Probabilistic Image Models and their Massively Parallel Architectures -
A Seamless Simulation- and VLSI Design-Framework Approach

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhems-Universität Bonn

vorgelegt von

Stephan C. Stilkerich

aus

Bonn Bad-Godesberg

München, 2006

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn

Erscheinungsjahr: 2007

Promotionskommission:
1. Referent: Univ.-Prof. Dr. Joachim M. Buhmann, ETH Zürich
2. Referent: Univ.-Prof. Dr. Joachim K. Anlauf, Universität Bonn
Fachnahes Mitglied: Univ.-Prof. Dr. Armin B. Cremers, Universität Bonn
Fachangrenzendes Mitglied: Univ.-Prof. Dr. Wolfgang Förstner, Universität Bonn

Tag der Promotion: 27. 04. 2007

Diese Dissertation ist auf dem Hochschulschriftenserver der ULB Bonn
`http://hss.ulb-bonn.de/diss_online` elektronisch publiziert.

Abstract

Probabilistic Image Models and their Massively Parallel Architectures -
A Seamless Simulation- and VLSI Design-Framework Approach

by

Stephan C. Stilkerich

Algorithmic robustness in real-world scenarios and real-time processing capabilities are the two essential and at the same time contradictory requirements modern image-processing systems have to fulfill to go significantly beyond state-of-the-art systems. Without suitable image processing and analysis systems at hand, which comply with the before mentioned contradictory requirements, solutions and devices for the application scenarios of the next generation will not become reality. This issue would eventually lead to a serious restraint of innovation for various branches of industry.

This thesis presents a coherent approach to the above mentioned problem. The thesis at first describes a massively parallel architecture template and secondly a seamless simulation- and semiconductor-technology-independent design framework for a class of probabilistic image models, which are formulated on a regular Markovian processing grid.

The architecture template is composed of different building blocks, which are rigorously derived from Markov Random Field theory with respect to the constraints of *massively parallel processing* and *technology independence*. This systematic derivation procedure leads to many benefits: it decouples the architecture characteristics from constraints of one specific semiconductor technology; it guarantees that the derived massively parallel architecture is in conformity with theory; and it finally guarantees that the derived architecture will be suitable for VLSI implementations.

The simulation-framework addresses the unique hardware-relevant simulation needs of MRF based processing architectures. Furthermore the framework ensures a qualified representation for simulation of the image models and their massively parallel architectures by means of their specific simulation modules. This allows for systematic studies with respect to the combination of numerical, architectural, timing and massively parallel processing constraints to disclose novel insights into MRF models and their hardware architectures.

The design-framework rests upon a graph theoretical approach, which offers unique capabilities to fulfill the VLSI demands of massively parallel MRF architectures: the semiconductor technology independence guarantees a technology uncommitted architecture for several design steps without restricting the design space too early; the design entry by means of behavioral descriptions allows for a functional representation without determining the architecture at the outset; and the topology-synthesis simplifies and separates the data- and control-path synthesis.

Detailed results discussed in the particular chapters together with several additional results collected in the appendix will further substantiate the claims made in this thesis.

Acknowledgment

I owe a lot to my adviser Prof. J. M. Buhmann. The dissertation on hand lives and significantly profits from the in-depth insights on statistical image analysis Prof. Buhmann is working on in his research department, and which he is also lecturing on. I appreciate Prof. Buhmann's inspiring discussions, his constructive help and the experience of being asked crucial and sometimes even unpleasant questions. I also want to emphatically thank Prof. Buhmann for having been spontaneously prepared to supervise this external dissertation. Thank you very much!

I would also like to thank my second thesis evaluator Prof. J. K. Anlauf for accepting to evaluate this thesis. Prof. Anlauf also contributed to my work as a discussion partner for research issues related to digital chip design and hardware relevant design-methodologies.

I am deeply indebted to my employer EADS Corporate Research Center Germany for part-time funding and continuously supporting my research work. The very inspiring and unique discussion atmosphere in the multi-discipline research group of Dr. Gerald Sobotta was an essential enrichment for the research direction of my work. Thank you very much Gerald Sobotta! At this point I also gratefully thank Dr. Helmut Zinner, the senior manager and former head of the EADS Corporate Research Center department "Optronic Systems and Signal Processing", for his never ending willingness to invest in expensive software- and FPGA testing-environments, which were indispensable for my research. This can definitely not be taken for granted in times of a substantial economic downturn and layoffs.

I also want to thank my research colleagues around the world. Their kind preparedness to discuss problems and conceptions at conference meetings were very helpful for this thesis. These discussions and remarks sometimes prevented me from pursuing directions and ideas, which have already been proved to fail by my colleagues, but are not published so far and perhaps will not be published at all.

My very special thanks go to Susanne A. Barr at Synplicity Inc. and her team. Synplicity Inc. patiently supported my research by means of special research software-licenses and by countless discussions on synthesis issues of hardware description languages.

I also have to thank Inge Prächtl and Helga Beyer for their administrative support, humor in all situations - sometimes even black humor - and their-all-time patience with me.

At most I would like to thank Maren for her patience, continuous support and her understanding. Thanks a lot Maren. With regard to the last months I also need to thank my daughter Olivia for her patience.

to my parents

Contents

1 Introduction **1**

2 Fundamentals - Bayesian Image Analysis **7**
2.1 Bayesian Image Analysis Paradigm 9
2.2 Markov Random Fields & Gibbs Fields 11
2.3 Optimization Schemes 15
2.4 Parallel Processing Strategies 20
2.5 Considered MRF-Class 23
2.6 Exemplary Image Processing Models 24
2.6.1 Cost Function - Edge Preserving & Noise Removing 25
2.6.2 Cost Function - Unsupervised Histogram Segmentation . . . 28
2.7 Analog Technologies versus Digital Technologies 33
2.8 Summary 38
2.9 Bibliographical Comments 42

3 System-Architecture Template **45**
3.1 Universal Constituents 47
3.2 Architectural Building Blocks 49
3.2.1 Topology & Structure Building Blocks 50
3.2.2 Processing Building Blocks 59
3.2.3 Control Building Blocks 64
3.3 VLSI Specific Issues 72
3.4 Cycle Scheme 75
3.5 Relation of Thesis Parts 78
3.6 Dealing with Large Images 79
3.7 Exemplary Model's Architectures 84
3.7.1 Edge Preserving & Noise Removing 84
3.7.2 Unsupervised Histogram Segmentation 87
3.8 Summary 90
3.9 Bibliographical Comments 91

4 MRF Simulation Framework **93**
4.1 Simulation Framework Overview 95
4.2 Simulation Modules 100
4.2.1 Supporting Modules 100
4.2.2 Topology & Structure Simulation Modules 100

4.2.3 Processing Simulation Modules . . . 108
4.2.4 Control Simulation Modules . . . 113
4.3 Building MRF Simulation Models . . . 117
4.3.1 Building the Simulation Model - Model Preparation . . . 117
4.3.2 Building the Simulation Model - Model Generation . . . 118
4.4 Results - Simulation Framework . . . 118
4.5 Implementation Issues . . . 127
4.6 Relation of Thesis Parts . . . 130
4.7 Summary . . . 131
4.8 Bibliographical Comments . . . 132

5 VLSI Design Framework 135
5.1 Design Framework Overview . . . 137
5.2 Canonical Design Representation . . . 142
5.2.1 Elementary Data Container . . . 144
5.2.2 Topology & Structure Graphs . . . 145
5.2.3 Processing Graphs . . . 154
5.2.4 Control Graphs . . . 157
5.3 High-Level Design Flow . . . 161
5.3.1 Expanding the Topology & Structure Graphs . . . 161
5.3.2 Expanding the Processing Graphs . . . 166
5.3.3 Expanding the Control Graphs . . . 171
5.4 Compiling - From Abstract Graphs to Circuit Descriptions . . . 176
5.4.1 Compiling Topology & Structure Parts . . . 176
5.4.2 Compiling Processing Parts . . . 181
5.4.3 Compiling Control Parts . . . 183
5.5 Results - Design Framework . . . 186
5.6 Implementation Issues . . . 198
5.7 Relation of Thesis Parts . . . 202
5.8 Summary . . . 202
5.9 Bibliographical Comments . . . 203

6 Conclusion 207

A Simulation Framework Results 211

B Design Framework Results 251

Bibliography 267

Index 280

Symbols

Common Symbols

$\mathbf{X}$, $\mathbf{Y}$	Finite state spaces
$\mathbb{R}, \mathbb{N}, \mathbb{B}$	Set of real, natural and Boolean numbers
$\emptyset$	Empty set
$\mathbb{E}[x]$	Expected value of variable x

Common Image Representation

Ω	Site grid
s_i	Destined site on grid
n	Total number of grid sites, $n = \|\Omega\|$
$\{< i, t >\}$	Neighbors of destined site s_i
$\mathcal{N}_\Omega$	Neighborhood system on Ω
$\mathcal{N}^{1-5}$	First five neighborhood systems on regular site-grids Ω
$\mathcal{C}^{\mathcal{N}}$	Set of cliques induced by $\mathcal{N}$
Π	Gibbs field induced by $\mathcal{H}$
$\mathcal{H}$	Energy functional on $\mathbf{X}$ inducing Gibbs field Π

Chapter 2 - Bayesian Image Analysis

$\widehat{X}$	Estimator; see Def. 2.2
$L(x, \widehat{x})$	Loss function
$R(\chi)$	Bayes risk; cf. Eq. 2.3
$S(p)$	Entropy of distribution p; $S(p) = -E_p[\log p]$
$\mathcal{F}$	Generalized free energy; cf. Eq. 2.15
$V(m)$	Site visitation scheme
T, β	Temperature resp. inverse temperature
$\mathbf{\Theta}$	Set of free model parameters; model dependent

Chapter 3 - System Architecture Template

$\mathbb{C}_{universal}$	Set of MRF specific universal constituents
$\mathcal{BB}^{TS}$	Set of Topology & Structure Building Blocks; cf. Def. 3.1
$\mathcal{BB}^{PC}$	Set of Processing Building Blocks; cf. Def. 3.2
$\mathcal{BB}^{CT}$	Set of Control Building Blocks; cf. Def. 3.3
$\mathcal{BB}^{System}$	Set of System Building Blocks; cf. Eq. 3.37
$\mathbf{\Lambda}$	Set of Mappings onto $\mathcal{BB}^{TS}$ components; cf. Section 3.2.1
$\mathbf{\Phi}$	Set of Mappings onto $\mathcal{BB}^{PC}$ components; cf. Section 3.2.2
$\mathbf{\Upsilon}$	Set of Mappings onto $\mathcal{BB}^{CT}$ components; cf. Section 3.2.3

Chapter 4 - Simulation Framework

$\mathcal{S}^{TS}$	Set of Topology & Structure representing simulation modules; cf. Def. 4.1
$\mathcal{S}^{PC}$	Set of Processing functionality representing simulation modules; cf. Def. 4.2
$\mathcal{S}^{CT}$	Set of Control functionality representing simulation modules; cf. Def. 4.3
$\mathbf{\Pi}$	Set of Mappings onto $\mathcal{S}^{TS}$ modules; cf. Section 4.2.2
$\mathbf{\Sigma}$	Set of Mappings onto $\mathcal{S}^{PC}$ modules; cf. Section 4.2.3
$\mathbf{\Delta}$	Set of Mappings onto $\mathcal{S}^{CT}$ modules; cf. Section 4.2.4

Chapter 5 - VLSI Design Framework

$\mathbb{C}^{P}$	Data Container - Parameter Container; cf. Def. 5.1
$\mathbb{C}^{ST}$	Data Container - Structure Core Container; cf. Def. 5.2
$\mathcal{G}^{TS}$	Set of Topology & Structure blocks representing Graphs; cf. Def. 5.3
$\mathcal{G}^{PC}$	Set of Processing blocks representing Graphs; cf. Def. 5.6
$\mathcal{G}^{CT}$	Set of Control blocks representing Graphs; cf Def. 5.10
$\mathbf{\Xi}$	Set of Mappings onto $\mathcal{G}^{TS}$ graphs; cf. Section 5.2.2
$\mathbf{\Psi}$	Set of Mappings onto $\mathcal{G}^{PC}$ graphs; cf. Section 5.2.3
$\mathbf{\Gamma}$	Set of Mappings onto $\mathcal{G}^{CT}$ graphs; cf. Section 5.2.4
$\mathcal{G}^{D}$	Complete graph theoretical design graph; cf. Corollary 5.6

Abbreviations

BS	Behavioral Synthesis
CDR	Canonical Design Representation
CPU	Central Processing Unit
DA	Deterministic Annealing
DAG	Directed Acyclic Graph
DFG	Design Flow Graph
DRC	Design Rule Checker
EDA	Electronic Design Automation
ESL	Electronic System Level
IEEE	Institute of Electrical and Electronics Engineers
FPGA	Field Programmable Gate-Array
FSM	Finite State Machine
GUI	Graphical User Interface
HDL	Hardware Description Language
HLS	High-Level Synthesis
ICM	Iterated Conditional Modes
MCMCM	Markov Chain Monte Carlo Method
MRF	Markov Random Field
RTL	Register Transfer Level
SA	Simulated Annealing
SoC	System on Chip
VLSI	Very large-scale Integration

Chapter 1

Introduction

"We are so familiar with seeing, that it takes a leap of imagination to realize that there are problems to be solved. But consider it. We are given tiny distorted upside-down images in the eyes, and we see separate solid objects in surrounding space. From the patterns of stimulation on the retina we perceive the world of objects and this is nothing short of a miracle"

Richard L. Gregory, Eye and Brain, 1966

Motivation

Definitely some of the vision "miracles" respectively processing principles of insects and mammals have been investigated and explained since R. L. Gregory's statement dated 1966. Especially the first processing stages of vision [43] and their underlying processing principles have been exhaustively explained and described over the last three decades and are thought to be well understood today. In contrast to these first image processing stages and their profound scientific findings, higher stages of processing, which, for instance, are responsible for depth- and form-perception, object forming, semantic assignment and the final task of image understanding, still pose challenging research questions today and the scientific picture is not yet coherent. This observation is especially true for the *binding problem*, which is discussed in neural science [91] [2] as the problem of how perception emerges from information processed independently in different cortical areas. The binding problem remains one of the essential unresolved problems in the understanding of perception.

The preceding short summary roughly characterizes the current situation in neural science with respect to insects' and mammals' vision systems [80]. However, all of the very detailed knowledge about the structure, topology, electrical respectively chemical signal transmission paths and elementary processing principles of these natural vision systems might create a misleading impression. One could rashly come to the conclusion that there are exhaustive answers and solutions derived from neural science to solve nearly all image analysis problems, which emerge along the entire image processing chain of technical systems from low-level image processing to complete image understanding. But these exhaustive answers and solutions are definitely not available. Till this day there are still essential problems to be solved [116] to

form a coherent comprehension of vision and to realize the corresponding technical counterparts with the same astonishing capabilities of natural vision systems.

On the basis of the preceding discussion it should become obvious that it is extremely difficult to establish a well-defined and systematic procedure, which transfers the diverse insights and scientific results on neural science vision to mathematically consolidated theoretical image processing approaches or even to technical realization of image processing systems [62] and its hardware-architectures. This fact is the reason why even today most of the feasible approaches and system realizations are based on *ad-hoc* models, algorithms or hand-tuned hardware-architectures. This state-of-the-art was acceptable in the past and above all feasible for industrial image processing platforms, which were guaranteed to operate in predefined environments and under controlled ambient lightning. The primary application area of these environmentally tuned image processing systems is currently to be found in the automated production of industrial goods in factories, where these systems perform their assigned specific task very well and reliable.

However, for numerous application scenarios currently discussed in the domains of video-, infrared- and radar-surveillance, medicine, man-machine interface and the significant autonomous vehicle guidance on ground, air and water, to name just a few of them, an ad-hoc approach is probably neither reliable nor put into practice in order to realize the appropriate image processing and analysis systems for these applications. Thus it is no longer possible to neglect the relevancy and meaningfulness of image processing and analysis systems as they have emerged over the last years as one of the crucial building blocks for these different applications in mind. Algorithmic robustness in real-world scenarios and real-time processing capabilities are the two essential and at the same time contradictory requirements modern image-processing systems have to fulfill to go significantly beyond state-of-the-art systems. Without suitable image processing and analysis systems at hand, that comply with the before mentioned contradictory requirements - *algorithmic robustness in real world scenarios* and *real-time processing capabilities* - solutions and devices for the application scenarios of the next generation will not become reality. This issue would eventually lead to a serious innovation restraint for various branches of industry.

Regarding the *low-level* image processing domain we can determine that it is principally possible to fulfill the contradictory requirements of algorithmic robustness in real world scenarios and real-time processing capabilities. The property of algorithmic robustness is achieved by formulating the considered *low-level* image processing problems within a Bayesian Image Analysis framework with probabilistic processing models. This statistical approach deals very well with modeling uncertainties, incomplete knowledge [167] and changing environmental settings in real world scenarios and thus gives its competitive edge with respect to algorithmic robustness in real world scenarios. The plain potential of the statistical image processing approach and its models foremost develops to its full extent by accommodating and ingraining the probabilistic models into the spatially regular topologies and parallel processing principles of Markov Random Field (MRF) grids.

These topological configurations and spatially parallel processing principles are promising and practicable structures respectively processing directives imitated from neural science vision, which were put into practice by a solid mathematically statisti-

cal framework. Thus real-time processing capabilities can be achieved by massively parallel hardware architectures, which efficiently exploit the inherent algorithmic parallelism of statistical image models on Markov Random Fields.

Contributions

This dissertation contributes to the scientific and technical progress by introducing: (a) a novel architecture template for a large class of MRF models [160] [154], which is rigorously derived on the basis of MRF theory. (b) a hardware-relevant and scalable simulation approach for MRF systems of industrial-relevant size. (c) a semiconductor technology independent VLSI design methodology for Markovian processing-grids. The advocated approaches are put into practice by a novel hardware-relevant Simulation-Framework and semiconductor independent VLSI Design-Framework. These two novel frameworks render it possible to systematically realize MRF processing grids with limited and regular neighborhood support, which are characterized by (i) an industrial-relevant size, (ii) a purely digital realization, (iii) massively parallel processing capabilities and (iv) an overall physical compactness (System-on-Chip).

Independent of the two frameworks' brought task-diversity, which range from modeling and simulation of massively parallel system architectures to the tool-supported generation of circuit representations in IEEE-standardized Hardware Description Languages (HDLs), both described frameworks rest upon a common fundamental data-structure. The Simulation-Framework and the VLSI Design-Framework, commonly rely on the solid, abstract and widely-approved fundamentals of graph-theory, its modeling capabilities, data-structures and algorithms. The uniquely and rigorously based graph theoretical approach for the VLSI Design-Framework, paved the way for the novel Development-Environment and its highly competitive properties.

Summarizing, the key contributions of the novel Simulation- and VLSI Design-Framework are:

- Firstly, the Simulation-Framework possesses simulation capabilities, which for the first time ever renders it possible to simulate complete MRF hardware-architectures with industrial relevant grid sizes of parallel processing elements and a limited neighborhood support [152] [153] [161] [156] [158].

- Secondly, the Simulation-Framework comprises simulation capabilities to investigate and study the parallel processing dynamics of these Markov Random Field based image processing systems. Additionally, the simulation models can be configured to operate with float-point numbers as well as with hardware-relevant fixed-point numbers [152] [153] [161] [156] [158].

- Thirdly, the Simulation-Framework possesses mechanisms to enable an automatic compilation-process of the specific MRF simulation models. Furthermore, the Simulation-Framework includes a simulation run-time, which is an order of magnitude faster than standard HDL simulators. The improved run-time is realized by means of the SystemC kernel [81] our Simulation-Framework is built up on as well as by the components of the Simulation-Framework itself [152] [153] [161] [156] [158].

- Fourthly, the VLSI Design-Framework is able to systematically handle the design complexity of industrial relevant MRF-systems, i.e. the graph-theoretical representation as well as the HDL-code of MRF-systems with grid sizes of up to 1024x1024 parallel processing elements has been successfully generated and tested [152] [153] [159] [155] [157].

- Fifthly, the VLSI Design-Framework is semiconductor-technology independent and thus applicable to all worldwide available digital semiconductor technologies, which support a standard HDL-based synthesis and back-end process, i.e. FPGAs, structured ASICs, standard cells to name just the widely used technologies can be utilized [152] [153] [159] [155] [157].

- Sixthly, the VLSI Design-Framework can generate automatically IEEE-conform HDL circuit descriptions, which are well-structured, modularly organized and above all still readable, checkable and modifiable by the framework users [152] [153] [159] [155] [157].

The novel Development-Environment is finally extensively validated by various artificial test cases and real-world scenarios to not only focus on the capabilities and advantages of the Simulation- and VLSI Design-Framework but also to disclose its so-far-known limitations and drawbacks.

Outline

Following Chapter 1, which serves as a short *introduction* regarding the motivation and topics of the thesis, Chapter 2 presents the required *theoretical fundamentals* on Bayesian Image Analysis, Markov Random Fields and the corresponding deterministic and stochastic optimization strategies. This discussion establishes the profound theoretical basics to which the consecutive chapters will refer, justifies their specific approaches and finally draws conclusions. Furthermore two vital image processing models, that of simultaneous noise removing and edge preserving and that of unsupervised segmentation, will be introduced, followed by the derivation of their complete cost functions. Throughout this dissertation both models will be used as exemplary low-level image processing models and realistic test cases, which are of significant practical and industrial relevance. The chapter closes with a discussion on semiconductor technologies, which provides a formal justification to exclusively consider digital implementation technologies.

Chapter 3 introduces the massively parallel *system architecture template* for the defined class of statistical image processing models, which are formulated on Markovian pixel-grids with a limited neighborhood support. The general topology, the particular architectural building blocks, support structures, wiring and the dependencies of the different components among each other are derived on the foundation of the theoretical fundamentals presented in Chapter 2. This architectural decomposition is the basis for later discussions and it links our formal description to graph theoretical modeling, simulation and representation of these massively parallel architectures and their circuits.

Chapter 4 finally describes the *Simulation-Framework* with its internal arrangement and the features issued thereof. By utilizing the specific module-structures and

their interdependence, a well-defined and systematic approach to system modeling, system simulation set-up, system complexity handling, simulation run-time improvement and the automated generation of simulation models has been established and exhausted on a large scale.

The *VLSI Design-Framework* introduced and explained in Chapter 5 is the constituent part of the novel Development Environment, which handles the abstract representation of the image processing architectures as graph structures. In a sequence of steps it automatically performs the generation of graph-representations and the following transformation from these graphs to concrete hardware architectures in standardized IEEE HDLs. The thesis is concluded with Chapter 6, which provides a summary of the scientific progress as well as statements about the corresponding technical novelties achieved.

Some additional appendices exhaustively illustrate the capabilities of the presented novel Simulation- and VLSI Design-Framework. The numerous illustrations of simulation-runs and VLSI Place&Route results have been moved to the appendices to enhance the readability of the text. Furthermore each single chapter is enriched by a "Bibliographical Comments" section at the end to improve the readability of each single chapter, to identify the international state-of-the-art and above all to stress the contributions and innovations of this dissertation in the respective fields of the academic and industrial research community.

Chapter 2

Fundamentals - Bayesian Image Analysis

This introductory chapter discusses the fundamentals, which are required to present the central topics of this thesis:

- *The derivation of an architecture template (Chapter 3) for massively parallel Markov Random Field based image processing devices.*
- *The introduction of a novel hardware-relevant simulation framework (Chapter 4) for massively parallel Markov Random Field based image processing devices.*
- *The introduction of a novel graph-theoretical VLSI design framework (Chapter 5) for massively parallel Markov Random Field based image processing devices.*

Hence, we will discuss the basic idea and motivation of a Bayesian image analysis approach in this chapter and will present a synopsis of the corresponding theoretical fundamentals, including optimization methods and parallel processing strategies with a guaranteed convergence behavior. Furthermore the considered class of image processing models and its requirements are defined in this chapter. Additionally, two exemplary types of image processing, noise removing with intensity preserving, as well as unsupervised segmentation are presented. These two models will serve as examples throughout the whole text. A vital discussion on semiconductor implementation technologies and novel VLSI high-level design methodologies will finally close this introductory chapter. All these fundamentals serve as common basis for the definitions, derivations, discussions and conclusions of the following chapters.

On an abstract level of consideration, image processing respectively image analysis[1] problems can mathematically be represented by optimization questions, i.e. by cost-functionals [167] and their corresponding costs. Thus an image processing problem is regarded as an abstract optimization problem. Such cost-functionals systematically capture the image processing problem in one coherent representation [126] [16]. The image configuration, which finally optimizes the costs of the cost-functional, represents the desired image processing solution. The specific class of cost-functionals, addressed in this thesis, combines two types of information:

[1] both denotations are used synonymously throughout this text

First, information regarding the generation process of the observed empirical data. Secondly, information about any kind of previous knowledge and expectations, in summary called *a priori* knowledge. Obviously such an cost-functional representation implies two elements of uncertainty: On the one hand the observed data, which is distorted respectively partially erased ideal-data and on the other hand the *a priori* knowledge, which is naturally imprecise, incomplete and just a vague perception of reality. Consequently, it would be advantageous to exchange rigid a priori constraints, which will merely result in an accept or reject decision regarding the solution, for constraints that permit a soft decision in form of a weighted answer regarding to what extent a solution is suitable. Within the Bayesian framework such a treatment of uncertainty is realized by an *a priori* probability distribution, which is then combined with the observed data. The prior distribution is adapted according to the extracted information of the observed data, which finally results in an *a posteriori* distribution. But such a Bayesian setting of image analysis makes it mandatory to systematically model and represent distributions in a numerically tractable form. This requirement leads us back to the concept of cost-functionals and its costs mentioned at the beginning.

But first we will discuss another important observation with respect to image analysis: Real world pictures own a lot of spatial regularity, i.e. neighboring pixels often have similar intensity values, edge-lines incline to be continuously linked, textures define homogeneous image regions and different objects show preferred orientations. Markov Random Fields are a suitable modeling structure to capture spatial regularities and to represent the distributions of the Bayesian paradigm in a numerically tractable form. Furthermore, Markov Random Fields represent a structure, which possesses inherent massively parallel processing capabilities. Thus a statistical image analysis approach, formulated on the basis of Markov Random Fields, ideally combines the two contradicting requirements that modern image-processing systems have to fulfill to go significantly beyond state-of-the-art systems: Firstly, the requirement of algorithmic robustness in real world scenarios, which is realized by means of probabilistic models. Secondly, the requirement of real-time processing capabilities, which is put into practice by massively parallel processing.

In summary the chapter is structured as follows: Section 2.1 informally presents the ideas, arguments and advantages of a statistical image analysis approach formulated in the Bayesian framework. This informal presentation is more precisely stated in Section 2.2, which establishes the fundamental definitions, theorems and literature-links to Markov Random Fields (MRFs). Hereafter, Section 2.3 discusses different common optimization schemes of the cost-functionals resulting from a Bayesian image processing approach with MRFs. Section 2.4 describes variants of theoretically well-founded parallel processing strategies of cost-functionals mapped on MRFs. In Section 2.5, we precisely define the class of MRFs, which will be regarded for simulations as well as for massively parallel VLSI implementations. Section 2.6 presents two exemplary image processing models and their particular cost functional derivation. These two image processing models are used throughout this thesis. Finally, Section 2.7 discusses the question of which semiconductor implementation technology variant - digital, analog or mixed-mode - is suitable and advantageous for hardware realizations. Section 2.7 also discusses the question of which VLSI design methodology is required to adequately address the system com-

plexity and specific design needs of massively parallel MRF based processing devices. This discussion also closes the first chapter.

2.1 Bayesian Image Analysis Paradigm

The Bayesian approach to image and signal analysis is characterized [167] by at least two features, which summarize the essential elements of this specific approach. At first this approach requires a detailed model of the statistical mechanisms, which has degraded the observed data. Hence the degradation model itself has to be a probabilistic model. Secondly, all rigid constraints and decisions are substituted by soft ones, i.e. rigid decision outputs are replaced by a measure, which represents the degree to which the outcome is accepted or rejected. If this measure is properly normalized, one receives a probability measure that is named *prior*. Within the Bayesian framework the *a prior* probability measure is systematically combined with the data model to form the *a posterior* distribution, in order to allow signal- and image-processing problems to be finally formulated with probabilities and Bayes law. In detail, Bayes law reads, with y denoting some observed data and x denoting one possible solution of the signal- or image-processing problem, as follows:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}. \tag{2.1}$$

The probability $P(x|y)$ is called the *a posteriori* probability, $P(x)$ the *a priori* probability and $P(y|x)$ the *likelihood* probability. Thus we are looking for a highly probable solution x. A careful interpretation of Equation 2.1 makes it obvious that this approach not only requires the modeling and representation of the probabilities $P(x)$ and $P(y|x)$ in a mathematical exact, flexible and tractable way (cf. Section 2.2) but we also have to provide some estimators for x. As already mentioned, the Bayesian approach to image analysis combines the prior expectations and the fit-to-data (likelihood) in a well-balanced manner. Nevertheless there are still several possibilities to systematically estimate a result, which is hopefully a proper solution of the image processing problem. The following paragraph provides an overview and describes selected estimators, including the *Maximum A Posteriori* (MAP) estimator, which is often used in the Bayesian image analysis community.

Formally an estimator is a mapping from the sample-space $\mathbf{Y}$ to the result-space $\mathbf{X}$, given by

$$\widehat{X} : \mathbf{Y} \longrightarrow \mathbf{X}, \quad \mathrm{y} \longrightarrow \widehat{\mathrm{X}}(\mathrm{y}), \tag{2.2}$$

where $\widehat{X}(y)$ denotes the estimated result - abbreviated by $\widehat{x}$ in the following - and $\mathbf{Y}, \mathbf{X}$ are finite spaces. In order to rigorously measure the quality of the estimated result $\widehat{x}$ compared with the unknown true result x, a loss function L is defined, with $L(x, \widehat{x}) \geq 0$ and $L(x, x) = 0$. The corresponding Bayes risk w.r.t. L and $\widehat{X}$ is given by

$$R(\widehat{X}) = E\left[L(x, \widehat{X}(y))\right] = \sum_{x,y} L(x, \widehat{X}(y)) \cdot P(x, y). \tag{2.3}$$

An estimator $\widehat{X}$, which optimizes, precisely minimizes, the Bayes risk $R(\widehat{X})$ is called Bayes estimator. Thus a Bayes estimator provides an estimation result $\widehat{x}$

which formally reads

$$\widehat{x} = arg \min_{\widehat{X}(y)} \sum_{x} L(x, \widehat{X}(y)) \cdot P(x|y). \tag{2.4}$$

Essentially there are several estimators that minimize the Bayes risk $R(\widehat{X})$ (cf. Equation 2.3) and all these estimators are called Bayes estimators. But merely the three widely used image-processing specific Bayes estimators $\widehat{X}(y)$ and their corresponding loss function L are described in the following text section.

The probably most popular Bayes estimator in image analysis, maximizes the posterior distribution $P(x|y)$ and is consequently denoted as *Maximum A Posteriori* (MAP) estimator. Its loss function is defined by

$$L(x, x^*) = 1 - \Delta_{x^*}(x), \tag{2.5}$$

where the delta function[2] $\Delta_{x^*}(x)$ represents the dirac point-action in x^*. Plugging 2.5 into 2.4 leads to the formulation of the MAP estimator

$$\widehat{x}^{MAP} = \arg \max_{x \in \mathbf{X}} P(x|y). \tag{2.6}$$

Image processing applications and their models, in which contextual information is of subordinated relevance, can be addressed by the *Marginal A Posteriori* (MPM) estimator, where the estimation is performed site by site. The loss function L of this MPM estimator is defined as follows

$$L(x, x_s^*) = \sum_{s \in \Omega} \left(1 - \Delta_{x_s^*}(x_s)\right), \tag{2.7}$$

which leads, by plugging 2.7 into 2.4, to the MPM estimator

$$\forall s \in \Omega : \widehat{x}_s^{MPM} = arg \max_{x_s \in \mathbf{X}} P(x_s|y). \tag{2.8}$$

The last Bayes estimator introduced here is called *Minimum Mean Square* (MMS) estimator. Its corresponding loss function L is given by

$$L(x, x^*) = \sum_{s \in \Omega} (x_s - x_s^*)^2. \tag{2.9}$$

This leads, if we plug 2.9 into 2.4, to the MF estimator

$$\forall s \in \Omega : \widehat{x}^{MF} = \sum_{x \in \mathbf{X}} x_s P(x|y). \tag{2.10}$$

The following section substantiates and formalizes the previous informal discussion on probabilistic image processing models, on the Bayesian approach to image analysis, on the representation of these probabilistic models as computational tractable cost-functionals with their corresponding costs and on the optimization of these costs to determine the solution of the image processing problem. In summary, the central ingredients of this specific image- and signal analysis approach are strictly positive probability measures, which possess differentiating features and Markov Random Fields as topological structures that are advantageous to model spatial interactions as well as to efficiently organize the calculations.

[2] $\Delta_{x^*}(x)$ covers the discrete Kronecker $\delta_{i,j}$, with $\delta_{i,j} = 1$ iff $i = j$ and otherwise $\delta_{i,j} = 0$ (i and j are integers) as well as the continues $\delta(X)$ with $\delta(X) = 0$ for $X \neq 0$ and $\int_{-\infty}^{\infty} \delta(X) dX = 1$

2.2 Markov Random Fields & Gibbs Fields

The concept of Markov Random Fields, although not explicitly denoted as such, was firstly presented by Lévy [106] in a paper published by the Académie française of science dated 1948. Lévy did not use the term Markov Random Field but instead introduced the term double Markov chains. The terminology used today goes back to the classical formalizations of Besag [11]. Foremost the groundbreaking paper of D. Geman and S. Geman from 1984 [59] renewed the interest in Bayesian modeling of signal- and image processing problems on Markovian site grids. Markov Random Fields are a theoretically well-founded setting to appropriately deal with probability distributions on sets of possible images.

The image data is assumed to be discrete and thus images are representable by components of finite product spaces. These product spaces are defined with respect to a set Ω of *sites*. Each site $s \in \Omega$ can have a state x_s from the state space $\mathbf{X}_\mathrm{s}$. Consequently, $\mathbf{X} = \prod_{\mathrm{s}\in\Omega} \mathbf{X}_\mathrm{s}$ represents the complete space of possible configurations, the site set Ω can adopt. Any strictly positive probability measure Π on $\mathbf{X}$ with $\forall x \in \mathbf{X} : \Pi(\mathrm{x}) > 0 \wedge \sum_{\mathrm{x}\in\mathbf{X}} \Pi(\mathrm{x}) = 1$ is called *random field*. Before Markov Random Fields can formally be defined, a structure with interrelating sites has to be established. The site interrelation and thus the global site structure is given by the following definition of site neighborhood systems $\mathcal{N}$.

Definition 2.1 Neighborhood-System $\mathcal{N}$ and Cliques C

- *Every site $s \in \Omega$ has an associated set $\mathcal{N}_s \subset \Omega$ of neighbors so that:*

  1. $s \notin \mathcal{N}_s$,
  2. $t \in \mathcal{N}_s \Leftrightarrow s \in \mathcal{N}_t$

  The collection $\mathcal{N} = \{\mathcal{N}_s, s \in \Omega\}$ of sets $\mathcal{N}_s$ is called a neighborhood system and consequently the sites t are called neighbors of site s. Exactly this relation, if t is a neighbor of s, will be denoted by $< s,t >$ in the sequel.

- *A subset C of Ω is a clique if all its elements are mutual neighbors: $s,t \in C \Leftrightarrow t \in \mathcal{N}_s$, or if C is a singleton $C = \{s\}$ or even the empty set $\emptyset$. In the following we write $C^{\mathcal{N}}$ to denote the set of cliques associated with the neighborhood system $\mathcal{N}$ defined on Ω and $C_i^{\mathcal{N}}$ denoting the cliques for one particular site.*

The pair $(\Omega,\mathcal{N})$ constitutes an undirected graph with nodes defined by the sites $s \in \Omega$ and edges defined by the neighbor relation $< s,t >$. From an opposite point of view an undirected graph constitutes a neighborhood system $\mathcal{N}$.

