path (see Eq. 5.8). Each DP_{VM} may expedite multiple CIs, where each CI has a different weighting factor (ω_{CI}, see Sect. 5.4.4) depending upon its execution frequency and execution pattern in the hot spot. Faster execution of the upcoming hot spot will reduce the overall leakage energy of both the core processor and the reconfigurable fabric. Therefore, leakage savings are computed for each virtually-muting candidate by considering ω_{CI} of the CIs that are accelerated by this candidate.

$$L_{Benefit}(\vec{p}, DP) = \sum_{\vec{x} \in CI_{next}} \left(\begin{array}{l} \omega_{CI}(\vec{x}.CI()) * \\ \left(\begin{array}{l} \vec{x}.CI().Fastest(\vec{p}).Latency() - \\ \vec{x}.CI().Fastest(\vec{p} + DP).Latency() \end{array} \right) \end{array} \right) \quad (5.8)$$

Leakage Energy Overhead ($E_{LeakOverhea}$d, line 10): Leakage occurs in a virtually-muted DPC (due to the powered-on Configuration SRAM) for the whole duration of the current hot spot execution. Therefore, this overhead needs to be considered for the energy benefit function (line 12).

Dynamic Energy Difference ($dynE_{diff}$, line 11): Different Implementation Versions of a CI vary in their dynamic power and energy consumption. Therefore, a DP_{VM} may bring an energy benefit or overhead due to a different CI Implementation Version as shown in Eq. 5.9.

$$dynE_{diff}(\vec{p}, DP) = \sum_{\vec{x} \in CI_{next}} \left(\begin{array}{l} \omega_{CI}(\vec{x}.CI()) * \\ \left(\begin{array}{l} \vec{x}.CI().Fastest(\vec{p}).E() - \\ \vec{x}.CI().Fastest(\vec{p} + DP).E() \end{array} \right) \end{array} \right) \quad (5.9)$$

The computational complexity for calculating the energy benefit is $\mathcal{O}(numMaxDP_{VM} \cdot |\vec{S}|)$. Figure 5.13 shows that $|\vec{s}|$ is typically much smaller than the total number of Data Paths that fit onto the reconfigurable fabric at a certain time.

Note: the wakeup energy for virtually-muted and fully-muted DPC are 3.5 and 7.0 pWs (for the sleep transistor design of [Te06]), respectively. However, the energy for reconfiguring one DPC is 147 µWs, i.e., more than 10^6 times bigger (see details in Sect. 6.3, p. 164). Therefore, the DPC reactivation energy overhead is not included in the cost function, as it does not affect the muting decision.

5.4.4 Hot Spot Requirement Prediction: Computing Weighting Factors for CIs

Different CIs of a hot spot may have different execution patterns (see Fig. 5.14). These execution patterns depend upon the following three parameters:

- expected execution frequency of CIs
- the time from the start of a hot spot until their first execution
- the average time between two executions of the same CI

The expected execution frequency is predicted by a light-weight online monitoring scheme (Sect. 2.3.5, p. 35), while the other two parameters are obtained using an average case from offline-profiling. Depending upon the above three parameters a weighting factor (ω_{CI}) is computed for each CI. It represents the relative contribution of a CI (compared to other CIs) for the accelerated execution of a hot spot. To calculate ω_{CI}, the time line is partitioned into multiple slots, each equal to the reconfiguration time of a Data Path. Since the performance of a CI may only change

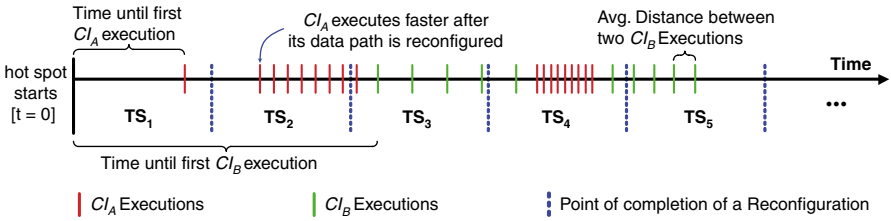


Fig. 5.14 Calculating the weighting factor for custom instructions w.r.t. the application context

after a reconfiguration is completed, the number of CI executions ($\#CIExec_{TSi}$) is computed for each time slot TS_i independently. Similarly, CIs executing in the earlier time slots have more weight than the later ones in the same hot spot (denoted by $Factor_{TSi}$). Considering there are 'n' time slots, ω_{CI} of a CI 'X' can be computed as shown in Eq. 5.10.

$$\omega_{CI}(X) = \sum_{i=1}^{n} (\#XExec_{TSi} * Factor_{TSi}) \tag{5.10}$$

An example can be found in the Motion Estimation hot spot of the H.264 encoder that requires two CIs: Sum of Absolute Differences (SAD) and Sum of Absolute Hadamard Transformed Differences (SATD). In the control flow, first SAD is required and then SATD, therefore the SAD CI has a higher importance for the earlier time slots. SATD becomes important in the later time slots. Let us assume that ME is the hot spot that is executed next. If only one DPC shall not be used in the current hot spot, but two virtually-muting candidates are available (containing Data Paths that are beneficial for the Motion Estimation), then it may be more beneficial to maintain the Data Path for SAD (i.e., setting it to virtually-muted mode) instead of SATD.

The complexity for online computation of the CI weighting factors is $\mathcal{O}(\#CIs \times n)$ per hot spot ('n' is the number of virtually-muting candidates, which is bound by #DPCs) with a memory overhead of $\mathcal{O}(\#CIs)$ to store the monitoring data (three 32-bit words per CI).

5.4.5 Evaluation of Selective Instruction Set Muting

Figure 5.15 illustrates the box-plot summary (over 408 different experiments of different performance and area constraints) of the benefit of using multiple CI muting modes and *Selective Instruction Set Muting*. The comparison is performed between the energy management scheme with one given muting mode (i.e., Fully Muting) and *Selective Instruction Set Muting* (where the decision of Fully or Virtually muting is evaluated at run time). Figure 5.15 shows that *Selective Instruction Set Muting* provides an energy benefit of up to 22% (on average 12%).

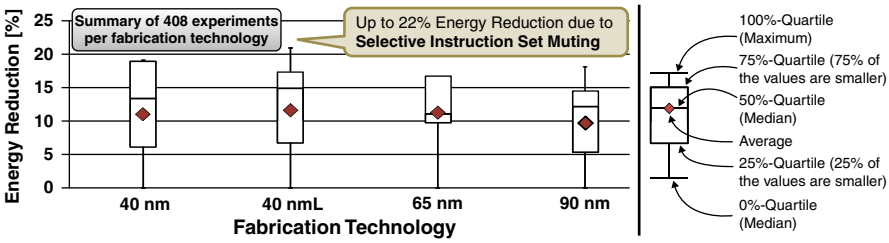


Fig. 5.15 Summary of energy benefit of using selective instruction set muting

5.4.5.1 Overhead of the Selective Instruction Set Muting Technique

The main compute-intensive part of the CI muting technique is the '**determine $\mathbf{DP_{VM}}$**' function (Algorithm 5.2) that determines the DP_{VM}. The energy benefit calculation (in lines 8–15, Algorithm 5.2) consumes 8.82 nWs, 11.56 nWs, 21.81 nWs for 40 nm, 65 nm, 90 nm, respectively. For 'n' virtually-muting candidates there are '$n(n-1)$' energy benefit calculations. For the H.264 video encoder application—in worst case—there are at most four candidates. The *overall energy overhead* of the CI muting technique is 105.89 nWs, 138.71 nWs, 261.68 nWs for 40 nm, 65 nm, 90 nm, respectively. However, the energy savings of the proposed technique are in multiples of mWs, i.e., more than 10^5 times bigger. Therefore, the energy overhead is negligible compared to the achieved energy savings.

The worst-case performance overhead of the CI muting technique is 1,356 cycles for the above-discussed experiments, which is negligible in comparison to the hot spot execution time (<<1%, depending on performance constraints). The CI muting technique is envisioned as executing on a Microblaze processor (a soft core provided by Xilinx) that along with monitoring and the reconfiguration controller requires only 5,564 slices in the current Xilinx Virtex-4-lx160 FPGA based prototype of the RISPP processor.

Note: the CI muting technique also requires a power-shutdown infrastructure in FPGAs to realize energy-aware adaptive computing that incurs additional area overhead. *Pika* [Te06] (a Xilinx low power FPGA research project) states an 8% area increase due to their power-shutdown infrastructure. However, they provide one sleep transistor per CLB, while in the envisioned power-shutdown infrastructure (see Sect. 5.2.2) two sleep transistors per DPC are required. Therefore, a much smaller area overhead is envisioned. Currently, such infrastructure is not available in today's commercial FPGAs. It would be far more energy efficient if FPGA vendors would provide a basic infrastructure that is necessary to exert the proposed CI muting technique.

5.4.6 Summary of Selective Instruction Set Muting

The *Selective Instruction Set Muting* technique uses various muting modes (requiring a power-shutdown infrastructure) that enable leakage energy reduction at the abstraction level of CIs. The CI muting technique selects one out of three muting

modes for each CI in the unused subset of CIs considering leakage as well as reconfigurable energy under run-time varying situations and constraints. The energy benefit function is evaluated at run time for each muting candidate which is a joint function of leakage, reconfiguration, and dynamic energy. Besides the execution length and requirements of the current and the upcoming hot spots, the weighting factors of different CIs (representing their relative contribution) in a hot spot are given as the input to compute the benefit of a particular muting mode. These weighting factors are determined by considering the expected execution frequency of CIs, the time from the start of a hot spot until their first execution, and the average time between two executions of the same CI. The experimental evaluation for various fabrication technology nodes corroborate the potential for far higher energy savings of dynamically reconfigurable processors which currently still suffer from a low efficiency as far as energy is concerned.

5.5 Summary of Adaptive Low-power Reconfigurable Processor Architecture

Once the energy requirements are reduced at the application/algorithm level, processor level energy management scheme needs to further trim the overall energy. First, different motivational scenarios were analyzed for different performance constraints of the video encoding application to highlight the energy reduction potential. Afterwards, a run-time energy management scheme is introduced. This scheme employs the novel concept of instruction set level muting that raises the abstraction level power-shutdown to the instruction set architecture. By doing so, it provides a much higher potential for energy reduction. Different muting modes are proposed considering a power-shutdown infrastructure that supports the independent shutdown control of the Logic and Configuration SRAM of the reconfigurable fabric. In the first step, the energy management scheme determines the energy minimizing instruction set while exploring the tradeoff related to the leakage, dynamic, and reconfiguration energy under run-time varying scenarios of performance and area constraints. It evaluates that for a given scenarios, whether it is beneficial to reconfigure more in order to meet the performance constraint, or it is beneficial to mute the CIs to reduce leakage. Afterwards, it determines the temporarily unused subset of CIs that are the possible muting candidates. Considering the area requirements of the currently executing and upcoming hot spots, the energy management scheme uses a *Selective Instruction Set Muting* technique to determines an appropriate muting mode for each CI. The benefits of determining the energy minimizing instruction set (i.e., exploiting the tradeoff of leakage, dynamic, and reconfiguration energy) and the *Selective Instruction Set Muting* are individually evaluated in Sect. 5.3.3 and 5.4.5, respectively. The evaluation is performed for different performance constraints and resolutions, while considering various fabrication technologies in order to demonstrate the technology independent efficiency of the proposed energy management scheme. This scheme is the key to realize an adaptive low-power reconfigurable processor architecture.

Chapter 6
Power Measurement of the Reconfigurable Processors

This chapter presents the details of building the power model described in Sect. 3.4 (p. 63). This power model is employed for power estimation, which is then used for the run-time adaptive energy management in reconfigurable processors (Chap. 5) and energy estimation for the adaptive low-power video encoding (Chap. 4). Section 6.1 presents the power measurement setup. Section 6.2 discusses the flow for creating the power model and parameter estimation. It further describes the procedure and different test cases for measuring the power of a complete Custom Instruction Implementation Version and different constituting components (i.e., computation, communication, and memory). Results for different measurements and estimated power are presented in this section. Section 6.3 presents the procedure and results for measuring the power of the reconfiguration process.

6.1 Power Measurement Setup

Figure 6.1a shows the power measurement environment which is developed in the scope of this monograph. It consists of a power supply board (Fig. 6.1b), two Agilent oscilloscopes (MSO6032A, and MSO6054A), a Xilinx Virtex-II v6000 FPGA (at 50 MHz and 1.5 V VCC_{INT}) based prototyping board (Fig. 6.1c), and a control program (running on a PC) for capturing the measurements from the oscilloscopes. Two oscilloscopes are used to simultaneously measure two different entities (e.g., FPGA and external memory). Precise measurement resistors of $R = 1.0\ \Omega$ are used. The voltage drop is measured across the two ends of a resistor using oscilloscopes as $V = V_2 - V_1$ and the current flowing through this resistor is obtained using $I = V/R$ formula. Here V_1 is the input voltage to the FPGA. The overall power consumption is $P = IV_1$.

Power supply board: To accurately measure the power consumption of FPGAs, a power supply board (Fig. 6.1b) is developed that supplies power to the FPGA prototyping board (Fig. 6.1c). This power board has several functions:

M. Shafique, J. Henkel, *Hardware/Software Architectures for Low-Power Embedded Multimedia Systems,*
DOI 10.1007/978-1-4419-9692-3_6, © Springer Science+Business Media LLC 2011

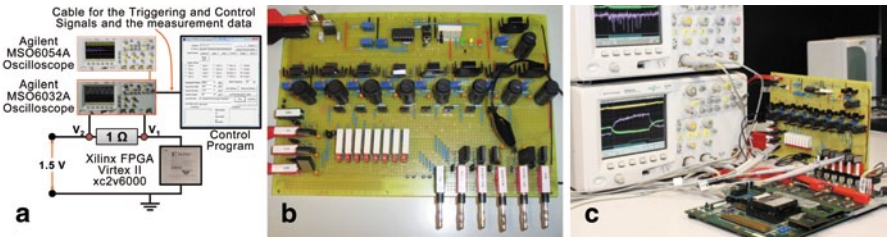


Fig. 6.1 **a** Measurement setup. **b** The in-house developed power supply board

- The voltage converters transform the input voltage (12 V) from the DC source to the desired voltages (VCC_{INT}, VCC_{AUX}, $VCC_{I/O}$ for the FPGA and voltages for peripherals)
- The fuses protect against the high current and the switching relays are activated only when all voltages are on
- The resistors measure the voltage drops at the output plugs

Agilent Oscilloscopes and VISA interface: The maximum sampling rate of MSO6032A and MSO6054A oscilloscopes are 2 and 4 GSamples/s, respectively, which is sufficient for the measurement. Both oscilloscopes support the VISA interface, which is an API for electronic control devices. It allows to send commands to the oscilloscopes and/or to receive data from the oscilloscopes. The Agilent IO Library (software) is used to create the control software (using Microsoft .Net Framework) for the oscilloscopes. Most of the functions performed with the knobs are also realized via VISA. This enables automation of the measurement process.

6.2 Measuring the Power of Custom Instructions

In this section, the flow for creating the power model is presented. Different test cases for measuring the power of different components are discussed. Afterwards, results for power measurement are presented.

6.2.1 Flow for Creating the Power Model

Figure 6.2 shows the flow to build the power model for CI Implementation Version. First, the HDL code of different Data Paths and Implementation Versions is synthesized with Xilinx Synthesis Technology (XST) [Xil10a]. Afterwards, Mapping, Place & Route, and Assemble are done using Xilinx Plan Ahead with the Early Access Partial Reconfiguration (EAPR) tool flow [Xil05] to obtain full and partial bitstreams.

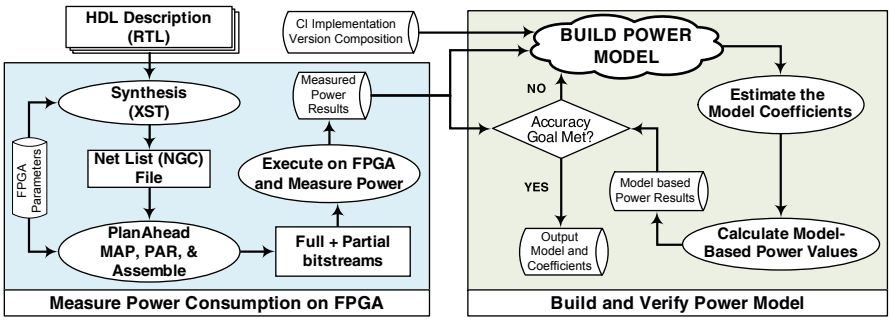


Fig. 6.2 Flow for creating the measurement-based power model

The partial bitstreams are then uploaded on the FPGA and power is measured for different Data Paths and Implementation Versions (see details in Sect. 6.2.2).

As discussed in Sect. 3.4 (p. 63), to estimate the dynamic power consumption of an executing CI Implementation Version (P_{CI_dyn}), the following needs to be considered:

- The types of Data Paths and how often they are executed.
- The number of write/read accesses on the local memory.
- The number of bus segments necessary for communicating the intermediate results.

Based on the analysis (Fig. 6.3), the dynamic power of a CI Implementation Version is modeled as:

$$P_{CI_dyn} = \alpha * P_{DataPath} + \beta * P_{SegBus} + \gamma * P_{Memory} + \delta \tag{6.1}$$

α, β, γ, δ are model coefficients. δ accounts for the measurement noise. $P_{DataPath}$, P_{SegBus}, and P_{Memory} are the average power consumption of a Data Path, a bus segment, and a single read or write operation, respectively (see details in Sect. 3.4). The model coefficients α, β, γ, and δ are estimated on Matlab using the Simulated Annealing algorithm ($T_{cool}=0.8$ T, $T_{stop}=10^{-12}$, $E_{allowable}=5\%$, and maximum consecutive rejections $=10^4$) and trained with a set of measured power values (see Table 6.4 in Sect. 6.2.3 for the final estimated values). T_{cool} is the cooling temperature that determines the temperature of the next iteration of the algorithm. T_{stop} is the tem-

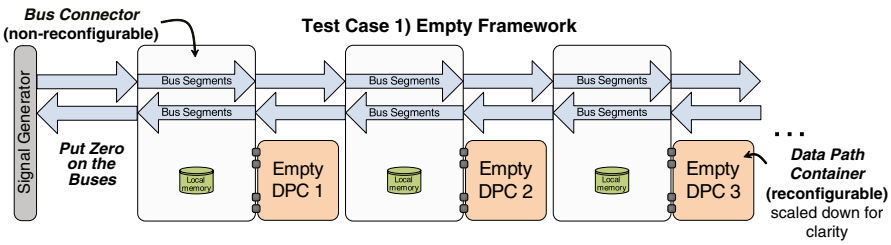


Fig. 6.3 Test case and setup for measuring the power of an idle (empty) framework

perature where the algorithm stops iterating and output the coefficients. Training is done by minimizing the difference error between the estimated (model-generated) and the measured power values of an Implementation Version for a given maximum error $E_{allowable}$ (5% in this case). The finalized model coefficients are then fed into the power model, which is then used for estimating the energy consumption of various CI Implementation Versions at run time.

Since the power can only be measured for the complete FPGA, in order to determine the power of the individual parameters (i.e., Data Path, segmented buses, and local memory) and the complete Implementation Version, several test cases are devised which are explained in the following.

6.2.2 Test Cases for Power Measurements

First, the computation- and communication infrastructure (Fig. 3.7, Sect. 3.4.1) is extended with a signal generator to realize a measurement framework. The power consumption of signal generator and leakage are surplus to the actual power consumption of an Implementation Version, thus, considered as the base offset (see test case 1, Fig. 6.3). The buses and the local memory consume energy only in case a toggle happens due to, e.g., a Data Path writes a new value into the local memory of its Bus Connector.

Test Case 1) Measuring power of the idle measurement framework ($P_{TestCase1}$): As discussed above, for a base offset, the power of the idle framework ($P_{TestCase1}$) is measured such that all DPCs contain blank bitstreams (i.e., a DPC is reconfigured to do nothing) and '0' value is transmitted on the bus segments. No writing to the local memory is performed. In this test case, the power consumption is mainly due to leakage and signal generator.

Test Case 2) Measuring power of a Data Path ($P_{DataPath}$): The dynamic power of a Data Path depends upon its type. To obtain the power of a Data Path, only one DPC is reconfigured to contain a particular Data Path and all other DPCs contain blank bitstreams (see Fig. 6.4). The output of the Data Path is not stored in the local memory. The average power of one Data Path is obtained as: $P_{DataPath} = P_{TestCase2} - P_{TestCase1}$. Numerous measurements are performed with varying

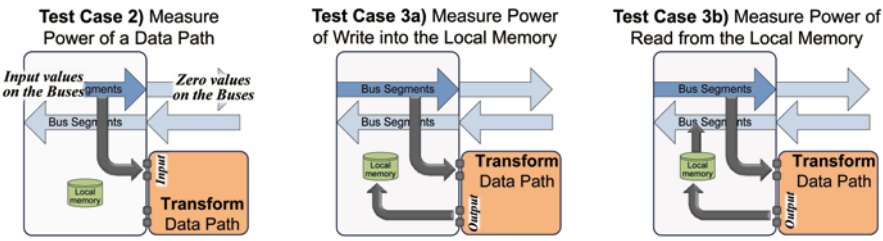


Fig. 6.4 Different test cases for measuring the power of different components of a custom instruction (CI) implementation version

stimulus to obtain the power of all Data Paths as used in the final experiments (see Table 6.2 in Sect. 6.2.3).

Test Case 3) Measuring power of the Local Memory (P_{Memory}): Two different tests are performed for measuring the power consumption of the local memory:

a. *Write to the local memory* (see Fig. 6.4): this test extends the test case 2 by writing the output of a Data Path into the local memory. The power of a write operation into the local memory is: $P_{WriteMem} = P_{TestCase3a} - P_{TestCase2}$
b. *Read from the local memory* (see Fig. 6.4): This test extends the test case 3a by writing the content of the local memory to a bus segment. The power of a read operation from the local memory is: $P_{ReadMem} = P_{TestCase3b} - P_{TestCase3a}$

Note, reading from a local memory causes toggles in the multiplexers of the Bus Connector and in the bus segment to which its contents are written. The power due to both types of toggles is considered as the power of one read from the local memory.

Test Case 4) Measuring power of a Bus Segment (P_{SegBus}): As discussed in Sect. 3.4, the communication power depends upon the number of bus segments, which directly depends on the relative placement of the communicating Data Paths on the reconfigurable fabric. In this test, the values from the local memory are written to one particular bus segment and '0' value is written to the other bus segments towards right to avoid toggles. To get the power of a bus segment, the test case 3b is extended such that the placement of a particular Data Path is shifted towards one DPC right and the difference of two measurements gives the power of one bus segment as: $P_{BusL2R} = (P_{TestCase3b})_{DPC_N+1} - (P_{TestCase3b})_{DPC_N}$. The average of several tests for different Data Path placements provides a more stable power value for a bus segment.

The measured power of Data Paths, local memory, and bus segments are used in the power model to estimate the power consumption of a CI Implementation Version. However, to tune the coefficients of the model and for model verification, several experiments were performed to measure the complete power of different Implementation Versions.

Test Case 5) Measuring power of a complete CI Implementation Version ($P_{CI_ImpVersion}$): The bitstreams of all Data Paths of an Implementation Version are reconfigured and the control signals determine the communication between these Data Paths. The power is measured for the complete execution of the Implementation Version. For an Implementation Version with latency L, the per cycle average power is computed as: $P_{CI_ImpVersion} = P_{TestCase5}/L - P_{TestCase1}$. Since there are several Data Paths processing in parallel, there might exist many Data Path placement combinations, i.e., a Data Path can be placed in one out of many DPCs and with increasing number of Data Paths the number of possible placement combination increases. An example of two Transform Data Paths is shown in Table 6.1 that was used in the measurement experiments. Each combination has different power consumption due to different amount of bus segments used. Therefore, different measured power values are used to tune the corresponding estimated power values for the same CI Implementation Version.

Table 6.1 Different placement combinations of two transform Data Paths for power measurement

1st Transform Data Path at	2nd Transform Data Path at				
DPC 2	DPC 3	DPC 4	DPC 5	DPC 6	DPC 7
DPC 3		DPC 4	DPC 5	DPC 6	DPC 7
DPC 4			DPC 5	DPC 6	DPC 7
DPC 5				DPC 6	DPC 7
DPC 6					DPC 7

6.2.3 Results for Power Measurement and Estimation

Table 6.2 shows the measured power results for different Data Paths and Implementation Versions. These power values are used for tuning the model coefficients ($\alpha, \beta, \gamma, \delta$, see Table 6.3 for finalized values) and verification of the power model (Sect. 3.4, p. 63). The reconfiguration power and time is obtained by measurements (see Sect. 6.3 for details of the power measurement procedure). Note, differently sized Data Paths may require different reconfiguration time due to their varying bitstream sizes.