Obviously there are two extreme cases of neighborhood systems; the first one is given by $\mathcal{N} = \emptyset$ and the second one by $\mathcal{N} = \{\Omega/\{s\} : \forall s \in \Omega\}$. Especially in the *low-level* image processing domain, Ω is very often organized as a regular and finite two-dimensional site-grid $\{(i,j) \in \mathbb{Z} \times \mathbb{Z} : -m \leq i,j \leq m\}$ and the imposed neighborhood system is represented by

$$\mathcal{N}_{i,j} = \{(k,l) : 0 \leq (k-i)^2 + (l-j)^2 \leq v\}, \tag{2.11}$$

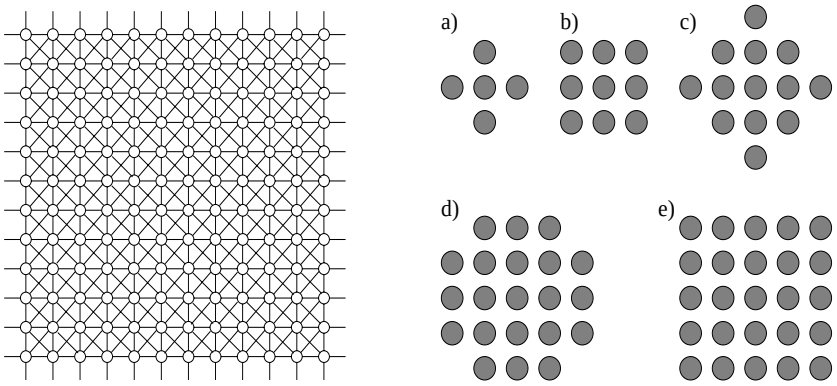


Figure 2.1: Regular two-dimensional site-grid Ω and first five complete neighborhood systems $\mathcal{N}^1$ - $\mathcal{N}^5$. (a)-(e) Neighborhood systems of 1st to 5th order with respect to the regular site-grid.

for some number $v \in \mathbb{Z}$ and w.r.t. to site s with coordinate (i, j). For the sake of simplicity the first five complete neighborhood systems of a regular two-dimensional site-grid Ω are denoted by $\mathcal{N}^1, \mathcal{N}^2, \mathcal{N}^3, \mathcal{N}^4$ and respectively $\mathcal{N}^5$ in the sequel. Figure 2.1 depicts these five neighborhood systems. Naturally, the sites $s \in \Omega$ directly at and near the grid boundaries have incomplete neighborhood systems.

Recapitulating, a strictly positive probability measure Π on $\mathbf{X}$, i.e. $\forall x \in \mathbf{X} : \Pi(\mathrm{x}) > 0 \wedge \sum_{\mathrm{x} \in \mathbf{X}} \Pi(\mathrm{x}) = 1$, is called random field. Incorporating any kind of dependencies between the sites $s \in \Omega$ by a neighborhood system $\mathcal{N}$ (cf. Definition 2.1) and thus influencing the characteristic of each site $s \in \Omega$ only locally, leads to the formal definition of Markov Random Fields.

Definition 2.2 Markov Random Field (MRF)
A random field Π *is called a* Markov Random Field *with respect to the imposed neighborhood system* $\mathcal{N}$ *if* $\forall x \in \mathbf{X}$,

$$\Pi(X_s = x_s | X_t = x_t, t \neq s) = \Pi(X_s = x_s | X_t = x_t, t \in \mathcal{N}_s). \tag{2.12}$$

Apparently small and spatially limited neighborhoods for all sites $s \in \Omega$ of a Markov Random Field are favorable in order to compute the conditional distribution. Exactly this spatially local site-dependency structure together with the upcoming general discussion on the representation of random fields in Gibbsian form [107] [167] lead to efficient and highly parallel algorithms (Section 2.3 and 2.4) and finally in Chapter 3 to the proposed massively parallel hardware architecture template for Markov Random Field based image processing systems.

So far only probability measures have been used to represent image processing problems in the Bayesian framework. Additionally, the introduction of neighborhood systems (cf. Definition 2.1) as well as the Markovian property of random fields with locally determined site characteristics derived from it, has not changed the actual mathematical representation of its distributions. Therefore a computationally tractable representation of distributions is mandatorily required to adequately model the probabilities and utilize them for calculations. This requirement is systematically realized with an approach, in which strictly positive probability measures (random

fields) are represented in the Gibbsian form and with the help of potentials. The formal arguments are introduced in the following.

Like a random field a Gibbs-distribution is a strictly positive probability measure and possesses the following mathematical form:

$$\Pi(x) = \frac{\exp\left(-\mathcal{H}(x)\right)}{\sum_z \exp\left(-\mathcal{H}(z)\right)} = \frac{\exp\left(-\mathcal{H}(x)\right)}{Z}, \tag{2.13}$$

where $\mathcal{H}$ is called the *cost functional* and the denominator $Z = \sum_z \exp\left(-\mathcal{H}(z)\right)$ is called a *partition function*. It follows that every random field, which establishes a strictly positive probability measure, can be written in the above introduced Gibbsian form. In the Gibbsian representation the cost functional $\mathcal{H}$ captures the dependencies respectively mutual interactions of the sites $s \in \Omega$. From a conceptual point of view, one can state that $\mathcal{H}$ encodes the complete image processing model. The representation of $\mathcal{H}$ can moreover be designed to be computationally advantageous, which will be explained directly following.

Despite the fact that probability measures in the Gibbsian form (2.13) are convenient for the representation and computation of distributions, these Gibbs measures Π also have an advantage compared to all other distributions. If one considers an arbitrary distribution p on $\mathbf{X}$, then the following inequality holds

$$\mathbb{E}[\mathcal{H}; p] - S(p) \geq Z, \tag{2.14}$$

where $S(p) = -\mathbb{E}_p[\log p] = -\sum_x p(x) \log p(x)$ denotes the entropy (with the convention $t \log t = 0$ if $t = 0$). The equality condition is only valid for $p = \Pi$. Inequality 2.14 is known in statistical physics as *Gibbs variational principle*. From inequality 2.14 one derives the property of Gibbs distributions being maximally uncommitted distributions, because among all distributions with expected costs $\mathbb{E}[\mathcal{H}; \Pi]$ the Gibbs measure Π has the greatest entropy and consequently is maximally disordered. In addition, the property that the Gibbs distribution optimizes, precisely minimizes, the so-called *free energy* at temperature $T \in \mathbb{R}^{+0}$

$$\mathcal{F} = \mathbb{E}[\mathcal{H}] - TS \tag{2.15}$$

is of crucial importance in the following sections. The temperature term T can be interpreted as a Lagrange parameter for the entropy or - when regarding the inverse temperature $\beta = 1/T$ - as a Lagrange parameter for the expected costs. More details are presented in Section 2.3 on optimization schemes. These fundamental insights into Gibbs measures Π guarantee that the representation of random fields in the Gibbsian form does not introduce any unintended constraints determined by the Gibbs distribution itself.

The overall costs, which are defined by the cost functional $\mathcal{H}$, are split up into their particular mathematical terms and thus into their contributions to the costs, generated by arbitrary subsets of the sites Ω in the general case and by a neighborhood $\mathcal{N}_\Omega$ system in the special case. This is formalized by the potential, which is defined as follows

Definition 2.3 Potential
A family of functions $\{U_A : X(A) \to \mathbb{R},\ A \subset \Omega\}$ *on* $\mathbf{X}$ *is called a* potential*, if the following holds*

1. $U_\emptyset = 0$,
2. $U_A(x) = U_A(y)$ *if* $x_s = y_s \, \forall s \in \Omega$.

The particular costs of potential U *are given by*

$$\mathcal{H}_A^U = \sum_{A \subset \Omega} U_A. \tag{2.16}$$

Furthermore, whenever A *is not a clique and it holds* $U_A = 0$ *w.r.t. the neighborhood system* $\mathcal{N}$*, then* U *is called* neighbor potential *also w.r.t.* $\mathcal{N}$.

Such potentials represent cost functionals and can be used in the probability measure representation in the Gibbsian form. This introduces the spatially local characteristic into the Gibbsian representation of random fields by means of a computational tractable potential. Formally we receive [167]

Definition 2.4 Gibbs Field
A random field Π *is called a Gibbs field w.r.t. the potential* U*, if it has the following form*

$$\Pi(x) = \frac{\exp\left(-\sum_{A \subset \Omega} U_A(x)\right)}{\sum_y \exp\left(-\sum_{A \subset \Omega} U_A(y)\right)}. \tag{2.17}$$

The probability measure Π *is called a* neighbor Gibbs field *whenever* U *is a neighbor potential.*

Now it becomes possible to formulate the equivalence theorem, which formally establishes the fact that a random field can be represented as a Gibbs field with a corresponding potential. Additionally, each Markov Random Field can be represented as a neighbor Gibbs field with the same neighborhood system. The equivalence theorem, sometimes also called Hammersely-Clifford theorem, states:

Theorem 2.1 Equivalence Theorem
If a pair $(\Omega, \mathcal{N})$ *with sites* $s \in \Omega$ *and an imposed neighborhood system* $\mathcal{N}$ *is given, then the following holds:*

1. *A random field is a Markov Random Field (MRF) w.r.t. the neighborhood system* $\mathcal{N}$ *if and only if the random field is a neighborhood Gibbs field w.r.t.* $\mathcal{N}$.
2. $\forall A \subset \Omega$ *of a Markov Random Field with the neighborhood system* $\mathcal{N}$.

$$\begin{aligned} &\Pi(X_s = x_s, \; s \in A | X_s = x_s, \; s \in \Omega / A) \\ &= \Pi(X_s = x_s, \; s \in A | X_s = x_s, \; s \in \mathcal{N}(A)). \end{aligned}$$

The formal proof of this essential equivalence theorem on Markov Random Fields and neighbor Gibbs Fields is technically involved and published in the contemporary literature (e.g. [167] [64] [61]) in various versions. By far the largest number of the equivalence theorem proofs are based on the Möbius inversion. Alternatively the proof can also be conducted based on the factorization theorem of D. Brook [24].

Recapitulating, the previously presented theoretical fundamentals imply that the neighborhood Gibbs representation Π of 2.17 factorizes over the cliques of the neighborhood system. Thus the neighborhood Gibbs field Π is represented by non-negative functions on cliques and shortly reads $\Pi(x) \propto \exp\left(-\sum_{C^{\mathcal{N}}} U_{C^{\mathcal{N}}}(x)\right)$. Consequently, the generic cost-functional $\mathcal{H}$ of a neighborhood Gibbs field respectively Markov Random Field is given by

$$\mathcal{H} = \sum_{i=1}^{n} \sum_{C_i^{\mathcal{N}}} U_{C_i^{\mathcal{N}}}, \tag{2.18}$$

with $C_i^{\mathcal{N}}$ denoting the cliques of site s_i with respect to the neighborhood-system $\mathcal{N}$ defined on Ω. The cost-functional representation 2.18 serves as basis for the derivation of the universal constituents in Section 3.1 of the following chapter. Exactly this functional representation by means of cliques establishes a numerically tractable and computationally efficient mathematical form of neighborhood Gibbs fields and consequently also of Markov Random Fields by characterizations merely defined by the neighborhood system. Obviously, the computational efficiency directly depends on the size of the neighborhood system and the locality of the characterizations derived out of it. Limited neighborhood systems with smaller cliques are computationally advantageous compared with proliferated neighborhood-systems spanning larger portions of the sites. In the following section we turn to a family of Markov Chain Monte Carlo Methods (MCMCM), which explore a Markov Random Field and optimize its corresponding costs, defined by the cost-functional $\mathcal{H}$.

2.3 Optimization Schemes

Systematically investigating the performance of developed probabilistic signal- and image processing models and thus exploring its Markov Random Fields is apparently a key task in the Bayesian approach to image analysis with MRFs. The cost functional $\mathcal{H}$ defines the costs of the model configurations and consequently, it poses a combinatorial optimization problem. As $\mathcal{H}$ is typically a non-convex function, it is advisable to develop global optimization methods, i.e. optimization methods which principally escape from local minima and converge to a global minimum in the cost landscape defined by $\mathcal{H}$. Often we have to restore to heuristic for shortcutting the exponentially slow global optimization problem.

The underlying ideas and concepts recapitulated in this section on optimization schemes undoubtedly have their roots in statistical physics and among others go back to the works of Jaynes [89] [88] on maximum entropy inference principles dated 1957. Especially the global optimization principle with annealing was originally motivated and developed on the basis of observations, which were made with respect to physical systems at thermal equilibrium. It was observed that the structure and solidness of materials, mainly metal and glass, improved with the help of an annealing process of controlled heating-up and cooling-down. Although the early simulation work of Metropolis et al. [122] on the dynamics of physical systems at different temperatures marked the foundation of annealed optimization methods, foremost Kirkpatrick et al. [95] [96] and independently Cerny [31] recognized the

fact that a stochastic search scheme, mimicking the annealing procedure normally applied to physical systems in order to settle the systems in its equilibrium ground-state, is favorable when having to solve optimization problems. Besides Metropolis et al. [122] also Pincus [135] and Khachaturyan et al. [94] contributed to the early conceptual and theoretical insights into the analogy of physical systems and combinatorial optimization problem-settings, which finally culminated in the work of Kirkpatrick et al. and Cerny. The physical roots are particularly important as they lead to the introduction of a varying temperature term T, controlling the annealing procedure, into the optimization algorithms of Kirckpatrick and Cerny. These optimization methods are called, in reminiscence of the physical jargon, *Simulated Annealing* (SA).

Formally the simulated annealing methods perform a random search process in the cost landscape of the function $\mathcal{H}$, whereby the shape of the cost landscape is controlled by the temperature term T and the state transition process is mathematically realized by an inhomogeneous and time-discrete Markov chain in order to determine a configuration with optimized costs. The temperature term T affects the cost landscape of $\mathcal{H}$ in such a way that configuration changes with an overall cost difference smaller than the actual temperature value will not be represented by the cost landscape. Consequently, the temperature term T influences the roughness of the cost landscape and thus the appearance of local and global maxima respectively minima of the cost landscape, in which the search is taking place. It can be shown that the SA induced Markov chain converges to the stationary distribution of the Gibbs distribution and finally establishes the relation to Jaynes maximum entropy principle. In the following we describe variants of simulated annealing methods and for this purpose an additional Definition 2.5 on site visitation schemes is required.

A site visitation scheme on Ω is defined as follows:

Definition 2.5 Site Visitation Scheme
An enumeration $V = \{s_1, ..., s_m, ..., s_{|\Omega|}\}$ of the sites Ω is called a site visitation scheme, where $V(m)$ denotes the distinct site s_m.

In Section 2.1 different estimators have been introduced. Estimators, which determine maximum modes, are of special interest within the Bayesian approach to image analysis because methods like simulated annealing [144] [49] - which will shortly be defined in more detail - can determine the maximum modes of the discrete configuration space $\mathbf{X}$.

The maximum a posteriori mode Estimator 2.6 for instance represents such an estimator [167]. With a Gibbs field and a cost functional given, $\Pi(x) \propto \exp(\mathcal{H}(x))$, we will introduce the previously explained temperature term as inverse temperature $\beta = 1/T$. Hence, the Gibbs field with cost functional $\mathcal{H}$ and inverse temperature reads $\Pi(x) = (Z)^{-1} \exp(-\beta\mathcal{H}(x))$. For a steadily increasing temperature the mass of the Gibbs distribution Π accumulates around the modes [165], which represent the global minimizers of $\mathcal{H}$. Consequently, sampling from this distribution extracts the maximum modes. Apparently, there exist different sampling strategies with corresponding transition probabilities to accept new configurations. Four acceptance rules to adopt new configurations are described in the following. This comprises

1. *Metropolis* criterion (cf. Definition 2.6),

2. *Heat-Bath* criterion (cf. Definition 2.7),

3. *Gibbs Sampler* (cf. Definition 2.8) and

4. *Iterated Conditional Modes* criterion (cf. Definition 2.9).

The generic structure of optimization procedures based on simulated annealing is presented in Algorithm 2.1, whereas this SA-algorithm is designed to be used with respect to the Metropolis criterion as well as with respect to the Heat-Bath criterion.

Algorithm 2.1 Simulated Annealing (SA) - Metropolis-Hastings Variant

Require: (1) Arbitrary initial configuration $\mathbf{x}_0$, (2) temperature schedule $\beta(n)$

INITIALIZE n=1
while $\neq$ *Convergence*() **do**
  for $z = 1, ..., z_{MAX}$ **do**
    SAMPLE $x^{new}_{V(m)} \in \mathbf{X}_{\mathrm{V(m)}}$ from the proposal distribution q on $\{\Omega,\mathbf{X}\}$
    if $T^n_{Prob} = 1$ according to 2.19 **then**
      $x_{V(m)} = x^{new}_{V(m)}$
    else
      if $T^n_{Prob} >$ RANDOM(0,1) according to 2.19 $\wedge$ 2.20 **then**
        $x_{V(m)} = x^{new}_{V(m)}$
      else
        $x_{V(m)} = x_{V(m)}$
      end if
    end if
  end for
  n=n+1
end while

The transition probabilities T^n_{Prob} for the temperature scheduling steps $\beta(n)$ and the cost-functional $\mathcal{H}$ on $\mathbf{X}$ of the Metropolis Markov chain [122] are defined as follows:

Definition 2.6 (Acceptance Rule - Metropolis Criterion)
The acceptance rule of the Metropolis criterion is given by the state transition probabilities T^n_{Prob}. The transition probabilities are given by

$$T^n_{Prob}(x', x) = \begin{cases} \min\left\{1, \exp\left[\dfrac{\mathcal{H}(x) - \mathcal{H}(x')}{\beta(n)}\right]\right\} & , \text{ if } x' \neq x \\ 1 - \displaystyle\sum_{z \in \mathbf{X}} \exp\left[\dfrac{\mathcal{H}(z) - \mathcal{H}(x')}{\beta(n)}\right] & , \text{ if } x' = x. \end{cases} \tag{2.19}$$

Alternatively, Definition 2.7 represents another acceptance rule called Heat-Bath criterion [67] with slightly different transition probabilities T^n_{Prob} compared to the Metropolis criterion. The structure of Algorithm 2.1 makes it obvious that for the Metropolis criterion as well as for the Heat-Bath criterion a two-phase sampling

procedure is required. In the first phase configuration candidates are sampled from the proposal distribution q and the following second phase realizes the acceptance decision. In its original formulation Metropolis [123] [122] has restricted the proposal distribution q to be symmetric, i.e. for the proposal distribution it holds $q(x_1, x_2) = q(x_2, x_1)$. This constraint has been weakened by Hastings [67] by formulating an arbitrary proposal distribution q.

Definition 2.7 (Acceptance Rule - Heat-Bath Criterion)
The acceptance rule of the Heat-Bath criterion is given by the state transition probabilities

$$T^n_{Prob}(x', x) = \begin{cases} \dfrac{1}{1 + \exp\left[\dfrac{\mathcal{H}(x) - \mathcal{H}(x')}{\beta(n)}\right]} & , \text{ if } x' \neq x \\ 1 - \displaystyle\sum_{z \in \mathbf{X}} \left(\dfrac{1}{1 + \exp\left[\dfrac{\mathcal{H}(z) - \mathcal{H}(x')}{\beta(n)}\right]} \right) & , \text{ if } x' = x. \end{cases} \tag{2.20}$$

Obviously the two-phase procedure of the Metropolis and Hastings algorithm implies a significant limitation. An improved variant of an one-phase sampling scheme with an increased acceptance rate for possible configurations is given by the Gibbs sampler, depicted in Algorithm 2.2, and its transition probabilities (cf. Definition 2.8). This acceptance scheme is advantageous, if it becomes possible

Algorithm 2.2 Simulated Annealing (SA) - Gibbs Sampler Variant

Require: (1) Arbitrary initial configuration $\mathbf{x}_0$, (2) site visitation scheme V, (3) temperature schedule $\beta(n)$

```
INITIALIZE n=1, m=1
while ≠ Convergence() do
   for z = 1, ...., z^MAX do
      COMPUTE W = {H_V(m)(x)|∀x ∈ X_V(m)}
      SAMPLE W according to 2.8
      x_V(m) = x^W_V(m)
      m=m+1
   end for
   n=n+1;
end while
```

to calculate the local characteristics fast and efficiently, i.e. the local costs of all configurations with respect to the site regarded and the neighborhood defined have to be evaluated effectively. The Gibbs sampler was first analytically studied in the seminal paper of D. Geman and S. Geman [59]. Additionally, this paper presents a convergence proof of a logarithmic temperature annealing schedule $\beta(n)$, which shows that the Gibbs sampler converges to the uniform distribution of the global

minimizers of the cost-functional $\mathcal{H}$.

Definition 2.8 (Acceptance Rule - Gibbs Sampler)
The transition probabilities of the Gibbs sampler are given by

$$T^n_{Prob}(x', x) = \frac{\exp[-\mathcal{H}(x')/\beta(n)]}{\sum_w \exp[-\mathcal{H}(w)/\beta(n)]}. \tag{2.21}$$

A fast variant to the optimization of the cost-functional $\mathcal{H}$ is the Iterated Conditional Modes (ICM) method illustrated in Algorithm 2.3. ICM was originally proposed by J. Besag [13]. This algorithm also belongs to the Simulated Annealing family but represents a special case of these methods. Essentially ICM is a temperature frozen Gibbs sampler at $\beta = 0$, i.e. ICM follows a solely greedy strategy and thus accepts only configurations with lower costs (Eq. 2.22) compared to the actual configuration. Consequently, the ICM optimization scheme converges to a

Algorithm 2.3 Iterated Conditional Modes (ICM)

Require: (1) Good initial configuration $\mathbf{x}_0$, (2) site visitation scheme V
1: INITIALIZE m=1
2: **while** $\neq$ *Convergence*() **do**
3: CALCULATE $\mathcal{H}_{V(m)}(x_m)\ \forall x_{V(m)} \in \mathbf{X}_{\mathrm{V(m)}}$
4: UPDATE value $x_{V(m)}$ according to 2.9
5: SET m=m+1
6: **end while**

local minimum of the cost landscape. Obviously a local minimum can be significantly worse than the global cost minimum and thus the generated solution of the ICM method might be of poor quality. In the worst case ICM might even produce completely useless solutions. Apart from the ICM variant, in general Simulated Annealing is an extremely robust optimization method.

Definition 2.9 (ICM Acceptance Rule)
The acceptance rule of the local and deterministic algorithm ICM (Iterated Conditional Modes) with a site visitation scheme V(m) is defined by

$$x_{V(m)} = \arg \min_{x_{V(m)} \in \mathbf{X}_{\mathrm{V(m)}}} \mathcal{H}_{Vm}. \tag{2.22}$$

However, the main disadvantage of SA optimization methods is the exhaustive usage of computing resources, manifested by their long run-times until a satisfactorily solution has been generated respectively found by the random search process. The reason for the long computational run-times is substantiated by the logarithmic temperature annealing schedules [59], and even the improvement of Hajek [65] did not eliminate this deficiency of SA methods. For pragmatic reasons a faster temperature cooling schedule is sometimes used, but as a far-reaching consequence the convergence to a global minimum of $\mathcal{H}$ is no longer guaranteed.

The optimization paradigm *Deterministic Annealing* (DA) represents an important alternative if run-time constraints are an essential application concern. In

contrast to SA approaches, which sample solutions from the Gibbs distribution, DA directly determines the distribution with its free parameters. The DA optimization scheme is depicted in Algorithm 2.4 and follows the presentation in [70]. Essentially, Deterministic Annealing performs a deterministic optimization process over a probabilistic configuration space. It has been shown in [137] that the Gibbs distribution factorizes under certain conditions, and consequently it becomes possible to represent the Gibbs distribution Π as a product of probabilities $\mathbb{E}[\mathcal{H}_i]$. Hence, the combinatorial optimization problem has been relaxed to a continuous optimization problem. It will be solved by a two-step alternating calculation scheme, which determines the average costs $\mathbb{E}[.]$ in the first step and optimizes the parameters in the second processing step. The computational temperature will not affect the search

Algorithm 2.4 Deterministic Annealing (DA)

Require: (1) Temperature schedule $\beta(n)$, (2) site visitation scheme V

1: INITIALIZE m=1
2: **while** $\neq$ *Convergence*() **do**
3: CALCULATE $E[\mathcal{H}_{V(m)}(x, \Theta)]$ w.r.t. $\Pi(x|\Theta)$
4: OPTIMIZE $E[\mathcal{H}_{V(m)}(x, \Theta)]$ w.r.t. Θ
5: SET m=m+1
6: **end while**

process, which is deterministic, as the randomness is encoded in the probabilistic configuration space in DA [70]. By freezing the temperature T to one we receive the classical *Expectation Maximization* algorithm [39]. For the unsupervised segmentation model, which will be presented in Section 2.6.2, the update formulas for the alternating deterministic annealing optimization scheme (E-step and M-step) will be derived.

2.4 Parallel Processing Strategies

The optimization methods previously described (cf. Section 2.3) either use site visitation schemes defined by the proposal distribution q or predefined and fixed site visitation schemes $V(m)$ (cf. Definition 2.5) in order to organize the update of the current configuration. Until now it has been assumed that the configuration x changes only on one site at the same time and sequentially over all sites $s \in \Omega$, i.e. for a given configuration x and one selected site s, determined by the proposal distribution or the site visitation scheme, the old site-local value x_s is eventually replaced by a sample y_s from the local characteristic $\Pi(x_s|x_{\Omega/\{s\}})$. Following this the next configuration-update eventually alters the actual configuration $y_s x_{\Omega/\{s\}}$, which has been generated in the previous step. After all sites have been selected and eventually updated their local value, a full site sweep is completed. Consequently, such a sequential site processing scheme requires $\mathcal{O}(n)$ steps, with $n = |\Omega|$ to finish one sweep over all sites. A significant speed-up of processing could principally be achieved, if several independent processing units work in parallel on the sites. The highest degree of parallelism is obtained, if each site $s \in \Omega$ represents an independent processing unit. In this massively parallel processing setting a single sweep can be

completed in $\mathcal{O}(1)$ steps, as all sites update their local values simultaneously.

However, when regarding the situation with respect to parallel processing strategies, it proves to be much more complicated. A naive implementation of a parallel processing scheme can lead to completely erroneous results, since the convergence is eventually no longer guaranteed and an oscillatory update behavior might occur. Thus it is indispensable to formulate fundamental rules, which ensure a correct parallel processing scheme for a concrete MRF model. The following two limit Theorems 2.2 and 2.3 establish the formal justification of parallel processing strategies based on the Gibbs sampler, which possess a guaranteed converge behavior [167] [59]. Obviously, these parallel processing strategies can equally be combined with the deterministic ICM method, because the ICM method is a temperature frozen Gibbs sampler. Thereby, it is ensured that the parallel processing strategy will not further affect the performance of the ICM method, which converges only to a local minimum.

For the rest of this section $\mathbf{I}$ denotes a set of sites from Ω and x as before denotes a given configuration. The simultaneous update of the sites $s \in \mathbf{I}$ is defined by the following transition probability

$$R_{\mathbf{I}}(x,y) = \begin{cases} \prod_{s\in\mathbf{I}} \Pi(y_s|x_t, t \neq s) & \text{if } y_{\Omega/\mathbf{I}} = x_{\Omega/\mathbf{I}} \\ 0 & \text{otherwise.} \end{cases} \tag{2.23}$$

The probability to generate the configuration y from the configuration x in one single site sweep is given by the composition $Q(x,y) = R_{\mathbf{I}_1}...R_{\mathbf{I}_\mathrm{k}}(x,y)$, where $\mathbf{I} = \{\mathbf{I}_\mathrm{i}|1 \leq \mathrm{i} \leq \mathrm{K}\}$ is the complete partitioning of Ω into K site-sets $\mathbf{I}$. A site set $\mathbf{I}$ is called *independent* with respect to the imposed neighborhood system $\mathcal{N}$ on Ω, if $\mathbf{I}$ contains no neighboring sites. Figure 2.2 shows the partition of a regular site-grid Ω with first and second order neighborhood system into independent sets $\mathbf{I}$. Consequently, the conditional probabilities of the sites s of an independent site set $\mathbf{I}$ solely depend on the values of $\mathbf{I}$. Thus the transition probability $R_{\mathbf{I}}(x,y) = \Pi_{s_1}...\Pi_{s_{|\mathbf{I}|}}(x,y)$ is valid for every permutation of the sites in the independent set $\mathbf{I}$. The transition probability $Q(x,y) = R_{\mathbf{I}_1}...R_{\mathbf{I}_\mathrm{k}}(x,y)$ thus corresponds to the transition probability generated by a single sweep, which runs serially over all sites $s \in \Omega$. If a cost functional $\mathcal{H}$ on $\mathbf{X}$ induces a Gibbs field Π, then it can be shown by extending the results [59] of serial site sweeps that sampling from the Gibbs field Π is characterized by

Theorem 2.2
For every initial distribution ν the marginals νQ^n converge to the Gibbs field Π if (1) the site-set Ω is partitioned into independent sets $\mathbf{I}$ and if (2) n goes to infinity.

In the following the annealing-procedure and the optimization of the cost-functional $\beta(n)\mathcal{H}$ with Gibbs field $\Pi_{\beta(n)}$, where $\beta(n)$ denotes a valid schedule of the temperature value on the n-th sweep over the sites, is regarded. The transition probability is given by $Q(x,y)_n = R_{\mathbf{I}_1,\mathrm{n}}...R_{\mathbf{I}_\mathrm{k},\mathrm{n}}(x,y)$, if $\mathbf{I}$ represents a partition of Ω into K independent sets $\mathbf{I}$. Again, it can be shown by extending the results of serial site-sweeps (cf. [166]) that the following holds true for the parallel updating of independent site sets $\mathbf{I}$.

Theorem 2.3
Let $\mathbf{I}$ *be a partition of* Ω *into independent sets* $\mathbf{I}$ *and* $\beta(n)$ *a valid schedule for the temperature value in annealing, then for an arbitrary initial distribution* ν *the marginals* $\nu Q_1 ... Q_n$ *converge to the uniform distribution on the minimizers of* $\mathcal{H}$ *as* n *goes to infinity.*

Consequently, the essential point for the design of convergent parallel processing schemes, is the partitioning of the sites $s \in \Omega$ into *independent* sets $\mathbf{I}$, i.e. the sets $\mathbf{I}$ contain no sites that are neighbors. For some models it is not a straightforward task to partition the sites into independent sets $\mathbf{I}$. However, for the MRF-class (cf. Definition 2.10) considered in this thesis, which will be formally defined in Section 2.5 directly following, the partitioning into independent sets $\mathbf{I}$ will be simplified as only regular two-dimensional site-grids with limited neighborhood systems are regarded (see e.g. Figure 2.2). The partitioning of a site set Ω with an imposed neighborhood system $\mathcal{N}$ can be illustrated by tinting the sites to provide all neighboring sites with different colorings. The smallest number of colors defines the cardinality of the independent sets and thus the maximum degree of parallelism, which can be realized by this strategy with regard to a specific set of sites Ω and its corresponding neighborhood system $\mathcal{N}$.

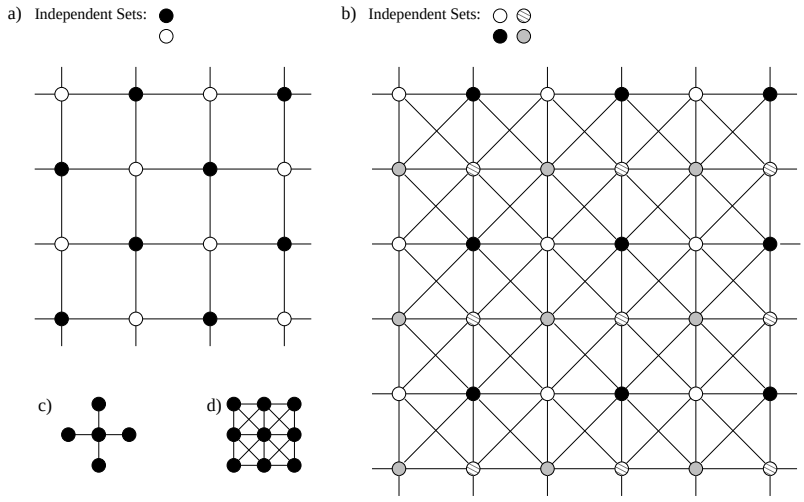


Figure 2.2: Independent Sets.(a) Two independent sets $\{\mathbf{I}_1, \mathbf{I}_2\}$ for a regular site-grid with first order neighborhood system (b) Four independent sets $\{\mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_3, \mathbf{I}_4\}$ for a regular site-grid with second order neighborhood system (c) First order neighborhood system $\mathcal{N}^1$ and (d) second order neighborhood system $\mathcal{N}^2$ for regular site-grid Ω

Obviously, there are other parallel processing strategies for Markovian site grids (see e.g. [5] or [165]), which are equally applicable to improve the calculation speed of Markov Random Field based statistical signal- and image processing models. But these strategies are of minor interest for the purpose of this thesis, as they are structurally not advantageous [4] [6] for massively parallel hardware-architectures and VLSI realizations.

2.5 Considered MRF-Class

Taking Definition 2.1 and Definition 2.2 without any restrictions into account, the class of Markov Random Fields is obviously quite large and diverse. Many structurally and functionally different MRFs can be composed, if no additional constraints are imposed on the arrangement and number of the sites s_i, the neighborhood systems $\mathcal{N}$, the neighbor potentials U and the optimization methods. Due to the arrangement of the distinct sites and the used neighborhood system not all of the MRFs, which are principally possible, are equally suitable both for massively parallel implementations and for VLSI realizations. This is the reason why we restrict ourselves to a sub-class of Markov Random Fields. This MRF sub-class is characterized by features and limitations, which are advantageous for massively parallel implementations and VLSI realizations.

This limitation to a sub-set of MRFs does not imply a severe restriction, as the regarded class of MRFs (see Definition 2.10) is still comprehensive enough to capture most of the Markov Random Field models, which have been proposed in the literature on *low-level* image processing. Thus Definition 2.10 ensures that the contemplated MRF class is of scientific and industrial relevance and does not only comprise an oversimplified and unrealistic caricature of a Markov Random Field.

Definition 2.10 MRF Class
The class of MRFs under consideration possesses the following features and restrictions:

- *The set of sites* $\{s_i : 1 \leq i \leq |\Omega|\}$ *are placed in the plane and arranged among each other, so that they form a regular and equally spaced site-grid* Ω*.*
- *Only neighborhood systems* $\mathcal{N}^1$ *-* $\mathcal{N}^5$ *(first to fifth order) are admissible.*
- *There are no restrictions in terms of the neighbor potentials* $\sum_{C_i^{\mathcal{N}}} U_{C_i^{\mathcal{N}}}$*.*
- *There are no restrictions with regard to the used optimization methods.*

Together, all these features and limitations simplify and support the VLSI design of massively parallel Markov Random Field processing devices and at the same time are general enough to define a comprehensive MRF image processing class. The arrangement of the particular sites s_i as a regular two-dimensional grid establishes a structure gantry, which is well-suited for large scale integration. This structure is meshed by neighborhood systems $\mathcal{N}$ of maximally fifth order. Hence, the connections between the sites remain spatially local and regular, which is again advantageous for VLSI realizations. In principle there are no restrictions regarding the neighbor potentials. However, not all arithmetic operations are equally advantageous for VLSI implementations with respect to the required chip area and the cycle-time behavior. This fact has to be taken into account, if a specific MRF is intended to be implemented in massively parallel VLSI structures.

The used optimization method is also not restricted, but undoubtedly the methods differ with respect to the computational complexity, the required implementation resources and the results produced. The Iterated Conditional Modes (ICM)

method is characterized by a low computational complexity as well as a low usage of implementation resources, but the produced results crucially depend on the initial configuration (see Algorithm 2.3). In contrast to the ICM method, stochastic methods of the simulated annealing family possess a high computational complexity as well as a significant usage of implementation resources, but the produced results are largely independent of the initial configuration (see Algorithm 2.1, 2.2).