Table 6.4 presents the power consumption and latencies of different Implementation Versions for two different cases of total DPCs at 40 nm (Virtex-6) and 65 nm (Virtex-5) technologies. It is notable that the power consumption of Implementation Versions on 40 nm is lesser than that on 65 nm due to the low-power architectural improvements in Virtex-6 [Kle10].

Table 6.2 Measured power results for various data paths & HT4×4 implementation versions

Data Path	Power [mW]	Implementation version: HT4×4 (Repack in DPC0)	Power [mW]
Clip3	15.9	Transform in DPC2*	178.9
PointFilter	55.4	Transform in DPC4*	180.8
LF_4	57.9	Transform in DPC6*	185.1
Cond	13.1	Bus_Power (P_{Bus})**	3.4
CollapseAdd	19.7	Mem_Power (P_{RW})	28.3
SADrow	28.2		
SAV	25.1		
Transform	64.9		
QuadSub	24.1		
Repack	14.4		

*Showing the effect of changing communication requirements, **power for a single toggling bus segment; many bus segments are used for communication to realize an Implementation Version

Table 6.3 Parameters of power model for the CI implementation versions

Attribute	Value
α	1.2387
β	0.4699
δ	0.0911
γ	0.8165

Table 6.4 Power consumption and latencies of different implementation versions (using different amount of DPCs) for various custom instructions for 65 nm and 40 nm technologies

Functional block	Custom instruction	Data paths	Using 4 DPCs			Using 20 DPCs		
			Latency [cycles]	Power [mW]		Latency [cycles]	Power [mW]	
				40 nm	65 nm		40 nm	65 nm
Motion Estimation (ME)	SAD16×16	SADrow	68	47.35	67.21	41	59.76	84.39
	SATD4×4	QuadSub, Transform, Repack, SAV	93	13.87	19.68	29	57.24	81.72
Motion Compensation (MC)	MC_Hz_4	PointFilter, Repack, Clip3	10	52.00	74.00	10	58.75	83.75
Intra Prediction (IPred)	IPred_HDC	CollapseAdd, Repack	130	7.46	10.54	130	7.46	10.54
	IPred_VDC	CollapseAdd, Repack	53	4.00	5.28	53	4.00	5.28
(Inverse) Transform	(I)DCT4×4	Transform, Repack, (QuadSub)	102	9.51	13.43	20	62.00	88.00
	(I)HT_2×2	Transform	2	50.00	70.00	2	50.00	70.00
	(I)HT_4×4	Transform, Repack	16	60.00	85.63	15	64.67	92.00
In-loop Deblocking Filter (LF)	LF_BS4	Cond, LF_4	10	52.00	74.00	10	52.00	74.00

6.3 Measuring the Power of the Reconfiguration Process

The smallest reconfigurable unit in a Xilinx FPGA (Virtex family) is a so-called frame [Xil05]. A Data Path Containers (DPCs) is composed of multiple frames. Due to technological reasons, DPCs are typically of rectangular shapes. The power measurements are performed using a Virtex-II FPGA where a frame covers the complete height of the FPGA. Therefore, the DPC consists of multiple Configurable Logic Block (CLB) columns[1]. Before performing a reconfiguration, the configuration data of the corresponding frames is read. Afterwards, the parts of the configuration data corresponding to the region(s) under reconfiguration are modified accordingly. In the last step, the frames are written back. Doing so assures the consistency of the static part and the other un-altered reconfigurable parts within the frames, when compared to their previous configuration.

For reconfiguring the DPCs, a dedicated Reconfiguration Controller IP core is developed ([Bau09]), which is connected to the MicroBlaze. It reads the partial bitstreams from an external EEPROM (KFG5616 32 MB OneNAND from Samsung [Sam05]), buffers data in a FIFO for burst transfer, and streams the data to the Internal Configuration Access Port (ICAP) of the FPGA for reconfiguration. Figure 6.5 shows an abstract diagram of this connection. Buffering is done because that the maximum data that can be continuously read in the burst read mode is one 1 KB memory page, which is much smaller than the size of a bitstream. As a result, the bitstream cannot be completely sent to ICAP in a burst, which does not comply with the input requirements of the ICAP. The MicroBlaze starts the reconfiguration by providing the starting address and length of a particular Data Path's bitstream in the external EEPROM. A checksum of the reconfigured bitstream is returned. After the MicroBlaze triggered a reconfiguration, the FIFO is filled with data from the EEPROM (Fig. 6.5). When the FIFO contains sufficient data to assure a continuous 50 MB/s burst to the ICAP of the FPGA, the data is sent from the FIFO to the ICAP port. The ICAP is operated at 50 MHz (same frequency as for the core pipeline and the MicroBlaze). The EEPROM delivers the remaining parts to the FIFO in parallel to the running reconfiguration. Due to the initial buffering (until sufficient data is available to perform a continuous 50 MB/s burst afterwards), the effective reconfiguration bandwidth for the whole process is 36 MB/s.

To measure the power consumption of the EEPROM and the reconfiguration via ICAP, a Data Path bitstream of 40 KB size is transferred from EEPROM to ICAP

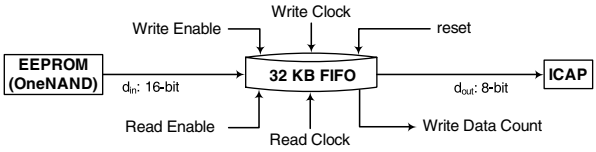


Fig. 6.5 Connection of FIFO between EEPROM and ICAP

[1] In the latest Virtex families (Virtex-4 and later), a frame does not span over the full FPGA height.

through the Reconfiguration Controller. Both oscilloscopes are used to simultaneously measure the power consumption of EEPROM and the FPGA.

6.3.1 Power Consumption of EEPROM

Figure 6.6a shows the measured signals while loading the bitstream of a Data Path from EEPROM to the FPGA. The V_2 and V_1 analog signals illustrate the voltage before and after a measurement resistor (1 Ω), respectively. This measurement resistor is connected with the EEPROM in series and thereby its current goes further to the EEPROM. As a result, the value of current through the EEPROM and the measurement resistor is identical. The analog signal indicating the voltage drop across the measurement resistor is computed by taking the difference between the V_2 and V_1 signals. The average value of current flowing through the measurement resistor is: $I = 25\ mV/1\ \Omega = 25\ mA$. The input voltage to the EEPROM (i.e., after the resistor) is measured as $3.26\ V$. Therefore, the corresponding power consumption is: $P = 3.26\ V * 25\ mA = 81.6\ mW$.

6.3.2 Power Consumption of the Reconfiguration via ICAP

Figure 6.6b illustrates the measured signals for transferring the bitstream of a Data Path to the ICAP and performing the corresponding reconfiguration. The analog signals indicate the voltage drop of the measurement resistor, which is connected with the FPGA in series. Moreover, the digital signals for the write enable of the FIFO and the write enable of the ICAP are also shown in Fig. 6.6b. The time for loading and reading bitstream from EEPROM is indicated by the write enable of FIFO.

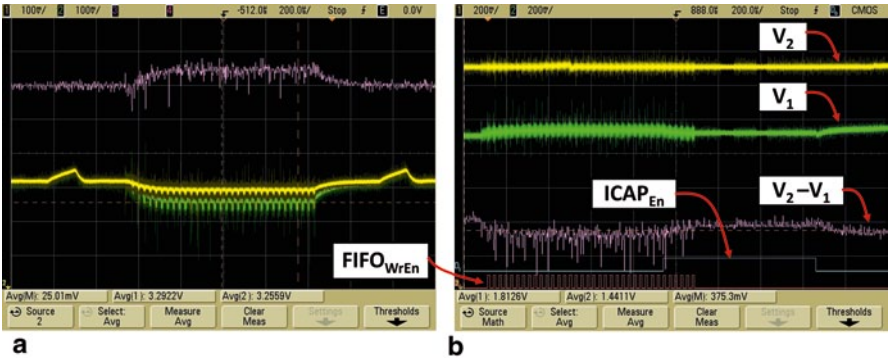


Fig. 6.6 **a** EEPROM voltage drop while loading one Data Path Bitstream from EEPROM to FPGA. **b** VCC_{INT} voltage drop for transferring one Data Path bitstream to ICAP and performing the corresponding reconfiguration

The ICAP write enable becomes high after sufficient data is loaded in the FIFO and it indicates the operating time of ICAP, namely the time for reconfiguration. The average voltage drop of the measurement resistor is *375.2 mV* and the average current flowing through the measurement resistor is: $I = 375.2\ mV/1\ \Omega = 375.2\ mA$. The input voltage to the FPGA is measured as *1.43 V*, thus the power consumption is: $P = 1.43\ V * 375.2\ mA = 536.5\ mW$. The power of the idle setup is *382 mW*; thus, the actual reconfiguration power (P_{DPC_reconf}) is *236 mW* ($536.5 + 81.5 - 382 = 236$).

6.4 Summary of the Power Measurement of the Reconfigurable Processors

The power model proposed in this monograph (see Sect. 3.4) is based on power measurements. To perform power measurements, a setup is designed and implemented that consists of a power supply board, two Agilent oscilloscopes, a Xilinx Virtex-II v6000 FPGA based prototyping board, and a control program for capturing the measurements from the oscilloscopes. To build a power model, the HDL code of different Data Paths and Implementation Versions was synthesized and implemented using the Xilinx tool chain with the Early Access Partial Reconfiguration tool flow. The resulting bitstreams were uploaded to the FPGA board and power consumption was measured for various input stimuli. In order to measure the power consumption of different components (i.e., Data Paths, communicating buses, local memory accesses), various test cases were devised. The measurements are used to train the model and evaluate the estimation accuracy. The reconfiguration power is measured by measuring the power consumption of the EEPROM (containing the configuration bitstreams) and the Internal Configuration Access Port (ICAP). The Data Path bitstream is transferred from the EEPROM to the ICAP through a reconfiguration controller. The reconfiguration energy mainly depends upon the amount of configuration data and the reconfiguration bandwidth. To simultaneously measure the power of both EEPROM and the ICAP, two oscilloscopes were deployed. The reconfiguration power measured is in the range of *236 mW*.

Chapter 7
Benchmarks and Results

In this chapter, the adaptive low-power application and processor architectures are benchmarked. The evaluation and analysis of the individual parts of the proposed adaptive low-power application and processor architecture are already presented in Chaps. 4 and 5, respectively. The first section will provide benchmarks for different algorithms at the Mode Decision and Motion Estimation levels for realizing adaptive low-power video coding. These algorithms are compared with different state-of-the-art fast and adaptive approaches. This section additionally provides the comparison with the *exhaustive Rate Distortion Optimized Mode Decision* (RDO-MD) and exhaustive search algorithms to benchmark against the optimal quality as it is typically done by the related work, too. The second section benchmarks the adaptive low-power reconfigurable processor architecture (with energy management scheme) against state-of-the-art reconfigurable processor. The following two different types of dynamically reconfigurable processor architectures are considered for comparison.

1. Dynamically reconfigurable processors that target at *maximizing the performance* for a given amount of reconfigurable fabric area. Kindly see Sect. 7.3.1 for comparison with architectures supporting *monolithic* Custom Instructions (CIs) and see Sect. 7.3.2 for comparison with architectures supporting *modular* CIs.
2. Dynamically reconfigurable processors *with the support of hardware-oriented shutdown* techniques for leakage power reduction *monolithic* CIs (see comparison in Sect. 7.3.3).

Both of these approaches are provided with the same set of low-power Custom Instructions (CIs) and Data Paths (see Sect. 4.2, p. 80) as they share the same CI model for accelerating the applications. For processor level benchmarking the complete H.264 video encoder is used, as the intricate processing behavior of the H.264 video encoder represents the increasing complexity of modern embedded multimedia applications. It exhibits different computational intensive parts (SAD for Motion Estimation, DCT, CAVLC, filters for Motion Compensation and Deblocking, etc.) that supersede the complexity of conventional benchmark applications in the benchmark suites (like *MiBench* [GRE+01] and *MediaBench* [LPMS97]). The evaluation is performed for various fabrication technologies ranging from 40 nm

M. Shafique, J. Henkel, *Hardware/Software Architectures for Low-Power Embedded Multimedia Systems,*
DOI 10.1007/978-1-4419-9692-3_7, © Springer Science+Business Media LLC 2011

to 150 nm (considering the reconfigurable fabric structure of Xilinx FPGAs, i.e., Virtex-II, 4, 5, 6).

7.1 Simulation Conditions and Fairness of the Comparison

For energy estimation, simulations are performed using the RISPP simulator extended with the proposed energy-management system (see Chap. 5) and power estimation methodology (see Sect. 3.4 and Chap. 6). Kindly refer to the Appendix B for further details on the simulation environment for the adaptive low-power reconfigurable processors. For video quality testing, algorithms' functional verification, and checking the standard compliance, the simulations are performed using the JM13.2 software of H.264 video encoder [JVT10]. For evaluation and validation of several QCIF and CIF video sequences (low to fast motion) [Ari08; Xip10] are encoded with different QPs (12, 16, 20, 24, 28, 32, 36, and 40) and bit rates. Common test conditions are: IPPP GOP (Group of Pictures) type, 1 reference frame, search range = 16. Note: all energy saving results include the overhead of the corresponding algorithm and the computation of video statistics. Moreover, all results include the leakage and dynamic energy consumption.

State-of-the-art techniques for different functional blocks of the H.264 video encoder were carefully implemented and simulated and their results were verified with their corresponding papers. Kindly note that several implementations were already available in the JM13.2 software.

Comparing state-of-the-art with the processor-level contribution is not straightforward. As discussed in Chap. 5, state-of-the-art approaches employ hardware-oriented shutdown, which has a different abstraction level for shutdown decision compared to the proposed energy management scheme with the *Selective Instruction Set Muting* technique (i.e., an instruction set oriented shutdown). In order to provide a fair comparison, the hardware-oriented shutdown concepts of [Ge04] and [MM05][1] (i.e., predetermining the components of DPCs that can be shutdown at run time) were deployed carefully to realize two predetermined muting modes as follows:

a. **Predetermined Virtually Muting (Pre-VM)** technique based on the hardware-oriented shutdown of [Ge04]: it always puts the temporarily unused CIs into virtually-muting mode as the hardware-oriented shutdown of [Ge04] only supports switching-off of Logic and it always keeps the Configuration SRAM powered-on.
b. **Predetermined Fully Muting (Pre-FM)** technique based on the hardware-oriented shutdown of [MM05]: it always puts the temporarily unused CIs into fully-muting mode as the hardware-oriented shutdown of [MM05] supports the combined switching-off of both Logic and Configuration SRAM.

[1] These approaches [Ge04; MM05] are considered for the comparison as they are the closest to the proposed technique in terms of shutdown options for different components of the fabric, thus representing a fair comparison.

In the following, the energy consumption comparison of the above-mentioned techniques will be presented for given performance constraints, such that in these particular scenarios, all techniques meet their performance constraints. Different performance constraints correspond to changing application contexts (like a change in the frame rate of the video coding). For further fairness of comparison, the same set of low-power CIs and Data Paths (see Sect. 4.2) is provided to all techniques. Therefore, the results reflect solely the impact when applying the proposed energy management scheme to realize an adaptive low-power reconfigurable processor architecture.

7.2 Adaptive Low-power Application Architecture

In this section the *Adaptive Computational Complexity Reduction Scheme* (see Sect. 4.4, p. 95) and the *energy-aware Motion Estimation with the integrated energy-budgeting scheme* (see Sect. 4.5, p. 104) are compared individually with different state-of-the-art to demonstrate their individual energy benefit. Note, these two schemes are the key contribution of this monograph to realize an application architecture for *adaptive low-power video coding*. In Sect. 7.3, the complete H.264 video encoder will be deployed as an application to evaluate the *adaptive low-power reconfigurable processor architecture*.

7.2.1 Comparing Complexity Reduction Scheme to State-of-the-art and the Exhaustive RDO-MD

The proposed Adaptive Computational Complexity Reduction Scheme (ACCoReS, Sect. 4.4) is compared to several state-of-the-art fast RDO-MD schemes for quality (a positive ΔPSNR shows PSNR loss) and energy reduction using similar coding conditions. Figure 7.1 shows that, compared to state-of-the-art approaches [JC04; KC07; PC08; SN06; Yu04], ACCoReS achieves up to 82% (average 56%) higher energy reduction at the cost of an average PSNR loss of 0.66 dB. The maximum energy saving of ACCoReS is achieved for the *Paris* sequence when compared to the approaches of [JC04] and [Yu04]. It is due to the fact that the approach of [JC04] processes on average five out of seven block types, while the approach of [Yu04] only considers mode correlation in the previous video frame. The significant energy saving of ACCoReS comes from the Prognostic Early Mode Exclusion and Hierarchical Fast Mode Prediction that curtails the set of candidate coding modes for further evaluation. This energy saving comes at the cost of a PSNR loss of up to 1.3 dB (average 0.66 dB). However, this loss is mainly at the PSNR value of more than 40 dB. It is worthy to note that, subjectively a loss of 1 dB is hard to be noticed by the human eye in case the overall PSNR is more than 40–45 dB [GW02; Pra01; WOZ02].

Figures 7.2 and 7.3 show the energy savings and the quality loss of ACCoReS compared to the *exhaustive RDO-MD* (that provides the optimal quality) for vari-

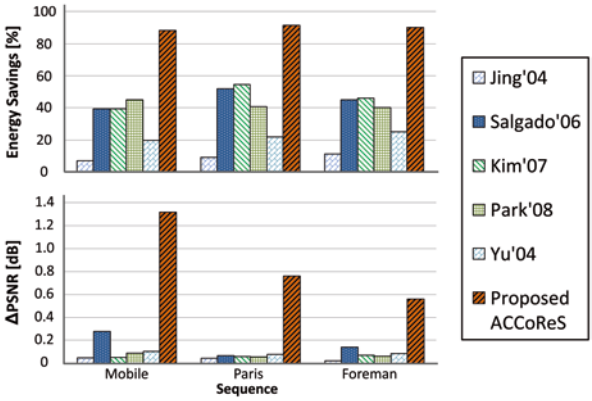


Fig. 7.1 Comparing the energy savings and quality loss of the ACCoReS with several state-of-the-art fast mode decision schemes

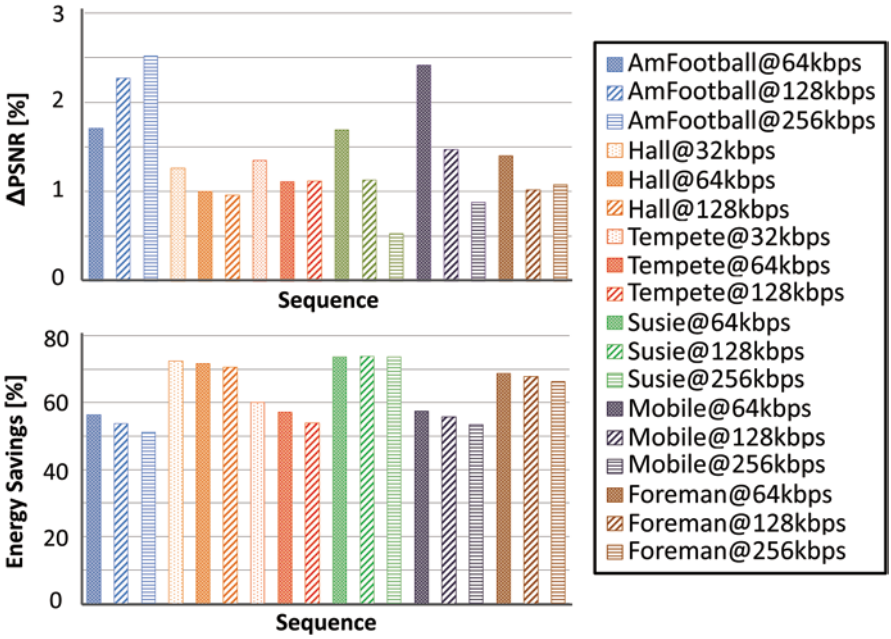


Fig. 7.2 Energy savings and quality loss of the ACCoReS compared to the exhaustive RDO-MD for CIF resolution video sequences

ous bitrates. The quality loss for CIF sequences is less than 2.5%, while for QCIF sequences the quality loss is less than 5%. On average, the quality loss of ACCoReS is 1.38% and 1% for CIF and QCIF sequences, respectively. However, the average energy savings of ACCoReS are 63.27% and 66.74% for CIF and QCIF sequences, respectively. Figure 7.2 shows that the highest energy savings (more than 70%) are achieved for *Susie* and *Hall* sequences which are slow motion sequences. In contrast, the lowest energy savings are achieved for the *American Football* sequence which a fast motion sequence. Still, in this case, the energy savings are more than

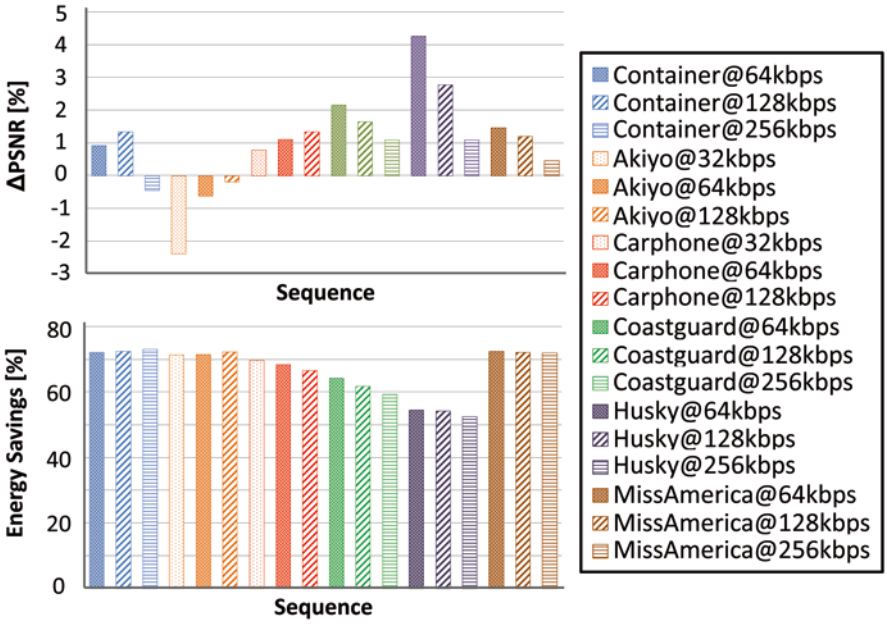


Fig. 7.3 Energy savings and quality loss of the ACCoReS compared to the exhaustive RDO-MD for QCIF resolution video sequences

50%. It is worthy to note that, in some QCIF cases (like *Akiyo* and *Container*), ACCoReS achieves a better PSNR. It is mainly due to the Macroblock based prioritization step in the Rate Control (see Appendix A.2).

Figure 7.4 shows the Rate-Distortion (R-D) curves of ACCoReS and the *exhaustive RDO-MD*. It demonstrates the quality loss of ACCoReS compared to the optimal video quality at a certain given bitrate. It can be noticed in Fig. 7.4 that for slow to medium motion sequences (*Akiyo*, *Container*, and *Susie*) the achieved quality of ACCoReS is close to that of the *exhaustive RDO-MD*. However, for the fast motion sequence (*American Football*), ACCoReS suffers from a PSNR loss of up to 5.7%. It is worthy to note that this PSNR loss occurs at the PSNR values of more than 40–45 dB. As discussed above, these discrepancies are insignificant as it is hard for a human eye to subjectively recognize a PSNR loss for the encoded videos having PSNR above 40–45 dB [GW02; Pra01; WOZ02]. Mostly, ACCoReS achieves a much closer R-D as compared to *exhaustive RDO-MD*.