2.6 Exemplary Image Processing Models

Two types of *low-level* image processing problems occur at regular intervals and are of central importance for nearly every image processing chain and its technical realization. The first type of early vision problems comprises the task of noise removing with simultaneously preserving distinct image features to avoid the blurring of these image features, which will in the end lead to the loss of structure and image details. One prominent and robust feature, representing a generic image structure, is definitely that of intensity changes in image data. These intensity changes tag boundaries of different objects respectively regions. Thus a device for noise removing and simultaneously preserving intensity changes, which is both algorithmically robust and capable of real-time processing would be desirable in view of projected technical machine vision systems. To further stress the practical and far-reaching relevance of this elementary and specific model for a brought scope of applications, we refer the reader to the summary section of this chapter for a more detailed discussion on some scenarios of industrial utilization. This discussion formally justifies the importance of this concrete model and the demand of realizing it as a massively parallel processing device. The model's assumptions and the derivation of its cost function is described in Section 2.6.1. Furthermore, the update equation for the model's free parameter is derived.

The second type of early vision problems comprises the task of image clustering, i.e. partitioning of the image sites into a set of disjoint clusters. One of the very first steps toward any kind of image understanding and object recognition is segmenting the image sites into a predefined number k of disjoint clusters based on a well-defined measure between distinct data features. Statistical similarity or homogeneity, measured in the spatially local neighborhood of each image site, are robust features. Histograms, locally generated at each site, represent empirical distributions and, from a statistical point of view, are robust and reliable features. These empirical distributions, the histograms, are compared with estimated prototypical distributions of each cluster. The assignment of a site to a specific cluster is then based on the outcome of this measurement. Again, we refer the reader to the summary section of this chapter for a discussion on essential scenarios of industrial utilization, which underpins the relevance of this image processing model. The model's structure and the derivation of its cost function, represented by its negative data log-likelihood, is described in Section 2.6.2. In addition, we derive the update equations for the cluster assignments and the free parameters in the Deterministic Annealing framework.

2.6.1 Cost Function - Edge Preserving & Noise Removing

The following paragraph describes the *Edge Preserving & Noise Removing* image processing model and derives its corresponding cost function $\mathcal{H}$ within the Bayesian framework. Essentially the complete *Edge Preserving & Noise Removing* image processing model is composed of two model parts: The first part of the model describes the degradation process of the observed data and thus represents the *likelihood* probability within the Bayesian setting. The second part of the model describes the assumptions about the expected results and thus represents the *a priori* probability within the Bayesian formulation. Because we can efficiently determine the costs of $\mathcal{H}$, which will become evident during the following discussion, it is possible to calculate and optimize the posterior. In this case maximizing the posterior leads to the MAP estimator (cf. Section 2.1, Eq. 2.6).

Let us assume that a set of image sites $\Omega = \{s_i | i = 1, ..., n\}$ is defined as follows: These sites are organized as a regular two-dimensional grid of size $N \times M$, $N, M \in \mathbb{N}$ with an imposed spatial neighborhood system of first order $\mathcal{N}^1$, i.e each site is connected with its directly neighboring sites - the site above, the site beneath, the leftmost side and the rightmost side respectively. Hence, the model respects the structural features of the considered MRF class, introduced in Section 2.5 and Definition 2.10. The measure of the intensity changes between neighboring sites is realized by a function with a cup shaped functional progression [167], which for instance can take on the form of $\kappa(u) = \frac{-1}{1+|u|/\gamma}$ or alternatively of $\kappa(u) = \frac{-1}{1+(u/\gamma)^2}$. The parameter γ adjusts the function to respect intensity changes of a specific difference, i.e. a single large intensity step is cheaper than smaller ones. Additionally, the noise, which is responsible for any distortion of the observed image data, is represented by a particular function. It is assumed that the noise can be described by an independent white Gaussian noise distribution, with m representing the mean- and σ^2 denoting the variance-parameter.

The free parameters of the model, which have to be estimated, are summarized and denoted by $\mathbf{\Theta} = \{\mathrm{m}, \sigma^2, \gamma\}$ in the sequel. The previous description of the *Edge Preserving & Noise Removing* image processing model introduced the structure of the model and commented on all its parts. Consequently, we can summarize the model with respect to *(1) observed data*, *(2) latent variables*, (3) *model specific functions* and *(4) model parameters* as follows:

- Observed data $\mathbf{y} = \{\mathrm{y_i} | \mathrm{i} = 1, ..., \mathrm{n}\}$
  - y_i, $1 \leq i \leq n$, describing a raw image datum, which is locally observed at site s_i, $1 \leq i \leq n$ of the site-grid Ω. The set of observations, summarized by $\mathbf{y} = \{\mathrm{y_i} | \mathrm{i} = 1, ..., \mathrm{n}\}$ represents the completely observed raw image, which in the case of that model is assumed to be corrupted by noise.
- Latent data $\mathbf{x} = \{\mathrm{x_i} | \mathrm{i} = 1, ..., \mathrm{n}\}$
  - x_i, $1 \leq i \leq n$, denoting a restored image datum at site s_i, $1 \leq i \leq n$, i.e. the additive white Gaussian noise component is removed while simultaneously preserving intensity changes. The set $\mathbf{x} = \{\mathrm{x_i} | \mathrm{i} = 1, ..., \mathrm{n}\}$ represents the completely restored image.

- Gaussian Degradation Model G
  - The observed data $\mathbf{y}$ is the result of a degradation process G. This noise process is additive with respect to each particular y_i and the noise variables are identically and independently distributed by a Gaussian law of mean m and variance σ^2, i.e. the pdf is given by $f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{(x-m)^2}{2\sigma^2})$.
- Intensity Change Grading $\kappa(v_1, v_2)$
  - Given two gray-scaled values v_1 and v_2, the intensity difference $v_1 - v_2$ is graded by the function $\kappa(v_1, v_2) = \frac{-1}{1+|v_1-v_2|/\gamma}$ or $\kappa(v_1, v_2) = \frac{-1}{1+(v_1-v_2/\gamma)^2}$, alternatively.
- Parameters $\mathbf{\Theta} = \{\mathrm{m}, \sigma^2, \gamma\}$
  - m, representing the *mean* parameter of the assumed identical independent Gaussian noise distribution,
  - σ^2, denoting the *variance* parameter of the Gaussian noise distribution.
  - γ, representing the scaling value of the intensity differences evaluating function.

Although the free model parameter can be estimated during the processing, we restrict the parameter estimation procedure to the variance σ^2 of the specific Gaussian noise, because the mean m is set to zero. Furthermore it is assumed that the remaining free parameter γ is predefined by the user. Thus the list of free parameters in the following discussion becomes reduced to the variance σ^2, i.e. $\mathbf{\Theta} = \{\sigma^2\}$. By means of the model fundamentals described above, we can define the posterior distribution in Gibbsian form (cf. Eq. 2.13) and the detailed posterior cost function of that model.

The posterior distribution in Gibbsian form is given by

$$P(\mathbf{x}|\mathbf{y}) = \frac{\exp(-\mathcal{H}(\mathbf{x}, \mathbf{y}))}{\sum_{\mathbf{z}\in\mathbf{X}} \exp(\mathcal{H}(\mathbf{z}, \mathbf{y}))}, \tag{2.24}$$

with $\mathcal{H}(\mathbf{x}, \mathbf{y})$ denoting the posterior cost function. The posterior cost function itself is composed of two components, the data degradation model and the a priori assumptions.

The a priori assumptions are summarizing covered by the intensity change grading function $\kappa(v_1, v_2)$ and thus the a priori cost function $\mathcal{H}_1$ reads as follows

$$\mathcal{H}_1(\mathbf{x}) = \sum_{\mathrm{i}=1}^{\mathrm{n}} \sum_{\mathrm{t}=1}^{4} \sum_{<\mathrm{x_i}, \mathrm{x_t}>} \kappa(\mathrm{x_i}, \mathrm{x_t}). \tag{2.25}$$

The degradation process is additive and Gaussian with mean m and variance σ^2. Furthermore it is assumed that $m = 0$. Hence, we receive the cost function $\mathcal{H}_2$ of the degradation process as

$$\mathcal{H}_2(\mathbf{y}, \mathbf{x}) = \sum_{\mathrm{i}=1}^{\mathrm{n}} \frac{(\mathrm{x_i} - \mathrm{y_i})^2}{2\sigma^2} \tag{2.26}$$

for the degradation model.

Consequently, putting Eq. 2.25 and Eq. 2.26 together, the posterior cost function $\mathcal{H}$ of the *Edge Preserving & Noise Removing* image processing model finally reads

$$\begin{aligned} \mathcal{H}(\mathbf{x},\mathbf{y}) &= \mathcal{H}_1(\mathbf{x}) + \mathcal{H}_2(\mathbf{y},\mathbf{x}) && (2.27)\\ &= \sum_i \sum_{t=1}^{4} \sum_{<x_i,x_t>} \kappa(x_i,x_t) + \frac{1}{2\sigma^2}\left(x_i - y_i\right)^2. && (2.28) \end{aligned}$$

The following section presents the optimization strategy for the described *Edge Preserving & Noise Removing* image processing model. The strategy is characterized by an alternating calculation scheme, which at first determines the free parameters and following the optimization of Eq. 2.28.

Alternating Optimization

We consider an optimization procedure, which simultaneously estimates the model parameters in an edge preserving way by removing noise [13] [107]. The particular steps of the optimization procedure are given below:

1. Start with an initial estimate of $\mathbf{x}$ as well as with an useful initialization of the free parameter $\mathbf{\Theta} = \{\sigma^2\}$.
2. Estimate [19] the free parameter by $\sigma^2 = \arg\max P(\mathbf{y}|\mathbf{x},\sigma^2)$.
3. Update the latent data by $\mathbf{x} = \arg\max \mathrm{P}(\mathbf{x}|\mathbf{y},\sigma^2)$ based on the current values of $\mathbf{x}$ and σ^2.
4. Continue with step 2 for a number of iterations.

For step 2 one has to maximize the likelihood of σ^2, which is given by

$$P(\mathbf{y}|\mathbf{x},\sigma^2) = \frac{1}{(\sqrt{2\pi\sigma^2})^{\mathrm{n}}} \exp\left(-\sum_{\mathrm{i}=1}^{\mathrm{n}} \frac{(\mathrm{x_i}-\mathrm{y_i})^2}{2\sigma^2}\right), \tag{2.29}$$

or equivalently the log likelihood

$$\ln P(\mathbf{y}|\mathbf{x},\sigma^2) = -\mathrm{n}\ln(2\pi) - \mathrm{n}\ln\sigma - \sum_{\mathrm{i}=1}^{\mathrm{n}} \frac{(\mathrm{x_i}-\mathrm{y_i})^2}{2\sigma^2}. \tag{2.30}$$

The free parameter σ^2 is optimized by setting the partial derivation of the log likelihood 2.30 equal to zero.

Hence the optimization condition for the model's free parameter σ^2 reads

$$\frac{\partial}{\partial\sigma}\left(-n\ln(2\pi) - n\ln\sigma - \sum_{i=1}^{n}\frac{(x_i-y_i)^2}{2\sigma^2}\right) = 0. \tag{2.31}$$

When finally solving Eq. 2.31, we receive the formula to determine the model parameter σ^2 in step 2, which reads

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i-y_i)^2. \tag{2.32}$$

For step 3 one of the optimization methods, described in Section 2.3 has to be utilized.

We continue the discussion with the description of the *Unsupervised Histogram Segmentation* model.

2.6.2 Cost Function - Unsupervised Histogram Segmentation

This section introduces the cost function $\mathcal{H}$ of the *Unsupervised Histogram Segmentation* image processing model. The model is motivated and influenced by histogram clustering respectively segmentation models, which have been proposed for instance in [138, 133, 137]. This specific cost function is naturally given by the complete negative data log-likelihood $\mathcal{L}$ of the model. Essentially the model described in the following represents the prototypical cluster distributions - exactly one prototypical distribution for each single cluster ν - by discrete histograms and thus by a nonparametric cluster representation. In contrast to these nonparametric representations of cluster distributions, recently suggested models (e.g. [169] [69]) have proposed and successfully tested parametric representations of cluster distributions.

Let us assume that a set of image sites, denoted by $\Omega = \{s_i | i = 1, ..., n\}$, is given in the following. These sites s_i are organized as a regular two-dimensional site-grid Ω. We further assume that the site-grid possesses the form and size $N \times M$, $N, M \in \mathbb{N}$ with an imposed spatial neighborhood system, i.e each site is connected, up to some extent, with its neighboring sites. As we are considering the MRF class introduced in Section 2.5, only neighborhood systems of first to fifth order are regarded and used for this model. Furthermore, each single site $s_i \in \Omega$ holds a finite set of discrete observations, summarized by x_i for a specific site and summarized by $\mathbf{X} = \mathrm{x}_1, ..., \mathrm{x}_\mathrm{n}$ for all sites of the grid Ω with $n = |\Omega|$. The finite set of discrete observations is collected in the defined neighborhood of s_i and further organizing the values of x_i by means of a histogram with m bins leads to an empirical distribution $\mathbf{h}_\mathrm{i} = \{\mathrm{x_{ij}} | 1 \leq \mathrm{j} \leq \mathrm{m}\}$ with x_{ij} denoting the number of entries in particular bins j. Additionally, it is assumed that the segmentation is done in k different clusters ν, where the total cluster number k typically ranges from 2 to 8. With p_ν we denote the probability of cluster ν with respect to the actual image data and with q_ν the prototypical distribution of cluster ν. During the actual segmentation pass, each site $s_i \in \Omega$ is assigned to exactly one specific cluster ν. Exactly this cluster membership of a site s_i is mathematically formalized by a boolean assignment variable $M_{i\nu}$, $\nu = 1, ..., k$ in the following. Thus we actually set $M_{i\nu} = 1$, if site s_i is assigned to cluster ν. The assignment variables of Ω are summarized in an overall assignment matrix $\mathbf{M} \in \mathfrak{M} = \{0, 1\}^{\mathrm{n} \times \mathrm{k}}$. We further impose the requirement that $\sum_{\nu \leq k} M_{i\nu} = 1$ holds true for avoiding multiple cluster assignments per site. Obviously each $\mathbf{M}_\mathrm{i}$ possesses the boolean vector form $\mathbf{M}_\mathrm{i} = (0, ..., 0, 1, 0, ..., 0)$, if site s_i is assigned to cluster ν.

The cluster probabilities p_ν with respect to the actual image data as well as to the prototypical cluster distributions q_ν represent free parameters of the model, which have to be estimated and are summarized by

$$\mathbf{\Theta} = \{\mathrm{p}_\nu, \mathrm{q}_\nu : 1 \leq \nu \leq \mathrm{k}\} \tag{2.33}$$

in the sequel.

The previous description of the *Unsupervised Histogram Segmentation* image processing model introduced and commented on all model ingredients so that we can summarize the model with respect to *(1) observed data*, *(2) latent variables* and *(3) model parameters* in the following way:

- Observed data $\mathbf{X} = \{\mathrm{x_i} | \mathrm{i} = 1, ..., \mathrm{n}\}$
  - x_i describing a set of values, observed in the defined neighborhood of site s_i. Further structuring x_i and simplifying the following notation we define histograms $\mathbf{h_i} = \{\mathrm{x_{ij}} | 1 \leq \mathrm{j} \leq \mathrm{m}\}$ with j bins.
- Latent cluster assignment matrix $\mathbf{M} \in \mathfrak{M} = \{0, 1\}^{\mathrm{n} \times \mathrm{k}}$
  - $M_{i\nu}$, $1 \leq i \leq n$, $1 \leq \nu \leq k$, signifying the boolean assignment variable that site s_i is assigned to exactly one cluster ν; $\sum_{\nu \leq k} M_{i\nu} = 1$.
- Parameters $\mathbf{\Theta} = \{\mathrm{p}_\nu, \mathrm{q}_\nu : 1 \leq \nu \leq \mathrm{k}\}$
  - p_ν, $1 \leq \nu \leq k$, representing the probability of a specific cluster ν with respect to the data currently available,
  - $q_{\nu j}$, $1 \leq \nu \leq k$, $1 \leq \jmath \leq m$, denoting the prototypical distribution of cluster ν, represented by an empirical distribution, which is realized as a histogram with j bins.

With these model fundamentals formally introduced we can derive the appropriate cost function of the model, which is given by its corresponding *negative complete data log-likelihood* . The derivation is started with the likelihood of the assignment matrix $p(\mathbf{M}|\mathbf{\Theta})$, followed by the likelihood of the observed data $p(\mathbf{X}|\mathbf{M}, \mathbf{\Theta})$ and finally put together to gain the complete data likelihood $p(\mathbf{X}, \mathbf{M}|\mathbf{\Theta})$. The likelihood of the specific boolean assignment vector $\mathbf{M_i}$ at site $s_i \in \Omega$, without regarding the observed image data x_i in its neighborhood, is defined by

$$p(\mathbf{M_i}|\mathbf{\Theta}) = \sum_{\nu=1}^{\mathrm{k}} \mathrm{M_{i\nu} p_\nu}, \tag{2.34}$$

where the parameter $p_\nu \in \mathbf{\Theta}$ represents the probability that a site s_i belongs to cluster ν. The following overall equation - extended by the product term of the last equation - holds true as the assignment variable $\mathbf{M_i}$ is a boolean vector with the imposed constraint $\sum_{\nu \leq k} M_{i\nu} = 1$. Thus the product term sums up to one and we can equivalently write

$$p(\mathbf{M_i}|\mathbf{\Theta}) = \sum_{\nu=1}^{\mathrm{k}} \mathrm{M_{i\nu} p_\nu} = \prod_{\nu=1}^{\mathrm{k}} \mathrm{p_\nu^{M_{i\nu}}}. \tag{2.35}$$

As it is assumed that the image sites $s_i \in \Omega$ are statistical independent among one another it follows that the likelihood for the complete assignment matrix $\mathbf{M}$ of the site-grid Ω is defined by

$$p(\mathbf{M}|\mathbf{\Theta}) = \prod_{\mathrm{i}=1}^{\mathrm{n}} \left[\sum_{\nu=1}^{\mathrm{k}} \mathrm{M_{i\nu} p_\nu} \right] = \prod_{\mathrm{i}=1}^{\mathrm{n}} \prod_{\nu=1}^{\mathrm{k}} \mathrm{p_\nu^{M_{i\nu}}}. \tag{2.36}$$

Equation 2.36 finalizes the derivation of the first part of the negative complete data log-likelihood so that the likelihood of the observed data $p(\mathbf{X}|\mathbf{M},\mathbf{\Theta})$ has to be derived in the next step to complete the model's data likelihood.

A first observation reveals that the probability of a site s_i, which belongs to cluster ν, of having a value x^j out of the value range defined by bin j, is given by

$$p(x^j|\nu,\mathbf{\Theta}) = \mathrm{q}_{\nu\mathrm{j}}. \tag{2.37}$$

The likelihood of observations x_i at site s_i, which belongs to cluster ν and given parameters $\mathbf{\Theta}$ is defined by

$$p(x_i|\nu,\mathbf{\Theta}) = \prod_{\mathrm{j}=1}^{\mathrm{m}} (\mathrm{q}_{\nu\mathrm{j}})^{\mathrm{x_{ij}}}\,. \tag{2.38}$$

For all $n = |\Omega|$ sites s_i and with respect to the complete cluster assignment matrix $\mathbf{M}$ the likelihood $p(\mathbf{X}|\mathbf{M},\mathbf{\Theta})$ of the observed data is given by

$$p(\mathbf{X}|\mathbf{M},\mathbf{\Theta}) = \prod_{\mathrm{i}=1}^{\mathrm{n}} \prod_{\nu=1}^{\mathrm{k}} \left[\prod_{\mathrm{j}=1}^{\mathrm{m}} (\mathrm{q}_{\nu\mathrm{j}})^{\mathrm{x_{ij}}} \right]^{\mathrm{M_{i\nu}}}. \tag{2.39}$$

Bringing the likelihood of the assignment matrix $p(\mathbf{M}|\mathbf{\Theta})$ (2.36) and the likelihood of the observed data $p(\mathbf{X}|\mathbf{M},\mathbf{\Theta})$ (2.39) together, we receive the complete data likelihood

$$\begin{aligned} p(\mathbf{X},\mathbf{M}|\mathbf{\Theta}) &= p(\mathbf{M}|\mathbf{\Theta}) \cdot \mathrm{p}(\mathbf{X}|\mathbf{M},\mathbf{\Theta}) \\ &= \prod_{i=1}^{n} \prod_{\nu=1}^{k} \left[p_\nu \cdot p(x_i|\nu,\Theta)\right]^{M_{i\nu}} \\ &= \prod_{i=1}^{n} \prod_{\nu=1}^{k} \left(p_\nu \left[\prod_{j=1}^{m} (q_{\nu j})^{x_{ij}} \right] \right)^{M_{i\nu}}. \end{aligned} \tag{2.40}$$

Hence the complete negative data log-likelihood function $\mathcal{L}$ of the *Unsupervised Histogram Segmentation* model reads

$$\begin{aligned} \mathcal{L}(\mathbf{\Theta}|\mathbf{X},\mathbf{M}) &= -\log p(\mathbf{X},\mathbf{M}|\mathbf{\Theta}) \\ &= -\sum_i \sum_\nu M_{i\nu} \left(\log p_\nu + \sum_j x_{ij} \log q_{\nu j} \right) \\ &= -\sum_i \sum_\nu M_{i\nu} \log p_\nu - \sum_i \sum_\nu M_{i\nu} \sum_j x_{ij} \log q_{\nu j}. \end{aligned} \tag{2.41}$$

Adding the empirical entropy of x, which does not depend on $\mathbf{\Theta}$ to the expanded complete negative data log-likelihood function $\mathcal{L}$ of Eq. 2.41, we receive

$$= -\sum_i \sum_\nu M_{i\nu} \log p_\nu - \sum_i \sum_\nu M_{i\nu} \sum_j x_{ij} \log q_{\nu j} - \sum_i \sum_j x_{ij} \log x_{ij}. \tag{2.42}$$

Additional rearranging and simplifying leads to

$$= -\sum_i \sum_\nu M_{i\nu} \log p_\nu - \sum_i \sum_\nu M_{i\nu} \sum_j x_{ij} (\log q_{j\nu} - \log x_{ij}) \tag{2.43}$$

and finally to the following representation

$$= -\sum_i \sum_\nu M_{i\nu} \log p_\nu + \sum_i \sum_\nu M_{i\nu} \sum_j x_{ij} \left(\log \frac{x_{ij}}{q_{j\nu}} \right). \tag{2.44}$$

The term $\sum_j x_{ij} \left(\log \frac{x_{ij}}{q_{j\nu}} \right)$ of Equation 2.44 is distinguished and represents an information theoretical measure between two distributions. This specific measure is called *cross-entropy* or *Kullback-Leibler* divergence $D_{KL}(P||Q)$ [101] [102] and serves as divergence respectively distance measure among the distributions P and Q, which are defined w.r.t. a discrete state space; with $D_{KL}(P||Q) \geq 0$ and $D_{KL}(P||Q) = 0$ iff $P = Q$. The Kullback-Leibler divergence is neither symmetric nor does it generally respect the triangle inequality and thus D_{KL} is not a metric.

Consequently, we write Equation 2.44 in the following as

$$= -\sum_i \sum_\nu M_{i\nu} \log p_\nu + \sum_i \sum_\nu M_{i\nu} D_{KL}(x_{\cdot|i}||q_{\cdot|\nu}). \tag{2.45}$$

Thus, the cost function $\mathcal{H}$ of the *Unsupervised Histogram Segmentation* image processing model finally reads

$$\begin{aligned} \mathcal{H} &= \mathcal{L}(\Theta|X, M) \\ &= \sum_i \sum_\nu M_{i\nu} \left[-\log p_\nu + D_{KL} \left(x_{\cdot|i}||q_{\cdot|\nu} \right) \right]. \end{aligned} \tag{2.46}$$

The following two paragraphs present in detail the update formulas of the alternating calculation scheme (M-Step and E-Step), which will finally optimize the costs of the unsupervised segmentation model's cost functional 2.46. The EM algorithm recalculates the cluster assignments in the E-step, whereas in the M-step the model's free parameters $\boldsymbol{\Theta}$ are optimized. The discussion starts with the E-step of the alternating optimization scheme.

Expectation Maximization - E-step

During the E-step calculation phase, the alternating optimization scheme recalculates the cluster assignment probabilities $m_{i\nu} = \mathbb{E}[M_{i\nu}]$, $1 \leq i \leq n$, $1 \leq \nu \leq k$, which are determined in accordance with the Gibbs distribution [68]. In [137] Puzicha proved that the distinct assignment variables $m_{i\nu}$ must fulfill the $n \times k$ equations

$$m_{i\nu} = \frac{\exp(\frac{1}{T} h_{i\nu})}{\sum_{\mu \leq k} \exp(\frac{1}{T} h_{i\mu})} \quad 1 \leq i \leq n, 1 \leq \nu \leq k. \tag{2.47}$$

The particular term $h_{i\nu}$ represents the costs of assigning site i to cluster ν with T being the computational temperature. For the unsupervised histogram segmentation

model and its costs, defined by the equation 2.46 of the cost-functional, the expected costs $h_{i\nu}$ read

$$h_{i\nu} = -\log p_\nu + D_{KL}(x_{\cdot|i}||q_{\cdot|\nu})) = -\log p_\nu + \sum_j x_{ij}\left(\log \frac{x_{ij}}{q_{j\nu}}\right). \tag{2.48}$$

The concrete calculation procedure for the particular $m_{i\nu}$ and $h_{i\nu}$, defined by Equation 2.47 and 2.48, for this image processing model is straightforward to perform. The outermost sum of Equation 2.46, which runs over all sites s_i of the site grid Ω, shows that all sites are absolutely independent of each other. Consequently, a single E-step can be performed by one complete run over all sites s_i, whereas the mean fields $h_{i\nu}$ are calculated according to Equation 2.48 and the cluster assignment probabilities $m_{i\nu}$ are finally updated with respect to Equation 2.47.

The most important feature of this image processing model, namely that of all sites s_i, $i = |\Omega|$ being decoupled, is of far-reaching relevance. First of all the derivation of Eq. 2.48 becomes simpler and the arithmetical structure less complex. Secondly each site s_i can conduct its calculations ($h_{i\nu}$ and $m_{i\nu}$) independent of the other sites in the grid. What follows is the fact that this particular model can be implemented *fully* in parallel, with all sites s_i being independent processing units. The equations of the M-step, which represent the calculation rules for the model´s free parameter $\mathbf{\Theta}$, are given in the next paragraph.

Expectation Maximization - M-step

During the M-step calculation phase, the alternating optimization scheme re-estimates the model´s two free parameters $\mathbf{\Theta}$ (cf. Equation 2.33). On the one hand p_ν, which represents the cluster-specific probabilities with respect to the data actually observed is re-estimated, and on the other hand q_ν, which represents the distributions of the cluster prototypes are updated. Obviously, the re-estimation of the free parameters is performed on the basis of the cluster assignment probabilities $m_{i\nu}$, which have been determined in the E-step.

The cost functional $\mathcal{H}$ (cf. Eq. 2.46) induces the Gibbs measure with a variational characteristic (see e.g. [167]), which essentially means that the Gibbs measure has least free energy $\mathcal{F} = \mathbb{E}[\mathcal{H}] - TS$ compared with all other distributions and thus marks this probability measure. The term S of the free energy denotes the entropy of the model´s cluster assignment. Consequently, the model´s free parameters $\mathbf{\Theta}$ are optimized by setting the partial derivatives of the free energy with respect to p_ν and q_ν equal to zero.

Thus the optimization condition for the model's parameter p_ν reads

$$\frac{\partial}{\partial p_\nu}\left(\mathcal{F} + \lambda \cdot \sum_{\mu=1}^{k} p_\mu\right) = \frac{\partial}{\partial p_\nu}\left([\mathbb{E}(\mathcal{H}) - TS] + \lambda \cdot \sum_{\mu=1}^{k} p_\mu\right) = 0. \tag{2.49}$$

In Equation 2.49 the term λ represents a Lagrange correction parameter, which establishes a normalization of the cluster specific probabilities p_ν. The partial derivation simplifies as the M-step re-estimates only the model´s free parameters $\mathbf{\Theta}$ and simultaneously leaves the cluster assignment probabilities $m_{i\nu}$ unaltered. Hence, it

is sufficient to particularly regard $\mathbb{E}[\mathcal{H}]$, as the entropy S of the cluster assignment remains unchanged. Expanding Equation 2.49 leads to the following optimality condition for p_ν

$$\frac{\partial}{\partial p_\nu}\left(\left[-\sum_{i=1}^{n}\sum_{\nu=1}^{k} m_{i\nu}\log p_\nu - \sum_{i=1}^{n}\sum_{\nu=1}^{k} m_{i\nu}\sum_j x_{ij}\left(\log\frac{x_{ij}}{q_{j\nu}}\right)\right] - TS + \lambda\cdot\sum_{\mu=1}^{k} p_\mu\right) = 0. \tag{2.50}$$

When finally solving Equation 2.50, we receive the formula to determine the model parameter p_ν, which reads

$$p_\nu = \frac{1}{n}\sum_{i=1}^{n}(m_{i\nu})\ ,\ \nu = 1,...,k. \tag{2.51}$$

Following up the same procedure for the second free model parameter q_ν, the condition to optimize q_ν is given by setting the derivation of the free energy $\mathcal{F}$ with respect to q_ν equal to zero. Consequently, the optimization condition for the model's parameter q_ν is given by

$$\frac{\partial}{\partial q_\nu}\left(\mathcal{F} + \lambda\cdot\sum_{\mu=1}^{k} q_\mu\right) = \frac{\partial}{\partial q_\nu}\left([\mathbb{E}(\mathcal{H}) - TS] + \lambda\cdot\sum_{\mu=1}^{k} q_\mu\right) = 0. \tag{2.52}$$

Again, λ represents a Lagrange correction parameter, which ensures a sufficient normalization of the cluster prototype distributions q_ν. Solving Equation 2.52 for q_ν we receive the formula to determine the model parameter q_ν, which reads

$$q_\nu = \frac{\sum_{i=1}^{n} x_i m_{i\nu}}{\sum_{i=1}^{n} m_{i\nu}}\ ,\ \nu = 1,...,k. \tag{2.53}$$

In summary the first phase of the alternating optimization scheme, the E-step, updates the cluster assignment probabilities $m_{i\nu}$ according to the formulas, which are given by Eq. 2.47 and Eq. 2.48. In the second phase, the M-step, all free parameters $\mathbf{\Theta}$ of the model are updated. The update formulas are given by Eq. 2.51 and Eq. 2.53. Thus the complete set of update Equations for the *unsupervised histogram segmentation* model are derived to systematically optimize the model´s cost-functional $\mathcal{H}$, given by Eq. 2.46.

2.7 Analog Technologies versus Digital Technologies

This section discusses in depth the advantages and limitations of possible semiconductor technologies for massively parallel processing devices, which are based on the processing principles of Markov Random Fields. This discussion establishes the basis and justification regarding the selection of a semiconductor implementation-technology made in this thesis.

Today there are principally only a few basic implementation-technology variants available, which systematically support the physical realization of Turing-universal processing devices. The industrial focus lies on semiconductor technologies with various integration capabilities and essentially comprises purely digital, purely analog

and mixed-mode technology-libraries for the implementation of processing devices. Optical and opto-electrical implementation technologies represent, in contrast to the technologies mentioned before, only niche technologies with extremely specific application domains. Furthermore, quantum computing and molecular computing approaches and their carrier-technologies are research directions with no current industrial relevance, but it is conceivable that these approaches will mature to mainstream technologies with a significant impact on future computing devices and their integration densities. Consequently, either pure digital, pure analog or mixed-mode semiconductor technologies are the logical choice of most processing device implementations. This is equally true for those kinds of processing devices, which are based on stochastic data processing models, and processing principles characterized by distributed calculation units arranged on a regular grid with spatially limited interactions, as dealt with in this thesis.

In the neural network community these processing approaches and their corresponding processing devices are in summary denoted by the term *Cellular Neural Networks (CNNs)* since the seminal publications of L.O. Chua and I. Yang [35] [34]. Generally neural networks have their theoretical roots and inspirational sources in biological- as well as neuro-science. These roots are the reason why artificial neural networks pursued to functionally imitate respectively copy the astonishing capabilities of biological nervous-compounds by means of idealized and often simplified models and their corresponding technical implementations. As biological systems use analog electrical signals during the fast signal transmission path along the nervous line [91], analog semiconductor technologies were and are still used to imitate these biological signal shapes and the integration of these signals in technical realizations of neural networks. This is especially true for the *Cellular Neural Network* community as analog realizations have a long-standing tradition in this research field [74] - [79]. Another argument for analog implementations is the possibility of spatially combining the sensing structures and the processing structures on one chip-die with high integration densities. Perhaps the most prominent and influential Cellular Neural Network example, presented in different variants and numerously realized in analog technologies (see e.g. [97], [113], [112]), is that of a simplified mammalian retina model. This artificial retina and all its variants are often used for early vision tasks in the neural network community.

In contrast to the neural network community, the image understanding community denotes the above mentioned processing approaches with the term Markov Random Field (MRF) or sometimes Gibbs Random Field. Whereas an image analysis approach, which uses Markov Random Field, is mostly more generally referred to as Bayesian image analysis. One of the most influential papers in this area is that of D. Geman and S. Geman [59] and earlier the contributions of Besage [10] [11] [12]. The Bayesian image analysis approach with Markov Random Fields as processing structures has its theoretical roots in mathematics and statistics, but also substantially borrows ideas and insights from statistical physics. Consequently, these signal and image-processing models, which are derived within a Bayesian framework and formulated on MRFs, solely require a standard digital number representation, well-defined operators working on these numbers and a serial processing machine. Obviously this can be achieved by means of digital number representations either as fix-point or as float-point type.

In summary, the previous discussion shows that the Cellular Neural Network community as well as the Bayesian image analysis community investigate, develop and popularize signal- and image-processing models, which share the same fundamental processing principles. In the end both communities are based on the same mathematical basics. However, both communities have found their fundamentals and derivations in different scientific disciplines. This mainly leads to the situation that the CNN community prefers analog semiconductor technologies for the implementation of CNN models in order to stress the biological aspect. In contrast to the CNN community, the Bayesian image analysis community prefers flexible and programmable digital processing platforms, in order to investigate the performance of proposed image processing models. The Bayesian image analysis community mainly uses commercial off-the-shelf computers, as it is the modeling aspect on which the community's focus is set and not the compact and real-time capable realizations.

Consequently, the discussion raises the central question of which implementation technology is more advantageous and forward-looking with regard to the realization of CNN/MRF based processing devices. Many different aspects have to be taken into account to finally come to a qualified conclusion regarding a suitable MRF/CNN implementation technology, which is more or less uncommitted to any scientific discipline. We advocate the position to essentially base the technology selection-process of digital, analog or mixed-mode technologies on the following aspects:

- Semiconductor technology trends,
- Design methodology trends,
- Advantageous CNN/MRF modeling and representation technology and
- Far-reaching application scenarios and their requirements.

Semiconductor Technology Trends

In the last decade a continuous and impressive progress in the field of semiconductor technology has taken place [83] [84]. The feature sizes of the individual transistors are steadily shrinking down to the nanometer scale. Consequently, with each single technology shrinking step the integration density tremendously increases. Furthermore, new surface polishing approaches [25] make it possible to stack up to 12 global wiring levels on top of the transistors, which further significantly supports the overall integration density. All these technology improvements are in principle equally available for analog as well as digital technologies. But the precise representation respectively generation of well-defined analog shaped signals reaches its limits as electron mobility decreases with shrinking feature sizes and as absolute voltage values decrease. Additionally, semiconductor-fabrication carriers are almost solely focused on digital design technologies with respect to the latest transistor feature sizes [93] and also do not offer the corresponding analog design technologies. Consequently, only digital design technologies fully profit from todays semiconductor process improvements. This situation, which will not change in the near future [17] [83], explicitly favors pure digital design approaches of CNN/MRF based processing devices over analog or mixed-mode implementations.

In addition to this, the argument of the CNN community that only analog design technologies can offer the capabilities to realize adequately large CNN/MRF site grids, which for a long time was correct and striking as analog CNN/MRF sites are smaller, is no longer valid because of todays integration densities of digital design technologies. Actually modern digital design technologies can theoretically outperform analog design technologies with respect to the overall size of the CNN/MRF site-grid. Consequently, compared to analog technologies, digital semiconductor technologies offer a far more advantageous technology basis for the implementation of massively parallel CNN/MRF processing devices, than analog technologies.

Design Methodology Trends

The research field of design methodologies for integrated circuits, independent of the technology variant - digital, analog or mixed-mode -, has developed almost as impressively as the semiconductor technologies itself. This fact is not astonishing as the continuously and tremendously increasing number of available and theoretically usable transistors per integrated circuit generates the demand to adapt, improve or newly develop electronic design automation methodologies and tools. Without the technical progress and the innovations of the electronic design automation (EDA) community it would not have been possible to design, to verify and to manufacture the complex devices prevalent today.

However, most of the research work, innovation and tool-development [98] has been conducted for the design-support of purely digital devices. Two internationally standardized hardware description languages (HDLs), VHDL and Verilog, are available and frequently used to design purely digital chips. Highly specialized and well-approved synthesis engines translate the corresponding hardware description into boolean equations, which can finally be mapped onto logical gates and transistors. Furthermore, modern synthesis engines optimize the boolean representation of the described chip and thus improve the chip-area utilization and the maximum clock-frequency the chip can operate in. Today the chip-layout generation with the main sub-tasks of placing & routing is a well-approved and tool-supported procedure too. Additionally, the simulation of digital circuits is systematically supported by different simulation-tools and can be conducted time-efficiently both for moderately sized and complex circuits; however, for extremely large and complex digital systems it quickly reaches its limit with respect to overall simulation-time and memory-usage.

In the analog domain the situation looks completely different. Foremost, hardware description languages, as VHDL-AMS and Verilog-A, have recently been internationally standardized to systematically represent analog circuits. But until now any hardware description of analog circuits using these languages is only suitable for simulation purposes. The generation of the corresponding transistor representation is currently not possible. Consequently, the design of analog circuits is still low-level handwork [26] for each single analog design technology with its specific parameters. The simulation of analog circuits is extremely time-consuming and memory-usage intensive as the analog circuit is represented by a non-linear, ordinary differential equation system, which must be solved to simulate the analog behavior. Obviously the simulation of large and complex analog circuits is affected and limited by this kind of circuit representation as a differential equation system. Thus com-

pared to analog design technologies, digital design technologies are advantageous with respect to design methodologies, available tools - including simulators - and industrial-approved design flows.

Suitable Representation Technology

The question of which technology is more suitable to model and technically represent CNNs/MRFs can be answered straightforwardly. At the first glance, if we do not consider any technological issues and limitations, each technology variant - purely digital, purely analog or mixed-mode, seems equally suitable to model CNN/MRF processing devices. No technology variant implies fundamental theoretical limitations with respect to signal representation and calculation capabilities, to become less powerful compared to other technology variants. Secondly, if we consider technological aspects, then purely digital technology variants become more advantageous for implementing CNN/MRF processing units than purely analog or mixed-mode technologies. This is substantiated by the fact that a precise representation of signal values with different co-domains and the arbitrary calculation with these signal representations is much easier and more stable to implement within purely digital technologies. Consequently, purely digital design technologies are definitely equivalent to analog or mixed-mode technologies with respect to their signal representation and calculation capabilities, and in addition offer several practical implementation advantages.

Applications and their Requirements

Potential and far-reaching application scenarios for CNN/MRF processing devices are affected by the chosen implementation technology, which in the extreme case leads to a complete failure when realizing specific CNNs/MRFs in that technology. For several application scenarios [42] it is essential to regard and process signal sources like *radar, infrared, sonar* and *laser* within the CNN/MRF paradigm. Therefore it becomes indispensable to process these signal sources within technical implementations of CNN/MRF devices.

All these sensors principally possess a digital interface and thus a digital representation of their output-signals. This fact favors digital CNN/MRF realizations to establish a seamless digital signal-processing flow. Principally, the digital sensor signals can be converted to appropriate analog signals and then passed over to an analog CNN/MRF for processing. But this conversion procedure, which has to guarantee the overall signal compatibility of the signals of both the digital-analog conversion process and the analog CNN/MRF signal representation, is technically rather complex and contains several analog-specific design issues. If we assume for the purpose of this discussion that it is possible to directly receive analog sensor signals, the signal-level adaptation process, the signal-integrity process and the overall electrical signal compatibility all need to be completely executed within the analog signal-domain before the analog sensor signals can pass over to an analog CNN/MRF device. Again, all these tasks are technologically involved and define additional analog design and system-integration challenges, which can only be mollified by an analog-digital-analog conversion process. Undoubtedly this analog-digital-analog

signal conversion effort can by no means be justified.

The situation previously described becomes even more complicated, if a data-fusion approach is addressed with the help of the CNN/MRF paradigm and realized with the corresponding analog CNN/MRF processing device. Generally in these data-fusion settings several analog signals from different sources have to be modified and adapted to be compatible with each other and with the analog CNN/MRF device. Consequently, certain generic application requirements, determined by the sensor-sources, raise serious analog design issues, which finally lead to a purely digital signal-processing flow and as a result purely digital CNN/MRF processing devices.

A completely different class of applications is defined by various aviation and space-flight scenarios, which could profit from the CNN/MRF signal- and image processing paradigm and its massively parallel processing devices. The technological key issue of this specific application-scenario class is the tolerance of the electronic components against harsh environments and radiation. Essentially - besides other parameters - temperature becomes an important parameter in harsh environments. All three technology variants - digital, analog and mixed-mode - are equally affected by the environment temperature and thus none of these implementation technologies shows any advantage with respect to this parameter.

Likewise, heavy ions, neutrons, and protons scatter the atoms in a semiconductor lattice [72] [120], independent of the technology variant. This introduces on the one hand short SEU (Single Event Upset) effects and on the other hand long lasting noise and error sources in semiconductor structures [110] [121], which finally alter the functionality of the devices. With regard to the topic radiation-tolerance also none of the technology variants owns any advantage compared to the others.

We conclude this section with the statement that modern ultra-deep sub-micron digital design technologies are more advantageous compared with purely analog and mixed-mode design technologies to implement signal- and image processing devices, which are based on the principles of CNNs/MRFs. This equally includes all high-density, digital and reconfigurable technologies available, which represent an extremely cost effective developing, testing and prototyping platform.

2.8 Summary

In this chapter we have introduced the theoretical fundamentals, which are required for the upcoming discussions and conclusions of this thesis. The fundamentals are divided up into two parts: The first part of the fundamentals covers Section 2.1 - 2.5 and presents more fundamental material on Bayesian image analysis, Markov Random Fields and different optimization schemes. The second part of describing the fundamentals, which covers Section 2.6 and 2.7, presents two concrete and practically relevant image processing models, used throughout the thesis as test models. Additionally, an in-depth discussion on suitable semiconductor implementation technologies and the conclusion drawn justifies the purely digital implementation approach advocated in this thesis.

Firstly, in Section 2.1 we introduced the central motivation and ideas of the Bayesian approach to image analysis and understanding. Starting with a general and flexible image-description as an array of components, representing different observed

or unobserved image-attributes, we have derived the principle of defining image processing problems as attribute-labeling problems. In the discussion exactly this attribute-labeling principle has been connected with the common statistical inference idea to direct the argumentation toward probability distributions.

Section 2.2 presented in greater depth the essential definitions of and theoretical insights into probability measures (random fields). Furthermore the representation of random fields in the Gibbsian form with respect to a particular potential has been defined. Finally, the equivalence theorem, also called Hammersley-Clifford theorem has been formulated. It formally states the fact that a random field can be represented by a Gibbs field with a corresponding potential, and reversely each Markov Random Field can be represented by a neighboring Gibbs field with the same neighborhood system.

Section 2.3 has presented and discussed optimization schemes in order to systematically investigate the performance of probabilistic image processing models formulated in the Bayesian framework on Markovian site grids. We have discussed the ideas and concepts of the optimization methods, which have their origin in statistical physics and partly go back to the maximum entropy inference principle of Jaynes. This led us to the family of Simulated Annealing optimization methods. The general Simulated Annealing (SA) algorithm (cf. Algorithm 2.1) has been presented as well as different acceptance rules. The described Gibbs sampler in particular means an improvement of the two-phase sampling scheme, used before the Gibbs sampler was introduced by Geman et. al. Additionally, a special variant of the general SA method, the temperature frozen ICM algorithm, which is often used in Bayesian image analysis community has been presented.

In Section 2.4 we have discussed parallel processing strategies for Markov Random Fields keeping the constraint of designing massively parallel VLSI hardware architectures in mind. The concept of independent sets (site-sets) has made it possible to formally establish conditions under which the convergence of the optimization methods to the optimizers of the cost-functional can be guaranteed. These conditions of parallel processing schemes have defined the fundamentals for the specification and definition of parallel processing hardware architectures for Markov Random Fields.

Since the general class of Markov Random Fields is large and diverse we have defined a smaller MRF sub-class in Section 2.5. The members of this specific MRF sub-class are structurally advantageous with regard to massively parallel VLSI implementations and are additionally comprehensive and flexible enough to cover low-level image processing problems.

Section 2.6 has presented two image processing models in detail. The first model at the same time removes noise and preserves intensity changes, which represent robust characteristics of object boundaries. The second model represents a nonparametric unsupervised segmentation; nonparametric as the cluster prototypes are modeled by an empirical distribution, namely by local histograms. For both models we have derived the corresponding cost function and the update equations for the model's free parameters. Especially for the unsupervised segmentation model, the Deterministic Annealing (DA) optimization approach has been utilized to determine the cluster assignments and the model's free parameters in a sequence of alternating optimization steps. In this section also the corresponding DA update equations for

the cluster assignment of the model as well as for the model's free parameters, have been derived.

In Section 2.7 we have discussed the advantage and disadvantage of three matured and industrial-relevant semiconductor implementation technologies - purely digital, purely analog and mixed-mode - for massively parallel Markov Random Field based processing devices. The technology comparison was performed on the basis of four central aspects: (1) common semiconductor technology trends (2) design methodologies trends (3) advantageous CNN/MRF modeling and representation technology and (4) potential and far-reaching application scenarios. The discussion has revealed that a purely digital semiconductor implementation technology is outclassing the analog and mixed-mode technologies with respect to aspect (1), (2) and (3). Regarding aspect (4) none of the technology variants has a clear advantage compared to the others. In summary, the discussion of Section 2.7 has shown that purely digital semiconductor implementation technologies are far more suitable for massively parallel VLSI hardware implementations of MRF based image processing devices than analog or mixed-mode semiconductor implementation technologies.

Application Scenarios

As already announced in the introductory part of Section 2.6 we will now discuss far-reaching industrial utilization scenarios for the two presented statistical image processing models in order to underpin their practical relevance and importance. Recapitulating the first model at the same time removes noise and preserves intensity changes, whereas the second model realizes an unsupervised segmentation process. The novel application scenarios for both image processing models herein discussed are resided in the domains of civilian aviation and automotive industries.

At first we discuss projected applications [42] for the image processing model, which simultaneously removes noise and preserves respectively enhances intensity changes. The first application comes from civilian aviation and the field of flight-assistance systems. Passive flight-assistance systems for helicopters, which merely display enriched image information to the pilot or co-pilot, are crucial for the helicopter safety during the operational flight-phase. Image data, which has undergone a preprocessing-step of noise removing and preserving or even enhancing of intensity changes, as represented by our first model, is a useful source of information for the pilot or co-pilot to grasp the essential ingredients of the current scene. Preserving respectively enhancing intensity changes and smoothing the regions between intensity changes through the noise removing process accentuates the different objects in the scene by their enhanced boundaries. Such objects are, among other uncritical objects, for instance obstacles like high-power transmission lines and their corresponding power poles. If the pilot or co-pilot perceives this obstacle information in the processed images earlier compared to his normal perception, he gets valuable seconds to alter the flight-path of the helicopter. Furthermore, this application scenario and its corresponding passive pilot or co-pilot assistance system is equally realizable for various sensor signals including standard video, infrared and laser radar - to name just a few with the presented image processing model, of course including small model-modifications for each sensor source. Consequently, passive

flight-assistance systems based on the image processing model of Section 2.6.1 could significantly support the overall flight-safety and all weather flight-capabilities of modern helicopters.

The second application scenario comes from automotive industries and covers the field of passive driver-assistance systems for motor cars and trucks, i.e. systems, which support the automotive driver by means of visually displayed information without directly interfering with the driving process by automatically steering or retarding. These driver assistance systems are conceptualized to support the driver during night and dawn rides as well as in bad weather rides; mainly visually enhancing obstacles on the road in time. For this purpose it is planned to display information together with complete images on the windscreen shifted with respect to the drivers field of view. Because the application field is focused on night, dawn and bad weather situations, infrared and laser radar sensors are the logical choice. As with helicopter assistance systems, automotive assistance systems are also intended to display processed image data to the driver, in order to raise the driver's attention and allow him to perceive possible obstacles earlier than with normal perception. Removing noise with simultaneously enhancing respectively preserving intensity changes and thus distinguishing object boundaries, as done by our image processing model, is a conceivable image processing step for these projected driver assistance systems. This finalizes the discussion on application scenarios for the first image processing model and underpins the practical relevance of this specific model.

The following applications are discussed with respect to the second image processing model introduced in Section 2.6.2, which segments image data in an unsupervised manner, i.e. the segmentation process is completely data driven. In contrast to the applications of the first image model, where the processed data will be directly displayed to the users of the assistance systems, the segmentation model has been projected to be part of an image processing chain and thus the model will pass its segmentation results over to further analysis stages of the image processing chain. The image processing chain is planned to generate results, where obstacles are marked and eventually tracked. Precisely in the field of civilian aviation the following applications are discussed and planned. Firstly the application of detecting in advance large and dangerous objects will be looked at, which come up on the runway while the aircraft is landing, in order to avoid serious accidents. These systems are interesting for airlines, which already operate at or want to expand to countries and airports with lower runway security standards. Again, this system will also be passive in the sense that it will not be connected to the central aircraft flight-control system; rather an appropriate information will be provided to the pilot or co-pilot. Secondly the application of detecting obstacles including other aircrafts at the gangway during the taxiing phase of an aircraft will be discussed. This application scenario is not mainly motivated, as could be thought of first, to prevent fatal aircraft accidents with personal injuries, but rather to prevent damages at the aircraft structure. Even small damages of the aircraft shell will ground the plane for a long time, which will in turn be very costly for the airline. Consequently, any system, which could significantly reduce the rate of these accidents and the resulting damages, would be most welcome and thus supported by airlines and insurances.

The third application is concerned with the detection and tracking of powerpoles and their corresponding high-power transmission lines for helicopters. Thus,

the projected system is an advanced system compared to the previously described system and can be enriched with information about the time-to-contact of the helicopter. In the field of automotive applications an image processing chain including the unsupervised segmentation model is discussed, which will detect pedestrians and cyclists to reduce the accidents between cars, trucks and these road users. Additionally, a system is discussed, which monitors the blind spot in order to detect cars and motorcyclists when changing to the passing lane.

In summary, the previous discussion reveals that the two exemplary image processing models, described in Sections 2.6.1 and 2.6.2, are of practical and far-reaching industrial relevance. Both models are critical components of an image processing chain for projected systems in the fields of civilian aviation and automotive industries. Consequently, from an industrial point of view, processing devices with physically compact packages - ideally realized as a single chip solution - and real-time processing capabilities are highly recommendable.

2.9 Bibliographical Comments

Next to contemporary conference and journal papers, there are some excellent monographs on the themes of Bayesian Image Analysis, Gibbs Fields, Markov Chain Monte Carlo methods and statistical inference for random fields, including the Gibbs sampler, the Metropolis-Hastings class of algorithms, model's free-parameter estimation and links to already known principles of statistical physics.

The monograph of Gerhard Winkler [167] is definitely the most exhaustive and profound presentation currently available. Starting with an intuitive introduction to Bayesian image analysis, the author continues with finite random fields, their neighborhood structure and their representation in the Gibbsian form as the fundamental mathematical objects. The Gibbs sampler is thereafter discussed and analyzed with respect to fully parallel updating schemes. One complete part discusses the topic of appointing the free parameters or *hyper-parameters* of the model - an essential topic of Markovian modeling -, which is often neglected or even ignored in textbooks. Maximum likelihood estimation is a general principle and approach to estimate these free parameters and becomes discussed in detail in this part.

The monograph of Bremaud [23] differs slightly from that of Winkler [167]. Bremaud devoted his monograph mainly to the studies of homogeneous Markov Chains (HMCs) with a countable state space, and thus not solely to probabilistic image analysis. Independent of this fact, the in-depth discussion of Markov chains on the real-line is later generalized to arbitrary discrete index sets by means of Gibbs fields, which establishes the link to image processing in this textbook of Bremaud. The formal presentation of Gibbs Fields and the Gibbs-Markov equivalence is profound and well structured. In the following, the discussion concentrates itself on Markov Chain Monte Carlo simulation algorithms with an introduction to simulated annealing optimization.

Chalmond's work [33] is dedicated to modeling issues and is divided into two parts, representing two model types. The first type are Bayesian models issued from statistics, and the second type are models derived from physics and mechanics (splines). The second part of this textbook considers inverse image processing

problems as a Markovian cost optimization task. Additionally, this monograph provides an in-depth discussion of the model's parameter estimation problem and thus differs profoundly from other textbooks. Several practical low-level image processing problems, like de-noising, de-blurring and detecting lines respectively curves are discussed in exhaustive detail and in different variants.

Finally the monograph of Stan Z. Li [107] is mainly devoted to applications of Markov Random Fields in the low-level domain and to object matching and recognition in the high-level domain. The author uses different examples in order to explain how one can systematically convert a specific image processing problem with modeling uncertainties and constraints into a pure optimization problem within the Bayesian framework on Markov Random Fields. But the most remarkable characteristic of this monograph is the detailed discussion of the model's parameter estimation problem.

The number of contemporary literature on probabilistic image analysis in a Bayesian framework and Markov Chain Monte Carlo methods is tremendously large and still growing. Hence it is impossible to provide a nearly complete list of publications. The most influential and basic publications on these topics are the papers of Geman et al. [59] and Besag et al. [10] [11] [12]. Furthermore [55] and [56] present vital ideas on texture- and object-boundary location in a Bayesian framework with MRFs. The restoration and recovery of discontinuities by means of Markovian site-grids as statistically robust feature of object boundaries is described in [58] [13]. A more general presentation and synopsis of Markov Random Fields and Bayesian image analysis is available in [14] [57]. Shannon derived the entropy principle axiomatically and established this concept in the field of communication and information theory as a measure of uncertainty [147]. Maximum entropy methods in statistical inference were first proposed by Jaynes [89] [88]. Additionally, P. Smolensky [149] [150] [148] [136] further generalized the cost-functional and optimization formalism, which finally led to the connectionist-based formalism of Harmonic Grammar.

The book of Salamon et al. [144] summarizes the theoretical foundations and ideas of SA methods - until then only published in a scattered way - in one single representation. An exhaustive introduction and overview on Simulated Annealing concepts can be found in [165]. In addition to the book of Laarhoven et al. [165], a comprehensive bibliography with respect to the Simulated Annealing theme is available in [36]. The temperature frozen SA variant Iterated Conditional Modes (ICM) is ascribed to the work of Besag [13] and frequently used in the context of Markov Random Fields, although this method is often trapped in local minima due to its purely greedy and local search strategy. Mean Field methods, which have been adopted from statistical physics are summarized for instance in the edition of Opper et al. [129].

Several basic parallel processing strategies for Simulated Annealing are summarized in Azencott et al. [5] and Salamon et al. [144]; where Salamon et al. is concerned with different application domains of SA and is not specifically focused on image analysis problems. The two book chapters [4] and [6] are in this context of particular interest. The SA variant Gibbs sampler discussed in detail and formally analyzed with respect to massively parallel processing schemes in [167]. Exactly this discussion on independent site sets is essential for parallel processing strategies on regular two-dimensional site grids, which are prevalent in low-level image analysis

settings. Furthermore, VLSI hardware architectures can ideally be adjusted and designed to match these parallel processing strategies.

The basic probabilistic model formulated in a Bayesian imaging framework, which simultaneously removes noise and preserves edges respectively local intensity changes is due to [59]. This model is recapitulated and revised in [56] and covers different terms for structure-regularities of boundaries, but the core idea presented in [59] remains unchanged. The authors of [16] presented a similar model with the essential difference compared to [59] of not using the Bayesian imaging approach but the special setting with the GNC algorithm. This restricts the flexibility of their model with respect to noise modeling and incorporating various statistically robust boundary features. In addition, in [54] and [58] edge respectively boundary features are presented and repeated, which have already been successfully tested in probabilistic models formulated in a Bayesian imaging approach.

On an abstract level of contemplation the image segmentation task can be regarded as a pixel-clustering problem, where particular pixels are classified and assigned to a specific cluster based on some predefined features. General data clustering methods, not exclusively dedicated to image processing, are presented in [85]. Image segmentation variants, based on the clustering idea, are for instance the central clustering method (see [90]), the non-parametric cluster-prototype representation models, to which histogram clustering belongs [137] [138], methods based on similarity-matrices [115] [71] and hierarchical variants [105]. An excellent segmentation model with a parametric cluster-prototype representation is presented in [69] [169] [68].

Analog realizations and image processing models based on the CNN paradigm are mainly presented in [74] - [79] [45] during the last decade. The initiation of this community and the seminal papers are due to Chua [35] [34]. The proposed models are diverse and rich, ranging from simple binary models for contour and letter enhancement, which are advantageous for analog VLSI designs with respect to their structure, to imitations of the mammalian retina [43].

Chapter 3

System-Architecture Template

In this chapter we present a novel system-architecture concept for statistical image processing models, which are formulated on a regular Markovian site-grid with spatially limited neighborhood support. The proposed concept serves as an architecture-template for the defined class of MRF models (cf. Section 2.5) and their purely digital, very large-scale integrated (VLSI) implementations. Hence the architecture-template is conceptualized to match with and exclusively adjusted to the two characteristics of (1) *massively parallel processing* and (2) *semiconductor technology independent System-on-Chip (SoC) integration.* The first attribute of the novel system-architecture template is substantiated by the fact that even if we rely on the impressive semiconductor progress to continue in the near future, all single serial processing approaches fall short of providing real-time processing capabilities for the class of statistical image processing models under consideration. Due to this unchangeable constraint and deficit we propose this massively parallel processing requirement in order to fulfill the real-time processing constraint posed by many application domains. The second feature is stated, as the majority of applications require physically compact image processing systems to be of practical and industrial relevance. Furthermore the second features is stated, in order to decouple the architecture characteristics from constraints of a specific implementation technology.

It is common practice to change or adapt mathematical models, solution strategies and algorithms for the sake of practicability, to allow them to match with specific structures of a processing architecture or even with a fully predefined architecture. This adaptation procedure, not exclusively limited to the image processing domain, is frequently influenced by experience values and *ad hoc* assumptions of the corresponding developer. In contrast to this prevalent approach, we advocate a systematic approach with different levels of abstraction in this chapter, where the definite architecture is finally solely derived and defined within the scope of Markov Random Field theory. Our approach thus systematically defines the system-architecture template by exhaustively taking advantage of the available properties provided by Markov Random Field theory, strictly respecting them and at the same time realizing the two before mentioned characteristics.

The proposed procedure for the derivation and definition of the system-architecture template comprises the following three main steps, which simultaneously represent the afore mentioned levels of abstraction (see Figure 3.1):

- In the first step we identify *universal constituents*. These universal constituents are exclusively derived from the theoretical fundamentals of Markov Random Field theory and thus represent an excellent architecturally uncommitted originator for the following architecture-derivation process. Additionally, as an inherent property, these universal constituents are - by derivation and definition - unrestricted with regard to any VLSI relevant constraints and concrete implementation details. The universal constituents represent the lowest level of abstraction (see Figure 3.1).

- The second step systematically details the universal constituents to become architectural building blocks, where the emphasis is exclusively set - as already said - firstly on parallel processing and secondly on semiconductor technology independent large scale integration. The process of detailing the particular universal constituents is formalized by mappings between the universal constituents and the architectural building blocks. This set of optimized architectural building blocks represents the average level of abstraction (see Figure 3.1).

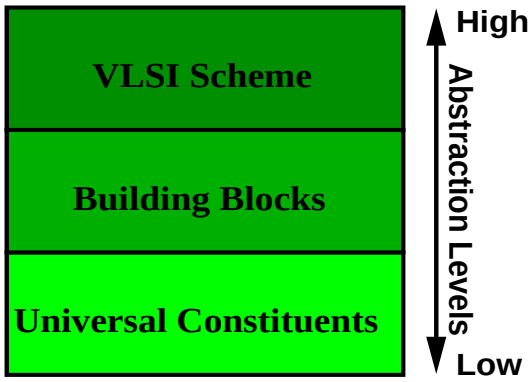


Figure 3.1: Abstraction levels. Derivation and refinement of VLSI appropriate building blocks.

- The systematic and controlled handling of VLSI specific tasks, like system complexity, technology synthesis, placing and global routing, clock-tree synthesis and memory-block integration is another fundamental issue. We have developed a method to cope with these problems, which in step three categorizes the architectural building blocks along a VLSI-appropriate scheme. This classification scheme differentiates between *structure and topology* relevant building blocks, *processing functionality* relevant building blocks and *control functionality* relevant building blocks. Exactly this well-defined classification scheme ideally addresses different large-scale integration questions, which in this systematic way is currently not provided for any system-architecture at all and for a massively parallel architecture-template of statistical image processing models less than ever before. This last step defines the highest level of abstraction (see Figure 3.1).

In order to complete the novel massively parallel architecture-template, a cycle

scheme for the ordering of operations and data transfers during the complete operational phase is introduced. With this massively parallel architecture-template and the cycle scheme at hand, we also introduce a processing strategy, which handles image sizes larger than the size of the Markov Random Field itself.

In summary, this chapter discusses the following topics: Section 3.1 introduces the universal constituents of MRF based signal- and image processing models. The architectural building blocks, organized along the VLSI-appropriate scheme, are introduced in the sequencing Section 3.2. In Section 3.4 a flexible and extensible cycle scheme template for the data-flow and operator ordering of the processing architecture is proposed. The relation diagram, which illustrates the context of the different chapters and its contents is presented in Section 3.5. Section 3.6 defines the different cases and at the same time systematically describes MRF device settings to process image sizes larger than the realized MRF field itself. Section 3.7 finally comments on the concrete massively parallel processing architectures of the two exemplary models, composed out of the defined architectural building blocks.

3.1 Universal Constituents

The derivation process of universal constituents merely based on the fundamentals of Markov Random Field theory is of vital significance for any further development and refinement of architectural building blocks. As the universal constituents are solely derived from theory, they are unrestricted regarding application and implementation assumptions or any other constraints and consequently form a set of basic elements. These universal constituents of Markov Random Fields thus represent an excellent, well-founded and architecturally uncommitted common originator for specific architectural developments and refinements. Independently of the scientific discipline one belongs to and thus which scientific perspective one takes, the one of a mathematician or statistician, of a computer scientist or of an electrical engineer, the identification outcome of the universal constituents of Markov Random Field based statistical image processing models is always comparable. This fact is by no means astonishing as the universal constituents are exclusively determined by the corresponding unique mathematical findings of Markov Random Field theory.

Thus it is per derivation and definition of these universal constituents excluded that our own constraints imposed, which are used during the definition and development process of the architectural building blocks in Section 3.2, interfere with already encoded constraints and architectural suggestions of other applications or implementations. To stress this point again, our approach ensures that the derived architectural building blocks in Section 3.2 are exclusively constrained by our own limiting conditions. The architectural building blocks will be developed and defined with respect to the capability of massively parallel processing and the ability to realize highly integrated and complex VLSI systems in an arbitrary purely digital semiconductor technology.

But first of all the architecturally uncommitted universal constituents of Markov Random Fields are derived and defined in this section. In order to enhance the readability and simplify the derivation of the universal constituents we shortly recapitulate the central Equation 2.18 from Section 2.2 enhanced by the $OPTIMIZE(\cdot)$

operator, here:

$$OPTIMIZE\left(\mathcal{H}_{Model}^{Image}\right) = OPTIMIZE\left(\sum_{i=1}^{n}\sum_{C_i^{\mathcal{N}}} U_{C_i^{\mathcal{N}}}\right). \tag{3.1}$$

Exactly this general equation of the energy functional of statistical image processing models formulated on Markov Random Field grids and its respective optimization strategy is the exclusive source of information for our derivation and definition of universal constituents. We will identify four different universal constituents in the following text passages. Three of the constituents are explicitly encoded and represented by components of Equation 3.1. Only one universal constituent is mathematically further encapsulated and has to be extracted and concluded from the original representation. But obviously for this last universal constituent the architecturally uncommitted principle is also fulfilled.

The first universal constituent of Markov Random Fields is explicitly expressed by the outermost sum with the summation index $i = 1, ..., |\Omega|$ of Equation 3.1. This sum runs over all sites s_i of a Markov Random Field site-grid Ω. Consequently, we identify the set of sites $\{s_i : 1 \leq i \leq |\Omega|\}$ as our first universal constituent. It should be pointed out that definitely each particular site s_i of the Markov Random Field site-grid Ω is considered. In the context of universal constituents, the site set $\{s_i : 1 \leq i \leq |\Omega|\}$ is in the sequel denoted by

$$\mathbb{S}_{sites} = \{s_i : 1 \leq i \leq |\Omega|\}. \tag{3.2}$$

The second universal constituent is also explicitly represented by a component of Equation 3.1. In this case the universal constituent is given by the family of functions at site s_i, called neighbor potentials $\{U_{C_i^{\mathcal{N}}} : C = \text{Cliques of } s_i, 1 \leq i \leq |\Omega|\}$; cf. Definition 2.3, 2.1. We notice here that the neighbor potentials are normally assumed to be identical for all sites $s_i \in \Omega$ and thus $\{U_{C^{\mathcal{N}}} : C = \text{Cliques } \forall s_i, 1 \leq i \leq |\Omega|\}$ is true. In the context of universal constituents we consider the general case, and thus the neighbor potential set is denoted by

$$\mathbb{P}_{pot} = \{U_{C_i^{\mathcal{N}}} : C = \text{Cliques of } s_i, 1 \leq i \leq |\Omega|\} \tag{3.3}$$

in the sequel.

The third universal constituent is induced by the neighbor potentials. Such a neighbor potential itself is in its part determined by the cliques C. The cliques C on the other hand are defined by an overall neighborhood system $\mathcal{N}$ on the site-grid Ω, which defines the set of neighbors at every site s_i. We write $< s_i, t >$, if t is a neighbor of site s_i defined by the neighborhood system $\mathcal{N}$ on the site grid Ω. Thus we identify another universal constituent as the neighborhood system itself and denote it by

$$\mathbb{N}_{sites} = \{< s_i, t >: 1 \leq i \leq |\Omega|,\ \forall t \in \mathcal{N}_i\}. \tag{3.4}$$

The three universal constituents derived earlier on, all formally justified by Equation 3.1, are summarized in the sequel by

$$\mathbb{C}_{universal} = \{\mathbb{S}_{sites} \cup \mathbb{P}_{pot} \cup \mathbb{N}_{sites}\}. \tag{3.5}$$

In a last step we identify the optimization method

$$OPTIMIZE(.)_{s_i} \tag{3.6}$$

as the final universal constituent. Thus Eq. 3.5 and 3.6 together represent the complete set of universal constituents, which form the architecturally uncommitted originator for the development and refinement of the architectural building blocks, discussed in the upcoming sections.

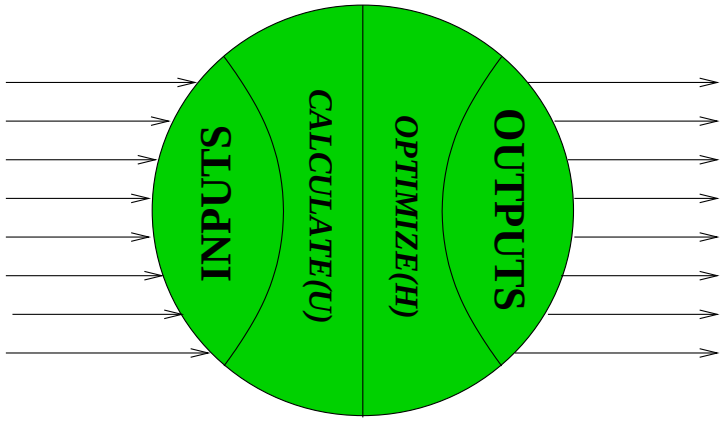


Figure 3.2: Diagrammatic representation of the identified universal constituents on the basis of one exemplary site s_i.

A summarizing diagrammatic preparation of the universal constituents (Eq. 3.5 and 3.6) mentioned before, solely derived on the fundamentals of Markov Random Field theory, by means of exactly one exemplary site s_i is depicted in Figure 3.2. The universal constituent site s_i is graphically represented by a circle and frames the carrier-structure, where all other universal constituents are embedded or linked to. The neighborhood system $\mathcal{N}_i$ of a site s_i is logically and for the sake of illustration split up into *INPUTS* transferring information from all neighbors $t \in \mathcal{N}_i$ to the site s_i, and secondly into *OUTPUTS* distributing the relevant data of site s_i to all its neighbors. Thus, as depicted in Figure 3.2, a flow of information takes place from the *INPUTS* to the *OUTPUTS* and, in the case of our illustration from, the left side to the right side of the exemplary site s_i. The two universal constituents left over, the neighbor potential set $\{U_{C_i^{\mathcal{N}}} : C = \text{Cliques of } s_i\}$ and the optimization method $OPTIMIZE(.)_{s_i}$, are embedded into the carrier-structure of the site s_i. The data from the *INPUTS* are processed by the neighbor potentials and finally optimized to calculate the value(s) for the *OUTPUTS* . As can also be seen from this schematic representation of the universal constituents and their dependencies among each other, neither hardware relevant aspects or assumptions nor VLSI specific features are hitherto directly or indirectly integrated into in the universal constituents and their representation.

3.2 Architectural Building Blocks

Based on the previously derived universal constituents, summarized by Eq. 3.5 and 3.6, the next step to systematically gain a well-founded definition of a system-

architecture template compiled of different building blocks can be conducted. During this architectural definition and refinement step we develop respectively detail hardware relevant building blocks, which are ideally tuned for massively parallel processing at first and secondly for a semiconductor technology independent representation, always to stress this point again, exclusively based on the earlier defined universal constituents.

Furthermore, all architectural building blocks are categorized along a VLSI-appropriate scheme, which differentiates *topology & structure* modeling building blocks $\mathcal{BB}^{TS}$, *processing functionality* modeling building blocks $\mathcal{BB}^{PC}$ and *control functionality* modeling building blocks $\mathcal{BB}^{CT}$. The complete set of building blocks is denoted by $\mathcal{BB}^{System}$ in the sequel. Exactly this categorizing scheme, covering the different architectural building blocks, represents a systematic method to reasonably deal with the diverse but specific issues of large scale integration, which occur at regular intervals. These are for example design complexity handling, controllable synthesis of complex and large systems, guided technology mapping and system floor-planning, to name just the most important ones.

In Section 3.3 we comment on the relevant large scale integration issues of the different building block since each block poses slightly different VLSI challenges. In order to tighten the discussion, the derivation and definition of the architectural building blocks and their categorization along the VLSI-appropriate scheme are summarized in one single step.

3.2.1 Topology & Structure Building Blocks

The first category of the VLSI-appropriate scheme covers architectural building blocks, which either directly determine the topology respectively structure of the massively parallel system-architecture or at least have a significant influence on the system topology. Obviously the two universal constituents $\mathbb{S}_{sites}$ (cf. Definition 3.2) and $\mathbb{N}_{sites}$ (cf. Definition 3.4) are the sources for further architectural definitions and refinements regarding the topology of the architecture template. As mentioned in the introduction of this chapter, the systematic process of developing architectural building blocks depends on the two constraints namely parallel processing and large scale integration, whereas the mapping of universal constituents to architectural building blocks is formalized by $\mathbf{\Lambda} = \{\Lambda^{\mathrm{Hulls}}, \Lambda^{\mathrm{W}}, \Lambda^{\mathrm{PM}}, \Lambda^{\mathrm{GMem}}\}$.

When taking these constraints and the above mentioned universal constituents into account, we at first derive the empty hulls of each site $s_i \in \Omega$ as a topology defining architectural building block. In addition to that, we derive the wiring among the sites, defined by the neighborhood system $\mathcal{N}$ on the site-grid Ω as the second topology defining architectural building block. Together with the assumption of the two-dimensional embedding of the site-grid in the plane as regular grid, a first version of the architecture gantry - with sites, site connections and site arrangements as regular grid - has been fixed.