Figure 7.5 presents the evaluation of the proposed ACCoReS for a power-aware test on a laptop (Intel Core2Duo T5500, 1.66 GHz) using its battery status. The *Mobile* test video sequence is encoded at 512 kbps@30 fps. Depending upon the current battery status, different states of the ACCoReS (i.e., Prognostic Early Mode Exclusion, Sect. 4.4.1, and Hierarchical Fast Mode Prediction, Sect. 4.4.2) are activated or deactivated. First the battery of the laptop is fully charged and then disconnected from the power outlet for the encoding test. Note after every 300 frames, there is a scene cut, where a sudden PSNR drop occurs, although relatively more modes are evaluated in this case. Kindly note that most of the evaluated modes are

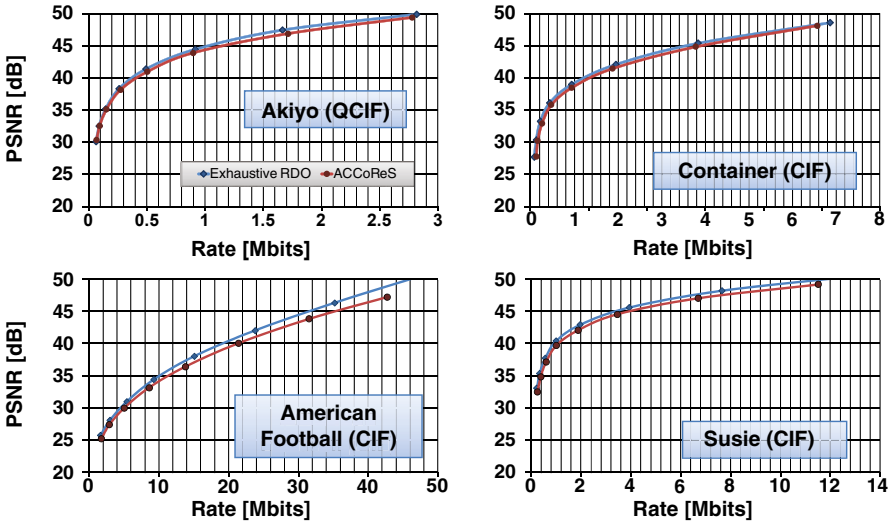


Fig. 7.4 Comparing the rate distortion curves for QCIF and CIF sequences

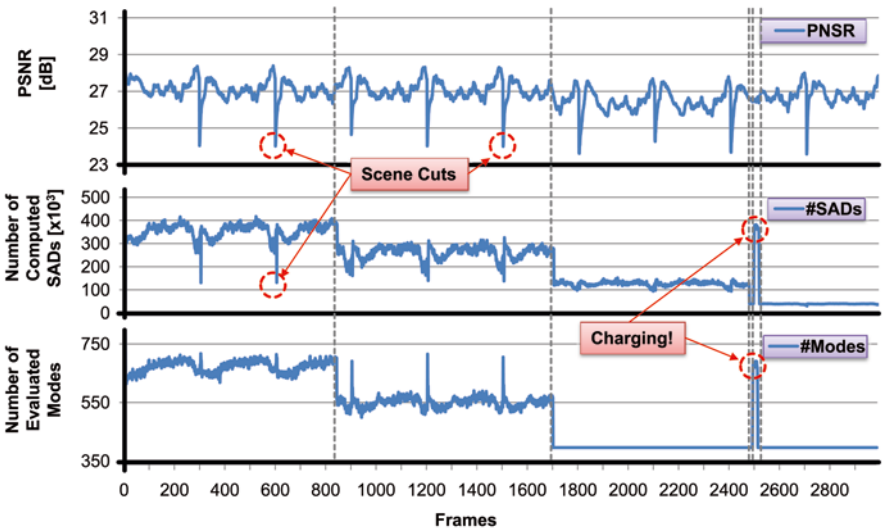


Fig. 7.5 Power test with a real battery using Mobile sequence

Intra modes, thus the number of SAD computations are dropped also in the corresponding frames (see dotted circles in Fig. 7.5 corresponding to the scene cuts). The PSNR drop is mainly due to the high amount of prediction error even in case of Intra Prediction (i.e., I-MB modes) and for a given bitrate this corresponds to a PSNR loss. As the battery level decreases to less than 25%, only aggressive exclusions are performed and only one mode per Macroblock is evaluated. When the battery status reaches 10%, the battery is charged again for a short time (see Fig. 7.5) to demon-

strate the quick response of ACCoReS. In this case ACCoReS switches to a high quality mode where relaxed decisions are taken and the Hierarchical Fast Mode Prediction is deactivated to maintain a good video quality. This experiment demonstrates the quality versus energy consumption tradeoff of the ACCoReS scheme.

7.2.2 Comparing the Energy-Aware Motion Estimation with Integrated Energy Budgeting Scheme to State-of-the-art

In the following, the proposed *energy-aware Motion Estimation with the integrated energy-budgeting scheme* (see Sect. 4.5, p. 104) is compared to three benchmark Motion Estimators, i.e., *Unsymmetrical-cross Multi-Hexagon-grid Search* (UMHexagonS) [CZH02], *simple UMHexagonS* [YZLS05], and *Enhanced Predictive Zonal Search* (EPZS) [Tou02] for energy and video quality (PSNR) using various video sequences [Ari08; Xip10]. The following experiments are performed using a bitrate of 256 kbps.

Summary of Two Battery States: Figure 7.6 shows the summary of the energy saving of the proposed energy-aware Motion Estimation compared to the state-of-the-art fast adaptive Motion Estimators. The box plot shows the summary of 96 experiments with 12 sequences and two different cases of initial battery states (1 Ws and 500 µWs). Figure 7.6 shows that the energy-aware Motion Estimation achieves an energy benefit of up to 93%, 93%, 90% (average 88%, 88%, 77%) for UMHexagonS [CZH02], UMHexagonS-Simple [YZLS05], EPZS [Tou02], respectively. Even in the worst case, the energy-aware Motion Estimation provides 65% energy savings compared to EPZS. The significant energy savings are mainly due to the switching between multiple *Energy-Quality Classes* depending upon the spatial and temporal properties of different Macroblocks (MBs). Although the battery is full, the energy is not wasted in case of homogeneous and slow moving MBs, and they are allocated less energy quota for the Motion Estimation process. Due to the slow motion properties, the reduced Motion Estimation effort and pixel decimation in SAD computation still provides a sufficiently good match of the current MB in the reference frame. Therefore, the incurred quality loss for sequences with homogeneous and slow moving MBs (see *Carphone*, *Clair*, *Akiyo* in Fig. 7.7) is insignificant.

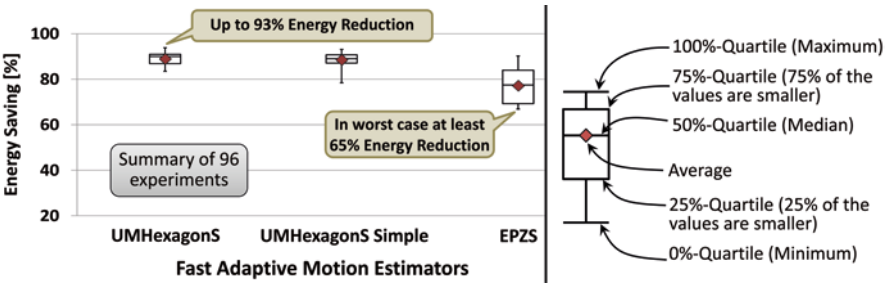


Fig. 7.6 Summary of energy savings of the enBudget scheme compared to various fast adaptive motion estimation schemes

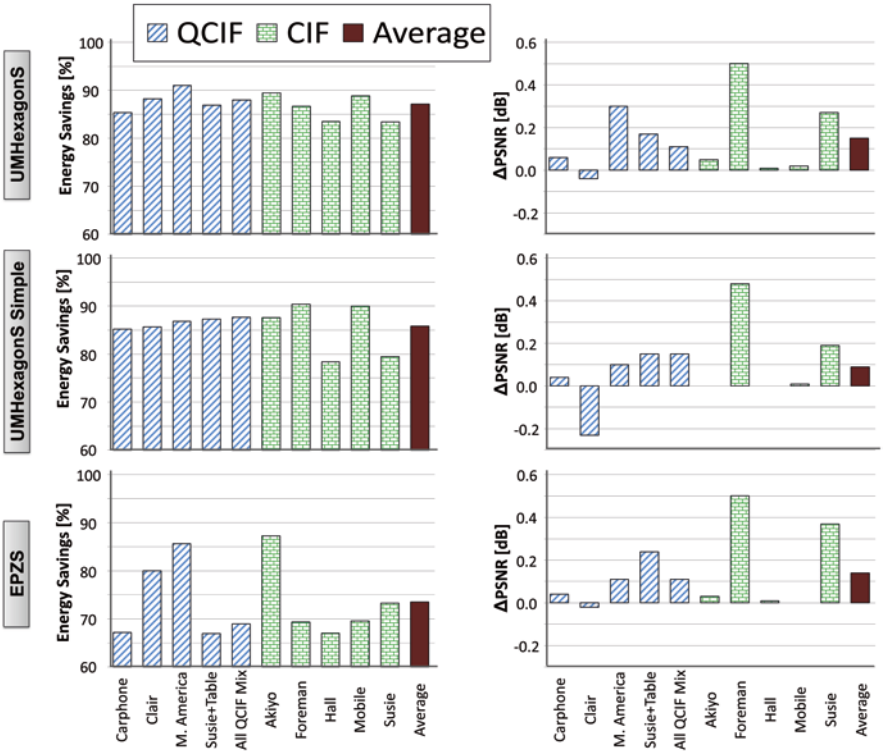


Fig. 7.7 Comparing energy saving and PSNR loss of the proposed energy-aware motion estimation and the enBudget scheme with various fast adaptive motion estimators. (* negative PSNR loss actually shows the PSNR gain of the scheme)

Details for One Battery State (1 Ws): Figure 7.7 presents the detailed energy savings and PSNR loss of the energy-aware Motion Estimation compared with state-of-the-art fast adaptive Motion Estimators. The major energy saving comes for low motion sequences (*MissAmerica*, *Akiyo*, *Clair*, and *Mobile*). Note, all the comparison partners are also adaptive Motion Estimators, thus they also react to different motion properties using their early termination criteria. However, as discussed earlier, the proposed energy-aware Motion Estimation scheme achieves higher energy saving due to the *Energy-Quality Classes* and video properties based downgrade/upgrade of *Energy-Quality Classes*. In several cases, the energy-aware Motion Estimation even achieves higher PSNR due to adaptive energy budget allocation, thus more Motion Estimation effort is spent for selective MBs (high texture, high motion). In this case, the overall average PSNR is improved. This is visible especially for low motion sequences (*Clair* and *Akiyo*). Compared to the *Full Search* Motion Estimator, the energy-aware Motion Estimation provides an energy saving of up to 99% at the cost of an average PSNR loss of 0.29 dB, which is visually insignificant. However, as discussed in Chap. 2 (see Sect. 2.2.3, p. 24), typically the *Full Search* is only used for video quality comparison.

7.3 Adaptive Low-power Processor Architecture

In this section the *Run-Time Adaptive Energy Management Scheme* (Sect. 5.2, p. 126) is benchmarked which is required to realize an adaptive low-power processor architecture. It determines an *energy minimizing instruction set* (see Sect. 5.3, p. 133) and employs the novel concept of *Selective Instruction Set Muting* (see Sect. 5.4, p. 146). In the following the adaptive energy management scheme is compared to different state-of-the-art, considering both with and without *Selective Instruction Set Muting* in order to demonstrate the individual energy benefit of the *energy minimizing instruction set* and *Selective Instruction Set Muting*.

7.3.1 Comparing the Adaptive Energy Management Scheme (Without Selective Instruction Set Muting) to RISPP with Performance Maximization [BSH08c]

Figure 7.8 shows the comparison between RISPP (with a performance-maximizing scheme (*PerfMax*)[2], performance is the main optimization goal [BSH08c]) and the proposed adaptive energy management scheme when executing the H.264 video encoder at three different performance constraints. In this case, the energy management scheme determines an energy minimizing instruction set (that minimizes the

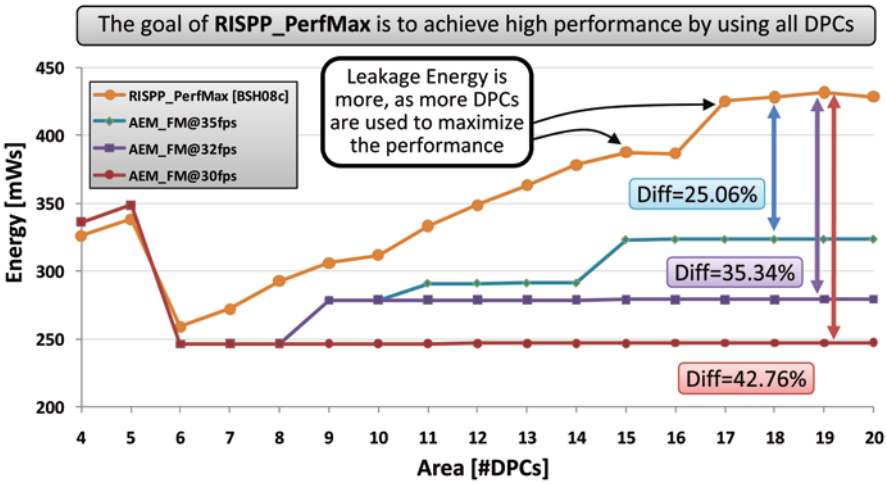


Fig. 7.8 Energy comparison of the AEM_FM and RISPP_PerfMax schemes for 65 nm

[2] Note: the power-shutdown is disabled in case of *RISPP_PerfMax*, as switched-off DPCs lose their configuration data and this typically degrades the performance, e.g., when the same Data Paths are needed soon afterwards. Comparison with *RISPP_PerfMax* also demonstrates the performance loss of *AEM_FM* compared to the peak performance.

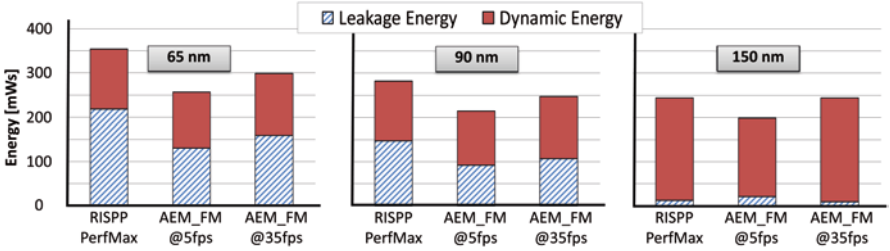


Fig. 7.9 Average energy comparison of the AEM_FM and RISPP_PerfMax for three technologies

energy for a given performance constraint) without *Selective Instruction Set Muting* using various muting modes. Rather the muting mode '*Fully Muted CIs*' is considered for all unused CIs. Here the purpose is to benchmark the energy management scheme with the *energy minimizing instruction set and CI-level shutdown in Fully-Muted mode* (denoted as *AEM_FM* for the ease of representation in the figures and the corresponding discussion) against the conventional reconfigurable processor approaches that maximize the performance. The energy management scheme with *Selective Instruction Set Muting* using various muting modes will be benchmarked in Sect. 7.3.3 against various hardware-oriented shutdown techniques.

Using the H.264 video encoder application, for 30 fps on 65 nm, *AEM_FM* achieves an energy saving of up to 40.78% (avg. 24.77%) compared to *RISPP_PerfMax* [BSH08c]. For 35 fps, *AEM_FM* achieves an energy saving of up to 25.06% compared to *RISPP_PerfMax*. In order to maximize the performance *RISPP_PerfMax* uses more DPCs thus leading to higher leakage and reconfiguration energy (see Fig. 7.8). However, compared to *RISPP_PerfMax*, the *AEM_FM* at 35 fps may suffer from an average performance loss of 8%.

Figure 7.9 shows the breakdown of leakage and dynamic energy averaged over 17 cases of available reconfigurable fabric area (i.e., 4–20 DPCs). It can be noted in Fig. 7.9 that major savings come from leakage energy reduction as a result of the CI-level shutdown. The energy management scheme determines the energy minimizing instruction set in such a way that the performance constraint is met, while the number of DPCs to be shutdown is also increased to achieve higher leakage reduction. The key is to shift the shutdown decision to the CI level such that all temporarily unused CIs are *muted* (i.e., deactivated by an instruction set level shutdown, see Sects. 5.2.1 and 5.2.2). It can be noticed that the leakage energy savings at 5 fps is more than that at 35 fps due to an increased number of switched-off DPCs.

7.3.2 Applying the Adaptive Energy Management Scheme (Without Selective Instruction Set Muting) to Molen [VWG+04] Reconfigurable Processor

In order to validate the applicability and benefits of the proposed adaptive energy management scheme with CI-level muting, the energy management scheme has been additionally evaluated for other state-of-the-art reconfigurable processors

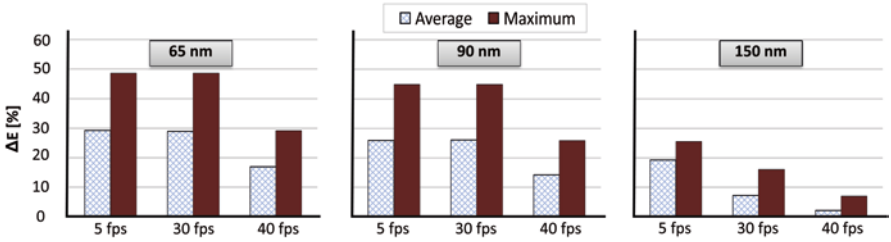


Fig. 7.10 Percentage energy saving of Molen [VWG+04] plus AEM_FM over Molen without AEM_FM for three technologies

(like Molen [VWG+04]) that support the *monolithic* CI model (see Sect. 2.3.4). Figure 7.10 shows the energy savings for Molen [VWG+04] with the proposed adaptive energy management scheme applied and Molen without the energy management scheme (averaged over 17 cases of area constraints, i.e., different sizes of the reconfigurable fabric). When compared to Molen without the energy management scheme (i.e., maximizing for performance), Molen plus the energy management scheme achieves an energy saving of up to 48.65% (average 28.93%) for 30 fps at 65 nm. This shows that the proposed adaptive energy management scheme is equally beneficial for various state-of-the-art reconfigurable processors as well.

Now, the adaptive energy management scheme with *Selective Instruction Set Muting* technique (that employs various CI muting modes) is benchmarked against different state-of-the-art hardware-oriented shutdown techniques.

7.3.3 Comparing the Adaptive Energy Management Scheme (with Selective Instruction Set Muting) to State-of-the-art Hardware-oriented Shutdown

As discussed in Sect. 7.1.1 and Chap. 5, comparing the proposed energy management scheme with state-of-the-art hardware-oriented shutdown techniques [Ge04; MM05] is not straightforward due to a different abstraction level of shutdown. Unlike state-of-the-art [Ge04; MM05], the proposed energy management scheme employs the *Selective Instruction Set Muting* technique (i.e., an instruction set oriented shutdown). Therefore, for a fair comparison, the hardware-oriented shutdown concepts of [Ge04] and [MM05] were deployed carefully to realize two predetermined CI muting techniques: (a) [Ge04]-based *Predetermined Virtually Muting* (Pre-VM) technique, and (b) [MM05]-based *Predetermined Fully Muting* (Pre-FM) technique (see Sect. 7.1.1 for further details). It is worthy to note that, the components of a DPC (i.e., Logic of Configuration SRAM, see Sects. 2.3.2 and 5.2.2) that can be shutdown are predetermined at design time in these state-of-the-art [Ge04; MM05], thus the muting mode is fixed for the unused CIs.

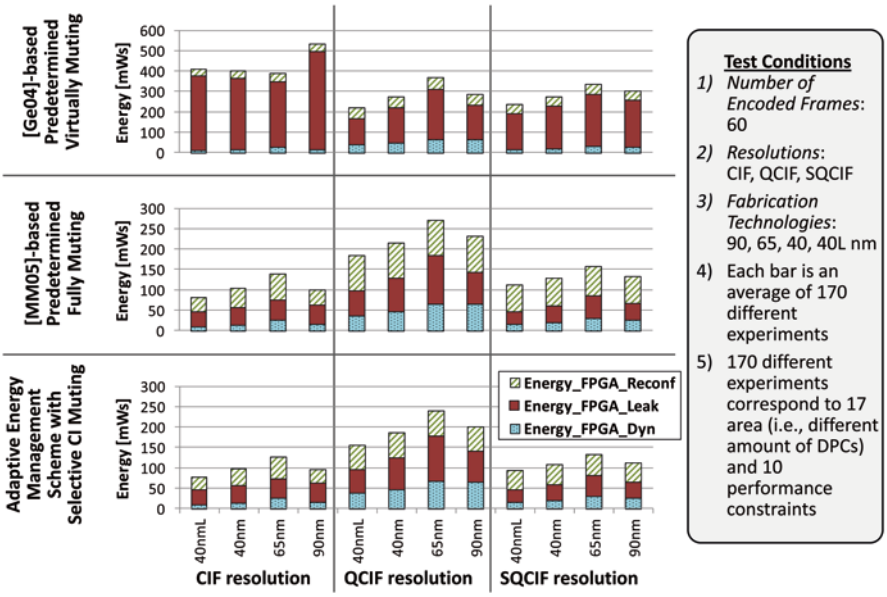


Fig. 7.11 Comparing the energy breakdown of the adaptive energy management scheme (with selective instruction set muting) to [Ge04]-based pre-VM and [MM05]-based pre-FM

Figure 7.11 compares the breakdown of the energy comparison for the proposed energy management scheme with *Selective Instruction Set Muting* and the two predetermined CI muting techniques. Each bar is the average value of 170 experiments (17 cases of the reconfigurable fabric area and 10 cases of different performance constraints). Figure 7.11 shows that the leakage energy is dominant in the [Ge04]-based *Pre-VM* technique due to high SRAM leakage, especially in case of CIF encoding. In contrast to this, the [MM05]-based *Pre-FM* technique reduces the leakage energy by shutting down the SRAM, but suffers from significant reconfiguration energy overhead. The proposed adaptive energy management overcomes the drawbacks of both of the above techniques by providing multiple CI muting modes and selecting an appropriate mode at run time for a temporarily unused subset of CI depending upon their predicted muting duration. It overcomes the reconfiguration overhead of *Pre-FM* by using the *virtually-muting mode* for a subset of CIs and eliminates the drawback of *Pre-VM* by putting the subset of CIs in *fully-muting mode* that are not used for a rather long period (see the details of different muting modes in Sect. 5.2.1 and 5.4). Figure 7.11 illustrates that the energy management scheme with multiple CI muting modes is superior to both state-of-the-art techniques in all cases. In particular, the benefit of the proposed scheme is noticeable in high-resolution encodings due to a rather long muting duration. It is worthy to note in Fig. 7.11 that the leakage energy of the energy management scheme is slightly lower than that of *Pre-FM*. This is because virtually-muted CIs may bring a leakage

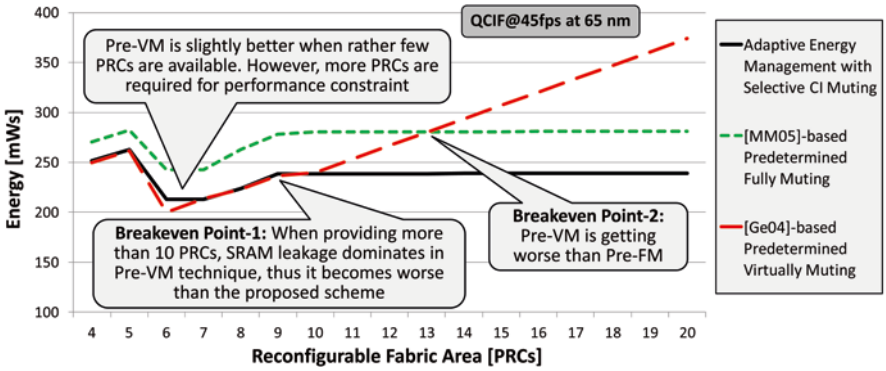


Fig. 7.12 Energy comparison of the adaptive energy management scheme with [Ge04]-based pre-VM and [MM05]-based pre-FM techniques for varying amount of reconfigurable fabric

reduction due to a faster execution[3] (as a result of the powered-on Configuration SRAM), which is not the case in the *Pre-VM* technique (see Sect. 2.3.2).

Figure 7.12 shows the energy consumption of the proposed adaptive energy management scheme with *Selective Instruction Set Muting* and the two predetermined CI muting techniques for a varying amount of the available reconfigurable fabric (i.e., different number of available DPCs) when encoding QCIF@45fps at 65 nm technology. There are two interesting cases in Fig. 7.12. The first breakeven point corresponds to the 10 DPCs case where the [Ge04]-based *Pre-VM* technique starts turning noticeably worse than the energy management scheme due to the increased leakage of the powered-on Configuration SRAM. The second breakeven point corresponds to the 13 DPCs case where the [Ge04]-based *Pre-VM* technique even worsens compared to the [MM05]-based *Pre-FM*. This shows that the *Pre-VM* technique is only beneficial when rather few DPCs are available. In this case, most of the available DPCs are always used. However, in order to meet tighter performance constraints, typically more DPCs are required, and in such cases the *Pre-VM* technique performs inefficiently. In contrast, the proposed energy management scheme is beneficial for almost all the cases (see Fig. 7.12). The performance constraints and the amount of available DPCs cannot be predicted at design- and/or compile-time as they depend on run-time specific scenarios, like changing application contexts or multi-tasking interactions.

It is noticeable in Fig. 7.12 that the proposed *energy management scheme with Selective Instruction Set Muting* performs always better than the [MM05]-based *Pre-FM* technique, whereas [Ge04]-based *Pre-VM* sometimes performs better. Therefore, Fig. 7.13 focuses on further comparisons with *Pre-VM*, showing the energy benefit summary (480 experiments per technology with various combinations of available fabric area and performance constraints) of the proposed energy management scheme when compared with the *Pre-VM* technique. Figure 7.13 shows

[3] The larger leakage *power* does not necessarily lead to larger leakage *energy* if the execution time is correspondingly shorter.