Another two topology & structure defining building blocks are essentially required with respect to digital large scale integrations. That is the site port memory of each site, which stores the incoming data from its neighbors. Of course, these memory parts are site internal structures, but their arrangement determine the shape of the site and thus the overall topology & structure of the system-architecture tem-

plate. Consequently, it is useful and justified to include these blocks into the topology & structure building blocks. The last building block is derived from the fact that data has to be dispensed to the distributed particular sites and collected after the processing. Hence we mandatorily require a distributed global memory hierarchy to organize the data transportation task within the site-grid.

Precisely, we formally define the set of architectural topology & structure building blocks, denoted by $\mathcal{BB}^{TS}$ in the sequel, as follows:

Definition 3.1 (Topology & Structure Building Blocks $\mathcal{BB}^{TS}$)
The set $\mathcal{BB}^{TS}$ of topology & structure defining architectural building blocks is

$$\mathcal{BB}^{TS} = \{S^{Hull}, W^{\mathcal{N}}, M^{Ports}, M^{GMem}\} \tag{3.7}$$

with

- $S^{Hulls} = \{s_i^{Hull} : 1 \leq i \leq |\Omega|\}$, the particular hulls of all sites $s_i \in \Omega$.
- $W^{\mathcal{N}} = \{w_i^{<i,t>} : 1 \leq i \leq |\Omega|,\ \forall t \in \mathcal{N}_i\}$, the neighborhood wiring between all sites $s_i \in \Omega$ and all its neighbors t, defined by the corresponding neighborhood system $\mathcal{N}$ on Ω.
- $M^{Ports} = \{m_i^{Port<i,t>_x} : 1 \leq i \leq |\Omega|,\ \forall t \in \mathcal{N}_i,\ x = \#bits\}$, the port memory of each site i, storing the incoming data from their neighboring sites t.
- M^{GMem}, the distributed global memory hierarchy in order to handle data transfers from and to each site $s_i \in \Omega$.

This finalizes the overview and formal definition of the topology & structure determining architectural building blocks $\mathcal{BB}^{TS}$. We continue with the detailed discussion of each of these building blocks, following the chronological order of Definition 3.1.

Site Hull

This specific architectural building block, denoted by $s_{\cdot}^{Hull}$ for an arbitrary site $s_{\cdot} \in \Omega$, represents the most elementary building block of our VLSI relevant system-architecture template. It is directly derived from the universal constituent $\mathbb{S}_{sites}$ under the constraints of massively parallel processing and large scale integration. First of all the constraint of massively parallel processing leads to the mapping Λ^{Hulls} of each universal site constituent $s_i \in \mathbb{S}_{sites}$ to its corresponding site hull $s_i^{Hull} \in S^{Hulls}$, formalized by

$$\Lambda^{Hulls} : s_i \longrightarrow s_i^{Hull}, \quad 1 \leq i \leq |\Omega|. \tag{3.8}$$

Consequently, the architecture template consists of $|\Omega|$ distinct site hulls summarized by $S^{Hulls} = \{s_i^{Hull} : 1 \leq i \leq |\Omega|\}$ and indexed by i. The strictly monotone increasing indexing-sequence i defines an ordering on the sites s_i and thus also on the site hulls s_i^{Hull}. Assuming a regular and entirely two-dimensional site-grid, which is definitely appropriate for the herein discussed and in Section 2.5 defined class

of Markov Random Fields, the indexing sequence $1 \leq i \leq |\Omega|$ uniquely defines matrix-like coordinates (x, y) of each site hull by

$$(x, y)_i = (i - (\lfloor i/Sites\_per\_Row \rfloor - 1) * Sites\_per\_Row, \lfloor i/Sites\_per\_Row \rfloor) \quad (3.9)$$

and with an algebraic sign change $(x, -y)_i$ also coordinates in the Cartesian system. Essentially, the $|\Omega|$ derived site hulls s_i^{Hull} are uniquely defined and embedded in the plane by their index i and thus pattern a topology gantry of the system-architecture template.

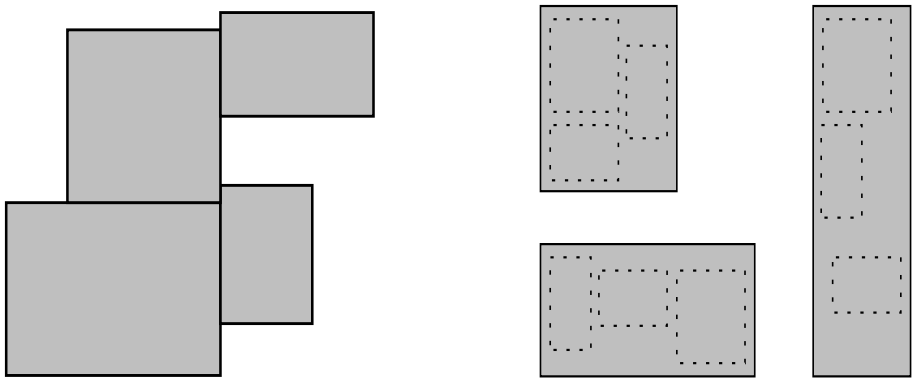


Figure 3.3: Diagrammatic representation of exemplary site hulls $s_.^{Hulls}$ with possible VLSI relevant shapes. (a) Modules without encapsulating site hull. (b) Modules with encapsulating site hull.

So far only parallel processing aspects have been called on for the derivation and definition of this first and elementary topology & structure constituting building block during the previous discussion. Large scale integration aspects have not been taken into consideration. But of course large scale integration aspects also significantly contribute to the particular peculiarity of this topology and structure defining building block. Purely digital VLSI semiconductor technologies mainly possess silicon structures, as the result of the etching process, with rectangular shapes and an orientation in the plane in 90 degree steps. When putting several mate VLSI structures respectively sub-modules together in an uncoordinated and loose way, one eventually comes up with a module-shape schematically illustrated in Figure 3.3a. Obviously, such module-accumulations and the resulting module-shapes will cause difficulties within the back-end steps of the design flow. This is the reason why the distinct site hulls s_i^{Hull} additionally take over a central role with respect to large scale integration issues. By forming a self-contained site hull object, encapsulating other sub-modules, the shape of the site hulls s_i^{Hull} can again be systematically realized as a regular rectangular in contrast to the loose accumulation of modules, as depicted in 3.3b. Thus the site hulls impose a shape constraint advantageous for the floor-planning step as well as for the place&route implementation step. In addition, the realization as distinct objects with the unique ordering index i systematically organizes the set of site hulls and allows it to separately identify each site hull s_i^{Hull}.

Site Wiring

Next to the previously introduced site hulls, the site wiring building block, denoted by $w_{\cdot}^{<\cdot,t>}$ for an arbitrary site of the site grid Ω and one of its neighbors t, is the second elementary topology & structure defining building block for the massively parallel and hardware relevant system-architecture template. Both building blocks, the site hulls and the site-neighborhood wirings, together form the connected grid-gantry of the architecture template.

The site hulls are assigned several wiring building blocks, one particular wiring building block for each neighbor t. This building block type is directly derived from the universal constituent $\mathbb{N}_{sites}$, again with respect the two constraints of (1) massively parallel processing and (2) large scale integration. With this building block, also similar to the site hulls, the applied constraint of massively parallel processing leads to a mapping Λ^W, which in this case assigns each universal neighbor constituent $< s_i, t >$ to its corresponding site-neighborhood wiring building block $w_i^{<i,t>}$, formally represented by

$$\Lambda^W :< s_i, t > \longrightarrow w_i^{<i,t>}, \quad 1 \leq i \leq |\Omega|, \forall t \in \mathcal{N}_i. \tag{3.10}$$

Thus the set of neighbor relations $< s_i, t >$, defined by the universal constituent $\mathbb{N}_{sites} = \{< s_i, t >: 1 \leq i \leq |\Omega|,\ \forall t \in \mathcal{N}_i\}$, translates into connections respectively wiring-structures between the site hulls s_i^{Hull} and all its neighboring site hulls t defined by $\mathcal{N}_i$. Consequently, the architecture template is composed of a set of site-neighbor wiring blocks, which is denoted by $W^{\mathcal{N}} = \{w_i^{<i,t>} : 1 \leq i \leq |\Omega|,\ \forall t \in \mathcal{N}_i\}$ and indexed by i and t, respectively. Both indexing sequences i as well as t determine the position of the site hull i and all its neighboring site hulls t within the two-dimensional regular site-hull-grid by Equation 3.9. If site hull i takes up a position $i = (x, y)$ within the regular site-hull-grid, then possible neighboring site hulls t can take the following positions with respect to site hull i:

$$t = \begin{cases} (x+u, y) & : \text{ right of site } i \\ (x+u, y-v) & : \text{ right up of site } i \\ (x, y-v) & : \text{ up of site } i \\ (x-u, y-v) & : \text{ left up of site } i \\ (x-u, y) & : \text{ left of site } i \\ (x-u, y+v) & : \text{ left down of site } i \\ (x, y+v) & : \text{ down of site } i \\ (x+u, y+v) & : \text{ right down of site } i \end{cases} \tag{3.11}$$

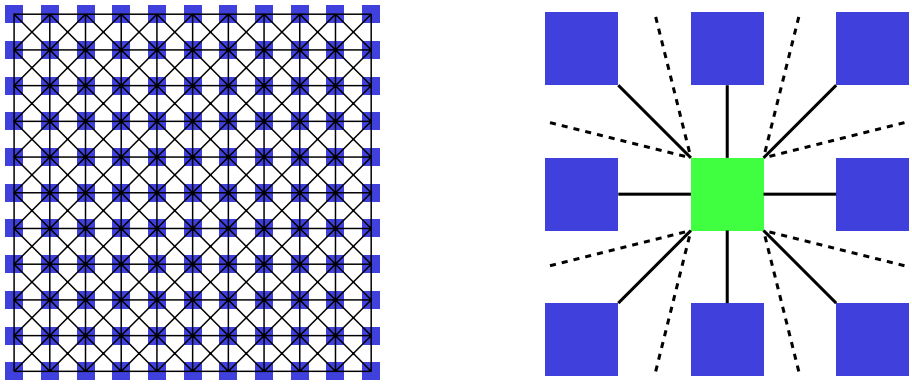
with

$$1 \leq |(x \pm u)| \leq \mathit{Sites\_per\_Row}, \quad 1 \leq u \leq \mathit{Sites\_per\_Row}$$

$$1 \leq |(y \pm v)| \leq \mathit{Sites\_per\_Column}, \quad 1 \leq v \leq \mathit{Sites\_per\_Column}$$

to ensure proper positions only within the definition-range of the site-hull-grid. Thus every wiring building block $w_i^{<i,t>}$ owns a preferred direction encoded by the indexes $i = (x_i, y_i)$ and $t = (x_t, y_t)$ within the site-hull-gantry of the architecture template.

The encoding of preferred directions by the indexes $i = (x_i, y_i)$ and $t = (x_t, y_t)$ has a far-reaching relevance, when we switch from the parallel processing constraint



(a) (b)

Figure 3.4: Illustration of site-hull-grid and imposed wiring structure together forming the architectural topology-gantry. Green: reference site, blue: neighbors. (a) Overview site-hull-grid with 1st order wiring. (b) Close-up: Site-hull with direct site-hull neighbors and their wirings. Solid lines: 2nd order wiring. Dashed lines: Higher order wirings to not shown neighbors.

to the large scale integration constraint with regard to the discussion of wiring building blocks $W^{\mathcal{N}} = \{w_i^{<i,t>} : 1 \leq i \leq |\Omega|, \ \forall t \in \mathcal{N}_i\}$. But firstly the large scale integration constraint results in each wiring block $w_i^{<i,t>} \in W^{\mathcal{N}}$ representing physical peer to peer wiring connections from site hull i to its neighboring site hull t. The concrete number of wires and thus the bus bit-width of each wiring block is per definition not fixed in the architecture template and can thus vary within a broad range, depending on application- and VLSI requirements. For instance, a $m$bit-width datum can be transmitted from site hull i to a neighbor site hull t in m steps by means of a purely serial wire building block or just in one step by a m bit-width wiring block, to mention the two extreme cases. All other useful combinations of bit-width and transmission steps are equally possible and represent a trade-off between transmission steps and wiring-lines. We shortly remark that processing speed has absolute priority in order to fulfill real-time processing capabilities and thus the setting is chosen where a $m$bit-width datum is transmitted in one step over a $m$bit-width wiring block. As long as a specific VLSI technology can not realize this wiring setting, one has to modify it.

Now we return to the preferred directions of the wiring building blocks encoded by the indexes i and t in our discussion. The consequence, which follows from this direction preference of the wiring building blocks with respect to large scale integration, is that each wiring block will be assigned to exactly one side of the rectangular site hull to ensure a straight and minimum length connection between each site hull i and all its neighboring site hulls t, which are uniquely defined by $\mathcal{N}_i$. Thus the wires of a specific wiring block will only arise from one predefined side of the site hull. This assignment scheme of wiring building block reads as follows:

$$TopSide \quad = \quad \begin{cases} (x, y - v) & : \text{ up of site} \\ (x + u, y - v) & : \text{ right up of site} \end{cases} \tag{3.12}$$

$$RightSide = \begin{cases} (x+u, y) & : \text{ right of site } i \\ (x+u, y+v) & : \text{ right down of site } i \end{cases} \tag{3.13}$$

$$BottomSide = \begin{cases} (x, y+v) & : \text{ down of site } i \\ (x-u, y+v) & : \text{ left down of site } i \end{cases} \tag{3.14}$$

$$LeftSide = \begin{cases} (x-u, y) & : \text{ left of site } i \\ (x-u, y-v) & : \text{ left up of site } i \end{cases} \tag{3.15}$$

Where *TopSide*, *RightSide*, *BottomSide* and *LeftSide* refers to the sides of the referenced site hull. This scheme is further illustrated in Figure 3.5 for neighborhood systems covered by our regarded class of Markov Random Fields (cf. Section 2.5 respectively Definition 2.10) and in addition to this for much larger neighborhood systems.

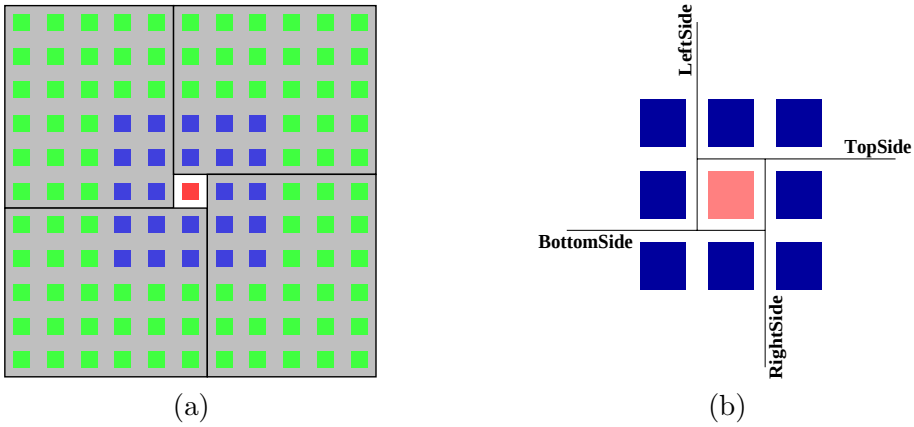


Figure 3.5: Origin and arrangement of wiring blocks at specific site-hull boarders. Red: reference site, blue: 5th order neighbors, green: 6th and higher order neighbors. (a) Overview, wiring origin for 5th order neighborhood system (blue) and larger system (green). (b) Close-up, wiring origin for direct neighbors (2nd order neighborhood system).

Obviously, the proposed scheme scales with arbitrary sized neighborhood systems on regular two-dimensional site-grids. Only such a systematic ordering and assignment of the wiring building blocks to specific sides of the site hull leads to VLSI architectures with a congestion-free wiring characteristic. Furthermore, the assignment of wiring building blocks to destined sides of the site hull, uniquely defined by Eq. 3.12, 3.13, 3.14 and 3.15, directly impinges on the site hull shape in VLSI realizations. A minimum side-length is mandatorily required to connect all wires, which are assigned to this specific site hull boarder.

Port Memory

The port memory building block for an arbitrary site $i \in \Omega$ and one of its neighbors t will be denoted by $m_i^{Port<i,t>_x}$, with $x = \#bits$, in the sequel. This module represents another fundamental topology & structure defining building block for the

architecture template. These port memory blocks $m_{\cdot}^{Port<\cdot,t>_x}$ are closely aligned with the site hulls $s_{\cdot}^{Hull}$ and conditioned by the wiring blocks $w_{\cdot}^{<t,\cdot>}$ of the corresponding neighbor site hulls, i.e. a particular memory block $m_i^{Port<i,t>_x}$ is connected with the correct wiring block $w_{\cdot}^{<t,\cdot>}$ of the neighboring site hull t in order to store these input data.

But despite this fact, the building block is solely derived from the universal constituent $\mathbb{N}_{sites} = \{< s_i, t >: 1 \leq i \leq |\Omega|,\ \forall t \in \mathcal{N}_i\}$, with respect to the two constraints of massively parallel processing and large scale integration. However, for this building block first of all the constraint of large scale integration is taken into account, which leads to the conclusion that storing elements are required to enable a proper data exchange between neighboring sites. Such a storing element is represented by the port memory building block $m_i^{Port<i,t>_x}$. When furthermore applying the parallel processing constraint to $\mathbb{N}_{sites}$, it finally comes to a mapping Λ^{PM}, where each universal neighbor constituent $< s_i, t >$ is assigned to its corresponding port memory building block $m_i^{Port<i,t>_x}$ with $1 \leq i \leq |\Omega|$ and t neighbors of site i. The mapping thus reads

$$\Lambda^{PM} :< s_i, t > \longrightarrow m_i^{Port<i,t>_x}, \quad 1 \leq i \leq |\Omega|,\ \forall t \in \mathcal{N}_i,\ x = \#bits. \tag{3.16}$$

Therefore, the set of neighbor relations $< s_i, t >$, defined by the universal constituent $\mathbb{N}_{sites} = \{< s_i, t >: 1 \leq i \leq |\Omega|,\ \forall t \in \mathcal{N}_i\}$, translate into storing elements respectively memory structures for each site $s \in \Omega$. By definition the memory structures are divided up in separate blocks for each neighbor relation $< i, t >$ of site s_i. Furthermore, the memory blocks are located within the site hull. Figure 3.6

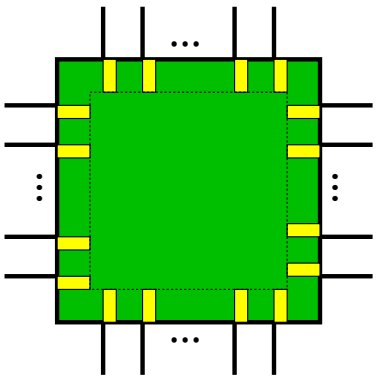


Figure 3.6: Arrangement of port memory building blocks within site hull and assignment to corresponding wiring blocks at the site hull boarders. Yellow: port memories, green: site hull.

schematically illustrates this arrangement. Consequently, the architecture template so far defined is extended by the set of port memory building blocks, which is represented by $M^{Ports} = \{m_i^{Port<i,t>_x} : 1 \leq i \leq |\Omega|,\ \forall t \in \mathcal{N}_i,\ x = \#bits\}$ in the sequel. Thus the recapitulations of the architecture template comprises

$$\mathcal{BB}^{System} = \{S^{Hull}, W^{\mathcal{N}}, M^{Ports}\}. \tag{3.17}$$

Again, the indexes i and t define an ordering on and among the so-far-defined building blocks of the architecture template. Naturally, by means of index i, all

building blocks, which belong together, can be uniquely identified. Precisely, by means of index i we can identify a site hull s_i^{Hull}, a set of wiring blocks $w_i^{<i,t>}$ and a set of port memory blocks $m_i^{Port<i,t>_x}$ for each site $s_i \in \Omega$. These three building block types form a tightly coupled unit for each index i. Within such a unit the index t orders the different wiring blocks as well as the particular port memory ports. At the same time the index t determines the position of the wiring blocks with respect to the sides of the site hull and thus also the position of the port memory blocks which are directly located beside them.

Such a unit, consisting of a site hull, the wiring blocks and the port memory blocks, is exemplary depicted for one site i in Figure 3.7a. If one puts n units together and arranges them onto a regular two-dimensional grid, the topology gantry of our massively parallel system-architecture template for statistical image- and signal processing models formulated on regular Markovian site grids emerges. A closer lock at Equation 3.16 (mapping wiring) but also at Figure 3.7b reveals that only the incoming side of the wiring block possesses a corresponding port memory block. The outgoing part of the wiring block is merely connected with the neighboring site

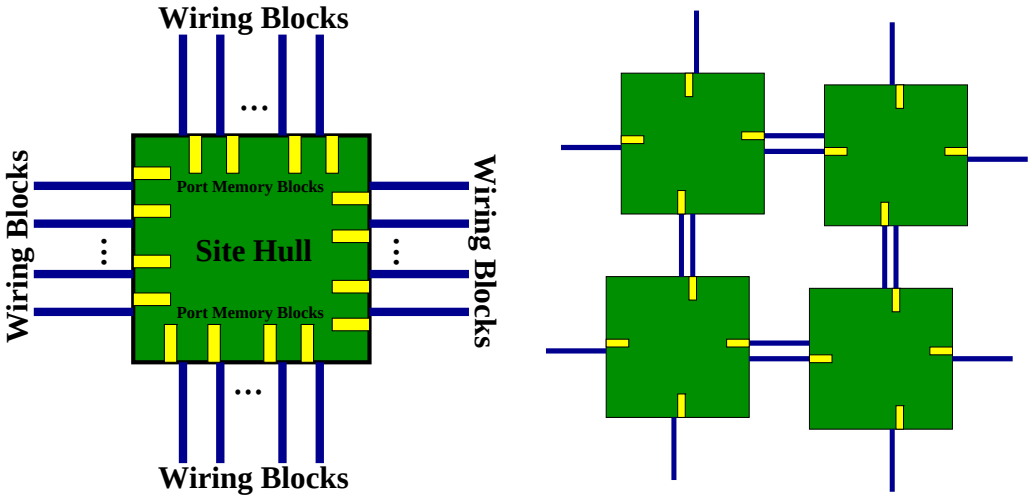


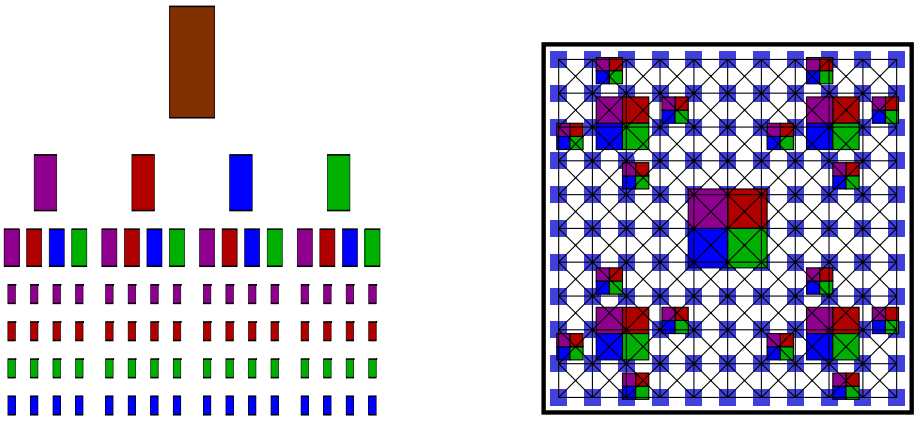
Figure 3.7: Site architecture (hull, port memories and wiring blocks) and small site cluster. Yellow: port memories, green: site hull, blue: wiring blocks. (a) Summary of so far derived site architecture. (b) Small site cluster and detailed relation wiring blocks to port memories.

hull and not buffered by memory elements. The data, which has to be transmitted from the site to its neighbors, is directly transfered to the outgoing wire-part.

Global Memory Hierarchy

The global memory hierarchy building block represents a single and self-contained structure within the building block compound of the system-architecture template. This building block is denoted in the sequel by M^{GMem}. This part of the architecture template also represents a basic topology & structure forming building block. The

complete memory hierarchy M^{GMem} is closely connected to the site hulls s_i^{Hull}. Again, this specific building block is solely derived from the universal constituent $\mathbb{S}_{sites} = \{s_i : 1 \leq i \leq |\Omega|\}$. Certainly the architectural building block memory hierarchy is also derived fulfilling the two constraints of massively parallel processing and large scale integration.



(a) (b)

Figure 3.8: Structure and memory-elements of the global memory hierarchy. (a) Abstract representation as quad-tree structure. (b) Topological embedding into two-dimensional regular site-grid.

As already done with respect to the port memory building blocks for the current building blocks, the constraint of large scale integration will once again regarded at first. When recapitulating, the sites - currently just site hulls and their connections - are spatially distributed and arranged on a regular two dimensional grid. This leads to the conclusion that a data distribution and collection structure is required to yield correct data-transportations to and from each site within the grid. Such a data distribution and collection structure is put into practice by the memory hierarchy building block M^{GMem}. When finally applying the parallel processing constraint to $\mathbb{S}_{sites} = \{s_i : 1 \leq i \leq |\Omega|\}$, a data transportation hierarchy and thus a mapping Λ^{GMem} is obtained, where each site $s_i \in \Omega$ is assigned to, respectively connected with the memory hierarchy building block M^{GMem}. This mapping Λ^{GMem} thus reads

$$\Lambda^{GMem} : \{s_i : 1 \leq i \leq |\Omega|\} \longrightarrow M^{GMem}. \tag{3.18}$$

Each site i, represented in the architecture template by its corresponding site hull s_i^{Hull}, is tightly affiliated and also logically connected with the memory hierarchy building block M^{GMem}, in order to receive raw data and to send processed data. Within the memory hierarchy both data paths, the one to the sites and the one back from the sites, are merged together in one single building block structure. Furthermore, both paths are absolutely independent from each other, to allow data to be passed to the sites and to be collected back from the sites simultaneously. This specific arrangement, respectively the separation of the two paths, supports the massively parallel processing constraint, as data can efficiently flow through the site-hull grid. Furthermore the memory hierarchy is additionally used for the

calculation of global statistic of Markov Random Fields. The details are discussed in paragraph *Parameter Estimation* of Section 3.2.2.

Additionally, the memory hierarchy is organized as a graph (cf. Figure 3.8a), to permit the data to flow through the graph, which will further improve the parallel processing capabilities by means of increasing the data flow throughput in the site-hull grid. The graph is planar to be embeddable in the plane and in order to support its weaving into the site-hull grid and the realization as VLSI structures. Such a combined site-hull grid and memory hierarchy arrangement is schematically shown in Figure 3.8b. Consequently, the system-architecture template so far defined becomes extended by the memory hierarchy building block M^{GMem}. With this additional building block at hand, the architecture template shows the following components

$$\mathcal{BB}^{System} = \{S^{Hull}, W^{\mathcal{N}}, M^{Ports}, M^{GMem}\}. \tag{3.19}$$

Each of the building blocks (cf. 3.19) so far defined contribute to the development of the system-architecture topology, which essentially determines the real-time processing capabilities by means of a regular mesh of processing units.

3.2.2 Processing Building Blocks

The second category of the VLSI-specific organization scheme covers all architectural building blocks, which represent any kind of data processing tasks within the system-architecture template. Apparently the universal constituent $\mathbb{P}_{pot} = \{U_{C_i^{\mathcal{N}}} : C = \text{Cliques of } s_i,\ 1 \leq i \leq |\Omega|\}$ as well as the universal constituent $OPTIMIZE(.)_{s_i}$ (cf. Definition 3.6) are definitely the originators of any derivations and definitions of architectural processing building blocks. For these processing building blocks the systematic process of developing is also guided by the two constraints of parallel processing and large scale integration and formalized by the set of mappings $\mathbf{\Phi} = \{\Phi^{\mathrm{EF}}, \Phi^{\mathrm{OPT}}, \Phi^{\mathrm{PAR}}\}$.

If we take the two universal constituents mentioned above and additionally these constraints into account, we derive the so-called energy functional of each site $s_i \in \Omega$, which is essentially represented by the potentials of s_i, as one of the processing building blocks. In addition to this block, we straightly derive the optimization method for each site s_i, if we regard the universal constituent $OPTIMIZE(.)_{s_i}$. Another processing building block is also extracted from the universal constituent $\mathbb{P}_{pot}$ and defines the task of parameter estimation, which is embedded in the energy functional of the image processing model. These three processing building blocks alone represent the complete set of modules, which is required to perform all processing tasks within Markov Random Field based image- and signal processing systems.

We formally define the architectural processing blocks, in the sequel denoted by $\mathcal{BB}^{PC}$, as follows:

Definition 3.2 (Processing Blocks $\mathcal{BB}^{PC}$)
The set $\mathcal{BB}^{PC}$ of architectural processing building blocks reads

$$\mathcal{BB}^{PC} = \{PC^{EF}, PC^{OPT}, PC^{PAR}\} \tag{3.20}$$

with

- $PC^{EF} = \{\mathcal{H}_i^{EF} : 1 \leq i \leq |\Omega|\}$, the set of building blocks representing the energy functionals of each site s_i.
- $PC^{OPT} = \{opt_i : 1 \leq i \leq |\Omega|\}$, the set of building blocks representing the optimization methods of each site s_i.
- PC^{PAR}, the parameter estimation block.

This short introductive discussion finalizes the overview and formal definition of the processing relevant architectural building blocks $\mathcal{BB}^{PC}$. The discussion continues with an exhaustive presentation of the different control building blocks, following the chronological order of Definition 3.2.

Energy Functional

The energy-functional respectively cost-functional building block, in the sequel denoted by $\mathcal{H}_\cdot^{EF}$ for an arbitrary site $i \in \Omega$, represents the most elementary processing building block within the system-architecture template. This basic processing building block is directly derived from the universal constituent $\mathbb{P}_{pot}$ (cf. Definition 3.3), once again subject to the constraints of parallel processing and large scale integration. Dealing with this building block first of all the constraint large scale integration is considered, which leads to the conclusion that a compact arithmetic data-path is mandatorily required to realize the calculation defined by this universal constituent in full-custom hardware. A subsequent applying the constraint of parallel processing finally leads to the mapping Φ^{EF}, where the functions $U_{C_i^{\mathcal{N}}}$ on the cliques of a site $i \in \Omega$ are mapped to their corresponding processing building block, represented by $\mathcal{H}_i^{EF}$. Thus the mapping formally reads as follows

$$\Phi^{EF} : \{U_{C_i^{\mathcal{N}}} : C = \text{Cliques of } s_i\} \longrightarrow \mathcal{H}_i^{EF},\ 1 \leq i \leq |\Omega|. \tag{3.21}$$

Consequently, the system-architecture template consists of $|\Omega|$ distinct and compactly represented energy-functionals, respectively cost-functional building blocks, solely indexed by i. Such an energy-functional building block is schematically depicted in Figure 3.9a. Together with this first processing building block the system-architecture template comprises hitherto

$$\mathcal{BB}^{System} = \mathcal{BB}^{TS} \cup \{\, PC^{EF}\}. \tag{3.22}$$

By means of the unique index i once again an ordering and relation of the so-far introduced system-architecture building blocks is defined. All building blocks, which functionally and logically belong together, are undoubtedly identified by their index i. When considering the building blocks already defined by Eq. 3.22, we - at this stage - identify with each index $i \in \Omega$ a site hull s_i^{Hull}, a set of wiring blocks $w_i^{<i,t>}$, a set of port memory blocks $m_i^{Port<i,t>_x}$ and an energy-functional block $\mathcal{H}_i^{EF}$. Furthermore, the index i uniquely determines the position of all these tightly coupled blocks within the overall site-grid arrangement. By the agglomeration of building blocks belonging together over the index i, the index t is further automatically constrained for the set of wiring blocks $w_i^{<i,t>}$ and the set of port memory blocks $m_i^{Port<i,t>_x}$. The index t organizes the arrangement of these blocks within each site hull s_i^{Hull}.

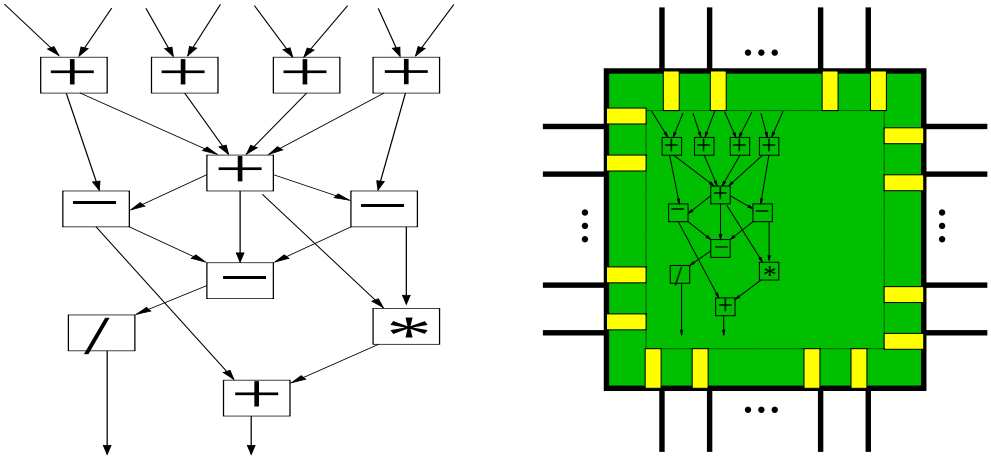


Figure 3.9: Energy-functional representation and corresponding site-hull embedding. (a) Abstract representation of energy-functional as data-flow graph. (b) Topological embedding of energy-functional into site hull.

Optimization

The next architectural building block is the optimization block. This optimization building block for each site $i \in \Omega$ is denoted by opt_i in the sequel. It represents the second processing building block within the system-architecture template and obviously forms a central element as the optimization method and its performance fundamentally determines the outcome of the whole calculation process. This fundamental processing building block is directly derived from the universal constituent $OPTIMIZE(\cdot)_{s_i}$, where the derivation process is typically determined by the two constraints of (1) parallel processing and of (2) large scale integration.

When first of all dealing with this building block, the constraint of parallel processing is taken into account. This fact leads to the conclusion that $|\Omega|$ distinct optimization building blocks $opt_.$ are required to optimize the outputs of the energy functional blocks $\mathcal{H}_.^{EF}$. After the application is finished the constraint of large scale integration leads to a mapping Φ^{OPT}, where the universal constituents $OPTIMIZE(.)_{s_i}$ are mapped to building blocks opt_i performing the optimization task. Formally the mapping thus reads

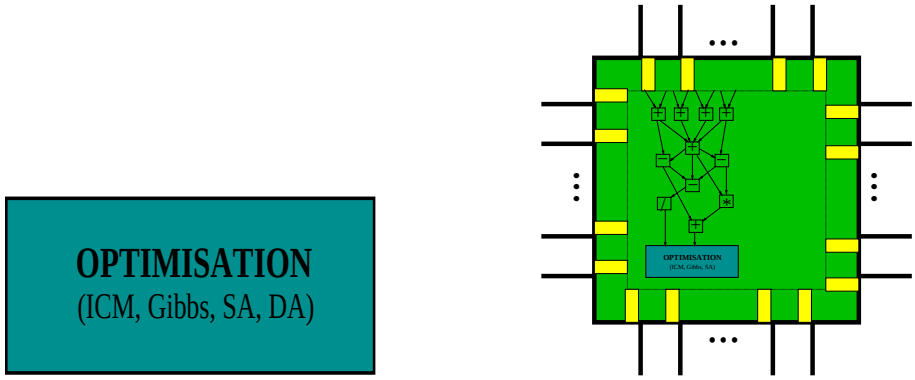
$$\Phi^{OPT} : OPTIMIZE(.)_{s_i} \longrightarrow opt_i,\ 1 \leq i \leq |\Omega|. \tag{3.23}$$

The set of optimization-functionality building blocks, defined by the universal constituent $OPTIMIZE(.)_{s_i}$, thus translates to VLSI-relevant and compact building blocks opt_i, again uniquely indexed by i. An optimization processing building block is schematically shown in Figure 3.10a. Putting all building blocks together, the system-architecture template thus comprises the following blocks

$$\mathcal{BB}^{System} = \mathcal{BB}^{TS} \cup \{PC^{EF}, PC^{OPT}\}. \tag{3.24}$$

By means of the global and unique index i we impose an ordering scheme upon on all building blocks, which is required to compose a system-architecture for a

specific Markov Random Field based signal- and image processing model. Primarily the index i brings all the different blocks so far defined (cf. Eq. 3.24) together, which are required to compile a site-module and its capabilities to connect and fit into a larger processing-grid. Each site-module currently identifies a site hull by the



(a) (b)

Figure 3.10: Illustration of optimization module and its corresponding site-hull embedding. (a) Block-schematic optimization module. (b) Topological embedding of optimization block into site-hull structure.

index i as well as a set of wiring blocks, a set of port memory blocks, an energy-functional block and an optimization block. A site-module compiled in this way is schematically shown in Figure 3.10b. Secondly the index i uniquely determines the position of all such site-modules on the grid.

Parameter Estimation

The last architectural building block, which defines the processing thread is the parameter estimation block. We denote this specific building block as PC^{PAR} in the sequel. In addition to the two processing building blocks introduced by Eqs. 3.22, 3.24, the parameter estimation building block PC^{PAR} completes the set of processing specific blocks within the system-architecture template (cf. Definition 3.2). This third processing building block is also derived from the universal constituent $\mathbb{P}_{pot} = \{U_{C_i^{\mathcal{N}}} : C = \text{Cliques of } s_i,\ 1 \leq i \leq |\Omega|\}$, once again guided by the two constraints of parallel processing and large scale integration.

At first the parallel processing constraint is regarded when looking at this specific processing building block PC^{PAR}. As the parameter estimation task represents a global calculation process of the complete actual data and thus over all sites $i \in \Omega$ of the grid, it does not become obvious how to organize and realize this important calculation step in parallel within the massively parallel system-architecture template. But the question of how to parallelize the parameter estimation task is crucial, as it is a serial bottleneck within this processing building block and will oppose our claim of defining a massively parallel system-architecture in order to achieve real-time processing capabilities. To recapitulate, the parallel processing constraint is required, because of the fact that even if we rely on the impressive semiconductor

progress to continue in the near future, all single serial processing approaches fall short of providing real-time processing capabilities for the class of statistical image processing models herein considered.

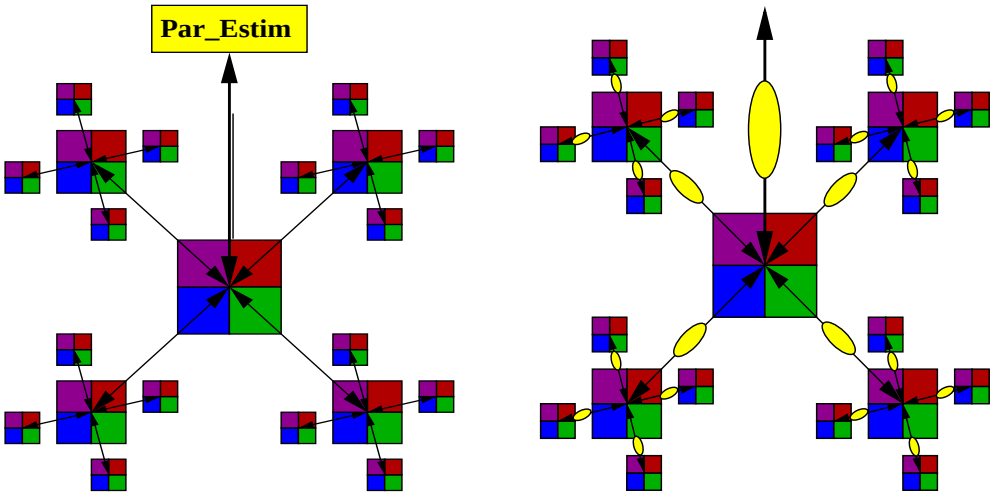


Figure 3.11: Parameter estimation schemes embedded along the memory hierarchy (Yellow: parameter estimation blocks). (a) Centered parameter estimation scheme; conducted at the highest level of the memory hierarchy in one step. (b) Local and distributed parameter estimation scheme; successively conducted on each memory hierarchy level.

Consequently, a purely serial parameter estimation process will lead our previous efforts ad absurdum. However, the global estimation process can be split up in several local calculations, which can be executed in parallel. The partitioning of the global estimation process into parallel and local calculations is organized along the memory-hierarchy structure and, therefore, shares the same algorithmic complexity as memory access. Additionally - taking the large scale integration constraint and the described memory hierarchy into account - we are able to define an ideally balanced processing scheme with respect to the serial and parallel nature of the parameter estimation process. Each memory-bank element of the memory hierarchy has to be equipped with the processing capabilities required for the parameter estimation. All these conclusions lead to a mapping Φ^{PAR}, where the universal constituents $\mathbb{P}_{pot} = \{U_{C_i^{\mathcal{N}}} : C = \text{Cliques of } s_i,\ 1 \leq i \leq |\Omega|\}$ are mapped to a building block structure PC^{PAR} performing the parameter estimation task. The mapping formally reads

$$\Phi^{PAR} : \{U_{C_i^{\mathcal{N}}} : C = \text{Cliques of } s_i,\ 1 \leq i \leq |\Omega|\} \longrightarrow PC^{PAR},\ 1 \leq i \leq |\Omega|. \quad (3.25)$$

The set of parameters hidden and embedded in the universal constituent $\mathbb{P}_{pot} = \{U_{C_i^{\mathcal{N}}} : C = \text{Cliques of } s_i,\ 1 \leq i \leq |\Omega|\}$, becomes translated into a processing building block structure arranged along the memory hierarchy, which exactly estimates these free model parameters. The system-architecture template is extended by the

parameter estimation building block and until now comprises the following building blocks

$$\mathcal{BB}^{System} = \mathcal{BB}^{TS} \cup \mathcal{BB}^{PC}. \tag{3.26}$$

Obviously, the parameter estimation building block PC^{PAR} and the memory hierarchy building block M^{GMem} form a tightly coupled structure within the architecture template, which structures and organizes both the data transportation within the site-hull grid and the parameter estimation task in a parallel manner.

The organization of the memory hierarchy, e.g. as quad-tree (cf. discussion in paragraph *Global Memory Hierarchy* of Section 3.2.1), directly determines the performance of the architecturally coupled parameter estimation process. To elucidate the dependence of the performance of the parameter estimation on the organization of the memory hierarchy one can call on multiscale approaches, which are used in applied mathematics to solve partial differential equations (PDEs), as analogy. This arrangement avoids a profound and critical serial data transportation or calculation bottleneck within the system architecture template.

Thus the global and putatively serial parameter estimation task as a first step has been successfully divided into independent and parallel processes and secondly has been seamlessly integrated and coupled with the VLSI appropriate memory-hierarchy structure. In summary we can state that the serial-parallel nature of the parameter estimation task is equivalent to the serial-parallel nature of the data transportation within the memory hierarchy. Definitely no additional serialization has been introduced into the system architecture template by the parameter estimation building block.

3.2.3 Control Building Blocks

The third category of the VLSI-appropriate scheme comprises architectural building blocks, which define all kinds of control tasks to ensure a coordinated processing-sequence within the massively parallel system-architecture. Obviously, as we have already derived VLSI relevant building blocks in the previous discussion, which need to be controlled, not all of the following control building blocks are directly derived from universal constituents but rather from already defined architectural building blocks. Nevertheless, the derivation and definition process of the control building blocks, formalized by the set of mappings $\boldsymbol{\Upsilon} = \{\Upsilon^{\mathrm{System}}, \Upsilon^{\mathrm{GMem}}, \Upsilon^{\mathrm{PortMem}}, \Upsilon^{\mathrm{EF}}, \Upsilon^{\mathrm{OPT}}\}$, is still guided by the two constraints of parallel processing and large scale integration.

Furthermore a global system control, denoted by CT^{System}, is required to control the complete hardware architecture. This system control block is neither derived from universal constituents nor from already defined building blocks. It is solely justified by the conditions of VLSI integration. The control units for the memory hierarchy CT^{GMem} and the port memories $CT^{PortMem} = \{ctrl_i^{PortMem} : 1 \leq i \leq |\Omega|\}$ represent control building blocks, which are derived from previously defined building blocks. The control units of the port memories take up a special position in the sense that certain conditions and architecture settings require these control units, but normally these control units are not needed at all. We will comment on this in the corresponding section dealing with these blocks. Nevertheless, to accentuate it, we did not violate or soften our claim of solely deriving all architectural building blocks within the boundaries of Markov Random Field theory and without

introducing theoretically unfounded features, although the above mentioned control units are not directly derived from these universal constituents. These control units are only required for hardware purposes and will not affect MRF relevant features. We will justify and put some light on this argument in the corresponding sections on control units.

Precisely, we formally define the architectural control blocks, denoted by $\mathcal{BB}^{CT}$ in the sequel, as follows:

Definition 3.3 (Control Blocks $\mathcal{BB}^{CT}$)
The set $\mathcal{BB}^{CT}$ of architectural control building blocks reads

$$\mathcal{BB}^{CT} = \{CT^{System}, CT^{GMem}, CT^{PortMem}, CT^{EF}, CT^{OPT}\} \tag{3.27}$$

with

- CT^{System}, the overall system control unit.
- CT^{GMem}, the control unit for the global distributed memory hierarchy.[1]
- $CT^{PortMem} = \{ctrl_i^{PortMem} : 1 \leq i \leq |\Omega|\}$, the set of port memory control units.
- $CT^{EF} = \{ctrl_i^{EF} : 1 \leq i \leq |\Omega|\}$, the set of energy functional control units.
- $CT^{OPT} = \{ctrl_i^{OPT} : 1 \leq i \leq |\Omega|\}$, the set of optimize control units.

This short introductory discussion finalizes the overview and formal definition of the control-relevant architectural building blocks $\mathcal{BB}^{CT}$. The discussion continues with an exhaustive presentation of the different control building blocks, thus following the chronological order of Definition 3.3.

System Control Unit

This single architectural control building block, denoted by CT^{System} in the following, represents a distinguished building block within the VLSI relevant system-architecture template and in many aspects differs from the other control building blocks, which will be introduced in the upcoming sections. The differences characterizing this control building block are profound.

First of all, this block is neither directly derived from any of the universal constituents defined by 3.5 and 3.6 nor is it derived from any building blocks already defined. Secondly, this structure is the only building block of the system-architecture template, which has direct connections with all site hulls s_i^{Hulls} and thus also with selected building blocks encapsulated in the site hulls. Furthermore the system control unit is directly connected with the control unit of the memory hierarchy. As this control building block is not derivable from the universal constituents and also not derivable from any building block of the system-architecture template at all,

[1] Due to the close linkage between the memory hierarchy and the parameter estimation, the control functionality for the parameter estimation is included in CT^{GMem}. This fact is valid for the remaining text.

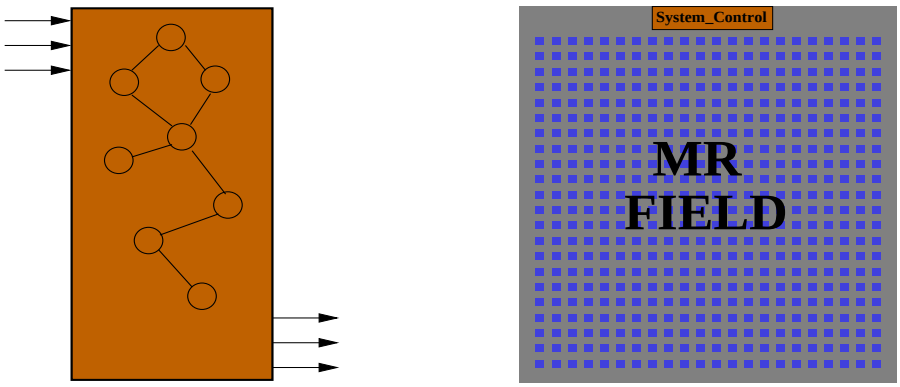
its formal existence is solely substantiated and justified by the constraint of large scale integration. Thus the mapping Υ^{System} to that specific system control building block CT^{System} is conducted by the empty set being exclusively motivated respectively justified by the large scale integration constraint. In this case the mapping formally reads

$$\Upsilon^{System} : \emptyset \longrightarrow CT^{System}. \tag{3.28}$$

Being extended by this system control building block CT^{System}, the set of introduced building blocks of the MRF system-architecture template hitherto comprises

$$\mathcal{BB}^{System} = \mathcal{BB}^{TS} \cup \mathcal{BB}^{PC} \cup \{CT^{System}\}. \tag{3.29}$$

Figure 3.12b schematically illustrates the architecture topology overview of an exemplary MRF system-architecture, composed of the building block set 3.29 so far defined. The internal structure of the system control building block is schematically represented by Figure 3.12a. At this point it should be remarked that the system control building block can principally be placed on any side around the site-grid, where - with respect to the routing task - a side-concentric position is favorable. It is definitely not limited to the position shown in Figure 3.12b, which was only chosen for the purpose of illustration.



(a) (b)

Figure 3.12: The MRF System Control Block, its representation and overall system embedding. (a) Schematic representation as finite automaton. Circles: States. Edges: Possible state transitions. (b) Embedding of system control block into overall site-grid structure.

Finalizing this section on the system control building block CT^{System} we will comment in detail on the fact that this building block is not derived from the universal constituents (cp. Eq. 3.5 and 3.6) or other system building blocks $\mathcal{BB}^{System}$, and on how this fact corresponds with our claim and principle of deriving all building blocks of the system-architecture template within the scope of Markov Random Field theory.

For all this the derivation process is determined by the two constraints of parallel processing and large scale integration. First of all we will not contradict to or weaken our claim, as this system control building block is influenced by the large scale

integration constraint even though it is not derived from any universal constituent or any other building block. Its existence is solely based on the requirements of large scale integration. Secondly, we will not violate any fundamentals of MRF theory, as this control block does not affect MRF relevant processing parts. The system control block is only responsible (1) for initiating the data transportation to the sites down through the complete memory hierarchy, (2) for initiating the calculation within the sites and finally (3) for initiating the data/result transportation from the sites up through the complete memory hierarchy. Consequently, the derivation process remains consistent with our claim although this building block essentially differs from all other blocks within the system-architecture template with respect to its derivation.

Memory Hierarchy Control Unit

The memory hierarchy control building block is denoted by CT^{GMem} in the following. In addition to the previously defined system control block, it is another central control block, which essentially coordinates and influences the overall processing sequence of the massively parallel system-architecture template. By means of this block the flow of data to and from the site hulls through the memory hierarchy is coordinated. The control-functionality of the system- as well as the memory hierarchy control block can be illustrated by processing-sequence frames (cf. [151] and Section 3.4), which uniquely define the outermost order of events within the system-architecture template. Following the upcoming derivation and definition of the mapping, we will further comment on this control frame scheme just mentioned. In Section 3.4 dealing with the cycle scheme of the system-architecture, this frame scheme is presented as an alternative to the prevalent state-transition model. The memory hierarchy control building block is derived from the memory hierarchy building block M^{GMem} already defined, and is also strictly influenced by the two constraints of parallel processing and large scale integration. Regarding this building block the constraint of large scale integration alone guides the definition process and finally leads to a mapping Υ^{GMem}, which reads

$$\Upsilon^{GMem} : M^{GMem} \longrightarrow CT^{GMem}. \tag{3.30}$$

Thus the mapping 3.30 translates the memory hierarchy building block already defined into its matching control block, which is mandatorily required with respect to large scale integration realizations. In summary the system-architecture template includes, CT^{GMem} being already added, the following building blocks

$$\mathcal{BB}^{System} = \mathcal{BB}^{TS} \cup \mathcal{BB}^{PC} \cup \{CT^{System},\ CT^{GMem}\}. \tag{3.31}$$

Figure 3.13 shows a simplified overview of this control building block and its possible location at the global memory hierarchy. The position of this control block on the chip should be chosen according to the position of the first level of the memory hierarchy and additionally should be located in the vicinity of the wires entering the chip and the first level of the hierarchy.

As mentioned at the beginning of this section the interplay of the different control blocks and the overall processing-sequence of the system-architecture is advantageous to explain and illustrate by a graphical representation of nested frames,

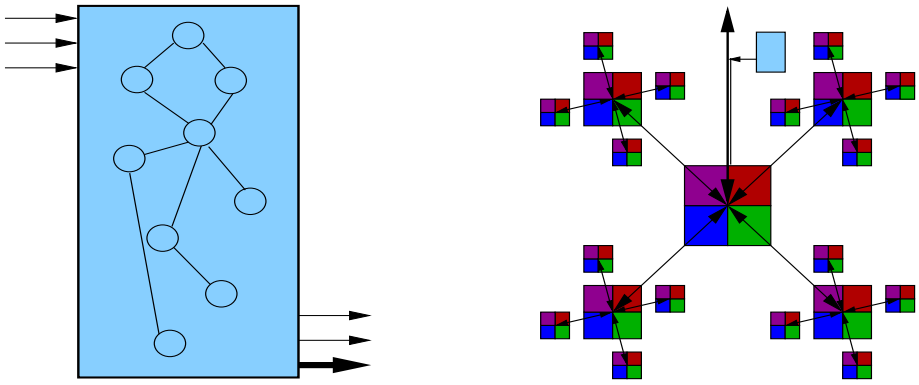


Figure 3.13: The Memory Hierarchy Control Block, its abstract representation and connection with the memory hierarchy. (a) Representation as finite automaton. Circles: States. Edges: Possible state transitions. (b) Topological connection with memory hierarchy.

where each frame represents a control building block. The ordering of the system-architecture's processing-sequence and the dependencies of the control blocks on each other are illustrated by nesting the control frames, which encodes that control frames nested in other control frames execute after the enfolding frame. Hence, the complete processing-sequence of the system-architecture, referring to the control sequence, proceeds from the outermost frame to the innermost frame and back in a pulsating way. The details are discussed in Section 3.4.

Port Memory Control Units

The set of port memory control building blocks, denoted by $CT^{PortMem}$ in the following, exactly represents these control building blocks of the system-architecture template, which directly coordinate the information flow as well as the information buffering between each site s_i^{Hull} and all its neighboring site hulls t; defined by the neighborhood system $\mathcal{N}$. All these port memory control building blocks $ctrl_i^{PortMem}$ are derived from the port memory building blocks $m_i^{Port<i,t>_x}$ already defined, whereas only one control block controls the complete set of port memory blocks embedded in each site hull i. As before the derivation process of this control unit is guided by the two constraints of parallel processing and large scale integration.

When considering the constraint of parallel processing it can be concluded at first that $|\Omega|$ distinct control building blocks are needed, one for each site hull i, to fulfill this condition. The large scale integration constraint leads to the observation that VLSI-appropriate control structures are required to coordinate the data-flow. Furthermore the large scale integration constraint justifies the decision to realize $|\Omega|$ distinct control blocks, because any other architectural arrangement with $ctrl_\cdot^{PortMem}$ building blocks controlling several site hull port memory blocks would break up the site hull container and thus generating needless floor-planning and

place&route problems. Hence, our proposed system-architecture template possesses $|\Omega|$ distinct $ctrl_{\cdot}^{PortMem}$ control blocks. The mapping $\Upsilon^{PortMem}$ formally reads

$$\Upsilon^{PortMem} : \{m_i^{Port<i,t>} : \forall t \in \mathcal{N}_i\} \longrightarrow ctrl_i^{PortMem}, \quad 1 \leq i \leq |\Omega|. \tag{3.32}$$

The set of port memory building blocks by the mapping 3.32 translates to hardware relevant and compact control building blocks $ctrl_i^{PortMem}$, which are also uniquely indexed by i. An exemplary memory port control block is schematically depicted in Figure 3.14a. Figure 3.14b shows a complete constellation of a site hull, the memory ports and its control block. At the current state, the system-architecture template is composed of the following building blocks

$$\mathcal{BB}^{System} = \mathcal{BB}^{TS} \cup \mathcal{BB}^{PC} \cup \{CT^{System}, CT^{GMem}, CT^{PortMem}\}. \tag{3.33}$$

The port memory control blocks are also uniquely indexed by i, which defines an ordering on these blocks and as already mentioned on all other building blocks of the system-architecture template. By means of this index i we identify the set of different building blocks, which belong together to form a specific site processing unit.

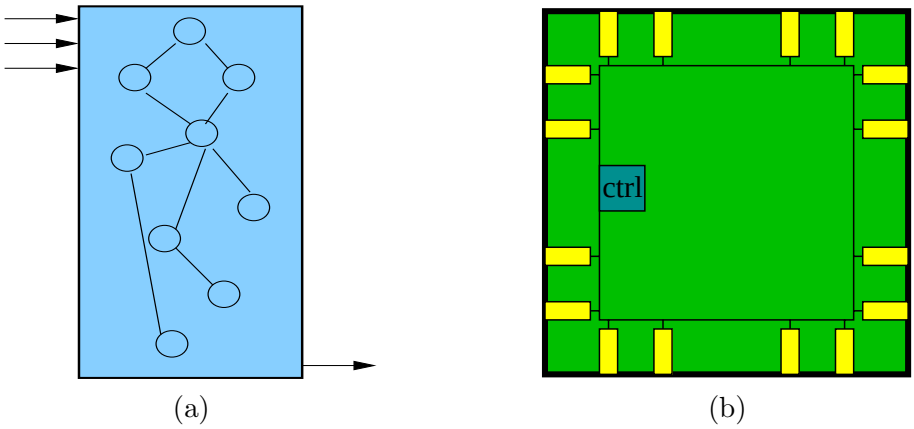


Figure 3.14: Illustration of Port-Memory Control Block and its embedding into the site hull structure. (a) Abstract representation as finite automaton. Circles: States. Edges: Possible state transitions. (b) Topological embedding.

Energy Functional Control Units

The energy functional control block, denoted by $ctrl_{\cdot}^{EF}$ for an arbitrary site, next to the port memory control block afore described, forms the second unit, which coordinates the flow of data within each site hull. Both site internal control units represent the first part of the overall site control structure, which is finally completed by the optimization control block introduced in the next section. This control building block is derived from the energy functional building blocks $\mathcal{H}_{\cdot}^{EF}$ already defined; where the derivation process is as before guided by the two constraints of parallel processing and large scale integration. The condition of large scale integration

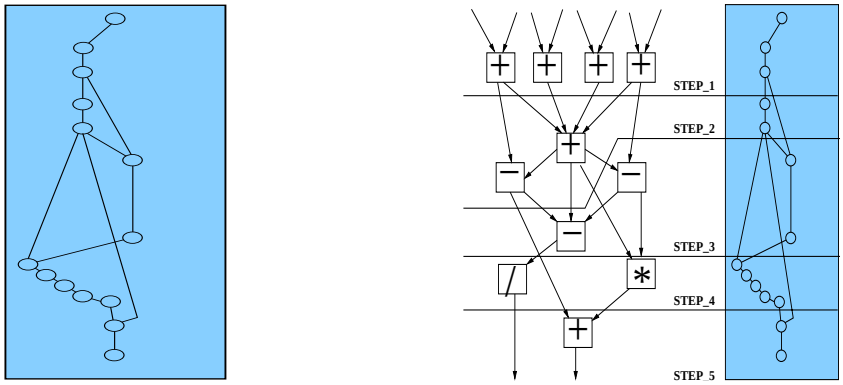
results in requiring a hardware relevant control structure in order to control the flow of data within the energy functional building block. Furthermore, the parallel processing constraint determines that for all $|\Omega|$ energy functional building blocks control structures are required. Additionally particular energy functional control blocks for each site hull conserve the container-functionality of the site hulls, which becomes advantageous for floor-planning and place&route tasks. This leads to the mapping Υ^{EF}, which is formally defined by

$$\Upsilon^{EF} : \mathcal{H}_i^{EF} \longrightarrow ctrl_i^{EF}, \quad 1 \leq i \leq |\Omega|. \tag{3.34}$$

Consequently, the set of already defined energy functional building blocks $\mathcal{H}_i^{EF}$ translates into distinct hardware relevant control structures for each of these energy functional building blocks. Figure 3.15a schematically illustrates such an energy functional control building block. The arrangement of the energy functional building block and its corresponding control block is shown in Figure 3.15b. In summary the system-architecture template comprises the following parts:

$$\mathcal{BB}^{System} = \mathcal{BB}^{TS} \cup \mathcal{BB}^{PC} \cup \{CT^{System}, CT^{GMem}, CT^{PortMem}, CT^{EF}\}. \tag{3.35}$$

Naturally these energy functional control building blocks are also uniquely indexed by i, in order to establish an ordering of these parts of the system-architecture and to assign respectively connect them over the index i to all other blocks, which belong together. By means of index i, the elementary processing units - the sites - are compiled out of the different building blocks (cf. Eq. 3.35).



(a) (b)

Figure 3.15: Energy-Functional Control Block. (a) Representation as finite automaton. (b) Relation of energy-functional's operators and states of the automaton.

Optimization Control Units

This architectural control building block, in the sequel denoted by $ctrl_{\cdot}^{OPT}$ for an arbitrary control optimization building block, represents a fundamental structure and at the same time forms the last missing part of the system-architecture template. Both parts, the optimization building block and its control block, together form

the essential unity, which significantly determines the performance of the complete image processing device. This control building block is derived from the processing building block *opt.* already defined and summarized by PC^{OPT}.

With regard to this building block of the system-architecture template, the derivation process is once again definitely guided by the two constraints of parallel processing and large scale integration. First the constraint of large scale integration is being looked at, which leads to the conclusion that a hardware respectively VLSI qualified structure is required to control the flow of data within the optimization block. Finally the constraint of parallel processing constraint leads to the fact that for each optimization building block such a control structure is needed. In the end all this results in the mapping $\Upsilon^{CtrlOPT}$, which is formally defined as follows

$$\Upsilon^{CtrlOPT} : opt_i \longrightarrow ctrl_i^{OPT}, \quad 1 \leq i \leq |\Omega|. \tag{3.36}$$

Thus the defined set of control optimization building blocks $CT^{OPT} = \{ctrl_i^{OPT} : 1 \leq i \leq |\Omega|\}$ by the mapping 3.36 translates into VLSI hardware qualified control structures; exactly one control block for each optimization building block. Figure 3.16a depicts an exemplary optimization control building block - whereas Figure 3.16b schematically shows the arrangement of the optimization building block and its control block within the site hull. With this last building block, the system-architecture template for massively parallel VLSI architectures of Markov Random Field based signal- and image processing devices is finalized and completely reads

$$\mathcal{BB}^{System} = \mathcal{BB}^{TS} \cup \mathcal{BB}^{PC} \cup \mathcal{BB}^{CT}. \tag{3.37}$$

As before the last building block is uniquely indexed by i and thus can be assigned to respectively combined with the rest of the building blocks, which together form the complete, basic processing unit within the system-architecture; extended by structures to transfer data to and from the sites and to exchange data between the sites.

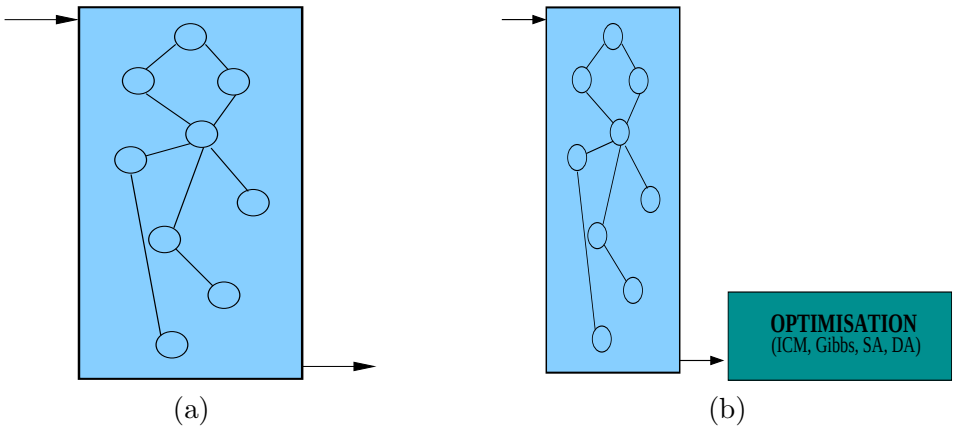


Figure 3.16: Optimization control block and its relation to the optimization block. (a) Abstract representation as finite automaton. Circles: States. Edges: Possible state transitions. (b) Arrangement of control block and optimization block.

3.3 VLSI Specific Issues

The different sets of building blocks, which have been introduced in the preceding sections, cause various difficulties during all phases of the VLSI design flow. Two groups of building blocks can be distinguished, which cause similar difficulties in nearly all processing steps of the VLSI realization chain. But these two groups also share similar strategies to alleviate these problems.

The first group comprises the following building blocks, with $1 \leq i \leq |\Omega|$, $\forall t \in \mathcal{N}_i$, $x = \#bits$: (1) The set of site hulls $S^{Hulls} = \{s_i^{Hull}\}$, (2) the set of wiring building blocks $W^{\mathcal{N}} = \{w_i^{<i,t>}\}$, (3) the set of port memories $M^{Ports} = \{m_i^{Port<i,t>_x}\}$, (4) the set of energy-functional blocks $PC^{EF} = \{\mathcal{H}_i^{EF}\}$, (5) the set of optimization-functionality blocks $PC^{OPT} = \{opt_i\}$, (6) the set of port memory control blocks $CT^{PortMem} = \{ctrl_i^{PortMem}\}$, (7) the set of energy-functional control blocks $CT^{EF} = \{ctrl_i^{EF}\}$ and (8) the set of optimization-functional control blocks $CT^{OPT} = \{ctrl_i^{OPT}\}$

The second group comprises the building blocks: (1) The global distributed memory hierarchy M^{GMem}, (2) the parameter estimation functionality PC^{PAR}, (3) the system control block CT^{System} and finally (4) the memory hierarchy control block CT^{GMem}.

$\mathcal{BB}$ **Types**	**HDL Compile Tasks**			
Topology&Structure	analyze	synthesis	mapping	copying
$s_{\cdot}^{Hull}$ prototype	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	-
s_i^{Hull}, $1 \leq i \leq \|\Omega\|$	$\mathcal{O}(\|\Omega\|)$	-	-	$\mathcal{O}(\|\Omega\|)$
$w_{\cdot}^{<\cdot,\cdot>}$ prototype	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	-
$w_i^{<i,t>}$, $1 \leq i \leq \|\Omega\|$	$\mathcal{O}(\|\Omega\|)$	-	-	$\mathcal{O}(\|\Omega\|)$
$m_{\cdot}^{Port<\cdot,\cdot>}$ prototype	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	-
$m_i^{Port<i,t>}$, $1 \leq i \leq \|\Omega\|$	$\mathcal{O}(\|\Omega\|)$	-	-	$\mathcal{O}(\|\Omega\|)$
[1] M^{GMem}	$\mathcal{O}(\|\Omega\| \log \|\Omega\|)$	$\mathcal{O}(\|\Omega\| \log \|\Omega\|)$	$\mathcal{O}(\|\Omega\| \log \|\Omega\|)$	-

Table 3.1: Processing Complexity for topology & structure representing graphs with respect to particular HDL compile tasks.

The main problem of the building blocks in the first group is essentially caused by the tremendous number of at least $|\Omega|$ distinct objects per building block set. We shortly remark that the number of wiring blocks $W^{\mathcal{N}}$ and port memories blocks M^{Ports} exceed $|\Omega|$, as every site hull s_i^{Hull}, $1 \leq i \leq |\Omega|$ needs several wiring and port memory building blocks (see Figure 3.7a-b), at least four for the first order neighborhood system. The total number of building blocks is unalterable, as we pursue a massively parallel processing approach in order to realize real-time processing capabilities. At the beginning of the design process, each particular building block has to be represented by a synthesizeable model in a standardized hardware description language for driving modern and approved design flows. Normally hun-

[1] Logarithmically bounded memory hierarchy.

dreds and thousands of textual descriptions, each with m lines, have to be created and stored in a bunch of huge files, which quickly reaches unmanageable file-sizes with respect to further analyzing, synthesis and mapping tasks. This disregards the fact of whether the building block descriptions are handcrafted or automatically generated.

But the overall textual description length can significantly be reduced, if we exploit the feature that all components of each building block set are structurally identical within the MRF processing grid. This leads to a strategy of organizing building block descriptions, where one prototype for each building block set is fully described (with m_{full} lines) and the $|\Omega|$ respectively $|\Omega| \times |\mathcal{N}_i|$ for $W^{\mathcal{N}}$ and M^{Ports} are i respectively i, t parametrized instantiations (each with m_{instance} lines) of the corresponding prototype. Consequently, the total description length is reduced from $|\Omega| \times m_{\text{full}}$ lines to no more than $|\Omega| \times m_{\text{instance}}$ respectively $|\Omega| \times |\mathcal{N}_i| \times m_{\text{instance}}$ lines, with $m_{\text{full}} \gg m_{\text{instance}}$. This basic optimization strategy of organizing the descriptions of the building blocks in the first group alone, overcomes the critical bottleneck of transferring large design descriptions to the following VLSI design steps of analysis, synthesis and technology-specific gate-mapping.

$\mathcal{BB}$ **Types**	**HDL Compile Tasks**			
Processing	analyze	synthesis	mapping	copying
$\mathcal{H}_{\cdot}^{EF}$ prototype	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	-
$\mathcal{H}_{i}^{EF}$	$\mathcal{O}(\lvert\Omega\rvert)$	-	-	$\mathcal{O}(\lvert\Omega\rvert)$
opt. prototype	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	-
opt_i	$\mathcal{O}(\lvert\Omega\rvert)$	-	-	$\mathcal{O}(\lvert\Omega\rvert)$
[1] PC^{PAR}	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	-
[2] PC^{PAR}	$\mathcal{O}(\lvert\Omega\rvert \log \lvert\Omega\rvert)$	$\mathcal{O}(\lvert\Omega\rvert \log \lvert\Omega\rvert)$	$\mathcal{O}(\lvert\Omega\rvert \log \lvert\Omega\rvert)$	-

Table 3.2: Processing Complexity for processing functionality representing graphs with respect to particular HDL compile tasks.

All processing complexities, subdivided into *analyze*, *synthesize*, *mapping* and *copying*, are illustrated in Table 3.1 for topology & structure defining building blocks, in Table 3.2 for processing-functionality defining building blocks and in Table 3.3 for control-functionality defining building blocks. These tables illustrate the savings, which can be obtained by systematically applying the above described strategy of prototypes and replacements of instantiations.

Likewise the large number of distinct building blocks alone cause problems during the chip floor-planning. However, the index i uniquely identifies each site hull s_i^{Hull}, $1 \leq i \leq |\Omega|$ and its corresponding position within the regular two-dimensional site-grid. Additionally, the index i uniquely identifies all other building blocks and above all interlink these blocks with the correspondingly correct site hull s_i^{Hull}. Furthermore, the index t assigns the wiring blocks $W^{\mathcal{N}}$ and port memories blocks

[1] Parameter estimation executed at the top of the logarithmically bounded memory hierarchy.
[2] Parameter estimation executed along the logarithmically bounded memory hierarchy.

$\mathcal{BB}$ **Types**	**HDL Compile Tasks**			
Control	analyze	synthesis	mapping	copying
CT^{System}	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	-
CT^{GMem}	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	-
$ctrl_i^{PortMem}$ prototype	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	-
$ctrl_i^{PortMem}$	$\mathcal{O}(\lvert\Omega\rvert)$	-	-	$\mathcal{O}(\lvert\Omega\rvert)$
$ctrl_i^{EF}$ prototype	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	-
$ctrl_i^{EF}$	$\mathcal{O}(\lvert\Omega\rvert)$	-	-	$\mathcal{O}(\lvert\Omega\rvert)$
$ctrl_i^{OPT}$ prototype	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	-
$ctrl_i^{OPT}$	$\mathcal{O}(\lvert\Omega\rvert)$	-	-	$\mathcal{O}(\lvert\Omega\rvert)$

Table 3.3: Processing Complexity for control functionality representing graphs with respect to particular HDL compile tasks.

M^{Ports} to their intended position within the site hull. Therefore the chip floor-planning can systematically be conducted. Evidently, a detailed floor-plan, with all site hulls assigned to their correct chip regions, interlinked with their other building blocks and related to each other, essentially speeds up the successful placing of these extremely large structures. The site hulls serve as strict boundaries in which only routing of the assigned other building blocks is allowed. This prevents the chip layout to become routed in a complicated way, which will finally prevent timing violations.