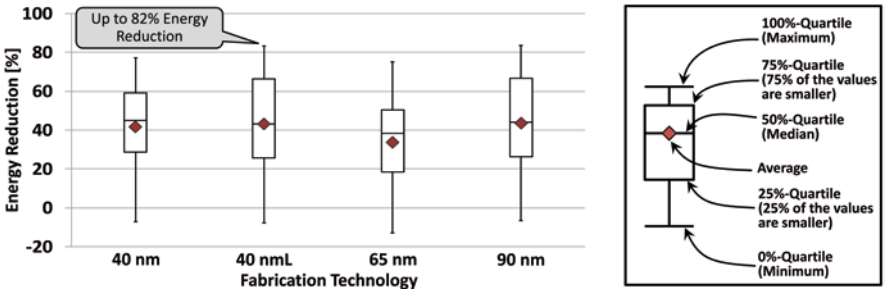


Fig. 7.13 Energy savings of the adaptive energy management scheme compared to the [Ge04]-based pre-VM technique

that, compared to the *Pre-VM* technique, the proposed scheme provides on average 41.64%, 43.11%, 33.75%, and 43.52% energy reduction for 40 nm, 40 nmL, 65 nm, and 90 nm, respectively. When rather few DPCs are available, the *Pre-VM* technique performs better than the proposed scheme as most of the DPCs are always used. However, in most of the cases (especially for larger resolutions), the proposed scheme outperforms the *Pre-VM* technique and provides an energy reduction of up to 82%.

It is worthy to note that the adaptive energy management with *Selective Instruction Set Muting* provides a compromise between [Ge04]-based *Pre-VM* and [MM05]-based *Pre-FM* techniques.

7.4 Summary of the Benchmarks and Comparisons

This chapter presented the benchmarks and evaluation of the proposed adaptive low-power application and processor architectures for various video sequences under different coding conditions. The adaptive energy management scheme is evaluated for different fabrication technologies under various performance and reconfigurable fabric area constraints. Moreover, both application and processor architectures are compared to state-of-the-art approaches.

At the application level, the proposed energy-aware H.264/AVC video encoder is evaluated for several video sequences with low to fast motion encoded at different bit rates. The proposed algorithms are compared for energy savings and quality degradation. Compared to state-of the-art approaches [JC04, KC07, PC08, SN06, Yu04], the proposed *adaptive complexity reduction scheme* achieves up to 82% (average 56%) higher energy reduction at the cost of an average PSNR loss of 0.66 dB. When compared to the *exhaustive RDO-MD*, the proposed complexity reduction scheme provides an average energy savings of 63.27% and 66.74% with an average PSNR loss of 1.38% and 1% for CIF and QCIF sequences, respectively. The highest energy savings (more than 70%) are obtained for slow motion sequences.

The proposed *energy-aware Motion Estimation with the integrated energy-budgeting scheme* achieves an energy benefit of up to 93%, 93%, 90% (average 88%, 88%, 77%) for UMHexagonS [CZH02], UMHexagonS-Simple [YZLS05], EPZS [Tou02] adaptive Motion Estimators, respectively. The major energy saving comes for low motion sequences (*MissAmerica*, *Akiyo*, *Clair*, *Mobile*, etc.) due to the switching between multiple *Energy-Quality Classes* depending upon the spatial and temporal properties of different Macroblocks. Even at a full battery level, the energy is not wasted for homogeneous and slow moving Macroblocks. Alternatively, more energy budget is allocated to the fast moving Macroblocks. Due to the slow motion properties, the reduced Motion Estimation effort still provides a sufficiently good match of the current Macroblock in the reference frame. Therefore, the incurred quality loss for sequences with homogeneous and slow moving Macroblocks is insignificant.

The proposed adaptive low-power processor architecture with run-time adaptive energy management scheme is evaluated for highly flexible Custom Instruction set architectures like in [Bau09; VWG+04]. The H.264 encoder is used as the application with various performance constraints and video resolutions. Applied to the RISPP architecture with *modular* CI model, the proposed adaptive energy management scheme with CI-level muting achieves an energy saving of up to 40.78% (average 24.77%) for 65 nm at the cost of an average performance loss of 8%, when compared to the original RISPP (i.e., having performance as the main optimization goal). The proposed adaptive energy management scheme with CI-level muting is additionally integrated with other state-of-the-art reconfigurable processors (like Molen [VWG+04]) with *monolithic* CI model where it provides an energy saving of up to 48.65% (average 28.93%) for 30 fps at 65 nm. This shows that the proposed adaptive energy management scheme is equally beneficial for various state-of-the-art reconfigurable processors as well. Experiments for different fabrication technologies demonstrate the technology independent efficiency of the proposed scheme. The adaptive energy management scheme with *Selective Instruction Set Muting* technique (using multiple CI muting modes) is additionally benchmarked against different state-of-the-art hardware-oriented shutdown techniques [Ge04; MM05], where it provides on average more than 30% energy savings.

Overall, the comparison with state-of-the-art and benchmarks for diverse experimental conditions demonstrate the superiority of the proposed adaptive low-power processor and application architectures, especially under run-time varying scenarios due to changing video properties, available energy resources, user-defined constraints, etc. The proposed adaptive energy management scheme with *Selective Instruction Set Muting* is particularly beneficial in applications with hard-to-predict behavior where conventional embedded (reconfigurable) processors operate inefficiently with respect to energy/power consumption. The results corroborate the potential for far higher energy savings of dynamically reconfigurable processors which currently still suffer from a low efficiency as far as energy is concerned. At the application level, the novel concept of *Energy-Quality Classes* and adaptive complexity reduction provides a foundation for *adaptive low-power video encoding*

to react to the unpredictable video data in an energy-efficient way. Altogether, the proposed processor and application architectures enable *adaptive embedded multimedia systems with low power/energy consumption* to provide means for next-generation mobile multimedia applications and emerging multimedia standards.

Chapter 8
Conclusion and Outlook

8.1 Monograph Summary

This monograph aims at exploiting the available potential of energy reduction in adaptive multimedia systems (based on dynamically reconfigurable processors) while meeting the performance and area constraints and keeping the video quality degradation unnoticeable, under run-time varying scenarios (due to changing video properties, available energy resources, user-defined constraints etc.). To enable this, novel techniques for adaptive energy management at *both processor architecture and application architecture levels* are proposed, such that both hardware and software adapt together in order to minimize the overall energy consumption under design-/compile-time unpredictable scenarios.

The key contribution at the processor architecture level is based on the novel concept of *Selective Instruction Set Muting*. Unlike state-of-the-art hardware-oriented shutdown techniques [CHC03; Ge04; MM05; Te06], the proposed *Selective Instruction Set Muting* allows to shun the leakage energy at the abstraction level of Custom Instructions (CIs), i.e., an instruction set oriented shutdown. Various so-called 'CI muting modes' are introduced, each leading to a particular leakage energy saving. This enables a dynamic tradeoff between 'leakage energy saving' and 'reconfiguration energy overhead' considering the application execution behavior under run-time varying performance and area constraints (e.g., in a multi-tasking environment). For dynamically reconfigurable processors, it is hard to predict at compile time which parts of the instruction set will be reconfigured on which part of the reconfigurable fabric. Therefore, raising the abstraction level of shutdown to the instruction set level provides a new way to save energy in dynamically reconfigurable processors by relating leakage energy reduction to the execution context of an application. It thereby enables a far higher potential for energy savings that results in a much higher energy efficiency for dynamically reconfigurable processors (and reconfigurable computing in general). The associated potential energy savings have not been exploited by state-of-the-art approaches [CHC03; Ge04; MM05; Te06].

Based on the concept of CI muting, an **adaptive low-power processor architecture** is proposed that integrates a novel run-time energy management scheme with dynamically reconfigurable processors. It exploits the higher potential for energy

M. Shafique, J. Henkel, *Hardware/Software Architectures for Low-Power Embedded Multimedia Systems,*
DOI 10.1007/978-1-4419-9692-3_8, © Springer Science+Business Media LLC 2011

savings due to the concept of CI muting with multiple shutdown modes and provides a high adaptivity. The energy management scheme investigates the tradeoff between leakage, dynamic, and reconfiguration energy for a given performance constraint, thus dynamically moving in the *energy-performance design space*. In the first step, the energy management scheme dynamically determines an energy minimizing instruction set under run-time varying performance and area constraints considering leakage, dynamic, and reconfiguration energy. Afterwards, it decides which subset of CIs shall be muted at what time and in which mode in order to minimize the overall energy. For this, the temporarily unused set of the CIs is determined which is the candidate for muting (i.e., power-shutdown) to reduce the leakage energy. Depending upon the Data Path requirements of the currently executing and the upcoming computational hot spots, a particular muting mode is determined for each CI by evaluating the possible associated energy benefit (a joint function of leakage, dynamic, and reconfiguration energy) at run time. Afterwards, the shutdown signals to the corresponding sleep transistors are issued. Note, these decisions may depend upon the number of CI executions that may vary at run time due to the application level adaptivity (e.g., changing control flow), changing input data, performance constraints, and the execution length of the hot spot. Therefore, the number of actual CI executions is monitored at run time. The algorithms for determining the energy minimizing instruction set and *Selective Instruction Set Muting* are explained on a formal basis and evaluated for various fabrication technologies.

To facilitate the adaptive energy management at both processor and application levels, *a comprehensive power model for dynamically reconfigurable processors* (i.e., ASIC-based core Instruction Set Architecture with an embedded FPGA) is developed, which is based on power measurements. The proposed power model estimates the power of *modular* Custom Instructions (CIs) executing on the reconfigurable fabric considering run-time choices of multiple CI Implementation Versions. Moreover, it can also be used to estimate the power of *monolithic* CIs that employed by the dynamically reconfigurable processors like Molen [VWG+04]. The leakage and dynamic power properties of both the core Instruction Set Architecture and the reconfigurable fabric are considered in the power model. The model parameters are estimated by performing an optimization using Simulated Annealing for an estimation error of less than 5%. The power of a CI Implementation Version depends upon (a) the types of Data Paths and how often they are executed, (b) the number of write/read accesses on the local storage, and (c) the number of bus segments necessary for communicating the intermediate results; this value depends on the relative placement of the communicating Data Paths on the reconfigurable fabric. As the power model is based on actual power measurements, a complete power-measurement setup for dynamically reconfigurable processors (a power supply board, two oscilloscopes, an FPGA based prototyping board, and a control program for capturing the measurements) is implemented and the power of various CI implementation versions in hardware is measured.

The proposed adaptive low-power processor architecture with run-time adaptive energy management scheme is evaluated for multiple Custom Instruction set architectures (RISPP [Bau09], Molen [VWG+04]) using an in-house developed H.264

encoder application (with various performance constraints and video resolutions) and various fabrication technologies. Applied to the RISPP architecture that supports the *modular* CI model, the proposed adaptive energy management scheme with CI-level muting achieves an energy saving of up to 40.78% (average 24.77%) for 65 nm at the cost of an average performance loss of 8% when compared to the original RISPP (i.e., having performance as the main optimization goal). In order to validate the applicability and benefits of the proposed adaptive energy management scheme, it has been additionally evaluated for other state-of-the-art reconfigurable processors (like Molen [VWG+04]) that support the *monolithic* CI model. When compared to Molen without the energy management scheme (i.e., maximizing for performance), Molen plus the energy management scheme achieves an energy saving of up to 48.65% (average 28.93%) for 30 fps at 65 nm. This shows that the proposed adaptive energy management scheme is equally beneficial for various state-of-the-art reconfigurable processors as well. The adaptive energy management scheme with *Selective Instruction Set Muting* technique (that employs multiple CI muting modes) is additionally benchmarked against different state-of-the-art hardware-oriented shutdown techniques [Ge04; MM05]. Compared to [Ge04]-based technique, the proposed scheme provides on average 41.64%, 43.11%, 33.75%, and 43.52% energy reduction for 40 nm, 40 nmL, 65 nm, and 90 nm, respectively. On overall, compared to [Ge04; MM05] based techniques, the proposed energy management scheme achieves on average more than 30% energy savings.

At the **application architecture level**, the adaptivity and energy reduction are demonstrated using an advanced video encoder (like H.264/AVC). An *optimized application architecture* for video encoders targeting dynamically reconfigurable processors is proposed. The finalized application architecture is implemented with optimized data flow and data structures. Various low-power Custom Instructions and Data Paths are designed for the H.264 video encoder. Several algorithms are proposed to realize *adaptive low-power video encoding*. First, *an analysis of spatial and temporal video properties* is performed with consideration of important Human-Visual System properties in order to categorize different video frames and their Macroblocks. Quantization Parameter based threshold models are developed to obtain precise categorization depending upon the coding configuration. Furthermore, an analysis of the energy consumption of different functional blocks of the video encoder is performed. This analysis is used by *an adaptive complexity reduction scheme* to reduce the energy requirements of the H.264/AVC encoder by excluding improbable coding modes from the mode-decision process. It solves the issue of choosing the final coding mode out of hundreds of possible combinations (without exhaustively searching the design space) by considering the spatial and temporal video properties. Unlike state-of-the-art, this scheme performs an extensive mode-exclusion before fast Mode Decision and Motion Estimation processes, thus providing a significant reduction in the computational complexity. Once the final set of candidate coding modes is determined, *an energy-aware Motion Estimation with integrated energy-budgeting scheme* is employed in order to adaptively predict the energy quota for the Motion Estimation corresponding to each candidate coding mode. It employs the novel concept of *Energy-Quality Classes* in order to

realize *adaptive low-power video encoding*. Each *Energy-Quality Class* represents a particular Motion Estimation configuration that requires a certain energy while providing a certain video quality. It thereby enables a run-time tradeoff between the energy consumption and the resulting video quality. A set of common optimal *Energy-Quality Classes* is obtained by performing a design space exploration for various test video sequences. The adaptive energy-budgeting scheme chooses a certain *Energy-Quality Class* for different video frames considering the available energy, video frame characteristics, and user-defined coding constraints while keeping a good video quality. The corresponding Motion Estimation configuration (i.e., set of initial search point predictors, search patterns, etc.) is forwarded to the energy-aware Motion Estimation. After the Motion Estimation is completed, the energy of *Energy-Quality Classes* is updated depending upon the current video statistics.

The proposed energy-aware H.264/AVC video encoder is evaluated for several QCIF and CIF video sequences (low to fast motion) [Ari08; Xip10] encoded at different bit rates, while considering the power model proposed in this monograph. The proposed algorithms are compared for quality degradation and energy savings. Compared to state-of-the-art approaches [JC04; KC07; PC08; SN06; Yu04], the proposed *adaptive complexity reduction scheme* achieves up to 82% (average 56%) higher energy reduction at the cost of an average PSNR loss of 0.66 dB. Compared to the exhaustive rate distortion optimized Mode Decision, the proposed *adaptive complexity reduction scheme* provides average energy savings of 63.27% and 66.74% with an average PSNR loss of 1.38% and 1% for CIF and QCIF sequences, respectively. The highest energy savings (more than 70%) are obtained for slow motion sequences. The proposed *energy-aware Motion Estimation with the integrated energy-budgeting scheme* achieves an energy benefit of up to 93%, 93%, 90% (average 88%, 88%, 77%) for *UMHexagonS* [CZH02], *UMHexagonS-Simple* [YZLS05], *EPZS* [Tou02] adaptive Motion Estimators, respectively. The proposed energy-budgeting scheme is equally beneficial for other state-of-the-art fast adaptive MEs as well. When integrated into *UMHexagonS* [CZH02] Motion Estimator, it provides an energy saving of up to 80% (avg. 70%) with a slight PSNR loss of 0.11 dB. Compared to the *Full Search*, the energy-aware Motion Estimation scheme provides an energy saving of up to 99% at the cost of an average PSNR loss of 0.29 dB, which is visually insignificant. Note, the comparison with the *Full Search* is mainly for video quality. The major energy saving comes for low motion sequences (*MissAmerica*, *Akiyo*, *Clair*, *Mobile*, etc.) due to the switching between multiple *Energy-Quality Classes* depending upon the spatial and temporal properties of different Macroblocks. Even at a full battery level, the energy is not wasted for homogeneous and slow moving Macroblocks. Alternatively, more energy budget is allocated to the fast moving Macroblocks. Due to the slow motion properties, the reduced Motion Estimation effort still provides a sufficiently good match of the current Macroblock in the reference frame. Therefore, the incurred quality loss for sequences with homogeneous and slow moving Macroblocks is insignificant.

To compensate the quality loss due to the energy-aware adaptations, a multi-level rate control is proposed. It allocates a bit budget to the Group of Pictures and then distributes this budget to different frames. It afterwards determines the final

Quantization Parameter value for each Macroblock inside a frame considering its spatial and temporal properties (see Appendix A). It allocates more bits to the complex Macroblocks and less bits to the less-complex ones. The complete H.264 video encoder application with the proposed run-time algorithms and low-complexity data flow is demonstrated by executing it on an in-house RISPP dynamically reconfigurable processor prototype [Bau09], Texas Instruments' multimedia processor, and laptop/desktop computers (see Appendix B). A video analysis tool with an easy-to-use graphical user interface is developed for quick and in-depth analysis of video sequences (see Appendix C).

Overall, the comparison with state-of-the-art and benchmarks for diverse experimental conditions demonstrate the superiority of the proposed adaptive low-power processor and application architectures, especially under run-time varying scenarios due to changing video properties, available energy resources, user-defined constraints, etc. The proposed adaptive energy management scheme with *Selective Instruction Set Muting* is particularly beneficial in applications with hard-to-predict behavior where conventional embedded (reconfigurable) processors operate inefficiently with respect to energy/power consumption. The results corroborate the potential for far higher energy savings of dynamically reconfigurable processors which currently still suffer from a low efficiency as far as energy is concerned. At the application level, the novel concept of *Energy-Quality Classes* and adaptive complexity reduction provide a foundation for *adaptive low-power video encoding* to react to the unpredictable video data in an energy-efficient way. Altogether, the proposed processor and application architectures enable *adaptive embedded multimedia systems with low power/energy consumption* to provide means for next-generation mobile multimedia applications and emerging multimedia standards.

8.2 Future Work

The novel conceptual contribution, benchmarking with diverse experimental conditions, and comparison with relevant state-of-the-art demonstrate that for designing an adaptive low-power multimedia system, there is a dire need to combat the power-related issues at all abstraction levels (i.e., at both processor and application levels). Moreover, for the next-generation low-power multimedia systems, both hardware and software need to adapt together at run time to efficiently utilize the available energy resources under design-/compile-time unpredictable scenarios. The promising results of this research work open new research avenues for power-management techniques in dynamically reconfigurable processors, improved power efficiency, energy-aware image and video processing, and compile-time automation. These new research avenues are summarized in the following.

Power-management techniques in dynamically reconfigurable processors: Significant energy efficiency has been obtained using the novel concept of *Selective Instruction Set Muting* that raises the abstraction level of power-shutdown to the

instruction set level. A power-shutdown infrastructure with multiple sleep transistors needs to be researched to support multiple muting modes considering the area and wakeup overhead. Such an infrastructure should be designed with consideration of partitioning of the reconfigurable fabric to support run-time partial reconfiguration (i.e., employing Data Path Containers). Design of the power-rail is an additional research challenge in this case. Furthermore, other factors like ground bounce noise may be considered while controlling the wakeup signals of different Data Path Containers. The proposed energy management scheme can also be extended towards multi-tasking systems, where multiple tasks share the same reconfigurable fabric. In such a scenario the fabric may be allocated to different tasks. Then the challenging question arises: whether it is beneficial to mute the temporarily unused subset of Custom Instructions of a task or temporarily re-allocate the corresponding fabric portion to the other tasks to achieve higher energy efficiency for them. When considering the sharing of a fabric among different tasks, the design of an energy management system becomes an additional research challenge. The contribution of this monograph provides an initial foundation for researching such power-related issues. Moreover, the benchmarks demonstrated that the provision of Dynamic Voltage and Frequency Scaling (DVFS) techniques may provide additional energy savings in cases of lower performance constraints.

Improved Power Efficiency: The proposed energy management system can be extended to a reconfigurable multicore processor where several cores share a centralized reconfigurable fabric. In such cases energy management becomes a complicated issue especially when different cores are executing tasks of varying complexities and constraints. First there will be a need for a power model for such reconfigurable multicore processor. The proposed power model can be considered a starting point for this research. Further challenges will be managing the energy consumption of cores and the fabric in a holistic way.

Energy-aware image and video processing: In the scope of this monograph, an *adaptive low-power video encoder* is proposed that employs the concept of *Energy-Quality Classes* in order to provide the run-time configurability for energy consumption and resulting video quality. Such a concept can be extended towards video pre- and post-processing where different filter algorithms are provided and a selection between different algorithms is performed depending upon the allocated energy budget. Moreover, different configurations of such algorithms may be switched at run time. An example can be the switching of a 5×5 kernel based filtering to a 3×3 kernel based filtering. Similar extensions may be provided for various image processing algorithms like image enhancement with variable sized kernels. Moreover, further extensions of the proposed concepts can be realized for upcoming video coding standards like Multiview Video Coding and H.265 coding standard.

Compile-time Automation: In this monograph, the application architecture and low-power Custom Instructions and Data Paths were designed manually. In order to reduce the development time, there is a need to research new design methodologies and tool chains to automate this process. For Custom Instructions a design tool

flow similar to that in ASIPs may be considered. However, for Data Paths and low-power optimizations, intelligent algorithms and transformation techniques need to be investigated. A new research project KAHRISMA [ITI; KBS+10] has started which focuses on researching such kind of compile-time tool flow and design methodology. The processor-specific adaptations in the application architecture emerge as a more challenging task. A modular design and re-targetable compilation methodology may be a suitable choice to investigate.

Appendix

Appendix A: A Multi-Level Rate Control

In this appendix, a multi-level rate control is presented that was developed in the scope of this monograph. The proposed rate control algorithm performs a non-linear target bit allocation to handle scene cuts and suddenly appearing high motion scenes in video encoding at a vast range of bit rates and resolutions. It copes with varying Rate-Distortion (RD) characteristics of different frame types (I, P, B) and different Macroblocks (MBs) of the same frame type by categorizing them depending upon their spatial/temporal properties. It thereby achieves a better video quality which is required to compensate for the quality loss occurred as a result of energy-aware adaptations in Chap. 4.

A.1 A Rate Control Algorithm

A rate control algorithm is a functional block of video encoders that fulfills the bandwidth and buffer constraints for given channel and application properties at a given target bitrate. Due to the unpredictable and varying nature of the input video data, different frames and their different MBs require a different bit budget for coding (even for the same coding conditions). In case a rate controller is not used, a possible loss of data may occur due to the buffer overflow. Moreover, ignorance of a rate controller may also result in significant quality fluctuations that are undesirable for the end user.

Figure A.1 illustrates the encoding of the *Susie* test video sequence with a rate controller (100 Kbps @15 fps, it corresponds to an average bit budget of 6.6 Kbits per frame) and without a rate controller (QP=28 provides a similar bit budget in this case). It can be noticed from the figure that without the rate controller the fluctuations in the number of the coded bits are significant, especially when there is a significant change in the video content due to the rapid motion (in this case it is due to the sudden waving of the girl's head between frames 38 and 70). This results in quality fluctuations and unsmooth buffer state (that directly corresponds to a higher

M. Shafique, J. Henkel, *Hardware/Software Architectures for Low-Power Embedded Multimedia Systems,*
DOI 10.1007/978-1-4419-9692-3, © Springer Science+Business Media LLC 2011

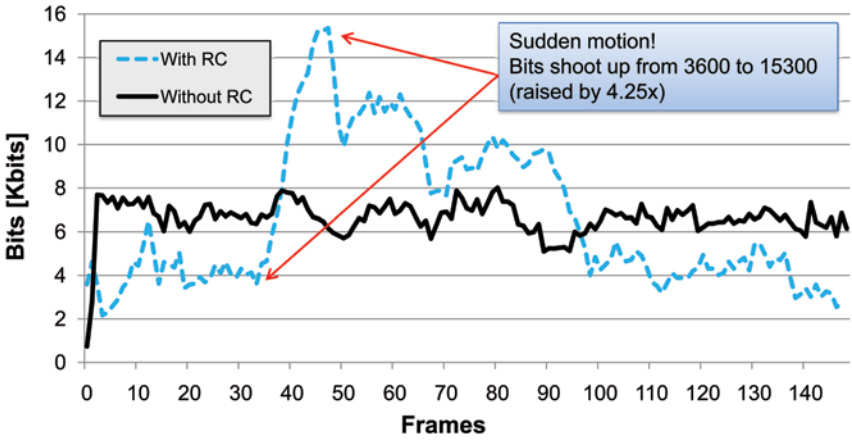


Fig. A.1 Comparison of produced bits with and without rate control

buffer cost or alternatively a risk of buffer overflow). In contrast to this, when using the rate controller provides reduced fluctuations in the number of the coded bits, thus providing a better buffer and quality smoothness. It is especially important in case the videos are transmitted over a channel under limited bandwidth constraints.