The problems of the building blocks in the second group are completely different compared to each other and to the first group. Essentially, the problems of M^{GMem} are, (1) how to flexibly link RAM model descriptions with predefined and technology specific RAM blocks, (2) how to separate the memory hierarchy gantry and the RAM models (RAM encapsulation) from each other, (3) how to flexibly exchange tri-state drivers with multiplexer and (4) how to merge the memory hierarchy with the complete site-grid. RAM-blocks represent very specific components within each digital semiconductor technology, as these blocks are hand-tuned to improve area, speed and power consumptions. These RAM components are either automatically provided by the semiconductor company in very common sizes or on request of the customer. Furthermore, it is also possible to compile larger RAM-blocks out of smaller RAM-blocks - this approach is prevalent in FPGA technologies. Within the memory hierarchy building block we have to link these technology specific RAM-blocks with our description. The complete RAM-block linking problem is even made worse by the fact that the memory hierarchy building block consists of RAM-blocks with different sizes at each level of the hierarchy. Thus we have to link a number of different RAM-blocks, which equals the levels of the memory hierarchy.

To organize and systematically conduct this linking process for a broad class of semiconductor technologies, we have developed and implemented the following approach consisting of two phases: In the first phase the description itself loosely links to several RAM-blocks. The second phase finally makes the linking non-ambiguous

by means of additional constraints or embedded directives. In order to further simplify and structure the linking task, we have to separate the memory hierarchy gantry from the RAM-blocks. This is done by means of special memory-wrappers and by encapsulating these memory-wrappers within the hierarchy. This allows us to change and modify the linking process without destroying the overall memory hierarchy gantry. As an ultimate consequence of this strategy, we can systematically study synthesis and place & route issues in changing RAM-block settings, including early design settings, where the RAM-blocks are represented by dummy filling blocks. By this arrangement and encapsulation strategy, it is possible to selectively pick up particular memory-wrappers, in order to assign tri-state bus drivers or multiplexers. Thus we can systematically exchange or mix these two types of bus-drivers, depending on the technology requirements and resources available; whereas only the data-path, which collects the results from the sites, requires bus-drivers.

The control block CT^{System} as well as CT^{GMem} only cause problems during the place&route implementation task. In the CT^{System} block a limited number of control connections arise, which run to all $|\Omega|$ site hulls and thus have to be routed over the complete chip. The connections, even though limited in number, have to be handled like clock-nets and have to be routed in a balanced signal-propagation tree. The overall routing effort of these control signals is therefore higher than compared with normal signal nets. This routing problem worsens depending on the absolute chip-area and the wire-length of these global signals.

In the CT^{GMem} block address wires traverse the complete hierarchy of this structure, which need signal buffering respectively signal refreshing as soon as the load and overall wire length increases a certain limit. This problem can become so serious that special and semiconductor technology specific buffers or logic-neutral signal refreshers need to be added into the hardware description itself to solve the problem. But this situation is only recognizable, if a complete VLSI design flow for this building, including placing and routing, has failed to meet the final timing-check. In order to diminish this building block specific problem, it is possible to regard the memory hierarchy and its control building block separately from the complete system-architecture. Hence, it is possible to isolate and study possible signal-buffering problems so that implementation technology specific strategies can be developed to solve these problems.

3.4 Cycle Scheme

The massively parallel architecture template, introduced in Section 3.2, was conceptualized and adjusted to comply with purely digital semiconductor design technologies. Purely digital design technologies offer several profound advantages with respect to the realization of massively parallel image processing devices, which are based on the calculation principles of Markov Random Fields, as discussed in Section 2.7. Thus, as we are dealing with purely digital and consequently with clocked systems, which evolve in discrete time steps respectively cycles, a template to order the sequence of events is required for the massively parallel system-architecture template to become a well-defined, complete and operational one. Without a template of ordered events no coordinated and correct data transportation and calculation is

possible within the clocked massively parallel processing devices.

An order of events template for the introduced massively parallel architecture template is described by means of two different representations. The first representation uses nested frames [151] to illustrate the order of events, whereas this specific representation hides certain details. Within this representation each frame depicts a discrete system state. The nested frames indicate that a state transition takes place starting at an outer frame and continuing to the inner frame directly following. State transitions from inner frames to arbitrary outer frames are also possible but not illustrated by this frame representation. The second representation makes use of a directed graph to define the order of events. In this representation nodes depict discrete system states and edges indicate the possible transition from one state to another.

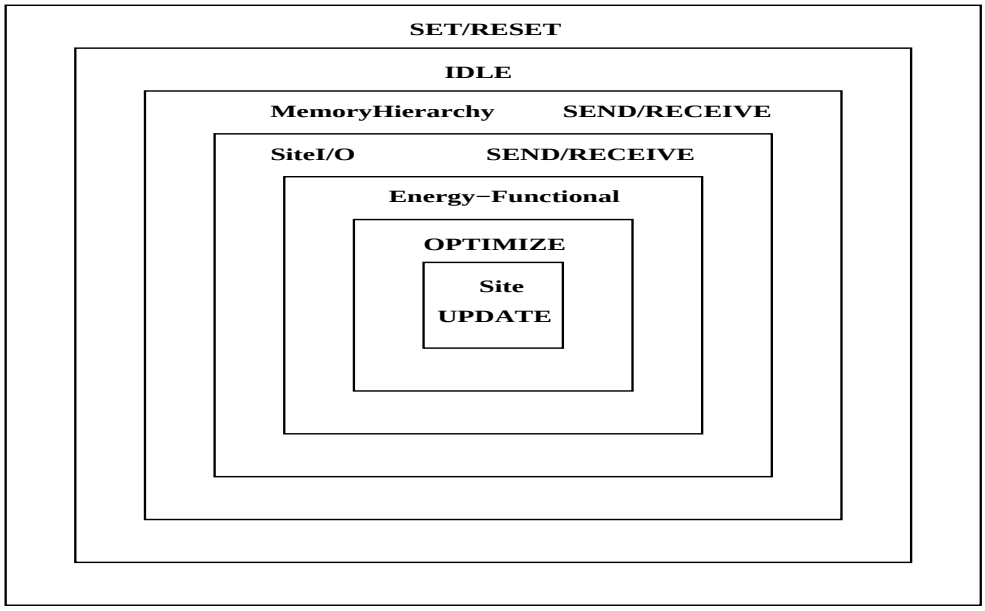


Figure 3.17: Order of events illustrated by frame representation. Discrete system states are represented by particular frames. State transitions take mainly part from outer frames to inner frames.

The nested frame notation is illustrated in Figure 3.17. Each frame represents a system super-state, which may contain further states, and thus Figure 3.17 shows the main system states each MRF processing device traverses through. The outermost frame represents the *SET/RESET* state of the system, which sets the complete device in a predefined starting state. Starting with this state the system migrates to the *IDLE* state, where it rests until the processing device gets a signal to start. When the device starts operating, it migrates to the *MemoryHierarchy SEND/RECEIVE* state. At this state the different levels of the memory hierarchy are periodically changing from the SEND to the RECEIVE state, in order to distribute or collect the data within the site grid. After having finished each SEND/RECEIVE state-change of the memory hierarchy, the system migrates to the *SiteI/O SEND/RECEIVE* state, where the sites are either set in the SEND or RECEIVE state or the sites are simultaneously set in the SEND and RECEIVE state, depending on their activity

status within the site-grid (see Section 2.4 and Figure 2.2).

Sites, whose I/Os have finished their receive or simultaneous send/receive phase migrate to the *Energy-Functional* state, whereas all those sites who have finished their send-phase remain in their current state and wait to change to the *SiteI/O RECEIVE* state. In the *Energy-Functional* state each active site calculates the value of the energy functional and migrates to the *OPTIMIZE* state. At this state the costs of the energy-functional are optimized, which leads to a recalculation of the energy functional and thus to an iterative state transition between the *Energy-Functional* and the *OPTIMIZE* state. When the *OPTIMIZE* state has finished and determined a new site value, the system migrates to the *Site UPDATE* state. From this state the system can migrate to the *SiteI/O SEND/RECEIVE* state to enable inactive sites to become active. Or, if all sites have become active exactly once, the system can migrate to the *MemoryHierarchy SEND/RECEIVE* to distribute new data and collect results. Obviously there are several improvements at hand to allow the order of events template to enhance the overall data throughput of the system and to decrease the processing time. However, various improvements depend on the actual MRF model and the overall integration of the MRF processing device in a larger systems' environment. Consequently, the introduced order of events template is flexible enough to cover the contemplated class of MRFs (cf. Definition 2.5) and additionally represent an adequate starting point for developing further improvements.

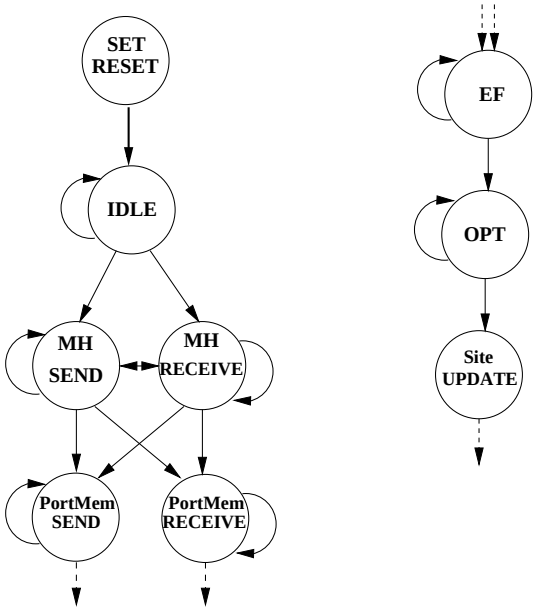


Figure 3.18: Order of events illustrated by a directed graph. Circles: system states. Edges: possible state transitions.

Irrespectively of the compact and elegant frame representation, one major drawback of this method is that it hides certain state transition details. Even though the overall order of the event's sequence is represented by the frame method, sometimes it is required to show additional state transition details. Because these modeling

details can not be illustrated in the frame representation, one has to fall back on the standard graph representation.

The graph representation method offers the features to illustrate all state transition details if necessary. In this representation method the different states are illustrated by nodes and the transition to other states is modeled by directed and marked edges between particular nodes. The edge marks define the requirements that have to be met to allow the state transition to take place. The order of events' template in graph notation for the massively parallel MRF architecture template is illustrated in Figure 3.18. Additionally the graph representation is helpful with regard to digital hardware realizations, because finite state machines (FSM), which realize control machines in hardware, are also modeled by means of these graph structures.

3.5 Relation of Thesis Parts

All main parts of this thesis, are interdependent but also closely linked to each, in order to form together a seamless simulation- and design-environment for massively parallel hardware architectures of Markov Random Field based image processing systems. The thesis comprises the following part: (1) the fundamentals of Markov Random Field theory (Chapter 2), (2) the building blocks of the system-architecture template introduced in the current chapter, (3) the simulation-modules of the simulation framework (Chapter 4), (4) the graph-theoretical representation of the MRF device in the design framework, and (5) finally the concrete VLSI implementation of a MRF model (Chapter 5). Exactly this interplay of the different thesis parts becomes formally expressed by a *relation diagram*, which is shown in Figure 3.19. Even though only the fundamentals of Markov Random Field theory, the univer-

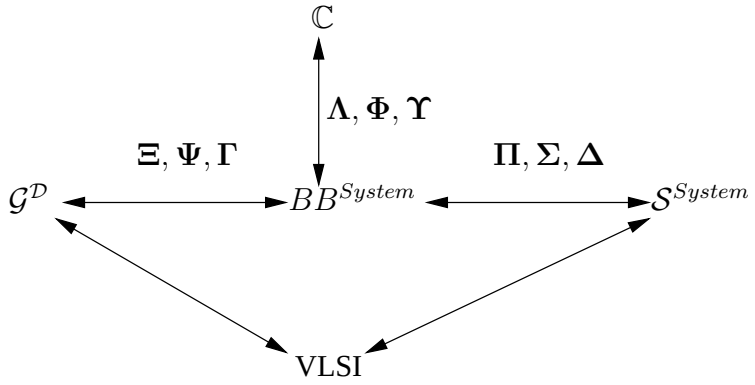


Figure 3.19: Relation diagram. Established relation of the different thesis parts.

sal constituents and the building blocks derived out of them, which makes up the system-architecture template, have been introduced, Figure 3.19 already shows the complete relation diagram this thesis establishes. Even though the details of the different parts of the relation will be presented in the corresponding chapters, the complete relation diagram already becomes presented here to introduce the line of thought this thesis pursues.

Up to now we have established the following parts and relations of the overall thesis part relation: At first we have established the relation of the fundamentals of Markov Random Field theory and a generic structure of models, which are formulated on regular two-dimensional site-grids. Based on this generic MRF model structure we have defined universal constituents, which are architecturally uncommitted and represent the first part of the relation. The second part comprises the building blocks $\mathcal{BB}^{System}$ of the system-architecture template. This part is connected with the universal constituents by the set of mappings $\mathbf{\Lambda}$ for *topology & structure* relevant parts, by the set of mappings $\mathbf{\Phi}$ for *processing functionality* relevant parts and finally by the set of mappings $\mathbf{\Upsilon}$ for *control functionality* relevant parts. As a result, the first detail of the relation has been established.

3.6 Dealing with Large Images

Several reasons and practical requirements exist, which make it necessary to regard system-settings, where image-sizes larger than the implemented Markov Random Field site-grid itself have to be processed. These system-settings with MRF devices realizing site-grids, which are smaller than the image-size, are solely caused either by semiconductor technology limitations or by economic limitations. Even with the ultra-deep sub-micron semiconductor technologies today available and the shrinking of the technology structures, which is expected to take place in the future, there is always an upper-bound of the chip integration density, which limits the overall MRF site-grid size. Obviously, by far the largest portion of the overall chip area of a MRF implementation is consumed by the sites. The absolute chip-area size of each site within the site-grid significantly depends on the implemented MRF model and its calculations, which have to be executed on each site.

Consequently, MRF models, which are more advanced with respect to their calculations on each site, will always be realizable as smaller MRF site-grid devices compared to VLSI realizations of simpler MRF models. Since the production of semiconductors for MRF devices, which implement large size-grids and thus exhaust the capabilities of modern ultra-deep sub-micron technologies, is extremely expensive, economic arguments eventually impose the restriction of producing the cheaper MRF devices with smaller site-grids and less chip area.

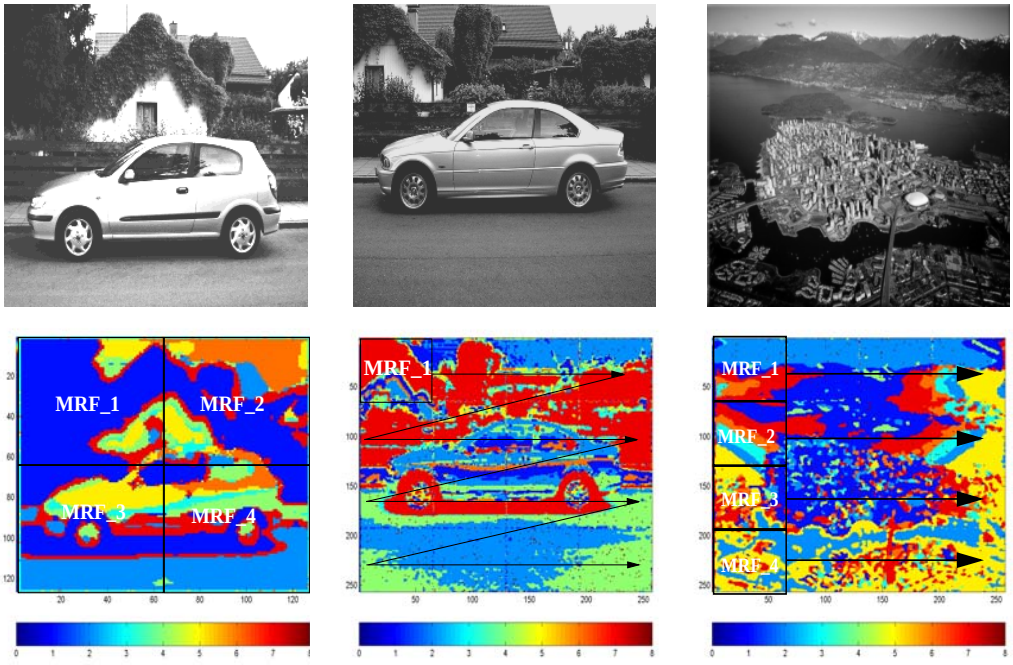
Primarily three different variants of system-settings with MRF devices can be distinguished, if image sizes have to be processed that are larger than the implemented Markov Random Field site-grid itself. These variants are completely covered by the previously introduced system-architecture template and will be described and discussed in the following section. The three MRF system-setting variants identified, are:

- Several physically identical MRF realizations, in which each specific MRF site-grid implementation is smaller than the overall image-size, are arranged in such a way that the complete image-size is covered by the site-grids. However, each MRF site-grid only processes a distinct and fixed part of the image at a time. Figure 3.20a schematically illustrates this system-setting.

- One physical MRF realization alone, where this MRF site-grid implementation

is smaller than the overall image-size, serially processes the whole image. Each time exactly one part of the image is processed, which equals the size of the site-grid. This system-setting is schematically shown in Figure 3.20b.

- This system-setting represents a combination of the previously described two settings. In this system-setting, where each of the MRF site-grid implementations is significantly smaller than the input image itself, several physical MRF realizations process the complete image in a serial manner. During the processing sequence each of the MRF site-grids at first work in parallel on one part of the image in one single step, which equals its site-grid size, and as a second step switches to a different part of the image. This scheme continues until the complete image is processed. Figure 3.20c schematically depicts this specific system-setting.



(a) (b) (c)

Figure 3.20: Illustration of three system-setting variants with MRF devices covered by the proposed architecture template. (a) Several MRFs cover the complete image and process the image data in parallel. (b) A single MRF device serially processes the complete image. (c) Several MRFs process different image parts in parallel and serially the complete image.

Regarding the first variant, Figure 3.20a illustrates an exemplary system-setting with exactly four realizations of a Markov Random Field site-grid working in parallel on the image. In this setting each MRF device processes exactly one quadrant of the image in total. If it is assumed that the total image size is $n \times m$ pixels and each MRF works on a not sub-sampled pixel set, then the site-grid of each MRF is

obviously $n/2 \times m/2$ large. From a practical point of view, any setting with two or four MRF devices represents the technically and economically most profitable configuration. System configurations with even six to nine distinct MRF devices - under certain circumstances - may be conceivable and justifiable; for instance as technology prototypes. However, any configuration with more than nine distinct MRF devices becomes easily technically impractical and fault-prone.

This statement is substantiated by the following facts: First of all, the MRF devices have to be arranged and mounted either on one printed circuit board or on several printed circuit boards, which need to be connected to form a complete subsystem. Technical constraints limit the integration capabilities of printed circuit boards and thus the absolute number of devices per printed circuit board area. Several long connections on or between the boards significantly slow down the overall data throughput and consequently the performance of the system, which directly affects the real-time processing capabilities, which are in any event difficult to fulfill. Furthermore, systems with more than nine distinct MRF devices physically become too large to be of practical relevance for industrial applications or products. It would also contradict the central claim of this thesis to make physically compact image- and signal processing systems possible, which are based on Markov Random Fields. The only consequent step would be to increase the overall site number and integration density per MRF device in order to limit the number of distinct MRF devices to a count of two to six devices in this system-setting variant.

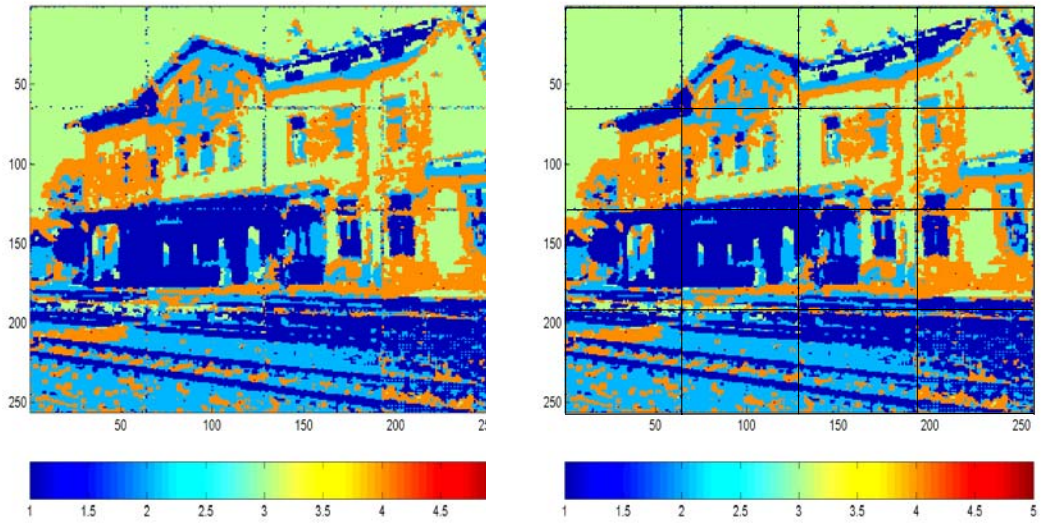
In addition to this, merely one MRF device setting of this variant makes it necessary to use memory-types within the device's memory hierarchy (cf. mapping 3.18 and corresponding section), which can simultaneously be written and read to capture the continuous flow of incoming data without latching the image data. In this specific constellation, exactly two MRF devices process their corresponding two image parts, which are horizontally aligned. By means of this constellation the data is written onto the first levels of the memory hierarchy of each MRF device, whereby each image-data row is split into two parts; the first half of each image-data row is transfered to the first MRF device and the second half of the data row to the second MRF device. The data on the first level of the memory hierarchy can be passed over to the next level as soon as the last image-data row is completely transfered. But following shortly after the last image-data row, the first image-data row of the next image appears and has to be stored onto the first level of the memory hierarchy. Thus we mandatorily need memory-types, which support simultaneous read- and write-actions on the same memory-cell and furthermore guarantee data-consistence and data-correctness. Any other constellation of this variant will not require these special memory types within the memory hierarchy, because of the image being vertically split. This results in assigning the first and last image-data row to different MRF devices, equalizes the data transfer within the memory hierarchy and allows the read- and write-actions to happen sequentially.

The second variant of a system-setting with exactly one MRF device, in which the site-grid of the implemented MRF device is once again smaller than the image itself, is exemplarily depicted in Figure 3.20b. In this specific configuration only one MRF device sequentially processes the complete image. The illustration of Figure 3.20b feigns the image data to be completely available in one single time step; however, this is definitely not the case in technical systems. The detailed flow of

data looks slightly differently there. Normally image data or, - more generally -, any kind of signal data is transmitted in a continuous row- or column-serial flow by the corresponding sensor device. Consequently, it is required to latch that part of the image data, which can not be immediately processed by the MRF device. This fact leads to the requirement that the single MRF device of this system setting variant mandatorily has to finish the calculations on one image before the next image-data arrives. Consequently it is guaranteed that the incoming data does not overrun the previously received data, which is currently being processed by the MRF device. Should the MRF device not be capable of processing the data with the incoming frequency, it would in principle be possible to drop as many complete images as required, to allow the MRF device to process one complete image. Obviously, this procedure of dropping data decreases the number of images the system can process in each second and is in addition to this not usable for application scenarios, where the complete image-sequence dynamic is required.

The third variant of a system-setting is a combination of the two previously described variants in terms of the first variant having several distinct MRF realizations. Their site-grid is smaller than the image itself and works in parallel on different parts of the image and with regard to the second variant sequentially processes the complete image by the different MRF devices. Consequently, this system-setting variant represents a trade-off between parallel and serial processing with several distinct MRF realizations. Exactly this system-setting is schematically and exemplarily depicted in Figure 3.20c by a setting with exactly four MRF site-grid devices. In this system setting Figure 3.20c feigns the complete image data to be concurrently available too, which is definitely not the case as the data normally arrives in a continuous row- or column-serial manner, when leaving the specific sensor device. Thus, the data which can not be directly processed by the MRF devices, has to be latched at first and later on transmitted to the MRF devices for processing. The arguments and conclusions presented in the discussion about the first system-setting variant regarding the overall number of distinct MRF devices in a technical system realization are also valid for this third system-setting variant. Additionally, the arguments and conclusions of the second system-setting variant regarding the frequency of the incoming data are also valid for this third variant, which is, as already mentioned, a combination of the previously introduced two system-setting variants.

For all three system-setting variants afore presented one specific peculiarity, which is caused by the fact that distinct physical MRF devices serially process the image-data, should be preconceived and adequately handled. Every realization of a MRF site-grid with a specific grid-size is essentially composed of the sites, which conduct the processing, and the neighborhood system, which establishes the connections between the MRF sites. But the neighborhood system is not complete for sites directly located at the boarder of the grid or sites beyond the boarder, which depends on the order of the used neighborhood system. These grid boarder-sites have definitely not the complete set of neighboring sites and thus also not a complete set of values for the calculation. In order to provide a principally complete set of values for the sites with an incomplete neighborhood system, in technical MRF realizations the missing values are provided by means of a fixed and hard-wired value. From a VLSI point of view it is advantageous - with respect to chip-area and wiring - to fix the missing values either to logical "one" or to logical "zero". Obviously, this



(a) (b)

Figure 3.21: Example of segmentation-result without overlapping boarders of the MRF devices. (a) Original segmentation result with 2nd order neighborhood system MRF (size 64×64) and probabilistic unsupervised segmentation model of Section 2.6.2 (b) Overlapping border structure generated by 64×64 MRFs.

procedure affects the outcome of the calculation on these sites, which is exemplarily illustrated in Figure 3.21a-b. The processing results of these specific boarder sites are not coherent with the results of sites, which possess a complete set of neighboring sites defined by the neighborhood system of the MRF site-grid.

The previous discussion explains the checkerboard-like-patterns respectively defects of the result image shown in Figure 3.21a. Figure 3.21b shows the result image with the overlapped MRF-device boarders, where defects can occur. For each of the three system-setting variants the distinct MRF devices process just a specific part of the image at one time, and due to the incomplete neighborhood system sometimes generates erroneous results on the grid-boarder. Exactly this peculiarity of the introduced system-setting variants can easily be handled and the defects respectively errors on the grid-boarder can be removed. The processed image parts merely have to overlap in order to allow only those results, which are generated by sites with a complete neighborhood system, to be stored. Of course, this results in some image data being processed several times, more precisely the data on those sites with an incomplete neighborhood system. This processing overhead is not vital and has to be accepted, if one of the three system-setting variants without site-grid boarder defects are used.

If one of the three system-setting variants are realized, the estimation of the model's free parameters for a complete image is a different issue. For the first variant the parameter estimation looks relatively simple. Each MRF device estimates the parameters with respect to the image part the specific MRF device is assigned

to. Since the complete image is covered by MRF devices, we just have to combine the parameter estimation results of each MRF to specify the model's final parameter set. The final parameter set is then transferred to each of the MRF devices to allow the the device to conduct the calculations of the implemented MRF model. For both system-setting variants two and three the situation with respect to the model's parameter estimation looks slightly different and is more involved. In principle there are two feasible approaches to organize the parameter estimation process for these two system-setting variants. The first proposed approach, which systematically organizes the parameter estimation task for the system-setting variants two and three uses exactly those parameters, which were determined by the part of the image the MRF devices started their processing with. For the second system-setting variant, where only one MRF device serially processes different parts of the complete image, the model's parameters are solely determined on the basis of partial data the MRF device starts its processing with. After this initial parameter determination process is finished, the parameters are fixed with regard to the rest of the image. This approach is also applicable for the third system-setting variant, with the difference that the parameter estimation results of the different distinct MRF devices - again solely calculated for the part of the image each MRF devices starts its processing with - are combined to form one single set of parameters. Again these parameters are fixed for the rest of the image. The second approach proposed, which defines the parameter estimation task for the second and third system-setting variant, essentially requires two complete processing passes over the image data to complete the calculation. During the first pass the parameters as well as the processing is executed. The different parameter sets are combined to form one single set, which is available after the first processing pass over the image data is finished. Exactly this parameter set is fixed for the second pass. During the second pass over the complete image data only the processing with the parameters of the first pass is done without changing respectively re-estimating the model's parameters.

3.7 Exemplary Model's Architectures

With the architecture template introduced in this chapter and its different building blocks for topology & structure $\mathcal{BB}^{TS}$, control functionality $\mathcal{BB}^{CT}$ and processing functionality $\mathcal{BB}^{PC}$ at hand, it becomes a well-defined and straightforward procedure to derive the concrete hardware relevant and massively parallel processing architectures for the two exemplary image processing models presented in Section 2.6.1 and Section 2.6.2. Both massively parallel system-architectures, the concrete characteristics of the building blocks as well as its possible architectural variants, are described in detail in the following subsections, to define a complete building plan for the specific architectures of the two models.

3.7.1 Edge Preserving & Noise Removing

The compiling of the specific massively parallel hardware architecture for the edge preserving and noise removing MRF model (for model-details see Section 2.6.1) begins with the topology defining site-hull gantry and thus with the set $\mathcal{BB}^{TS}$, which comprises the required topology & structure defining building blocks = S^{Hulls}, $W^{\mathcal{N}}$,

M^{Ports}, M^{GMem}. This specific massively parallel hardware architecture consists of $|\Omega|$ site hulls $S^{Hulls} = \{s_i^{Hull} : 1 \leq i \leq |\Omega|\}$. These particular site hulls s_i^{Hull} are arranged in the plane as a regular and equally spaced site-hull grid (cf. Figure 2.1). This topology setting establishes the first part of the architecture gantry and at the same time the overall shape of the chip floor-plane, because all other building blocks are embedded into this site-hull gantry structure. By means of the neighborhood wiring $W_{\mathcal{N}}$ all site hulls s_i^{Hull} are connected among each other in a specific and model-dependent kind.

For the *Edge Preserving & Noise Removing* model a first order $\mathcal{N}^1$ neighborhood system is sufficient, because the intensity change grading function κ operates on the four direct neighbors of each site hull (see Section 2.6.1 and Eq. 2.6.1), i.e. the values of the right and left as well as the top and bottom neighbors with respect to s_i^{Hull} are needed to determine κ. Within this architecture all neighborhood wiring blocks $w_i^{<i,t>}$ are characterized by their maximum bus-width of 8-bit, as it has been assumed for this model that it processes gray-valued images with a gray value range of $[0, ..., 255]$. Beside the 8-bit bus-width variant of the wiring blocks, there are other architectural variants with a 1-bit, 2-bit or 4-bit width busses, in order to transfer the 8-bit datum in 8-steps, 4-steps or respectively 2-steps. These variants on the one hand significantly reduce the risk of wiring congestion during the chip routing process but on the other hand introduce additional steps to transfer the data. Depending on the processing time requirements and the used implementation technology, one variant has to be chosen.

The next category of building blocks are the port memory blocks M^{Ports} each having a size of 8-bit to store the incoming data of the neighbors. Either the 8-bit datum is stored in one step, if the wiring blocks have 8-bit width buses, or the 8-bit datum becomes transfered in several steps and successively stored within the corresponding memory places of the 8-bit register bank. The global memory hierarchy building block M^{GMem} completes the topology of the model's architecture. In view of VLSI implementations and the demand to use regular structures, it is advantageous to realize the building block M^{GMem} as a completely balanced quad-tree, i.e. at the top of the quad-tree there is merely an interface to an external memory, which represents the root of the tree. At first the first level of the quad-tree is incorporated into the topology and thus into the VLSI implementation. This kind of organization saves important chip resources, especially for large MRF site-grids. All memory elements of the quad-tree have to store the corresponding input- and output values as well as additionally the single free parameter $\mathbf{\Theta} = \{\sigma^2\}$ and the user defined value γ of the intensity change grading function κ.

The memory hierarchy (cf. Figure 3.8) is separated into two distinct paths: One down-path, i.e. the input values are transferred to the sites, and one up-path, which transfers the output-results of the sites out of the hardware-structure. This makes it necessary to regard both paths separately. The bus-width of the memory's down-path is 8-bit in order to transfer the assumed gray scaled pixel values (range $[0, ..., 255]$) to the particular sites. For the free parameter σ^2 and the user defined value γ a bit-length-representation can be chosen, which is a multiple of the 8-bit memory hierarchy bus-width. Even 16-bit are mostly sufficient for σ^2 and γ and 32-bit are definitely sufficient for the just discussed model. The up-path of the memory

$\mathcal{BB}$ **Types**	**MRF site-grid size**			
(number of)	64x64	128x128	256x256	512x512
s_i^{Hull}	4096	16384	65536	262144
$w_i^{<i,t>}$	16384	65536	262144	1048576
wires $w_i^{<i,t>_8}$	131072	524288	2097152	8388608
wires $w_i^{<i,t>_4}$	65536	262144	1048576	4194304
wires $w_i^{<i,t>_2}$	32768	131072	524288	2097152
$m_i^{Port<i,t>}$	16384	65536	262144	1048576
bits $m_i^{Port<i,t>}$	131072	524288	2097152	8388608
kByte $m_i^{Port<i,t>}$	16	64	256	1024
M^{GMem}	1	1	1	1
$\mathcal{H}_i^{EF}$	4096	16384	65536	262144
opt_i	4096	16384	65536	262144
PC^{PAR}	341	1365	5461	21845
CT^{System}	1	1	1	1
CT^{GMem}	1	1	1	1
$ctrl_i^{PortMem}$	4096	16384	65536	262144
$ctrl_i^{EF}$	4096	16384	65536	262144
$ctrl_i^{OPT}$	4096	16384	65536	262144

Table 3.4: Required architectural building block resources for the *Edge Preserving & Noise Removing* MRF model with $\mathcal{N}^1$. The number of building blocks for each type as well as the detailed number of wires and bits for particular building blocks are given. Four different site-grid sizes have exemplary be used for the calculations.

hierarchy is 8-bit wide to transfer the results, which are the restored gray-values in the range $[0, ..., 255]$, from the particular sites and out of the hardware architecture. This finalizes the architecture topology of the edge preserving and noise removing MRF model.

All building blocks $\mathcal{BB}^{PC} = \{PC^{EF}, PC^{OPT}, PC^{PAR}\}$, which represent any kind of processing functionality within the model's architecture are incorporated into the so-far-established topology. The energy-functional building block is for all sites of the site grid Ω functionally identical. Hence, Eq. 2.27 has to be implemented in hardware. The mathematical operations are, despite the division-operators, resources efficiently implementable in VLSI structures. A 16-32bit number representation (cf. discussion Section 4.4) is sufficient for all results, signals and variables. If the architecture will use the ICM-optimization method we can prevent to use the exp function. Should the architecture have to use a SA-optimization method then we have to realize exp and the temperature $1/T$ schedule. The operator exp is realized as look-up table with a piecewise linearization, whereas the temperature

schedule is realized by a simple shift operation. Obviously, this will not mimic the logarithmic temperature decrease dictated by theory, but this approach is a good compromise in view of hardware-area consumption. Again 16-32bit are sufficient for all results, signals and variables. Finally, the processing block for the parameter estimation task has to be incorporated into the architecture. Principally there are two possibilities: Either as a single block at the root of the memory hierarchy or as a distributed structure along the memory hierarchy. The calculation of the free parameter is given by Eq. 2.32. It is a summing followed by a division operation; should the term $1/n$ be a power of two, the division simply translates in hardware to a shift-operation. Thus also the processing building blocks have been incorporated into the topology. In order to finalize the architecture the control-functionality representing blocks have to be added.

The set $\mathcal{BB}^{CT} = \{CT^{System}, CT^{GMem}, CT^{PortMem}, CT^{EF}, CT^{OPT}\}$ of control units is finally added to the hardware-relevant architecture in order to complete it. At first an overall control unit, realized as time discrete finite automaton, is added to the architecture. This control unit guides the MRF processing device through the repeated phases of (1) data-transfer, (2) processing and (3) parameter estimation. The control unit CT^{GMem} of the memory hierarchy is triggered by the system control unit and signals back to the system control when this phase is finished. In case the wiring blocks possess a smaller bus-width as the port memory, the data has to be transfered in several steps, which will be controlled by the port-memory control unit $CT^{PortMem}$. This module is realized as a finite automaton. The control block for the energy-functional is realized $|\Omega|$-times and located in each site hull in close conjunction with the energy-functional data-path. This unit controls the flow of data and results within the energy-functional data-path. The optimize control unit block is also realized $|\Omega|$-times and located within each site hull in close conjunction with its optimize data-path. Finally the parameter estimation control unit is either realized as a single unit at the root of the memory hierarchy, if the parameter estimation is completely done at the root-node or realized several times, if the parameter estimation becomes executed along the memory hierarchy. With this we have completed the more detailed description of the massively parallel architecture for the *Edge Preserving & Noise Removing* model presented in Section 2.6.1.