A rate controller determines the Quantization Parameter (QP) by considering the varying number of bits for each frame due to their diverse spatial and temporal characteristics. A good rate controller prevents buffer overflow (frame skipping) and/or underflow (improper bandwidth utilization) and maintains the buffer and quality smoothness between frames. It additionally provides a good visual quality within a frame (i.e., for different MBs) considering their Rate Distortion (RD) characteristics. Unlike other video coding standards, H.264/AVC [ITU05] exhibits a complex RD-model and a variety of frame type coding structures. A large body of research has been conducted in different Rate Control (RC) schemes to determine QP at frame and Basic Unit (BU, a group of Macroblocks that share the same QP value) level to achieve a target bit rate. Most of these RC schemes use (a) QP of P frames to determine the QP of I or B frames using (e.g., Mean Absolute Difference based) predicted RD characteristics [LPL+03; LSW04]; (b) Mean Absolute Difference or variance based QP adjustments at BU level [SZFH08; ZuHNS07]. Moreover, these approaches incorporate compute-intensive models (e.g., quadratic model with run-time adaptation of model parameters in [LPL+03; LSW04; LT07] and a standard deviation based model at BU level in [WAW07]). However, these RCs suffer from several drawbacks. First, the frame level target bits estimation ignores the image statistics and motion properties [LPL+03; LSW04], thus a sudden change in the frame (e.g., abrupt motion, scene cut) leads to discontinuities in the visual quality. Significantly, more bits are allocated for encoding the earlier frames in a Group of Picture (GOP) leaving a smaller bit budget for the successive frames. Another problem is the accuracy of Mean Absolute Difference estimation, as a linear model is susceptible to high prediction inaccuracy due to the unpredictable nature of the vid-

eo content, e.g., scene cuts. Therefore, the predicted QP is too small (or big) which leads to undesirable buffer and visual quality fluctuations. RC-Mode-0 [LPL+03] in H.264 encoder reference software treats all frames types in the same way not considering the diverse RD characteristics of different frame types. Therefore, it suffers from high buffer and quality fluctuations especially when encoding videos with multiple GOPs and multiple frame types at low bit rates. RC-Mode-3 [LT07, 08] treats I, P, and B frames in a different way considering the hierarchical levels. However, this approach does not handle those scenarios efficiently where a scene cut may occur at the B frame. Moreover, changing QP for each hierarchical level may introduce undesirable quality fluctuations. Another drawback of RC-Mode-3 is that it requires a priori knowledge about the content [LT08], which is unlikely in real-world applications due to the unpredictable nature of the video content.

Summarizing: frequently injected I frames, scene cuts, and scenes with hectic motion require more bits than normal P and B frames. Under scenarios of varying RD characteristics of different frame types (I, P, B) and different MBs in one frame (e.g., bright, textured, static/moving MBs), a low-complexity rate control with non-linear bit budgeting is desirable.

A.2 The Proposed Multi-Level Rate Control

In this monograph, a novel Rate Control (RC) scheme is proposed and employed that covers Group of Pictures (GOP), frame/slice, and Basic Unit levels (see Fig. A.2). It treats different frame types (I, P, B) in a different fashion with consideration of whether they are referenced or non-referenced frames. The proposed RC scheme prioritizes Macroblocks (MBs) depending upon their spatial and temporal characteristics (considering eye-catching regions) for refined Quantization Parameter (QP) allocation. It handles various bit rate scenarios, poorly predicted frames, and videos with dark/bright, blurry/noisy, high/low-textured, slow/fast motion properties. The variation in QP is restricted depending upon the target bit rate, frame type, and buffer status etc. In the following, different blocks of the RC scheme are presented in their corresponding processing sequence.

GOP Level Rate Control: For ensuring a smooth quality variation in consecutive GOPs, each GOP is provided with a separate bit budget. However, this is only beneficial for small-sized GOPs where target bits of a GOP serve as a hint for the frame level bit budgeting strategy. For large-sized GOPs (e.g., ≥100 frames), the target bit budget is not used as the RC scheme ensures buffer and quality smoothness at frame level. The target bit budget for the *i*th GOP (TB_{GOPi}) is determined by:

$$TB_{GOPi} = [(S_{buff} > 2*TBR \ \& \ is \ VBR())?2*TBR : TBR]*(N_{GOP}/FR) \quad (9.1)$$

TBR is the target bit rate, *FR* is the frame rate, N_{GOP} is the number of frames in a GOP, and S_{buff}=*mf***TBR* is the bitstream buffer size. In case of *Variable Bit Rate* (VBR) and large-sized buffers, a higher bit budget is allowed for the GOP. The

multiplication factor *mf* depends upon (a) the type of RC, i.e., either *Constant Bit Rate* (CBR) or VBR; and (b) channel/encoder delay constraints. The maximum buffer fullness constraint is defined as $BuffFullness_{Max}=(1-BOPF) * S_{buff}$ where the *buffer overflow prevention factor* (BOPF) acts as a factor of safety. Kindly note that, the buffer size affects the transmission delay and the overall memory cost of the system. A relatively smaller buffer is cost effective but increases the risk of data loss due to overflow. Alternatively, a relatively bigger buffer increases the transmission delay and memory requirements. The size of the bitstream buffer may be determined by the provided buffer smoothness of a rate controller.

Non-Linear Target Bit Budgeting: Since different frame types or different Basic Units (BUs) of the same frame may exhibit varying RD characteristics, a linear target bit budgeting may lead to undesirable/unacceptable quality variations. A scene cut in P or B frames will require more bits than the previous P/B frame to avoid sudden PSNR variation. In RC-Mode-0 [LPL+03], the QP for the I frames depends upon the target bit rate and resolution. Some RC schemes consider spatial variance of frame for linear translation into bits. The RC scheme in [LT08] uses $R_I=\gamma_I*R_P$ and $R_B=\gamma_B*R_P$ to predict the target bits of I and B frames. However, linear functions may lead to quality fluctuations due to the unpredictable nature of video data thus they are inefficient in sudden textural changes. Therefore, a non-linear target bit budgeting is performed to handle scene cuts, I frames, and high-textured images. First, the amount of texture difference ($EdgeDiff_{AVG}$) between two consecutive frames is calculated (using Sobel Operator). Afterwards, the target bits for jth frame of ith GOP ($T_{Bits_Fi,J}$) are determined as:

$$
\begin{aligned}
&IF\ (EdgeDiff_{AVG} > \varepsilon_1\ or\ isI_Slice()\ or\ isSceneCut()) \\
&\xi = [\max(\min(EdgeDiff_{AVG}/\varepsilon_1,\ \varepsilon_2),\ \varepsilon_3)]\ ELSE\ \xi = 1 \\
&T_{BitsFi,j} = \xi * [((TB_{GOPi} * N_{GOP}/FR) - Bits_{TotalUsed})/N_{GOPi_Rem}] \\
&EdgeDiff_{AVG} = \sum_{i=0}^{\#MBs} \left|Edge_{Fcurri} - Edge_{Fprevi}\right|/\#MBs
\end{aligned}
\tag{9.2}
$$

where ε_1, ε_2, and ε_3 classify the complexity difference of consecutive frames. N_{GOPi_Rem} is the number of remaining non-encoded frames in the *i*th GOP, and $Bits_{TotalUsed}$ is the total amount of bits spent till the last encoded frame. If $EdgeDiff_{AVG}$ does not exceed ε_1, it is sufficient to perform a linear target bit allocation.

Frame/Slice Level Rate Control: At frame/slice level, a *Normalized PID Controller* (see Fig. A.2) is deployed to compute the ΔQP. The QP of current frame/slice (QP_{slice}) is obtained by adding ΔQP and QP_{prev}. The normalized PID controller keeps the buffer occupancy close to the target buffer fullness using its three gain factors.

a. *Normalized Proportional Gain (K_P')* reduces the error between the achieved and target bits.
b. *Normalized Integral Gain (K_I')* eliminates the steady error effect by considering the accumulated error of previously encoded frames.
c. *Normalized Derivative Gain (K_D')* ameliorates the system stability.

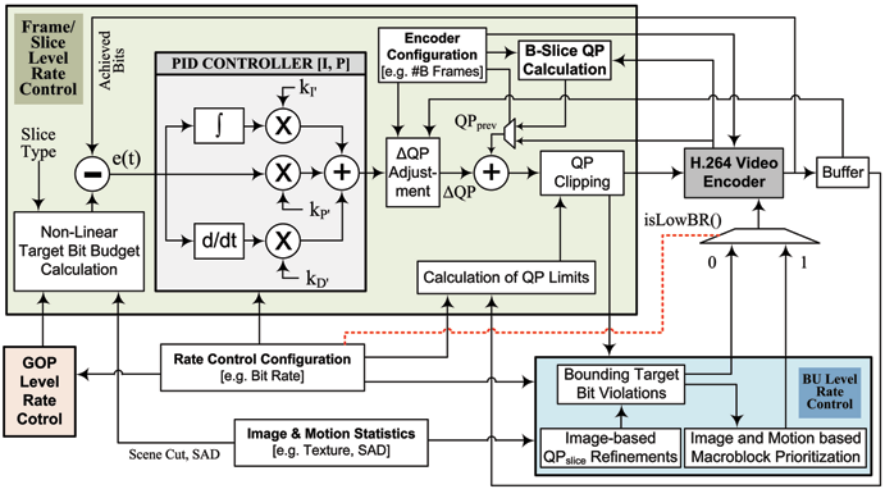


Fig. A.2 The multi-level rate control scheme covering GOP, frame/slice, & BU levels along with image and motion based macroblock prioritization

The ΔQP is calculated by:

$$\begin{aligned} \Delta QP &= K_{P'} * e(t) + K_{I'} * \sum e(t) + K_{D'} * [e(t) - e(t-1)] \\ e(t) &= A_{BitsFi,j} - T_{BitsFi,j}, \{K_{P'}, K_{I'}, K_{D'}\} = \{K_P, K_I, K_D\}/TBR \end{aligned} \tag{9.3}$$

K_P, K_I, and K_D are obtained using the *Ziegler-Nichols-Method* (ZN-Method) that uses an online experiment followed by the use of rules to compute the numerical values of the PID coefficients [JM05]. The dynamic vibration behavior of the complete control loop is investigated using the following procedure:

- First, only the P-controller is setup and the I- and D-parts are disable.
- For small *P-coefficient* values, the signal will result in a transient oscillation after some disturbance at the beginning. With increasing *P-coefficient* values, the signal will build up and the oscillation becomes mixed up.
- By successive iterations, the goal is to find a *P-coefficient* value at which the closed control loop swings with constant amplitude—the so-called *Critical Ziegler-Nichols-Point*. The period of this oscillation is called $T_{Critical}$ and the proportional boost $K_{P_Critical}$. Figure A.3 illustrates the case for the *American Football* test video sequence with 1 Mbps@15 fps, where $K_{P_Critical}$=0.8 and $T_{Critical}$=2.
- These values are utilized in the *Ziegler-Nichols-Rules* to compute the three PID coefficients:

$$K_P = 0.6 * K_{P_Critical},\ K_I = K_P/(0.5 * T_{Critical}), K_D = K_P * (0.125 * T_{Critical}) \tag{9.4}$$

The proposed PID controller is different from the related work as it directly outputs the ΔQP after scaling (considering the fact that an increase of 6 in QP value doubles

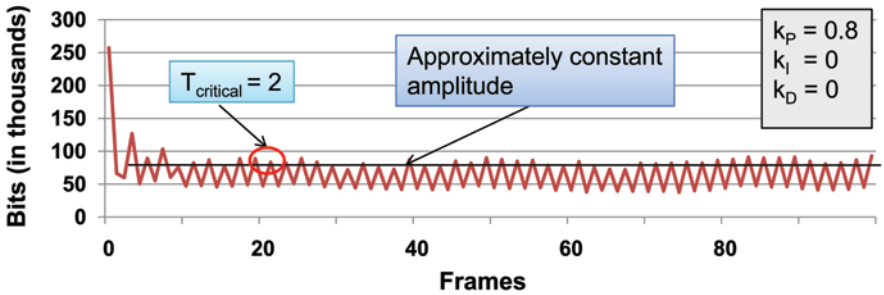


Fig. A.3 Critical Ziegler-Nichols-point for American Football

the quantization step) of PID gains to achieve an embedded translation of PID output into ΔQP. Moreover, the gains of the PID controller are normalized to make it generic for a vast range of bit rates (32 Kbps–4 Mbps). The output of the PID controller (i.e., ΔQP) is then clipped between $\pm\Delta QP_{Limit}$ (to keep a smooth visual quality):

$$\begin{aligned} &IF(QP_{Fprev_Avg} - 36 \geq 0)\ \Delta QP_{Limit} = MIN(MAX(QP_{Fprev_Avg} - 35, 3), 5) \\ &ELSE \qquad\qquad \Delta QP_{Limit} = 3 \end{aligned} \tag{9.5}$$

Note: ΔQP is calculated in a similar way for all frame types using the normalized PID controller but QP_{prev} is calculated in a different way for P and B frames. For P frames QP_{prev} is simply the QP of last encoded frame. However, for calculating QP_{prev} in B frames a Temporal Distance (td) based scheme is deployed (as shown in Fig. A.4) due to their dissimilar RD characteristics compared to P frames.

The QP_{prev_Bi} for the ith B-frame is calculated using the minimum of (1) average QP of previously encoded frame, and (2) weighted average of QPs of two referenced frames, as shown below:

$$\begin{aligned} &QP_{Bi_wAvg} = (td_{ref2} * QP_{ref1} + td_{ref1} * QP_{ref2}) / (td_{ref1} + td_{ref2}) \\ &QP_{Bi_prev} = MIN(QP_{Bi_wAvg}, \textstyle\sum_{i=0}^{\#MBs} QP_{Fi_prev} / \#MBs) \end{aligned} \tag{9.6}$$

The RC scheme treats B frames in two categories: '*used as referenced frame*' and '*not used as referenced frame*'. Due to this reason, it results in a far lesser PSNR

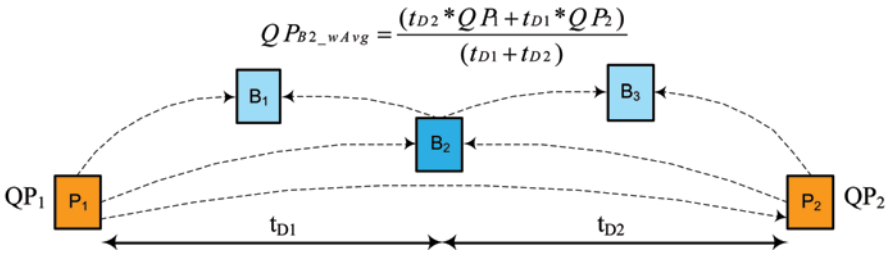


Fig. A.4 Temporal distance based QP calculation for B frames/slices

variation (see Sect. A.3) compared to state-of-the-art. Finally, the QP for a B frame is computed as:

$$QP_{Bi_slice} = QP_{Bi_prev} + \Delta QP + \alpha(!\,isRef()) - \beta(isRef()\,\&\,isHighTexture())$$
$$IF(isSceneCut())\alpha = 0, \beta = 2 \quad ELSE\ \alpha = 1, \beta = 1 \tag{9.7}$$

After adding ΔQP in the QP_{prev}, the resulting QP is clipped between QP_{MIN} and QP_{MAX} that are determined as follows:

$$QP_{MIN} = 12 - (TBR \text{ in bps} > 2650 * \#MBs) * 2$$
$$QP_{MAX} = 42 + (TBR \text{ in bps} < 165 * \#MBs) * 3 \tag{9.8}$$

In case of a buffer management system, the control effort is relaxed or tightened based on buffer status and buffer size considering the target bit rate. For VBR and large buffer-sized systems where data is not read from the buffer after encoding each frame (rather the reading from buffer is scheduled based on task switching considering encoder and buffer as two different tasks), QP_{MAX} is adjusted as:

$$QP_{MAX} = QP_{MAX} - 4 + \left\lfloor \left(1 - \frac{BuffFullness}{BuffFullness_{Max}}\right) * 4 \right\rfloor \tag{9.9}$$

Basic Unit (BU) Level Rate Control: The BU-level rate control reacts to changing image content within one video frame and performs refined QP allocations inside a frame depending upon the spatial and temporal complexity of the BU. Unlike state-of-the-art approaches (e.g., [LPL+03; SZFH08]), a BU-level rate control is used for both P and B frames. Since I frames deploy *spatial* prediction using neighboring MBs, varying QP at BU-level may lead to an unacceptable PSNR variation. Therefore BU-level rate control is disabled for an I frame. Figure A.5 shows the BU-level rate control operating in the following three steps.

Step 1) *Refined QP_{slice} Adjustments*: In the first step, QP_{slice} is refined by an amount of ΔQP_{dec} which depends upon the spatial/temporal properties of the BU. The Eq. A.5.1 in Fig. A.5 show the computation of ΔQP_{dec} for a dark BU in a dark frame to avoid the white noise effects in the darker regions. TH_{BU_QP}, TH_{BU_Br}, TH_{BU_SAD}, and TH_{B_Low} control the amount of decrement and categorize a BU as dark or bright. Afterwards, Eqs. A.5.2–A.5.4 compute the decrement step for ΔQP_{dec} for bright regions with less texture or slow motion to avoid the loss of details. Irrespective of the brightness of a BU, strong quantization may distort the slow moving regions therefore it needs to be preserved (see Eqs. A.5.5 and A.5.6). In case the MB inside a BU is stationary but not skipped, there is a high probability that there will be some transformed coefficients that needs to be protected to avoid the quantization noise. Therefore, ΔQP_{dec} is adjusted (depending on the SAD and TBR) to preserve the transformed coefficients of stationary MBs (see Eqs. A.5.7–A.5.12).

Step 2) *Controlling Target Bit Violations at BU-Level:* As image based decision may lead to a target bit violation when operating at BU level, the proposed scheme computes $\Delta QP_{dec_control}$ to adjust ΔQP_{dec} depending upon the error be-

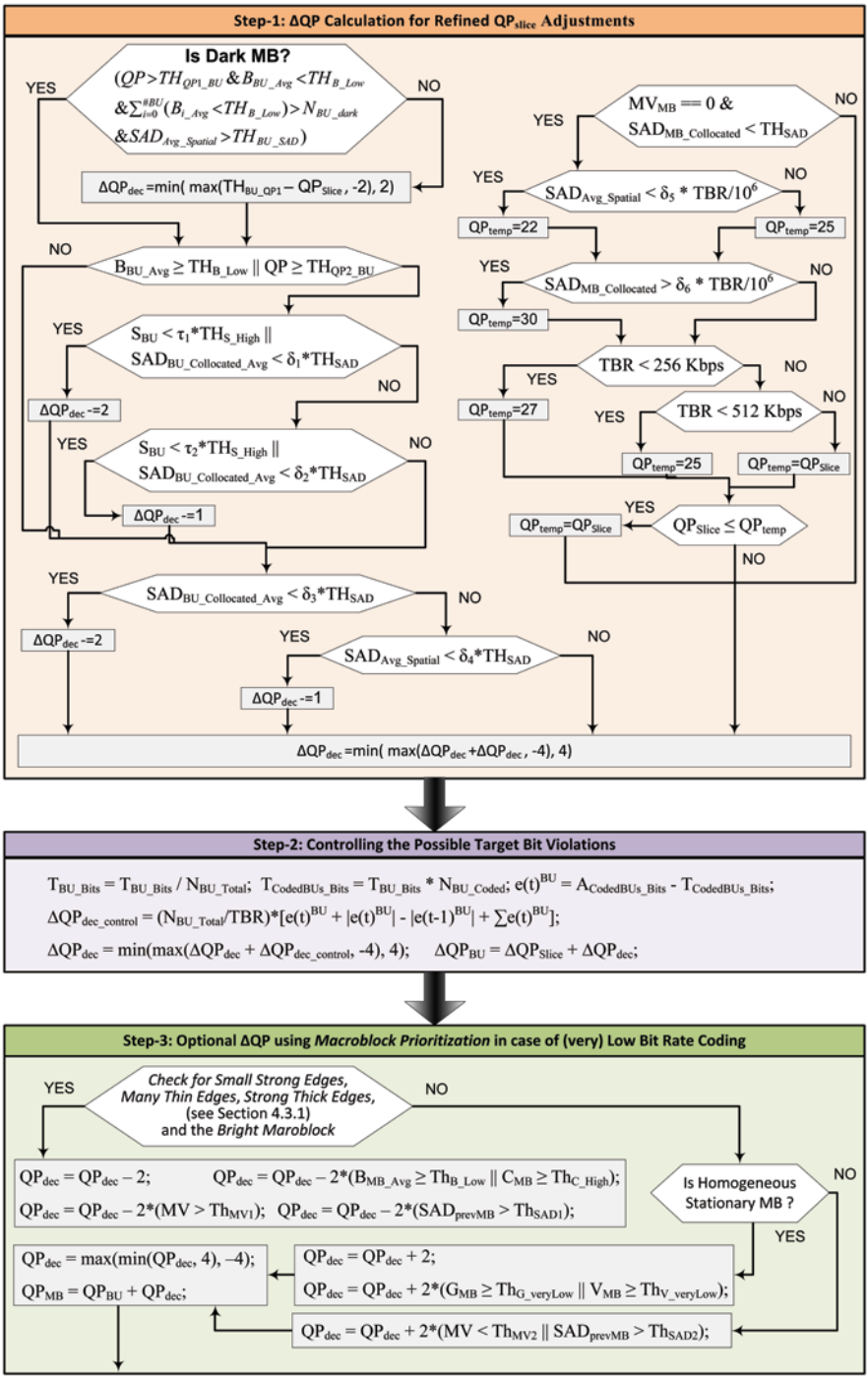


Fig. A.5 Basic unit (BU) level RC with texture and motion based QP adjustments

tween achieved and target bits till the coded BU ($e(t)_{BU}$). At first, the target bits for one BU (T_{BU_Bits}) and for all coded BUs ($T_{CodedBUs_Bits}$) are computed, where N_{BU_Coded} is the number of already encoded BUs. Afterwards, the bit error is computed which is then used to calculate $\Delta QP_{dec_control}$. A_{BUk_Bits} is the number of already produced bits for all coded BUs in the current frame/slice. $\Delta QP_{dec_control}$ is added in ΔQP_{dec} which is then clipped between ± 4 to restrict the possible violations of target bit budget and is added in QP_{slice} to get QP_{BU}.

Step 3) *Image-/Motion-Based Macroblock Prioritizations:* This step is optional for low bit rate coding scenarios and performs an image-/motion-based Macroblock Prioritization to capture eye-catching regions. The human eye is sensitive to fast motion and highly textured scenes that are hard to encode at low bit rates. Therefore, it is beneficial to spend more bits to such regions at the cost of a small degradation in stationary background regions. MBs with high texture and motion information are prioritized as regions of interest (that capture the attention of the human eye) and QP_{dec} is lowered in that case (see Eq. A.5.13). On the contrary, QP_{dec} is increased for homogeneous and stationary MBs, which are not of high user interest (see Eq. A.5.14). Afterwards, QP_{dec} is clipped between ± 4 and added in QP_{BU} to get QP_{MB}.

A.3 Evaluation and Results

The proposed multi-level Rate Control (RC) scheme is compared with various RC modes (especially RC-Mode-3 [LT08] which is the latest one to handle multiple frame types and offers better quality than other RC Modes) of H.264. For a pure video quality comparison following test conditions are considered: *exhaustive RDO-MD*, UMHexagonS, 16 search range, 1 reference frame, GOP = 100, CAVLC, using different coding structures. The thresholds and test conditions presented in Sect. 4.3.2 are used for the following experiments. Figure A.6 shows the R-D curves for Carphone (QCIF, IPPP) and American Football (SIF, IBBP). Figure A.6 shows that the multi-level RC scheme achieves always better PSNR (avg. 1 and 0.5 dB) than RC-Mode-3.

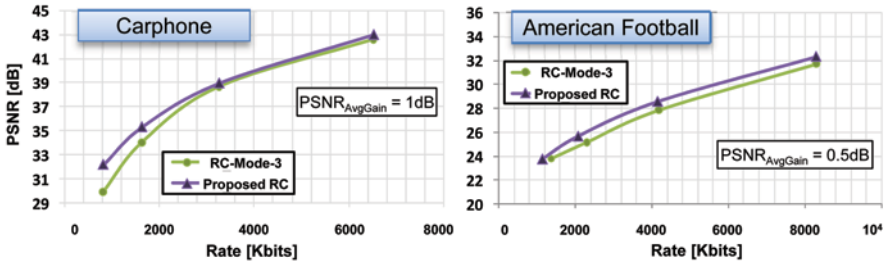


Fig. A.6 RD-curves comparison of the proposed multi-level RC with RC-mode-3 for carphone (QCIF, IPPP) and American Football (SIF, IBBP)

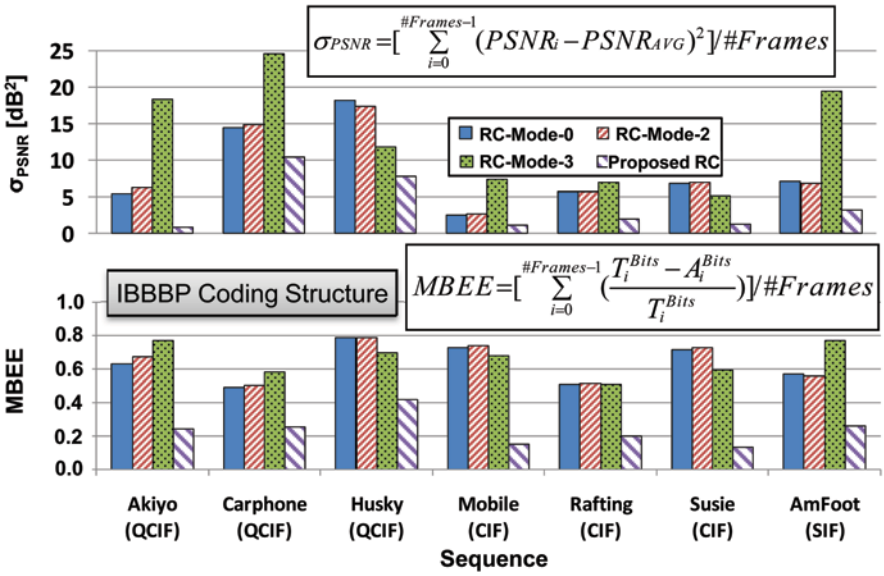


Fig. A.7 MBEE comparison of the multi-level RC with three different RC modes

Figure A.7 shows the *Mean Bit Estimation Error* (MBEE) and the *PSNR variation* (σ_{PSNR}) for encoding various sequences using the multi-level RC and RC-Mode-0,2,3. Each bar is the averaged (MBEE, σ_{PSNR}) value over 7 bit rates (64 Kbps–4 Mbps).

Figure A.7 shows that the multi-level RC outperforms all RC modes in terms of the buffer and visual quality smoothness and achieves the target bit rate more accurately. Figure A.7 illustrates that the multi-level RC achieves up to 81.4, 81.9, and 77.8% (avg. 61.7, 62.3, and 63.9%) reduced MBEE compared to RC-Mode-0,2,3, respectively. Moreover, the multi-level RC provides up to 86, 87.9, and 95.9% (avg. 61.9, 62, and 72.4%) reduced σ_{PSNR} compared to RC-Mode-0,2,3, respectively.

Figure A.8 compares the PSNR and Rate of the multi-level RC with RC-Mode-3 on frame-basis when encoding a combination of Rafting and Football CIF sequences. This fast motion sequence contains scene cuts at every 50th frame. Compared to RC-Mode-3, the multi-level RC achieves on average 67% less σ_{PSNR} and 65.46% reduced MBEE. At the start, RC-Mode-3 performs well but requires a much higher amount (≈2×) of bits, therefore the overall RD ratio is similar to the multi-level RC. As soon as scene cuts occur, the quality of the RC-Mode-3 decreases. After 100 frames, RC-Mode-3 is already worse than the multi-level RC and after 150 frames, the quality of the RC-Mode-3 degrades severely. After 15–20 frames the multi-level RC achieves a smooth buffer fullness and less σ_{PSNR}. After 100 frames, it achieves an always better PSNR.

Figure A.9 presents frame-wise PSNR and Rate comparison of the multi-level RC scheme with RC-Mode-0 for encoding the adapted Susie sequence for checking the robustness of RCs. During the first 60 frames, the multi-level RC achieves a slightly better PSNR while requiring almost the same amount of bits. From frame 61 onwards, the frames contain heavy noise. The multi-level RC scheme recognizes

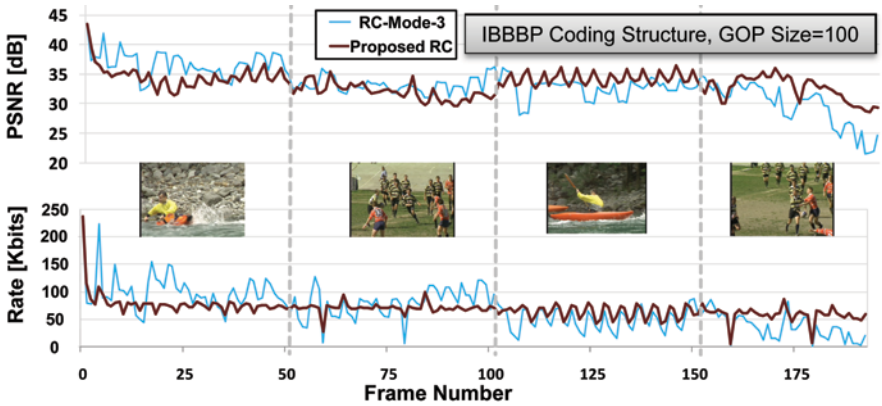


Fig. A.8 Frame-wise comparison of the multi-level RC with RC-mode-3 for fast motion combined CIF sequences encoded at 2 Mbps@30 fps

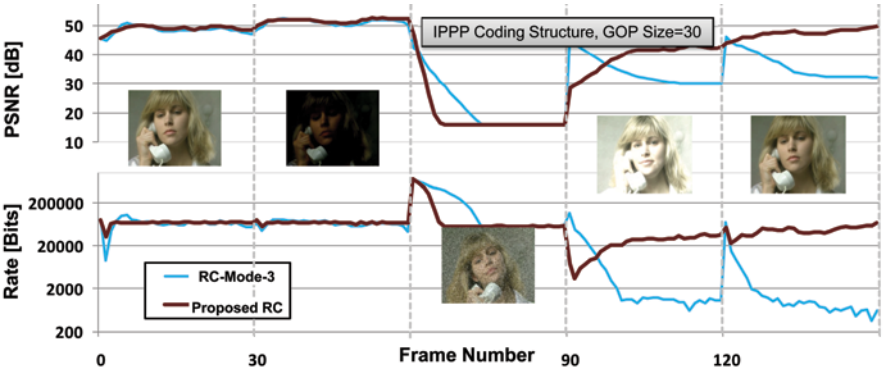


Fig. A.9 Frame-wise comparison of the multi-level RC with RC-mode-0 for Susie mixed CIF sequence (*Bright*, *Dark*, *Noisy*) at 2 Mbps@30 fps

this fact and adjusts faster to the target bit budget without wasting extra bits. On the contrary, RC-Mode-0 adjusts itself slower (frames 60–75) to the target bit budget when facing the transitions from dark-to-noisy frames. As a result, RC-Mode-0 suffers from lower PSNR in the subsequent frames and gives high σ_{PSNR}. Overall, RC-Mode-0 achieves an average PSNR of 38.21 dB while the multi-level RC achieves 41.18 dB (i.e., a gain of 2.97 dB).

The efficiency of image-statistics and motion based Macroblock Prioritizations can be seen in Fig. A.10 that shows the 14th reconstructed frame of American Football when encoded with the multi-level RC (Left) and RC-Mode-0 (Right). The multi-level RC encodes the moving helmets and arms of the players (eye-catching regions) with better quality compared to RC-Mode-0 while blurring the grassy background, which is less important than the players. Therefore, the proposed multi-level RC scheme is superior in terms of user interests.

Fig. A.10 Evaluating the image and motion based MB prioritizations (Note: All excerpts are 2× zoomed using nearest neighbor interpolation)

Complexity: On Intel Core2Duo T5500 (1.66 GHz), on average, the multi-level RC requires 0.54 MCycles while RC-Mode-0 requires 9 MCycles for encoding one frame, i.e., the multi-level RC is 16.6× faster than RC-Mode-0.

Appendix B: Simulation Environment the H.264 Video Encoder Demonstration

This appendix presents the simulation environment (used in this monograph) and the demonstration of the in-house developed H.264 video encoder (in the scope of this monograph) on RISPP dynamically reconfigurable processor and Texas Instruments' DM6437 Digital Media Processor. For researching the adaptive low-power reconfigurable processor architectures, the simulator for dynamically reconfigurable processors [Bau09] was extended with run-time energy management modules. Before moving to the details of the video encoder demonstration, the simulation methodology is described in the following.

B.1 Implementation and Simulation Environment

The implementation and simulation environment consists of (a) *ArchC-Simulator* for a SPARC-V8 architecture [ARB+05] in order to generate a branch trace of the application and functional testing on the core processor, (b) Simulator for dynamically reconfigurable processors (in this case it is a *RISPP Simulator* [Bau09]), (c) Xilinx ISE for implementing the Data Paths and ModelSim for simulations in order to perform functional testing, (d) *gcc* compiler on a Linux machine for PC-based

evaluation. The simulation methodology is partitioned into four phases, (1) Design phase, (2) Implementation phase, (3) Power Measurement and Estimation phase, and (4) Simulation phase.

In the **design phase**, first the application is executed in the *ArchC-Simulator* and the output is compared with the output of the original *PC-based execution* (i.e., using the original target platform of the application). When compiling for the *ArchC-Simulator* several modifications might be required, e.g., it is not possible to use the same name for global variables and methods twice. Afterwards, the application is executed in the *ArchC-Simulator*, to verify the correct output. For designing the Custom Instructions (CIs), information about the computational hot spot(s) of the application is required. This is obtained by profiling the application using the '*valgrind tool suite*' [Net04a, b]. After gathering this information CIs for the hot spots are designed, while considering the constraints predetermined by the architecture (i.e., RISPP [Bau09] in this case).

In the **implementation phase**, the *ArchC-Simulator* and the *RISPP Simulator* are made aware of the new CIs (by adding, e.g., name and opcode to an XML-file containing the information about all CIs). The CIs are programmed as assembly instructions in the application and the data structures are adapted accordingly for integrating the CIs. Furthermore, the behavior of CIs is added to the *ArchC-Simulator* for functional correctness and valid output generation. The modified application is tested on the extended simulator and in order to assure correct functionality the output of the original and modified applications are compared. Moreover, the Data Paths are implemented in VHDL. The behavior of Data Paths is simulated and finally the VHDL-Code is synthesized using the Xilinx ISE tool chain in order to provide information about the hardware requirements (number of flip flops, slices, frequency, etc.). This information is added to the XML-file which is provided to the *RISPP Simulator*. Additionally the functionality of the complete CI and the composing Data Paths is implemented in software (i.e., using the core instruction set architecture, cISA) and the execution times are measured. In the next step, the CI graph is generated showing the Data Paths as nodes connected with edges (repre-

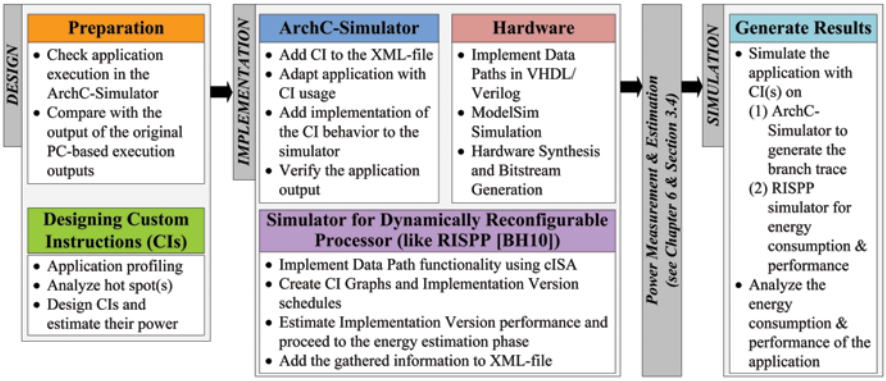


Fig. B.1 Simulation methodology showing various steps of the simulation procedure

senting the connections). This graph is also fed as an input to the *RISPP Simulator*. An in-house developed tool automatically generates schedules for various Implementation Versions considering different area constraints (i.e., different number of given Data Path Containers, DPCs). The information is stored in the data structures internal to the *RISPP Simulator*.

The **power estimation and measurement phase** is explained in Chap. 6 and Sect. 3.4. The measured power of the Data Paths is added to the XML-file that contains the area and latency information of each Data Path. Moreover, the average power and energy consumption of different Implementation Versions is estimated using the proposed power model (see details in Sect. 3.4), which is later on stored in the Custom Instructions data structure of the *RISPP Simulator*.

In the **simulation phase**, the application is simulated using the *ArchC-Simulator* which provides output files (e.g., branch trace) that serves as the input to the *RISPP Simulator* that provides an estimate of the energy consumption and performance of the application. At the end the results (energy consumption, performance, etc.) are analyzed. To investigate the concepts and algorithms for run-time energy management with CI-level muting (as proposed in the scope of this monograph), the *RISPP Simulator* [BSH09a] is extended with several new modules. These modules are (a) Run-Time Energy Management, (b) Power-Estimation, and (c) Muting Manager (see Fig. B.2). Besides the application binary, branch trace, and the core instruction set architecture (cISA), the power model for dynamically reconfigurable processors

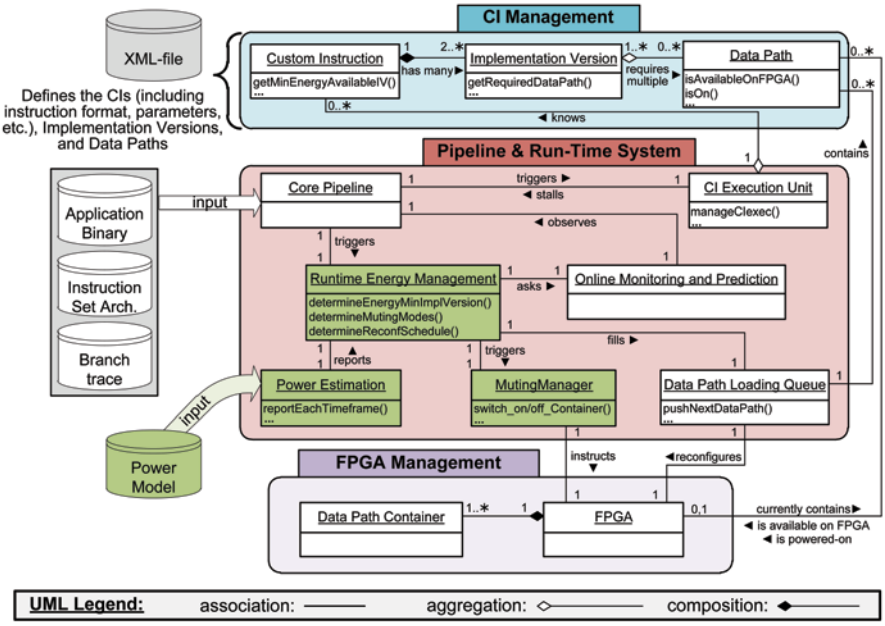


Fig. B.2 Reconfigurable processor simulator with the extensions implemented in the scope of this monograph for run-time adaptive energy-management

(see details in Sect. 3.4) is passed as input. Figure B.2 shows the extended simulator as a UML class diagram that consists of three major parts:

- the pipeline of the core processor and the run-time system with energy management scheme; it simulates the pipeline behavior and manages the executions, energy management, power estimation, reconfigurations, etc.
- the Custom Instructions (CIs) with their composing Data Paths and various Implementation Versions; it is represented by a data structure containing the performance and energy properties of different Data Paths and Implementation Versions, etc., and
- the FPGA with various DPCs with management of the Data Paths loaded in the DPCs, the muting mode of different DPCs, etc.

The information about the CIs, Implementation Versions (like name, CI opcode and instruction format, latency and average power consumption, etc.) and their composing Data Paths is fed through an XML-file (as discussed above). Moreover, the area and power information about the Data Paths is also provided in this XML-file. The pipeline simulates the application binary and the branch trace is considered to imitate the control flow of the application. In the current setup, the model of a SPARC-V8 architecture is modeled with five pipeline stages. Note that the register file contents and the data memory accesses are not simulated in the *RISPP Simulator* (see further details in [Bau09]). Each load and store instruction requires two cycles considering a 100% cache hit.

When a *Forecast Instruction* is encountered to hint about the upcoming CIs, the run-time energy management scheme is triggered that determines the energy minimizing instruction set and the appropriate muting modes for the set of temporarily unused CIs. The information about the execution frequency of CIs is obtained from the Online Monitoring and Prediction module. The corresponding muting mode is then sent to the Muting Manager module that issues the shutdown signals to the Logic and/or Configuration SRAM of the DPCs corresponding to the muted CIs. The energy consumption of the application is estimated at run time at different time intervals, between two *Forecast Blocks*, or after the complete application execution. After the muting mode is configured, the Data Paths to be reconfigured are pushed into the Data Path Loading Queue (see Fig. B.2) and the reconfiguration sequence is determined. The CI Execution Unit controls the execution of CIs using cISA or using the available Implementation Versions.

B.2 H.264 Video Encoder on the RISPP Hardware Prototype

Figure B.3a shows the H.264 video encoder (developed in the scope of this monograph) executing on the RISPP hardware prototype (based on an Avnet Xilinx Virtex-4 LX160 Development Kit; "ADS-XLX-V4LX-DEV160-G", [Avn09])[1]. The

[1] This board contains a Xilinx Virtex-4 XC4VLX160-FF1513 FPGA [Xil08b].

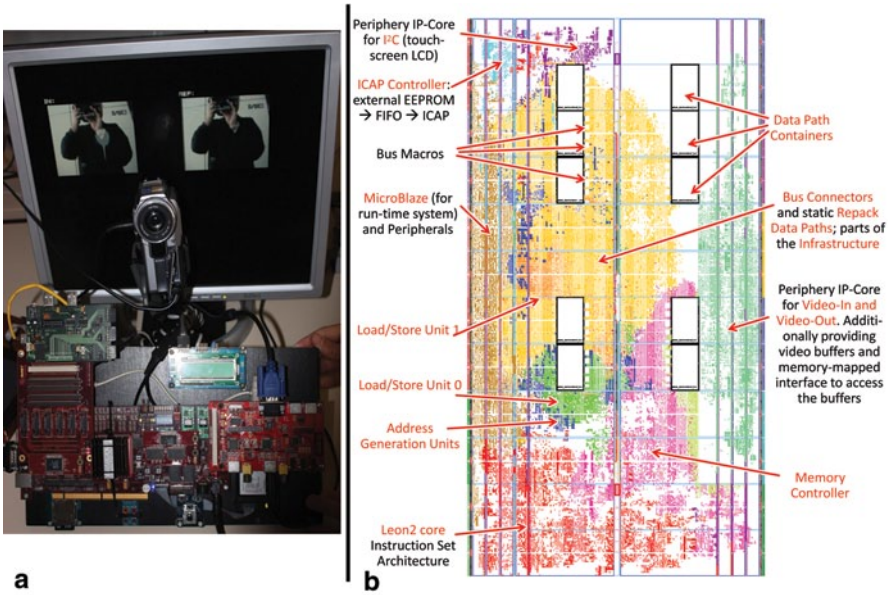


Fig. B.3 **a** H.264 video encoder executing on the RISPP prototype; **b** Floorplan of the RISPP prototype implementation on the Xilinx Virtex-4 LX 160 FPGA

internal floorplan (after place & route) of the RISPP processor prototype (executing at 50 MHz) with the video preprocessing IP-core is provided in Fig. B.3b [Bau09]. First the raw (RGB) video in an interlaced format is obtained from the camera. The video preprocessing core performs the de-interlacing, format conversion (RGB to YUV 4:4:4), and color sub-sampling (YUV 4:4:4 to YUV 4:2:0). A triple circular buffer mechanism is implemented that provides storage for the current and reference video frames along with the next frame written by the camera (while the current frame is being encoded). The CIs (e.g., for Motion Estimation) access the current and the reference frame buffers using one 128-bit port for each buffer. After the current frame is encoded and overwritten by the reconstructed Macroblocks data, the buffer rotation is performed, i.e., the next frame becomes the current, the current frame becomes the reference, and the reference frame buffer becomes the next frame in which the camera writes the new data. Note, this rotation is performed in hardware in order to simplify the software implementation [Bau09]. The address of the current and reference frames are unchanged. The reference frame is displayed via a VGA output periphery module. The main encoder program is executing on the core processor (in this case Leon2 core pipeline), while the Data Paths for the Custom Instructions (CIs) are loaded on the DPCs. It can be noticed in Fig. B.3b that there are currently 10 DPCs connected with Bus Connectors. The run-time system executes on the MicroBlaze. Further details on the RISPP prototype can be found in [Bau09].

B.3 H.264 Video Encoder on the Texas Instruments' DM6437 Digital Media Processor

Figure B.4 demonstrates the H.264 video encoder (developed in the scope of this monograph) executing on the Texas Instruments' (TI) DM6437 Digital Video Development Platform (TMDSVDP6437) [Ins08a] with TMS320DM6437 Digital Media Processor [Ins08b]. Figure B.5 illustrates the processing flow of different functional blocks (video capture, format and resolution conversion, encoding, and video display) of the video recording system executing on the TMDSVDP6437 platform. The step-by-step flow is explained in the following.

- The video data from charge-coupled device (CCD) is obtained by the *Video Processing Front End* (VPFE) driver using an on-board tvp5146 decoder. The format of the captured video is YUV 4:2:2 interleaved with a resolution of 720×576 (D1, PAL).
- To support various video resolutions, down-scaling is performed as an optional step using the *Resizer* module of VPFE. For instance, in the demonstration of Fig. B.4, the input video is down-scaled from 720×576–352×288 (Common Intermediate Format, CIF) resolution. The resizer is a hardware implemented poly-phase filter for image scaling operations with a capability of scaling up to four times. Note, the choice of the filter type (four-phase seven-tap filter or eight-phase four-tap filter) is done automatically by the hardware based on the scaling ratio and it is not changeable by the software. The resizer can operate on either YUV 4:2:2 interleaved format or separated single color plane.
- The video encoders typically require videos in YUV 4:2:0 format. Therefore, the format of the input video is converted to YUV 4:2:0 planar using EDMA3

Fig. B.4 H.264 video encoder executing on the TI' DM6437 DSP board

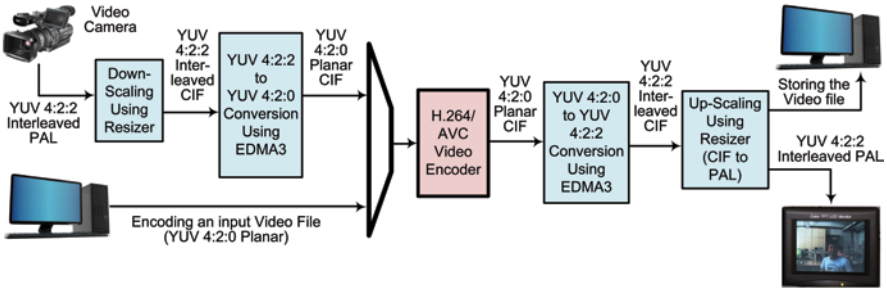


Fig. B.5 Flow for porting H.264 Encoder on DM6437 digital signal processor

(Enhanced Direct Memory Access) module. EDMA3 provides user-programmed data transfers between two memory-mapped slave endpoints on the device.

- Afterwards, the video encoder (executing on the core DM6437 processor with C64x+instruction set and DaVinci video technology) processes the video frame for encoding. Various modules are optimized using the specialized assembly with operations of sub-word level processing.
- The reconstructed video is displayed on the TFT LCD monitor. For displaying, the format of the reconstructed video is converted from YUV 4:2:0 planar to YUV 4:2:2 interleave using the EDMA3 module. This format is required by the *Video Processing Back End* (VPBE). Afterwards, the video is up-scaled to 720×576 (D1, PAL) and sent to display using the VPBE driver.

Appendix C: The CES Video Analyzer Tool

In the scope of this monograph, a video analysis tool (Fig. C.1) has been developed named "CES Video Analyzer" in order to analyze the subjective quality of various algorithms. Moreover, it is also used to subjectively learn about the relationship between the optimal coding modes and various video properties. The CES Video Analyzer tool has various features like playback of raw YUV video files, computing and displaying spatial and temporal video properties, a framework for researching new Motion Estimators. The fetures of the CES Video Analyzer tool are described in the following.

- Playback of raw YUV video files of different resolutions with different frame rates (see Label ① in Fig. C.1)
  - possibility to view separate components of video frames Y, U and V
  - open and playback of multiple video files for comparison in a frame-wise synchronized fashion
- Extract, display, and output the properties of a video (Gradient, Variance, Texture, Brightness, Contrast, etc.) at frame and/or Macroblock levels (see Label ⑧)

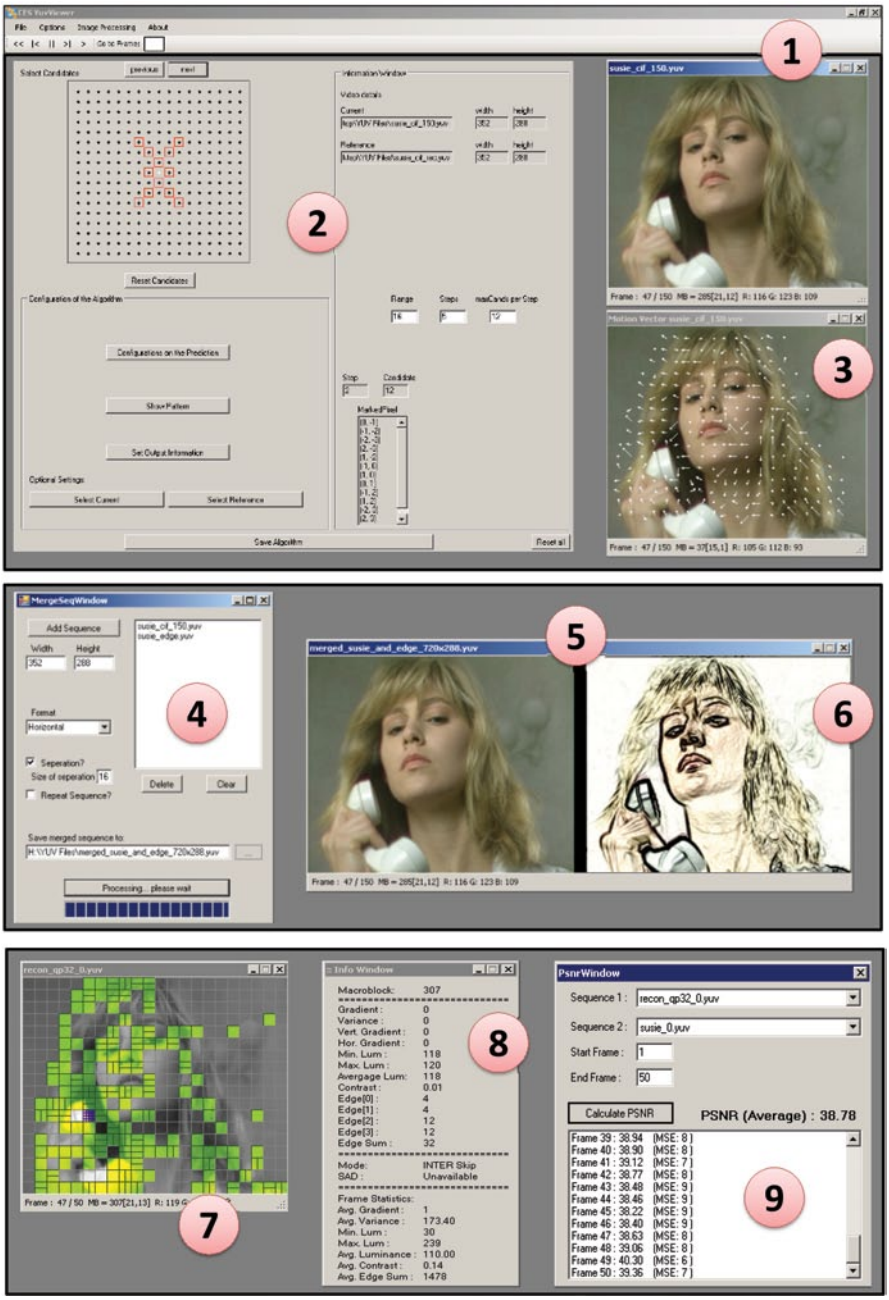


Fig. C.1 The CES video analyzer tool showing the research framework for motion estimation, video merging, and texture analysis

  - display the edges in different colors or threshold based edge coloring
  - display the edge maps for texture analysis of various edge-detection algorithms (see Label ⑥)

- Coding mode distribution analysis (see Label ⑦)
- A framework for researching and evaluating different Motion Estimators (see Label ②)
  - multiple Motion Estimation stages can be defined with Initial Search Point Prediction and pattern types
  - different patterns can be easily configured and compared
  - search range can be configured
  - the final configuration can be stored in the list of pre-stored Motion Estimators
- Comparing different Motion Estimators
  - plotting the motion vectors for subjective motion analysis (see Label ③)
  - color of motion vectors can be selected
  - different standard Motion Estimation algorithms are provided for comparison (Full Search, Spiral Search, Three Step Search, and UMHexagonS)
  - output the information about the motion vectors in a text file (comma separated format)
- Computing the Peak Signal-to-Noise Ratio (PSNR) at frame or video level (see Label ⑨)
- Create new test video sequences with different brightness, noise, blur factors
  - save the complete video file or a specified set of frames
- Create new test video sequences by merging different video sequences in order to realize scene cuts and videos with diverse properties within a given frame (see Label ④)
  - a border between the video frames can be added (see Label ⑤)
  - the color and size of the border between the frames can be selected
  - the video after the last frame can be stopped or replayed from the start in case videos with different number of frames are merged with each other
- Zoom/Upscale using different filters
- Save individual frames in different formats (Bitmap, Jpeg, Png, Gif, Tiff or Windows Metafile)

Kindly note that this tool is actively used in further research projects. It is developed to help the research community (researchers, developers, students, etc.) of embedded multimedia systems in their research and educational projects (i.e., to perform quick analysis/evaluation of videos and different algorithms). Further extensions of this tool are to support the playback and analysis of multiview video sequence and analyzing various video pre- and post-processing filters for video quality enhancement and restoration.

Bibliography

[ADVLN05] E. Arsura, L. Del Vecchio, R. Lancini, and L. Nisti, "Fast macroblock intra and inter modes selection for h.264/avc", in *Proceedings of the 2005 International Conference on Multimedia and Expo (ICME)*, July 2005, pp. 378–381.

[Ae06] K. Agarwal and et, "Power gating with multiple sleep modes", in *IEEE International Symposium on Quality Electronic Design (ISQED)*, 2006, pp. 633–637.

[Aer] Aeroflex Gaisler, "Homepage of the Leon processor", http://www.gaisler.com/leonmain.html.

[AKL+07] C. Arbelo, A. Kanstein, S. Lopez, J. F. Lopez, M. Berekovic, R. Sarmiento, and J.-Y. Mignolet, "Mapping control-intensive video kernels onto a coarse-grain reconfigurable architecture: the h.264/avc deblocking filter", in *Proceedings of the 10th conference on Design, Automation and Test in Europe (DATE)*, April 2007, pp. 1–6.

[Ama06] H. Amano, "A survey on dynamically reconfigurable processors", *IEICE Transaction on Communication*, vol. E89-B, no. 12, pp. 3179–3187, December 2006.

[AML07] E. Akyol, D. Mukherjee, and Y. Liu, "Complexity control for real-time video coding", in *Proceedings of the 2007 International Conference on Image Processing (ICIP)*, October 2007, pp. I–77–I–80.

[AN04] J. H. Anderson and F. N. Najm, "Power estimation techniques for fpgas", *IEEE Transaction on Very Large Scale Integration (TVLSI)*, vol. 12, no. 10, pp. 1015–1027, 2004.

[ARB+05] R. Azevedo, S. Rigo, M. Bartholomeu, G. Araujo, C. Araujo, and E. Barros, "The ArchC architecture description language and tools", *International Journal of Parallel Programming*, vol. 33, no. 5, pp. 453–484, October 2005.

[ARC] ARC International, "ARCtangent processor", http://www.arc.com/configurables/.

[Ari08] Arizona State University, "Video Traces Research Group", http://trace.eas.asu.edu/yuv/index.html, 2008.

[ASI] ASIP Solutions, Inc., "Homepage of ASIP Meister", http://asip-solutions.com/.

[Avn09] Avnet, Inc., "Avnet electronics marketing", http://avnetexpress.avnet.com, 2009.

[Bau09] L. Bauer, "Rispp: A run-time adaptive reconfigurable embedded processor", in *PhD Dissertation, University of Karlsruhe, Germany*, December 2009.

[Ber09] C. V. Berkel, "Multi-core for mobile phones", in *Proceedings of the 12th conference on Design, Automation and Test in Europe (DATE)*, April 2009, pp. 1260–1265.

[BHU03] J. Becker, M. Huebner, and M. Ullmann, "Power estimation and power measurement of xilinx virtex fpgas: Trade-offs and limitations", in *Symposium on Integrated Circuits and Systems Design (SBCCI)*, 2003, pp. 283–288.

M. Shafique, J. Henkel, *Hardware/Software Architectures for Low-Power Embedded Multimedia Systems,*
DOI 10.1007/978-1-4419-9692-3, © Springer Science+Business Media LLC 2011

[BKD+05] M. Berekovic, A. Kanstein, D. Desmet, A. Bartic, B. Mei, and J. Mignolet, "Mapping of video compression algorithms on the adres coarse-grain reconfigurable array", in *Workshop on Multimedia and Stream Processors*, November 2005.

[BL00] F. Barat and R. Lauwereins, "Reconfigurable instruction set processors: A survey", in *Proceedings of the 11th IEEE International Workshop on Rapid System Prototyping (RSP)*, June 2000, pp. 168–173.

[Bob07] C. Bobda, *Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications*. Springer Publishing Company, Incorporated, June 2007.

[BSH08a] L. Bauer, M. Shafique, and J. Henkel, "A computation- and communication- infrastructure for modular special instructions in a dynamically reconfigurable processor", in *18th International Conference on Field Programmable Logic and Applications (FPL)*, September 2008, pp. 203–208.

[BSH08b] L. Bauer, M. Shafique, and J. Henkel, "Efficient resource utilization for an extensible processor through dynamic instruction set adaptation", *IEEE Transactions on Very Large Scale Integration Systems (TVLSI), Special Section on Application-Specific Processors*, vol. 16, no. 10, pp. 1295–1308, October 2008.

[BSH08c] L. Bauer, M. Shafique, and J. Henkel, "Run-time instruction set selection in a transmutable embedded processor", in *Proceedings of the 45th annual Conference on Design Automation (DAC)*, June 2008, pp. 56–61.

[BSH09a] L. Bauer, M. Shafique, and J. Henkel, "Cross-architectural design space exploration tool for reconfigurable processors", in *Proceedings of the 12th conference on Design, Automation and Test in Europe (DATE)*, April 2009, pp. 958–963.

[BSH09b] L. Bauer, M. Shafique, and J. Henkel, "Mindeg: A performance-guided replacement policy for run-time reconfigurable accelerators", in *IEEE International Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS)*, October 2009, pp. 335–342.

[BSKH07] L. Bauer, M. Shafique, S. Kramer, and J. Henkel, "RISPP: Rotating Instruction Set Processing Platform", in *Proceedings of the 44th annual Conference on Design Automation (DAC)*, June 2007, pp. 791–796.

[BSKH08] L. Bauer, M. Shafique, S. Kreutz, and J. Henkel, "Run-time system for an extensible embedded processor with dynamic instruction set", in *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*, March 2008, pp. 752–757.

[BSTH07] L. Bauer, M. Shafique, D. Teufel, and J. Henkel, "A self-adaptive extensible embedded processor", in *First International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, July 2007, pp. 344–347.

[BTM00] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations", in *International Symposium on Computer Architecture (ISCA)*, 2000, pp. 83–94.

[CC01] J. E. Carrillo and P. Chow, "The effect of reconfigurable units in superscalar processors", in *Proceedings of the ACM/SIGDA eighth international symposium on Field Programmable Gate Arrays (FPGA)*, February 2001, pp. 141–150.

[CC07] C.-M. Chen and C.-H. Chen, "An efficient pipeline architecture for deblocking filter in h.264/avc", *IEICE Transactions on Information and Systems*, vol. E90-D, no. 1, pp. 99–107, 2007.

[CCH+06] 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 Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 16, no. 6, pp. 673–688, 2006.

[CCLR07] N. Cherniavsky, A. C. Cavender, R. E. Ladner, and E. A. Riskin, "Variable frame rate for low power mobile sign language communication", in *Proceedings of the 2007 ACM SIGACCESS Conference on Computers and Accessibility (ASSETS)*, October 2007, pp. 163–170.

[CCT+09] 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 Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 19, no. 8, pp. 1118–1128, 2009.

[CCW+09] 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 Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 19, no. 12, pp. 1739–1754, 2009.

[CH02] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software", *ACM Computing Surveys (CSUR)*, vol. 34, no. 2, pp. 171–210, June 2002.

[CHC03] B. Calhoun, F. Honore, and A. Chandrakasan, "Design methodology for fine-grained leakage control in mtcmos", in *Proceedings of the 2003 International Symposium on Low Power Electronics and Design (ISLPED)*, August 2003, pp. 104–109.

[CJMP03] S. Choi, J.-W. Jang, S. Mohanty, and V. K. Prasanna, "Domain-specific modeling for rapid energy estimation of reconfigurable architectures", *The Journal of Supercomputing*, vol. 26, pp. 256–281, 2003.

[CK08] Y.-K. Chen and S. Y. Kung, "Trend and challenge on system-on-a-chip designs", *Journal of VLSI Signal Processing Systems (JSPS)*, vol. 53, no. 1–2, pp. 217–229, 2008.

[CLC06] T. C. Chen, C. J. Lian, and L. G. Chen, "Hardware architecture design of an h.264/avc video codec", in *Asia and South Pacific Conference on Design Automation (ASP-DAC)*, 2006, pp. 750–757.

[CLZG06] Y.-K. Chen, E. Q. Li, X. Zhou, and S. L. Ge, "Implementation of h.264 encoder and decoder on personal computers", *Journal of Visual Communications and Image Representations (JVCIR)*, vol. 17, no. 2, pp. 509–532, April 2006.

[CoW] CoWare Inc., "LISATek", http://www.coware.com/.

[CWL+05] L. Cheng, P. Wong, F. Li, Y. Lin, and L. He, "Device and architecture co-optimization for fpga power reduction", in *Proceedings of 42nd ACM IEEE Design Automation Conference (DAC)*, 2005, pp. 915–920.

[CZH02] Z. Chen, P. Zhou, and Y. He, "Fast integer pel and fractional pel motion estimation for jvt", in *Joint Video Team (JVT) of ISO/IECMPEG & ITU-T VCEG 6th Meeting*, December 2002, pp. Document JVT–F017.

[Dal99] M. Dales, "The Proteus processor—a conventional cpu with reconfigurable functionality", in *Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications (FPL)*, August 1999, pp. 431–437.

[Dal03] M. Dales, "Managing a reconfigurable processor in a general purpose workstation environment", in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, March 2003, pp. 980–985.

[Das99] I. Das, "On characterizing the 'knee' of the pareto curve based on normal-boundary intersection", *Structural and Multidisciplinary Optimization*, vol. 13, no. 3, pp. 107–115, 1999.

[DGHJ05] L. Deng, W. Gao, M. Hu, and Z. Z. Ji, "An efficient hardware implementation for motion estimation of avc standard", *IEEE Transactions on Consumer Electronics (TCE)*, vol. 51, no. 4, pp. 1360–1366, November 2005.

[Esp04] M. Esponda, "Trends in hardware architecture for mobile devices", in *Institut für Informatik, Freie Universität Berlin*, November 2004.

[EY05] M. Etoh and T. Yoshimura, "Advances in wireless video delivery", *Proceedings of the IEEE*, vol. 93, no. 1, pp. 111–122, 2005.

[FHR+10] G. Frantz, J. Henkel, J. Rabaey, T. Schneider, M. Wolf, and U. Batur, "Ultra-low power signal processing", *IEEE Signal Processing Magazine*, vol. 27, no. 2, pp. 149–154, 2010.

[Ge04] A. Gayasen and et, "Reducing leakage energy in fpgas using region-constrained placement", in *ACM Internaltional Symposium on Field Programmable Gate Arrays (FPGA)*, 2004, pp. 51–58.

[Ge07] A. H. Gholamipour and etAl, "Energy-aware co-processor selection for embedded processors on fpgas", in *International Conference on Computer DDesign (ICCD)*, 2007, pp. 158–163.

[GE08] M. Goraczko and EtAl, "Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems", in *Proceedings of 45th ACM IEEE Design Automation Conference (DAC)*, 2008, pp. 191–196.

[GRE+01] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "MiBench: A free, commercially representative embedded benchmark suite", in *Annual IEEE International Workshop Workload Characterization (WWC)*, December 2001, pp. 3–14.

[GW02] R. C. Gonzales and R. E. Woods, *Digital Image Processing*. Upper Saddle River, New Jersey, USA: Prentice-Hall Inc., 2002.

[GY05] C. Grecos and M. Y. Yang, "Fast inter mode prediction for p slices in the h264 video coding standard", *IEEE Transactions on Broadcadting (TB)*, pp. 256–263, 2005.

[Har01] R. Hartenstein, "A decade of reconfigurable computing: a visionary retrospective", in *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*, March 2001, pp. 642–649.

[Hen03] J. Henkel, "Closing the soc design gap", *IEEE Computers*, vol. 36, no. 9, pp. 119–121, September 2003.

[HL07] P.-A. Hsiung and C.-W. Liu, "Exploiting hardware and software low power techniques for energy efficient co-scheduling in dynamically reconfigurable systems", in *17th International Conference on Field Programmable Logic and Applications (FPL)*, 2007, pp. 165–170.

[HLLW08] C. H. Ho, P. H. W. Leong, W. Luk, and S. J. E. Wilton, "Rapid estimation of power consumption for hybrid fpgas", in *18th International Conference on Field Programmable Logic and Applications (FPL)*, 2008, pp. 227–232.

[HM09] H. P. Huynh and T. Mitra, "Runtime adaptive extensible embedded processors—a survey", in *Proceedings of the 9th International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, July 2009, pp. 215–225.

[HP07] H.Alzoubi and W. D. Pan, "Efficient global motion estimation using fixed and random subsampling patterns", in *Proceedings of the 2007 International Conference on Image Processing (ICIP)*, October 2007, pp. I–477–I–480.

[Hui10] Hui Yong Kim, "Next generation video coding standardization", http://www.itforum.kr/board/include/download.php?no=144&db=board3&fileno=2, 2010.

[Ins08a] T. Instruments, "TMDSVDP6437: DM6437 Digital Video Development Platform", http://focus.ti.com/docs/toolsw/folders/print/tmdsvdp6437.html, 2008.

[Ins08b] T. Instruments, "TMS320DM6437 Digital Media Processor", http://focus.ti.com/docs/prod/folders/print/tms320dm6437.html, 2008.

[ITI] ITIV & CES, "KAHRISMA: KArlsruhe's Hypermorphic Reconfigurable-Instruction-Set Multi-grained-Array processor", http://www.kahrisma.org/.

[ITU05] ITU-T Rec. H.264 and ISO/IEC 14496-10:2005 (E) (MPEG-4 AVC), "Advanced video coding for generic audiovisual services", 2005.

[ITU09] ITU-T Rec. H.264 and ISO/IEC 14496-10:2005 (E) (MPEG-4 AVC), "Advanced video coding for generic audiovisual services", 2009.

[JC99] J. A. Jacob and P. Chow, "Memory interfacing and instruction specification for reconfigurable processors", in *Proceedings of the ACM/SIGDA 7th international symposium on Field Programmable Gate Arrays (FPGA)*, February 1999, pp. 145–154.

[JC04] X. Jing and L.-P. Chau, "Fast approach for h.264 inter mode decision", in *Electronic Letters*, 2004, pp. 1050–1052.

[JL03] B. W. Jeon and J. Y. Lee, "Fast mode decision for h.264", in *Joint Video Team (JVT) of ISO/IECMPEG & ITU-T VCEG 8th Meeting*, 2003, pp. Document JVT–J033.

[JM05] M. A. Johnson and M. H. Moradi, *PID Control: New Identification and Design Methods*. New York, NY, USA: Springer-Verlag New York, Inc., 2005.

[Joi08] Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, "JVT-AB204: Joint draft 8.0 on multiview video coding", 2008.

[Joi10] Joint Collaborative Team (JCT) on Video Coding Standard Development, "H.265: High efficiency video coding", http://www.h265.net/2010/01/final-call-for-proposals-on-hngvc-hvc-issued-jointly-by-vceg-and-mpeg.html, 2010.

[JVT10] JVT, "H.264 codec", http://iphome.hhi.de/suehring/tml/index.htm, 2010.

[KBS+10] R. König, L. Bauer, T. Stripf, M. Shafique, W. Ahmed, J. Becker, and J. Henkel, "KAHRISMA: A novel hypermorphic reconfigurable-instruction-set multi-grained-array architecture", in *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*, March 2010, pp. 819–824.

[KC07] B.-G. Kim and C.-S. Cho, "A fast inter-mode decision algorithm based on macroblock tracking for p slices in the h.264/avc video standard", in *Proceedings of the 2007 International Conference on Image Processing (ICIP)*, October 2007, pp. V–301–V–304.

[KL07] H. Kalva and J.-B. Lee, "The vc-1 video coding standard", *IEEE Transactions on Multimedia (TM)*, vol. 14, no. 4, pp. 88–91, October-December 2007.

[Kle10] M. Klein, "Wp298: Power consumption at 40 and 45 nm", http://www.xilinx.com/support/documentation, 2010.

[KLHS06] S. D. Kim, J. H. Lee, C. J. Hyun, and M. H. Sunwoo, "Asip approach for implementation of h.264/avc", in *Asia and South Pacific Conference on Design Automation (ASP-DAC)*, Jan 2006, pp. 758–764.

[KR07] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics", *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 26, no. 2, pp. 203–215, 2007.

[KRD+03] U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, J. H. Ahn, P. Mattson, and J. D. Owens, "Programmable stream processors", *IEEE Transaction on Computer (TC)*, vol. 3, no. 8, pp. 54–62, 2003.

[KSK06] M. G. Koziri, G. I. Stamoulis, and I. X. Katsavounidis, "Power reduction in an h.264 encoder through algorithmic and logic transformations", in *Proceedings of the 2006 ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, October 2006, pp. 107–112.

[KXVK06] C. Kim, J. Xin, A. Vetro, and C.-C. J. Kuo, "Complexity scalable motion estimation for h.264/avc", in *Proceedings of the 2006 SPIE Visual Communications and Image Processing (VCIP)*, January 2006, pp. 109–120.

[LBM+06] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford, "Enhanced architectures, design methodologies and CAD tools for dynamic reconfiguration of Xilinx FPGAs", in *Proceedings of the 16th International Conference on Field-Programmable Logic and Applications (FPL)*, August 2006, pp. 1–6.

[LCHC03] F. Li, D. Chen, L. He, and J. Cong, "Architecture evaluation for power-efficient fpgas", in *ACM Internaltional Symposium on Field Programmable Gate Arrays (FPGA)*, 2003, pp. 175–184.

[LH02] Z. Li and S. Hauck, "Configuration prefetching techniques for partial reconfigurable coprocessor with relocation and defragmentation", in *Proceedings of 8th international symposium on Field Programmable Gate Arrays (FPGA)*, February 2002, pp. 187–195.

[LK06] W. H. Lee and J. H. Kim, "H.264 implementation with embedded reconfigurable architecture", in *IEEE International Conference on Computer and Information Technology (CIT)*, 2006, pp. 247–251.

[LL08] J. Lamoureux and W. Luk, "An overview of low-power techniques for field-programmable gate arrays", in *IEEE NASA/ESA Conference on Adaptive Hardware and Systems*, 2008, pp. 338–345.

[LLH07] F. Li, Y. Lin, and L. He, "Field programmability of supply voltages for fpga power reduction", *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 26, no. 4, pp. 752–764, 2007.

[LPL+03] Z. G. Li, F. Pan, K. P. Lim, G. Feng, X. Lin, and S. Rahardja, "Adaptive unit layer rate control for jvt", in *Joint Video Team (JVT) of ISO/IECMPEG & ITU-T VCEG 7th Meeting*, March 2003, pp. Document JVT–G012r1.

[LPMS97] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: A tool for evaluating and synthesizing multimedia and communications systems", in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, December 1997, pp. 330–335.

[LSV06] R. Lysecky, G. Stitt, and F. Vahid, "Warp processors", *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 11, no. 3, pp. 659–681, June 2006.

[LSW04] K.-P. Lim, G. Sullivan, and T. Wiegand, "Text description of joint model refer methods and decoding concealment methods", in *Joint Video Team (JVT) of ISO/IECMPEG & ITU-T VCEG Meeting*, March 2004, pp. Document JVT–K049.

[LT07] A. Leontaris and A. M. Tourapis, "Rate control reorganization in the jm (joint model) reference software", in *Joint Video Team (JVT) of ISO/IECMPEG & ITU-T VCEG 23rd Meeting*, April 2007, pp. Document JVT–W042.

[LT08] A. Leontaris and A. M. Tourapis, "Rate control for video coding with slice type dependencies", in *Proceedings of the 2008 International Conference on Image Processing (ICIP)*, October 2008, pp. 2792–2795.

[LWW+03] K. P. Lim, S. Wu, D. J. Wu, S. Rahardja, X. Lin, F. Pan, and Z. G. Li, "Fast inter mode selection", in *Joint Video Team (JVT) of ISO/IECMPEG & ITU-T VCEG 9th Meeting*, 2003, pp. Document JVT–I020.

[Lys07] R. Lysecky, "Low-power warp processors for power efficient high-performance embedded systems", in *Proceedings of the 10th conference on Design, Automation and Test in Europe (DATE)*, 2007, pp. 141–146.

[MAWL03] B. Meng, O. Au, C.-W. Wong, and H.-K. Lam, "Efficient intra-prediction mode selection for 4x4 blocks in h.264", in *Proceedings of the 2003 International Conference on Multimedia and Expo (ICME)*, July 2003, pp. III–521–III–524.

[May04] F. May, "Pact xpp virtual platform based on axys maxsim 5.0", in *PACT Corporation*, Revision 0.3 2004, pp. 12–12.

[MBNN10] J. Meehan, S. Busch, J. Noel, and F. Noraz, "Multimedia ip architecture trends in the mobile multimedia consumer device", *Elsevier Signal Processing: Image Communication (SPIC)*, vol. 25, no. 5, pp. 317–324, 2010.

[MC07] K.-Y. Min and J.-W. Chong, "A memory and performance optimized architecture of deblocking filter in h.264/avc", in *International Conference on Multimedia and Ubiquitous Engineering (MUE)*, 2007, pp. 220–225.

[Mic27] A. Michelson, *Studies in Optics*. Chicago, IL, USA: University of Chicago Press, 1927.

[Mic10a] Microsoft, "Audio video standard", http://www.avs.org.cn/en/, 2010.

[Mic10b] Microsoft, "The vc1 video coding standard", http://www.microsoft.com/windows/windowsmedia/howto/articles/vc1techoverview.aspx, 2010.

[MM05] S. Mondal and S. Memik, "Fine-grain leakage optimization in sram based fpgas", in *IEEE Great Lakes Symposium on VLSI (GLSVLSI)*, 2005, pp. 238–243.

[MMFS06] M. Martina, G. Masera, L. Fanucci, and S. Saponara, "Hardware co-processors for real-time and high-quality h.264/avc video coding", in *14th European Signal Processing Conference (EUSIPCO)*, 2006, pp. 200–204.

[MRS07] S. Momcilovic, N. Roma, and L. Sousa, "An asip approach for adaptive avc motion estimation", in *Proceedings of the IEEE Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*, July 2007, pp. 165–168.

[MSH+08] 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 Journal of Solid-State Circuits (JSSC)*, vol. 43, no. 11, pp. 2354–2362, 2008.

[MSM+96] S. Mutoh, S. Shigematsu, Y. Matsuya, H. Fukuda, and J. Yamada, "A 1v multi-threshold voltage cmos dsp with an efficient power management technique for mobile phone application", in *42nd IEEE International Solid-State Circuits Conference (ISSCC)*, 1996, pp. 168–169.

[MVM05] B. Mei, F. J. Veredas, and B. Masschelein, "Mapping an h.264/avc decoder onto the adres reconfigurable architecture", in *15th International Conference on Field Programmable Logic and Applications (FPL)*, 2005, pp. 622–625.

[MYN+06] A. Major, Y. Yi, I. Nousias, M. Milward, S. Khawam, and T. Arslan, "H.264 decoder implementation on a dynamically reconfigurable instruction cell based architecture", in *IEEE International SOC Conference*, 2006, pp. 49–52.

[Ne08] T. Nishimura and etAl, "Power reduction techniques for dynamically reconfigurable processor arrays", in *18th International Conference on Field Programmable Logic and Applications (FPL)*, 2008, pp. 305–310.

[Net04a] N. Nethercote, "Dynamic binary analysis and instrumentation", in *PhD Dissertation, University of Cambridge*, November 2004.

[Net04b] N. Nethercote, "Valgrind Tool Suite", http://valgrind.org, 2004.

[Nok10] Nokia Research Center, "Mobile 3d video", http://research.nokia.com/page/4988, 2010.

[NWKS09] M. Nadeem, S. Wong, G. Kuzmanov, and A. Shabbir, "A high-throughput, area-efficient hardware accelerator for adaptive deblocking filter in h.264/avc", in *IEEE/ACM/IFIP 7th Workshop on Embedded Systems for Real-Time Multimedia ESTIMedia*, October 2009, pp. 18–27.

[OBL+04] 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 and Systems Magazine*, vol. 4, no. 1, pp. 7–28, 2004.

[PBV06] E. M. Panainte, K. Bertels, and S. Vassiliadis, "Compiler-driven FPGA-area allocation for reconfigurable computing", in *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*, March 2006, pp. 369–374.

[PBV07] E. M. Panainte, K. Bertels, and S. Vassiliadis, "The Molen compiler for reconfigurable processors", *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 6, no. 1, February 2007.

[PC08] I. Park and D. W. Capson, "Improved inter mode decision based on residue in h.264/avc", in *Proceedings of the 2008 International Conference on Multimedia and Expo (ICME)*, 2008, pp. 709–712.

[PH06] M. Parlak and I. Hamzaoglu, "An efficient hardware architecture for h.264 adaptive deblocking filter algorithm", in *First NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2006, pp. 381–385.

[Phi10] Philips Nexperia, Inc., "Nexperia documentation", http://www.nxp.com, 2010.

[PLR+05] F. Pan, X. Lin, S. Rahardja, K. Lim, Z. Li, D. Wu, and S. Wu, "Fast mode decision algorithm for intraprediction in h.264/avc video coding", *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 15, no. 7, pp. 813–822, 2005.

[PP08] J. Peddersen and S. Parameswaren, "Energy driven application self-adaptations at run-time", *Journal of Computers (JC)*, vol. 3, no. 3, pp. 14–24, 2008.

[Pra01] W. K. Pratt, *Digital Image Processing*. Los Altos, California, USA: John Willy & Sons Inc., 2001.

[PSB05] R. Puri, L. Stok, and S. Bhattacharya, "Keeping hot chips cool", in *Proceedings of 42nd ACM IEEE Design Automation Conference (DAC)*, 2005, pp. 285–288.

[PWY05] K. K. W. Poon, S. J. E. Wilton, and A. Yan, "A detailed power model for field-programmable gate arrays", *ACM Transaction on Design Automation of Electronic Systems (TODAES)*, vol. 10, no. 2, pp. 279–302, 2005.

[PYL06] F. Pan, H. Yu, and Z. Lin, "Scalable fast rate-distortion optimization for h.264-avc", *EURASIP Journal on Applied Signal Processing (EURASIP)*, pp. 117–117, January 2006.

[Qe07] Y. Qu and etAl, "Using dynamic voltage scaling to reduce the configuration energy of run time reconfigurable devices", in *Proceedings of the 10th conference on Design, Automation and Test in Europe (DATE)*, 2007, pp. 1–6.

[RB05] C. A. Rahman and W. Badawy, "Umhexagons algorithm based motion estimation architecture for h.264/avc", in *Proceedings of the 5th International Workshop on System-on-Chip for Real-Time Applications*, July 2005, pp. 207–210.

[Ric03] I. E. Richardson, *H.264 and MPEG-4 Video Compression*. John Wiley & Sons, 2003.

[Ric10] I. E. Richardson, *The H.264 Advanced Video Compression Standard*. John Wiley & Sons, 2010.

[RP04] A. Rahman and V. Polavarapuv, "Evaluation of low leakage design techniques for field programmable gate arrays", in *ACM Internaltional Symposium on Field Programmable Gate Arrays (FPGA)*, 2004, pp. 23–30.

[Sam05] Samsung Electronics Company, Ltd, "OneNAND specification", http://origin2.samsung.com/global/system/business/semiconductor/product/2007/6/11/OneNAND/256Mbit/KFG5616Q1A/ds_kfg5616x1a_66mhz_rev12.pdf, 2005.

[SCL06] S. Y. Shih, C. R. Chang, and Y. L. Lin, "A near optimal deblocking filter for h.264 advanced video coding", in *Asia and South Pacific Conference on Design Automation (ASP-DAC)*, 2006, pp. 170–175.

[SF04] S. Saponara and L. Fanucci, "Data-adaptive motion estimation algorithm and vlsi architecture design for low-power video systems", *IEE Computers and Digital Techniques*, vol. 151, no. 1, pp. 51–59, January 2004.

[SGS98] S. Sawitzki, A. Gratz, and R. G. Spallek, "CoMPARE: A simple reconfigurable processor architecture exploiting instruction level parallelism", in *5th Australasian Conference on Parallel and Real-Time Systems (PART)*, September 1998, pp. 213–224.

[SHS08] M. Z. S. Hu, Z. Zhang and T. Sheng, "Optimization of memory allocation for h.264 video decoder on digital signal processors", *Congress on Image and Signal Processing (CISP)*, vol. 2, pp. 71–75, 2008.

[SJ04] J. W. Suh and J. Jeong, "Fast sub-pixel motion estimation techniques having lower computational complexity", *IEEE Transactions on Consumer Electronics (TCE)*, vol. 50, no. 3, pp. 968–973, 2004.

[SJJL09] D. Schneider, M. Jeub, Z. Jun, and S. Li, "Advanced h.264/avc encoder optimizations on a tms320dm642 digital signal processor", in *DSP'09: Proceedings of the 16th international conference on Digital Signal Processing*, 2009, pp. 1187–1190.

[SKB02] L. Shang, A. S. Kaviani, and K. Bathala, "Dynamic power consumption in virtex[tm]-ii fpga family", in *ACM Internaltional Symposium on Field Programmable Gate Arrays (FPGA)*, 2002, pp. 157–164.

[SLIS07] Y. Song, Z. Liu, T. Ikenaga, and S.Goto, "Low-power partial distortion sorting fast motion estimation algorithms and vlsi implementations", *IEIEC Transactions on Information and Systems (IETISY)*, vol. E90-D, no. 1, pp. 108–117, January 2007.

[SN06] L. Salgado and M. Nieto, "Sequence independent very fast mode decision algorithm on h.264/avc baseline profile", in *Proceedings of the 2006 International Conference on Image Processing (ICIP)*, October 2006, pp. 41–44.

[Ste09] G. R. Stewart, "Implementing video compression algorithms on reconfigurable devices", in *PhD Dissertation, University of Glasgow, United Kingdom*, June 2009.

[STM06] STMicroelectronics, Inc., "Stmicroelectronics: Nomadik mobile multimedia application processor", http://www.alldatasheet.com/datasheet-pdf/pdf/134714/ST-MICROELECTRONICS/STN8815.html, 2006.

[Str] Stretch Inc., "S6000 family software configurable processors", http://www.stretchinc.com/products/s6000.php.

[SZFH08] Y. Sun, Y. Zhou, Z. Feng, and Z. He, "A novel incremental scheme for h.264 video coding", in *Proceedings of the 2008 International Conference on Image Processing (ICIP)*, October 2008, pp. 1612–1615.

[TCW+05] T. Todman, G. Constantinides, S. Wilton, O. Mencer, W. Luk, and P. Cheung, "Reconfigurable computing: architectures and design methods", *IEE Proceedings Computers & Digital Techniques*, vol. 152, no. 2, pp. 193–207, March 2005.

[Te06] T. Tuan and et, "A 90nm low-power fpga for battery-powered applications", in *ACM Internaltional Symposium on Field Programmable Gate Arrays (FPGA)*, 2006, pp. 3–11.

[Ten] Tensilica Inc., "Tensilica: Customizable processor cores for the dataplane", http://www.tensilica.com/.

[Tex10a] Texas Instruments, Inc., "Omap_tm technology", http://focus.ti.com/general/docs/gencontent.tsp?contentId=46946&DCMP=WTBU&HQS=Other+OT+omap, 2010.

[Tex10b] Texas Instruments, Inc., "Ti documentation", http://www.ti.com/, 2010.

[THLW03] P.-L. Tai, S.-Y. Huang, C.-T. Liu, and J.-S. Wang, "Computation-aware scheme for software-based block motion estimation", *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 13, no. 9, pp. 901–913, September 2003.

[TL03] T. Tuan and B. Lai, "Leakage power analysis of a 90nm fpga", in *IEEE Custom Integrated Circuits Conference*, 2003, pp. 57–60.

[Tou02] A. M. Tourapis, "Enhanced predictive zonal search for single and multiple frame motion estimation", in *Proceedings of the 2002 SPIE Visual Communications and Image Processing (VCIP)*, January 2002, pp. 1069–1079.

[VS07] S. Vassiliadis and D. Soudris, *Fine- and Coarse-Grain Reconfigurable Computing*. Springer Publishing Company, Incorporated, 2007.

[VSWM05] F. J. Veredas, M. Scheppler, and B. M. W. Moffat, "Custom implementation of the coarse-grained reconfigurable adres architecture for multimedia purposes", in *15th International Conference on Field Programmable Logic and Applications (FPL)*, 2005, pp. 106–11.

[VWG+04] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. Panainte, "The MOLEN polymorphic processor", *IEEE Transactions on Computers (TC)*, vol. 53, no. 11, pp. 1363–1375, November 2004.

[WAW07] C.-W. Wong, O. C. Au, and R. C.-W. Wong, "Advanced real-time rate control in h.264", in *Proceedings of the 2007 International Conference on Image Processing (ICIP)*, October 2007, pp. I–69–I–72.

[WC96] R. Wittig and P. Chow, "OneChip: an FPGA processor with reconfigurable logic", in *IEEE Symposium on FPGAs for Custom Computing Machines*, April 1996, pp. 126–135.

[WOZ02] Y. Wang, J. Ostermann, and Y.-Q. Zhang, *Video Processing And Communications*. Upper Saddle River, New Jersey, USA: Prentice-Hall Inc., 2002.

[WSBL03] T. Wiegand, G. J. Sullivan, G. Bjntegaard, and A. Luthra, "Overview of the h.264/avc video coding standard", *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 13, no. 7, pp. 560–576, 2003.

[WSK+07] J.-H. Woo, J.-H. Sohn, H. Kim, J. Jeong, E. Jeong, S. J. Lee, and H.-J. Yoo, "A low power multimedia soc with fully programmable 3d graphics and mpeg4/h.264/jpeg for mobile devices", in *Proceedings of the 2006 ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2007, pp. 238–243.

[WSLL07] X. Wang, J. Sun, Y. Liu, and R. Li, "Fast mode decision for h.264 video encoder based on mb motion characteristic", in *Proceedings of the 2007 International Conference on Multimedia and Expo (ICME)*, July 2007, pp. 372–375.

[WUS+08] K. Willner, K. Ugur, M. Salmimaa, A. Hallapuro, and J. Lainema, "Mobile 3d video using mvc and n800 internet tablet", in *3DTV Conference: The True Vision—Capture, Transmission and Display of 3D Video*, May 2008, pp. 69 –72.

[Xil05] Xilinx, Inc., "Xilinx development system: Partial reconfiguration", http://toolbox.xilinx.com/docsan/xilinx8/de/dev/partial.pdf, April 2005.

[Xil07] Xilinx, Inc., "Virtex-II platform FPGA user guide, v2.2", http://www.xilinx.com/support/documentation/user_guides/ug002.pdf, November 2007.

[Xil08a] Xilinx, Inc., "Spartan and Spartan-XL FPGA families data sheet, v1.8", http://www.xilinx.com/support/documentation/data_sheets/ds060.pdf, June 2008.

[Xil08b] Xilinx Inc., "Virtex-4 FPGA user guide, v2.6", http://www.xilinx.com/support/documentation/user_guides/ug070.pdf, December 2008.

[Xil09] Xilinx, Inc., "Virtex-4 FPGA configuration user guide, v1.11", http://www.xilinx.com/support/documentation/user_guides/ug071.pdf, June 2009.

[Xil10a] Xilinx, Inc., "Xilinx documentation", http://www.xilinx.com/support/documentation, 2010.

[Xil10b] Xilinx, Inc., "Xilinxpowersolutions", http://www.xilinx.com/products/design_resources/power_central, 2010.

[Xip10] Xiph.org, "Test Media, Video Sequences", http://media.xiph.org/video/derf, 2010.

[XPP02] XPP_Team, "The xpp white paper", in *PACT Corporation*, Release 2.1 2002, pp. 1–4.

[YCL05] Z. Yang, H. Cai, and J. Li, "A framework for fine-granular computational-complexity scalable motion estimation", in *Proceedings of the 2005 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2005, pp. 5473–5476.

[YCW09] L. Yu, S. Chen, and J. Wang, "Overview of avs-video coding standards", *Signal Processing: Image Communication, Special Issue on AVS and its Application*, vol. 24, no. 4, pp. 247–262, April 2009.

[Yu04] A. C. Yu, "Efficient block-size selection algorithm for inter-frame coding in h.264/mpeg-4 avc", in *Proceedings of the 2004 International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, May 2004, pp. III169–III172.

[YWV05] S. Yang, W. Wolf, and N. Vijaykrishnan, "Power and performance analysis of motion estimation based on hardware and software realizations", *IEEE Transactions on Computers (TC)*, vol. 54, no. 6, pp. 714–726, June 2005.

[YZLS05] X. Yi, J. Zhang, N. Ling, and W. Shang, "Improved and simplified motion estimation for jm", in *Joint Video Team (JVT) of ISO/IECMPEG & ITU-T VCEG 16th Meeting*, July 2005, pp. Document JVT–P021.

[Ze07] P. Zipf and etAl, "A power estimation model for an fpga-based softcore processor", in *17th International Conference on Field Programmable Logic and Applications (FPL)*, 2007, pp. 171–176.

[ZuHNS07] S. M. Ziauddin, I. ul Haq, M. Nadeem, and M. Shafique, "Method for providing low cost robust operational control for video encoders (US Patent))", Patent Pub. No. US-2007-0206674-A1, Class: 375240050 (USPTO), 2007.

[ZWFS07] L. Zhuo, Q. Wang, D. D. Feng, and L. Shen, "Optimization and implementation of h.264 encoder on dsp platform", in *IEEE International Conference on Multimedia and Expo (ICME)*, July 2007, pp. 232–235.

Index

A
ACCoReS, 94–96, 100–104
Adaptive, 72, 95, 96, 104, 105, 108, 111, 118
Application, 69–72, 74–78, 121
Application Architectural Adaptations, 69, 70, 72, 75
Architecture, 57, 60, 63, 66, 74, 75, 77, 78, 121, 184
ASIC, 114

B
Bitstream, 77
Bus Connector, 64, 66

C
CAVLC, 72, 76, 77
cISA, 75
Clip3, 76, 87
CollapseAdd, 76, 87
Common Intermediate Format, 2, 15, 78, 79, 207
Complexity Reduction Scheme, 24, 25, 95, 96, 100, 104, 169
Computational Complexity, 25, 95, 96
Cond, 76, 83, 87
Constant Bit Rate, 194
Custom Instruction, 8, 10–13, 23, 32, 35–41, 45, 46, 56, 57, 59, 60, 63–65, 67, 69, 71, 75, 76, 80, 81, 83–87, 121, 183–185, 188, 203–206

D
Data Flow Diagram, 78
Data Path, 69, 71, 73, 75, 76, 80, 81, 83–87, 121, 184, 185, 188
Data Path Container, 35, 36, 42, 63, 66, 81, 188, 204
Deblocking Filter, 21, 23, 24, 27, 28, 32, 52, 72, 76, 77, 81–84, 121
Digital Signal Processor, 4, 208
Discrete Cosine Transform, 20, 32, 72, 76, 86
Dynamic Power, 65

E
Early Mode Exclusion, 96, 97, 100, 102
enBudget, 105, 110–112, 114, 115, 117–121
Energy-Minimizing Instruction Set, 126, 133, 136, 139
Energy-Quality Class, 10, 11, 13, 27, 46, 53, 60, 61, 69, 121, 122, 185–188
Enhanced Predictive Zonal Search, 26, 173
EPZS, 110
EQ-Class, 105, 109–115, 117, 119
Error, 117

F
Fast Mode Decision, 24
Fast Mode Prediction, 96, 97, 99
FPGA, 87, 120
Fractional-pixel Motion Estimation, 22
Frame, 78, 89, 92, 94, 102, 103, 111, 115, 119, 121
Full Search, 71, 105, 106, 110, 112, 113, 186
Functional Blocks, 17, 24, 51, 54, 71

G
General Purpose Processor, 6
GOP, 111, 115, 121
Group of Pictures, 17–19, 57, 61, 111, 117, 121, 186, 193

H
H.264, 69–72, 74, 76–78, 80, 81, 87, 89, 95, 104, 105, 110, 121
Hadamard Transform, 19, 21, 64, 72, 76, 77, 86
Hardware, 74, 75, 77, 78, 120
Hardware Pressure, 74, 75, 77, 78

M. Shafique, J. Henkel, *Hardware/Software Architectures for Low-Power Embedded Multimedia Systems,*
DOI 10.1007/978-1-4419-9692-3, © Springer Science+Business Media LLC 2011

Human Visual System, 53, 55, 69, 87, 95, 121
HVS, 87, 88, 92, 95, 96, 104, 121, 122

I
(I)DCT4x4, 76
(I)HT_2x2, 75, 76
I16x16, 97, 98
I4x4, 89, 97–100, 102, 103
I-MB, 72, 73, 76, 89, 92–94, 96, 100
Implementation Version, 34, 36–41, 62–67, 83, 126, 131–136, 138, 139, 142, 145, 146, 150–152, 184, 204, 205
Instruction Set, 4, 10, 11, 13, 33, 34, 57, 58, 60, 63, 66, 75, 183–185, 187, 212
Integer-pixel Motion Estimation, 22
Internal Configuration Access Port, 30, 164, 166
IPred, 71, 72, 76
IPred_HDC, 76
IPred_VDC, 76

J
Joint Video Team, 17, 213, 215, 216, 220

L
Leakage Power, 66
Level, 102, 103, 111, 112, 115, 119, 121
LF_4, 76, 83, 87
LF_BS4, 76, 82, 83, 85
Low-Power, 80, 121

M
Macroblock, 69, 70, 76, 81, 82, 92, 95, 112, 120–122, 186, 187
Macroblock Categorization, 92
MC_Hz_4, 76, 86
Mean Bit Estimation Error, 200
Mode Decision, 50, 51, 54, 67, 72, 75, 76, 80, 122, 185, 186
Mode Elimination, 96, 99, 100
Motion Compensation, 20, 22, 23, 71, 76, 77, 79, 80, 86
Motion Estimation, 10, 11, 13, 17, 19, 21–25, 27, 32, 45, 46, 50–55, 60, 61, 67, 69, 71, 75–77, 80, 85, 89, 95, 96, 104, 105, 111, 114, 117–122, 185, 186, 206, 209, 210
Motion Vector, 52, 106
Multimedia MPSoC, 6
Multiview Video Coding, 2, 51, 53, 188

O
On-Demand Interpolation, 72
Overhead, 104, 120, 121

P
P16x16, 89, 93, 94, 97–100, 103
P16x8, 89, 99, 100
P4x4, 89, 100
P4x8, 89, 100
P8x16, 89, 99, 100
P8x4, 89, 100
P8x8, 89, 93, 94, 97–100, 103, 104
Performance, 121
P-MB, 72, 76, 89, 92, 96
PointFilter, 76, 87
Prediction, 71, 73, 76, 77, 86, 96, 97, 99, 100, 106, 107, 109, 112–114
Processing Time Distribution, 51
Processor, 69
Profile, 70
Proportional Gain, 194
PSNR, 100–105, 109–113, 117–119

Q
QCIF, 71, 73, 78, 101, 108, 119–121
QuadSub, 76
Quantization Parameter, 20, 57, 60, 61, 69, 77, 78, 89, 90, 95, 109, 122, 185, 187, 192, 193
Quarter Common Intermediate Format, 2, 78

R
Rate Controller, 77, 78
Rate Distortion, 17, 19, 51, 54, 72, 75, 192
RDO, 72, 75–77, 80, 89, 94–96, 99
RDO-MD, 72, 75–77, 80, 89, 94–96, 99, 100, 102–104
Reconfigurable Functional Unit, 32
Reconfigurable Processor, 11, 28, 32, 43, 56, 63, 69, 130, 155, 166, 176, 204
Reconfiguration, 34, 41, 66, 87
Repack, 75, 76, 80, 86, 87
RISPP, 71
Rotating Instruction Set Processing Platform, 13, 34, 35, 212
Run-Time, 115

S
SAD16x16, 76, 85
SADrow, 76, 85, 87
SATD, 71, 75, 85
SATD4x4, 76, 85, 86
Spatial, 87, 91, 92, 95
Standard Definition, 2
State-of-the-art, 8, 10, 24, 26, 46, 58
Sum of Absolute Differences, 19, 55, 71, 76, 85, 153
Sum of Absolute Transformed Differences, 19, 71, 85

T
Temporal, 87, 91, 92, 95
Transform, 76

U
UMHexagonS, 71, 110, 112, 117–119, 186
Unsymmetrical-cross Multi-Hexagon-grid Search, 25, 173

V
Variable Bit Rate, 193
Variable Length Coding, 21, 72
Video Coding, 49, 51–53, 188
Video Conferencing, 13, 50, 51
Video Encoder, 53, 54, 76, 87

X
Xilinx Synthesis Technology, 158