For the massively parallel architecture of the *Edge Preserving & Noise removing* model Table 3.4 shows the number of building blocks $\mathcal{BB}$, which the different grid-sizes will have. This demonstrates the large number of elements, which are required to compile a massively parallel architecture for this specific model. The simulation framework (Chapter 4), as well as the VLSI design framework (Chapter 5), has to adequately cope with this complexity of the system architecture and the complexity of the particular elements.

3.7.2 Unsupervised Histogram Segmentation

The derivation procedure of the concrete massively parallel hardware architecture for the unsupervised histogram segmentation model (for details see Section 2.6.2) starts with the set $\mathcal{BB}^{TS} = \{S^{Hulls}, W^{\mathcal{N}}, M^{Ports}, M^{GMem}\}$ made up of topology & structure defining building blocks. The massively parallel hardware architecture of

the segmentation model consists of $|\Omega|$ site hulls $S^{Hulls} = \{s_i^{Hull} : 1 \leq i \leq |\Omega|\}$. These particular site hulls s_i^{Hulls} are arranged as a regular grid in the plane (cf. Figure 2.1), which establishes the first part of the architecture gantry and, at the same time, a coarse shape of the chip floor-plane. All sites are connected among each other by means of the neighborhood wiring $W^{\mathcal{N}}$. In the case of the segmentation model and with respect to the given constraints of the contemplated MRF class, the neighborhood wiring can be realized to implement neighborhood systems ranging from the first order $\mathcal{N}^1$ up to the maximum fifth order $\mathcal{N}^5$. The segmentation model operates for all these neighborhood systems, but larger neighborhood systems will tend to lead to more homogeneous segments (see e.g. Figure A.5). Each particular neighborhood wiring block $w_{\cdot}^{<\cdot,t>_8}$ is further characterized by its bus-width of 8-bit, as it has been defined to segment gray-scaled images with a gray value range of $[0, ..., 255]$. Additionally useful architectural variants of the wiring blocks are realizations with a 1-bit, 2-bit or 4-bit width bus, in order to transfer the 8-bit datum in 8-steps, 4-steps or 2-steps, respectively. The next building blocks are the port memory blocks with each having a size 8-bit to store the incoming data of the neighbors. Finally the global memory hierarchy block M^{GMem} completes the topology of the segmentation model's architecture. This block is realized as a completely balanced quad-tree, which is organized to render possible a memory element with the size of the image plus the number of parameter values to be located at the quad-tree root. Furthermore it results in the memory elements, which have the size to store values of four sites plus the parameters on the leaves. The bus-width of the memory's down-path is 8-bit in order to transfer the assumed gray scaled pixel values (range $[0, ..., 255]$) respectively 24bit to transfer the values (range $[0, ..., 255]$ per channel) of the RGB channels as well as the parameters p_ν and q_ν, which are represented by 8-bit values, too. In contrast to this the up-path of the memory hierarchy is up to 3-bit wide to encode eight different classes plus 8-bit respectively 24bit for the pixel values, which are required to estimate the q_ν parameters (cf. update Eq. 2.53). This setting is not influenced by the fact if the parameter estimation is executed along the memory hierarchy or if it is done completely on the root of the memory hierarchy (see Figure 3.8).

The set of processing building blocks $\mathcal{BB}^{PC} = \{PC^{EF}, PC^{OPT}, PC^{PAR}\}$ is incorporated into the so-far-established topology of the segmentation model's architecture. Especially with regard to the unsupervised segmentation model, the processing building blocks PC^{EF} and PC^{OPT} are combined and represented by Eq. 2.47 and Eq. 2.48. These equations contain the operators log and exp, which are resource-intensive with respect to VLSI implementations. For this reason we suggest to realize the operators log and exp as look-up tables with a piecewise linearization as suggested by Vassiliadis et al. [143]. It is also suggested to realize the expression $1/T$ as a look-up table or alternatively by a shift-operation, which will not exactly implement $1/T$. The bit-width in this combined data-path varies from 16-bit to 32-bit. Each site hull possesses its own combined PC^{EF} and PC^{OPT} processing block. Finally the processing block for the parameter estimation has to be incorporated into the architecture. There are two possibilities: Either as a single block at the root of the memory hierarchy or as a distributed structure along the memory hierarchy. The calculations are given by Eq. 2.51 and Eq. 2.53. For the p parameter it is a simple summing followed by a division operation. If the term $1/n$ is a power of

$\mathcal{BB}$ **Types**	**MRF site-grid size**			
(number of)	64x64	128x128	256x256	512x512
s_i^{Hull}	4096	16384	65536	262144
$w_i^{<i,t>}$	98304	393216	1572864	6291456
wires $w_i^{<i,t>_8}$	786432	3145728	12582912	50331648
wires $w_i^{<i,t>_4}$	393216	1572864	6291456	25165824
wires $w_i^{<i,t>_2}$	196608	786432	3145728	12582912
$m_i^{Port<i,t>}$	98304	393216	1572864	6291456
bits $m_i^{Port<i,t>}$	786432	3145728	12582912	50331648
kByte $m_i^{Port<i,t>}$	96	384	1536	6144
M^{GMem}	1	1	1	1
$\mathcal{H}_i^{EF}$	4096	16384	65536	262144
opt_i	4096	16384	65536	262144
PC^{PAR}	341	1365	5461	21845
CT^{System}	1	1	1	1
CT^{GMem}	1	1	1	1
$ctrl_i^{PortMem}$	4096	16384	65536	262144
$ctrl_i^{EF}$	4096	16384	65536	262144
$ctrl_i^{OPT}$	4096	16384	65536	262144

Table 3.5: Required architectural building block resources for the *Unsupervised Segmentation* MRF model with $\mathcal{N}^5$. The number of building blocks for each type as well as the detailed number of wires and bits for particular building blocks are given. Four different site-grid sizes have exemplary be used for the calculations.

two, the division operation simply translates in hardware to a shift-operation. The operations for the parameter q are more involved and with respect to the division operator rather VLSI hardware resources-intensive. Again the bit-width varies from 16-bit to 32-bit.

The set $\mathcal{BB}^{CT} = \{CT^{System}, CT^{GMem}, CT^{PortMem}, CT^{EF}, CT^{OPT}\}$ of control units is finally incorporated into the architecture to complete it. At first an overall control unit, realized as time-discrete finite automaton, is added to the architecture. This unit mainly guides the MRF processing device through the repeated phases of data-transfer, processing and parameter estimation. The control unit of the memory hierarchy is triggered by the system control unit and signals back to the system control when this phase is finished. If the wiring blocks possess a smaller bus-width as the port memory, then the data has to be transfered in several steps, which will be controlled by the port-memory control unit. This module becomes realized as a finite automaton. The energy-functional control block is realized $|\Omega|$-times and located in each site hull together with the energy-functional data-path. Finally

the optimization control unit is either realized as a single unit, if the parameter estimation is finished, at the root of the memory hierarchy or realized several times, if the parameter estimation is executed along the memory hierarchy. Consequently we have completed the more detailed description of the massively parallel architecture for the unsupervised segmentation model presented in Section 2.6.2.

Table 3.5 summarizes the number of building blocks $\mathcal{BB}$, which are required for different grid-sizes of the massively parallel architecture. The particular table values refer to the *Unsupervised Segmentation* model with a fifth order neighborhood system $\mathcal{N}^5$. This table clearly illustrates the tremendous number of building block elements, which are required to compile a massively parallel architecture for the unsupervised segmentation model. Concurrently the table values demonstrate that larger neighborhood system become quite difficult to realize in semiconductor technologies, due to the number of wires and port memories. The simulation framework (Chapter 4) as well as the VLSI design framework (Chapter 5) have to adequately cope with this overall complexity of the system architecture and the complexity of the particular elements.

3.8 Summary

In this chapter we have introduced a novel massively parallel system-architecture template for a large class of statistical image processing models - including the contemplated MRF class of Section 2.5 and Definition 2.10 -, which are formulated on regular Markovian site grids with a spatially limited neighborhood support. The Markov Random Field system-architecture template has been conceptualized and specifically adjusted to simultaneously fulfill two constraints. At first the constraint of *massively parallel processing* in order to put real-time processing capabilities of MRFs into practice and secondly the constraint of *semiconductor technology independent large-scale integration* to realize physically compact electronic MRF processing systems in different purely digital semiconductor design technologies.

The complete derivation and definition process of the system-architecture template was exclusively achieved within the theoretical scope of Markov Random Field theory. At first we have therefore extracted from and within the scope of MRF-theory *universal constituents*, which are characterized by the feature that they are architecturally uncommitted, i.e. the universal constituents are free from inherent implementation and VLSI assumptions. Exactly this architecturally uncommitted originator for the derivation and definition of the system-architecture template has been described in Section 3.1.

Based on the universal constituents in Section 3.2 we have systematically and formally derived the different building blocks of the system-architecture template with respect to our two driving-on constraints of *parallel processing* and *large scale integration*. The building blocks were systematically categorized either as *topology & structure* building blocks, *processing* building blocks or as *control* building blocks in order to structure and guide the compilation process of concrete MRF architectures and also to simplify large-scale integration issues. To finalize the description of each building block, we have added a discussion (see Section 3.3) on VLSI issues and strategies to adequately deal with them.

In Section 3.4 we have introduced a cycle scheme template for the system-architecture template, which defines the sequence of operations the MRF architecture has to pass through to complete a calculation run. The cycle scheme template represents a valid but not finally optimized time-discrete cycle scheme.

Section 3.5 has presented the first part of the thesis part relation, which formally establishes the links between the different thesis parts to each other and especially the connections to the fundamentals of Markov Random Field theory. The relation represents the condensed content of this thesis.

An essential topic of massively parallel Markov Random Field VLSI hardware architectures has been discussed in Section 3.6. In this section we have systematically analyzed three system-settings, where image sizes larger than the site-grid of the MRF architecture have to be processed. All three system settings can be covered by massively parallel MRF architectures composed of the system-architecture template.

In the last Section 3.7 we have successfully used our proposed system-architecture template in order to define the massively parallel architectures of the two concrete and industrially relevant MRF image-processing models (cf. Section 2.6). These results underpin the features and the applicability of the proposed system-architecture template for the contemplated class of Markov Random Field based image- and signal processing systems. Additional results in terms of hardware-relevant simulations (cf. Section 4.4) and prototypical VLSI implementations (cf. Section 5.5) support the features of the proposed system-architecture template.

3.9 Bibliographical Comments

Due to the novelty of the introduced massively parallel system-architecture template and the applied derivation approach - the rigorous derivation of building blocks from the architecturally uncommitted universal-constituents, which by their own have been defined based on the fundamentals of MRF theory as well as the focus on purely digital semiconductor design technologies - no directly related additional literature is currently available. Massively parallel hardware architectures, which are based on the processing principles of MRFs or CNNs are almost exclusively proposed in publications of the CNN or Neural Network community [75] [76] [77] [78] [45]. The CNN community investigates a special architectural setting, in which the sensing elements and the processing structures form an inseparable and spatially closely arranged unit [74] [79], realized in analog design technologies. By far the largest number of proposed architectures is exclusively defined for one image processing model, one specific analog technology and one system-environment [131] [132] [124]. As to the authors best knowledge, until now no systematically derivation of an system-architecture template for purely digital semiconductor implementation technologies has been discussed within contemporary literature.

Chapter 4

MRF Simulation Framework

One of the very first steps toward a VLSI realization of massively parallel MRF devices, which are based on the introduced architecture template (Chapter 3), is simulating the complete hardware architecture. Only detailed simulations offer the option to systematically study architecturally, timing and numerically caused effects, which otherwise can not be determined analytically. Without hardware-relevant simulations of the complete architecture unpredictable error-sources would remain, which would never justify a costly VLSI implementation of a MRF device. The mask costs of modern semiconductor-technologies alone are in the range of several million dollars. But up to now neither a systematic approach nor a flexible simulation environment is available to allow for hardware-relevant simulations of complete MRF devices.

In this chapter we introduce a novel simulation framework for massively parallel Markov Random Field based processing devices, which is specifically tuned to support the distinct hardware-relevant simulation requirements these MRF architectures pose. Our proposed simulation framework

- offers the unique option to systematically study MRF models with respect to the combination of numerical, architectural, timing and massively parallel processing constraints and thus to receive novel insights into MRF models and their hardware architectures.

- overcomes the existing hardware-relevant simulation deficit of MRF architectures and thus paves the way for massively parallel VLSI implementations of MRFs.

The reasons for the hardware-relevant simulation deficit of MRFs are diverse. It is state-of-the-art to model, simulate and synthesize digital hardware with one of the IEEE-standardized hardware description languages VHDL or Verilog. But both languages suffer from the profound limitation that they are not intended to simulate complete systems. Rather these hardware description languages are mainly dedicated to logic simulation of particular modules; a fact that also has not been changed with recently revisions of the standards. Additionally, the simulation-run times are tremendous and quickly reach the order of magnitude of days and even weeks. For large distributed systems, as represented by massively parallel MRF

architectures, already the simulation set-up fails with a fatal error due to the overall complexity. In summary VHDL and Verilog are not suitable to offer simulation capabilities for complete massively parallel MRF architectures of an application relevant size. Simulation environments [128] [127] for large distributed systems are almost exclusively focused and tuned on communication networks and thus are not suitable for hardware relevant simulation settings. Likewise specialized image processing environments fail in supporting distributed processing features as well as hardware relevant features right from the outset.

An appropriate hardware-relevant simulation and modeling environment for massively parallel MRF architectures has to provide the following key-capabilities:

1. complexity handling of complete MRF architectures, owning a realistic MRF grid size,

2. reproduction of massively parallel processing dynamics,

3. systematic support for architecture modeling with a minimum of error-prone hand coding.

Our proposed MRF simulation framework combines the previously mentioned requirements, which are all necessary to simulate and investigate our regarded class of MRF models and their architectures within one single environment. The required simulation capabilities are put into practice by means of reusable and parameterizable simulation modules, disposed with respect to similar functionality and collected within different module libraries. A specific MRF simulation model becomes realized by the *Simulation-Model Generator* module, which automatically compiles the simulation model based on user-defined specifications. The automated generation process of simulation models is supported by inherent self-generating and referencing capabilities of the library modules. In order to prevent a time-consuming and error-prone development of an event-driven simulation-kernel from scratch, we decided to build up our framework on the SystemC [81] class hierarchy, its event-driven simulation kernel and its support for hardware-relevant modeling.

The structure and arrangement of the proposed MRF simulation framework will be introduced in Section 4.1 directly following. Furthermore, this introductory section will provide a short synopsis of the most important framework parts. In the subsequent Section 4.2 the different parts of the MRF simulation framework will be exhaustively explained and illustrated by the corresponding simulation prototypes, which cover the common structure and functionality of the particular simulation modules. Section 4.3 presents the methodology to automatically construct respectively generate the simulation model of the intended MRF and the overall simulation environment. Section 4.4 exhaustively presents selected simulation results to underpin the capabilities of the proposed simulation framework. Additionally, this section summarizes the novel insight into the combination of MRF models and their architectures yielded by our proposed simulation framework. Section 4.5 comments on implementation issues of the MRF simulation framework version so far realized. Finally, Section 4.6 discusses the extension of the relation diagram.

4.1 Simulation Framework Overview

The novel hardware relevant simulation framework is conceptualized as a modular and open software system, built up on the event-driven SystemC [81] simulation kernel. Figure 4.1 diagrammatically depicts the overall structure of the novel simulation framework and its contributing components. All components of the proposed simulation framework are functionally complete standalone units and thus from the first neither linked to nor interrelated with each other. The procedure of linking, relating and parameterizing the different simulation framework components in order to form a coherent simulation-model for a specific MRF architecture is finally realized by the central *Simulation-Model Generator* module. Consequently, this

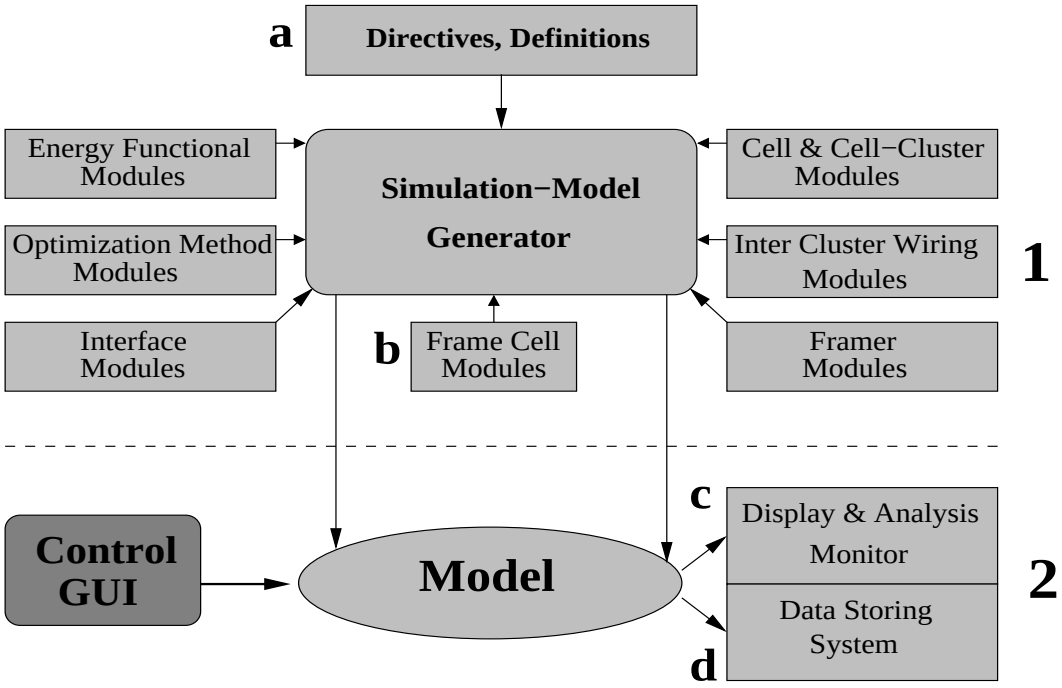


Figure 4.1: Arrangement and components of the proposed Simulation Framework for massively parallel Markov Random Field based image processing devices.

generator component represents a central instance within the simulation framework during the MRF simulation-model generation and set-up process. However, the central simulation-model generator does not affect or contribute to the simulation-run itself. Thus the overall task of simulating massively parallel hardware architectures for MRF based signal- and image processing models divides into two steps: In the first step the simulation-model of a specific MRF architecture is generated with respect to predefined specifications, whereas in the second step the generated simulation-model is executed. These two steps of a particular MRF simulation also have a correspondence within the proposed simulation framework illustrated by the dotted line in the schematic presentation of Figure 4.1, which separates the model's generation part from the simulating part.

Essentially, the two main steps and the limited number of additional sub-steps

execute the following actions, in order to generate and execute a MRF specific simulation model. Step **1** analyzes general directives and user-defined parameters before the central module, named *Simulation-Model Generator*, compiles the MRF specific simulation model. Step **2** executes the afore generated model, whereas the simulation-run is user-controlled by means of a GUI. In additional to that, any kind of data or provisional results of the model can be displayed or simultaneously stored for later off-line analysis. In order to complete step **1** and to produce a correct simulation-model, the user has to execute two sub-steps by hand. Firstly, in sub-step **a**, the user has to provide and adjust basic parameters, and secondly in sub-step **b** a special module has to be written to allow the automatic generation process of the simulation-model to be executed. To realize step **2** the user eventually has to modify some routines in sub-step **c** as well as sub-step **d** to display and store the relevant data of the specific simulation-run.

The modular and open character of the proposed simulation framework for massively parallel MRF-architectures implies vital advantages with respect to the simulation-model's generation process, the simulation itself and any further framework modifications respectively extensions. Adding new modules to the simulation framework or modifying existing modules is systematically supported by means of module-libraries, which comprise functionally similar modules. Furthermore, the separated simulation-model generation process significantly supports the extensibility of the framework, as module-libraries and final simulation-models are decoupled with the help of this arrangement. Additionally, the execution of the model itself represents a separate part of the simulation framework. Thus all three main parts of the simulation framework, the module-libraries, the simulation-model generation process and the simulation itself, do not form a monolithic and heavily interdependent unit, which is complicated to be modified or extended, but rather represents an open and upgradeable simulation environment with loosely connected parts. All these feature have been regarded during the conception-phase and the implementation of the simulation framework. Thus, the introduced simulation framework from the outset is not exclusively limited to the class of MRFs, which has been defined in Section 2.5, respectively Definition 2.10. Rather, the simulation framework is also usable for other classes of MRFs, if the necessary modifications were added to the framework.

Up to now a rich set of module-libraries has been implemented and integrated together with the model-generator module and the SystemC simulation kernel [81] in order to establish a functionally complete simulation framework carrying the above mentioned features and module arrangements. In the following we will shortly introduce and describe each single part or module of the simulation framework exactly in the chronological order, which is used during simulation set-up and execution. This concludes the overview of the proposed simulation framework for massively parallel architectures of MRF based signal- and image processing models. The set-up procedure and the features of self-generating simulation-models will be exhaustively explained and discussed in Section 4.3; right after the formal mapping of architectural building blocks to their simulation modules has been described. The modules each library can principally represent are illustrated by means of a prototype. Based on these prototypes we show in Section 4.2 that the framework is powerful enough to represent the derived MRF architecture-template for simulation.

Overview Framework Modules

In the following we will shortly introduce and describe the central features of each type of simulation module exactly in the chronological order, which is used during simulation set-up and execution. This discussion concludes the overview of the proposed simulation framework for massively parallel architectures of MRF based signal- and image processing models.

Additionally, this overview prepares for the discussion of Section 4.2 where the concept of simulation Prototypes is used to verify the powerfulness of the proposed framework to represent the architecture-template for simulation.

Cell & Cell-Cluster Modules

The collection of *Cell* and *Cell-Cluster* modules comprises differently sized simulation models, which represent the cell processing units within the MRF site-grid. The single cell models as well as the cell-cluster models are equipped with trigger ports, the ports of the neighborhood system and a mechanism to automatically incorporate inter-cluster wiring modules in order to wire the cell components of the cluster. Furthermore each particular simulation module possesses specific parts, which will be exclusively used during the generation respectively self-expansion (see Section 4.3.2) process of the complete MRF simulation model. One of these specific module parts realizes the automatic embedding of sub-cluster modules in the corresponding parent-module. Thus, larger cell-clusters automatically fill them with smaller cell-clusters until finally only particular cells will be embedded. Another module part, which will not be used during simulation, automatically connects the ports of the cell-cluster modules to the inter-cluster wiring module in order to establish a correct connection scheme. Due to the recursive self-expanding scheme, the numbering of the particular cells does not correspond to a standard column by row numbering. Thus it is not possible to uniquely address the top most left cell number 1 or the bottom most right cell number $n = |\Omega|$ in the MRF grid. Hence each cell-cluster possesses routines, which re-number the cells to form a standard column by row numbering.

Inter-Cluster Wiring Modules

The *Inter-Cluster Wiring* modules are a collection of parameterizable models, which wire particular cells or cell-clusters among each other. These modules form together with the cells and cell-clusters the topology gantry of MRF simulation models, which are covered by the MRF-class of Definition 2.10. Consequently, the wiring modules in principle support neighborhood systems of up to an order of five. Due to the tremendous usage of memory and processing resources of larger neighborhood systems, we often used fully-modeled second order neighborhood systems and pseudo-wirings with direct data access for third to fifth order wirings.

Frame-Cell Modules

The *Frame-Cell* modules represent supporting simulation structures, which encapsulate the complete cell-grid and ensure that cells with an incomplete neighborhood

system receive predefined signals at the defect neighborhood positions. Furthermore the encapsulation into the frame-cell generates a single object, which can easily be referenced to and addressed by other components, especially by the *Simulation-Model Generator* of the simulation framework.

Framer Modules

The *Framer* modules represent supporting structures within the simulation framework that collect all other modules of a specific MRF simulation model to combine them in one object. This ensures that all data-interfaces and interrelations of the different simulation modules are well-defined and that the expanding of the simulation model can systematically take place.

Energy Functional Modules

The block of *Energy Functional* modules comprises a collection of more elementary functions, which can be used by different MRF models as well as very specific functions, which are tuned for just one specific MRF model. Consequently, a complete energy functional of a specific MRF model can be compiled by these functions and, if necessary, additional functions can easily be added to this collection. Each component of this energy functional collection is seamless pluggable into the cell modules to form an operationally correct cell processing unit.

Optimization Method Modules

This library of simulation modules comprises different functionalities to form optimization methods, which have been described in Section 2.3. We have decided to realize sub-functionalities instead of complete optimization methods, since this approach offers a greater flexibility and adaptability without significantly affecting the integration effort of the user. Furthermore it is easier to realize flexible and adaptable sub-functionalities compared to a complete optimization method.

Interface Modules

This collection of modules within the simulation framework comprises blocks, that are required to establish standard data-interfaces with the control GUI as well as with the display & analysis monitor and the data storage system. Mechanisms to systematically incorporate the Matlab engine into the simulation framework are also be included.

Simulation-Model Generator

The *Simulation-Model Generator* block is the central unit within the simulation framework, which is also illustrated by its central emplacement in Figure 4.1. This specific block of the simulation framework establishes the connection between the other components and module libraries, which are from the outset independent from each other. Consequently, in a first step this block integrates the different module components and put them into relation with each other in order to form the condition for a coherent MRF simulation model. During an additional step the simulation-

model generator block triggers the expansion of the full MRF simulation model by executing the self-expanding features of the different modules.

Control GUI

The *Control GUI* of the proposed simulation framework represents a well-defined man-machine interface in order to guide the user through the different set-up steps and the actual simulation. Consecutive set-up steps are only unlocked for the user if the previous step was successfully and error-free executed. This strict user guidance prevents misleading simulation set-ups and simultaneously allows it to identify problems as early as possible during the simulation run. Consequently, operating errors, which otherwise might have falsified the simulation results, are systematically addressed by the control GUI. Obviously, a GUI also makes the overall handling more comfortable.

Display & Analysis Monitor

The block *Display & Analysis Monitor* comprises an interface and various mechanisms to visualize any kind of data the simulation models hold and generate during the actual simulation-run. Naturally, this block is closely linked to the *Data Storage System* to save these data for later off-line analysis. One of the elementary visualization tasks is displaying the input data, which has been transfered to the simulation model as well as displaying the results after each processing iteration. These elementary visual information is already sufficient to allow the experienced user to identify problems with respect to the model, the optimization schemes or the simulation framework. Additionally, the displayed internal signals and data of the model reveal further valuable insights with respect to the processing dynamics, numerical sensitivity and quantitative convergence properties.

Data Storage System

The *Data Storage System* is a collection of user-defined routines, which process the data and results of the simulation run. It is currently possible to save the raw data as well as different image formats. It has been shown that a Matlab interface and the integration of the complete Matlab engine is a comfortable and highly flexible solution for the data storage system and the processing of the generated data. But the Matlab engine approach has a clear drawback caused by its inferior processing speed and its tremendous memory-usage. Consequently, the complete simulation is slowed-down, in case Matlab is excessively used. Hence, it is advantageous to only store raw-data and have an off-line process after the complete simulation-run is finished.

Based on the architectural building blocks $\mathcal{BB}^{System}$, which have been derived in Section 3.2 and the previously described overall structure of the simulation framework, we will define in the following sections a set of simulation modules for the set of architectural building blocks $\mathcal{BB}^{System}$. Thus the simulation framework can both represent and simulate the massively parallel MRF architectures compilable by $\mathcal{BB}^{System}$ blocks. The types of simulation modules are organized along the organi-

zation scheme, which has been introduced in Chapter 3. Consequently, we discriminate between *topology & structure* representing simulation modules $\mathcal{S}^{TS}$, *processing* functionality representing simulation modules $\mathcal{S}^{PC}$ and finally *control* functionality representing simulation modules $\mathcal{S}^{CT}$.

This specific organization scheme systematically structures the compiling of MRF simulation models and simplifies the extension of the corresponding collections of simulation modules in order to realize and study different MRF model/architecture combinations. Furthermore the strict separation of the different types of simulation modules supports the formal verification that the simulation modules can represent the architectural building blocks for simulation.

4.2 Simulation Modules

Next to the different core simulation modules that comprise the processing cells, the cell wiring, the energy-functionals, the optimization methods the proposed simulation framework contains additional support-modules. All these different collections of simulation modules together form the operationally approved framework for massively parallel architectures, which are based on the principles of Markov Random Fields.

4.2.1 Supporting Modules

The support-modules are indispensable in order to coordinate the set-up process of the specific simulation model as well as the simulation-run itself. Without these supporting simulation modules it becomes difficult to form a flexible and coherent simulation environment for massively parallel and MRF-based image processing devices. The proposed simulation framework (cf. Figure 4.1) possesses the following collections of support-modules: (1) Interface Modules, (2) Framer Modules, (3) Frame-Cell Modules and the (4) Control GUI module. Each of these module collections has previously been described and shortly commented on regarding its functionality. We forgo to further detail the description of these supporting modules, because they do not represent any essential part of the proposed simulation framework.

4.2.2 Topology & Structure Simulation Modules

In the first category of the organization scheme simulation modules $\mathcal{S}^{TS}$ are summarized, which model topology & structure defining architectural building blocks $\mathcal{BB}^{TS}$. There does not exists a strict one-to-one mapping between the particular members of the two sets $\mathcal{S}^{TS}$ and $\mathcal{BB}^{TS}$. Consequently, several architectural building blocks from the set $\mathcal{BB}^{TS}$ are simultaneously modeled by one simulation module taken from the set $\mathcal{S}^{TS}$. That is the three topology & structure representing types of architectural building blocks, S^{Hulls}, M^{Ports} and M^{GMem} are modeled by simulation modules of $\mathbb{S}^{C}$. In contrast to that the wiring building blocks $W^{\mathcal{N}}$ are represented by distinct simulation modules of $\mathbb{S}^{W}$.

The set of topology & structure representing simulation modules will be denoted by $\mathcal{S}^{TS}$ in the sequel. This set comprises the sets of simulation modules $\mathbb{S}^{C}$ and $\mathbb{S}^{W}$,

and reads as follows:

Definition 4.1 (Topology & Structure Modules $\mathcal{S}^{TS}$)
The set $\mathcal{S}^{TS}$ of topology & structure representing simulation modules reads

$$\mathcal{S}^{TS} = \{\mathbb{S}^C, \mathbb{S}^W\} \tag{4.1}$$

with

- $\mathbb{S}^C = \{c_z^{\mathbb{S}}, cc_y^{\mathbb{S}} : |z|, |y| \gg 1\}$; $c_\cdot^{\mathbb{S}}$ denoting simulation modules of single cells and $cc_\cdot^{\mathbb{S}}$ denoting simulation modules of cell-cluster. [1]
- $\mathbb{S}^W = \{w_u^{\mathbb{S}} : |u| \gg 1\}$, representing the wiring simulation modules, which model the architectural building blocks $W^{\mathcal{N}}$.

The decision to model the building block types S^{Hulls}, M^{Ports} and M^{GMem} together is justified by the following unchangeable constraints and arguments: (1) The simulation module for the site hulls S^{Hulls} contains the ports of the neighborhood system in order to provide a well-defined communication to the outside as well as to the inside of the site hull. The signals on the ports can directly be taken and stored into port memory variables located inside the side hull. An additional encapsulation of the port memory variables into a distinct simulation module would generate ports - input- as well as output ports - for this module and wires between the ports of the site hull and the port memory model. All together the simulation module would become more complex without offering further capabilities to investigate the complete massively parallel MRF processing device. (2) The memory usage and the processing time for executing the simulation would increase by a separate modeling of S^{Hulls} and M^{Ports} without causing any advantage for the particular simulation as well as for the overall simulation framework. (3) Furthermore, the functional and hardware-relevant representation of a combined S^{Hulls} and M^{Ports} model is equal to that of a separated S^{Hulls} and M^{Ports} simulation model. (4) The globally distributed memory hierarchy block M^{GMem} also becomes incorporated into the simulation modules $\mathbb{S}^C$, because $\mathbb{S}^C$ contains agglomerations of cells into which the corresponding parts of M^{GMem} are seamlessly integrated. Again, this simplifies the structure of the simulation model, improves the overall memory- and processing-time-usage but does not affect the functional and hardware-relevant equivalence between a separated and combined model.

The set $\mathcal{S}^{TS}$ of simulation modules represents the most basic part of modules within the simulation framework, as this type of modules determine the fundamentals for compiling the architecture gantry of massively parallel MRF-based processing systems. All other simulation modules become plugged into such a simulation module later on. The pooling of topology-relevant building blocks in the two sets $\mathbb{S}^C$ and $\mathbb{S}^W$, as well as the pooling of several building blocks in one single type of simulation module (cf. Definition 4.1) systematically structures the compilation of larger MRF simulation models and interrelates the corresponding parts, which structurally belong together. This creates the fundamental conditions for the self-generating capabilities of the MRF models.

[1] The term *cell* is tightly associated with the terms *site* and s^{Hull}, but the particular objects are not identical. Hence the term *cell* was chosen to clearly separate the objects during the discussion.

The simulation prototypes, which will be introduced directly following are composed of three parts. At first a $STRUCTURE$ part, which defines the basic structural components. Secondly, a $INIT\_RULES$ part, which conducts all kinds of initialization and thirdly, a $FUNCTIONALITY$ part, which realizes various calculations and actions executed during the MRF model set-up procedure and during the simulation itself. All operational simulation modules of the framework are derived from their corresponding prototypes and hence possess an identical structure.

The prototype of one single cell simulation module as well as the prototype of a cell-cluster module, which will be proved to realize the architectural building blocks of S^{Hulls}, M^{Ports} and M^{GMem} are depicted in Prototype 4.1 and Prototype 4.2.

Topology & Structure Prototypes - Single Cell

The structure part of Prototype 4.1 for single cell modules comprises a list of $TRIGGER$ signals, which start and stop particular actions. Additionally, the structure part contains a $WATCH\_PORT$ list, which allows it to connect internal signals to these ports and observe them from outside. It further contains the COM_PORT

Simulation Prototype 4.1 Single Cell Modules

1: **BEGIN** *STRUCTURE*
2: TRIGGER$< a, b, ... >$
3: WATCH_PORT$< A, B, ... >$
4: COM_PORT$< \mathcal{N}^1, \mathcal{N}^2, \mathcal{N}^3, \mathcal{N}^4, \mathcal{N}^5 >$
5: PORT_MEM$< \mathcal{N}^1, \mathcal{N}^2, \mathcal{N}^3, \mathcal{N}^4, \mathcal{N}^5 >$
6: INTERNAL_VAR$< \alpha, \beta, ... >$
7: **END** *STRUCTURE*
8: ———————————
9: **BEGIN** *INIT_RULES*
10: CALL_MODULE$(CT_k^{\mathbb{S}})$
11: CONNECT$< (a, FUNC\bullet), (b, FUNC\bullet), ... >$ // $\bullet = 1, 2, ...$
12: INIT_VAR$< \alpha(Value), \beta(Value), ... >$
13: GENERATE_MODULE$< EF_{i,k}^{\mathbb{S}} >$
14: **END** *INIT_RULES*
15: ———————————
16: **BEGIN** *FUNCTIONALITY*
17: FUNC1: SyncReset_Behavior
18: FUNC2: AsyncReset_Behavior
19: FUNC3: StartInit_Behavior
20: FUNC4: CALL_MODULE$(EF_{i,k}^{\mathbb{S}})$
21: **END** *FUNCTIONALITY*

list, which defines the ports for the neighborhood wiring and the $PORT\_MEM$ list to store the incoming data from the neighboring sites. Finally the structure part of Prototype 4.1 possess an $INTERNAL\_VAR$ list for various support variables and constants. Within the initialization part ($INIT\_RULES$) the triggers are equipped with specific rules and conditions and in the following will be connected with functions of the $FUNCTIONALITY$ part. Thus, specific functional parts

are executed if the conditions of a trigger are fulfilled. Furthermore, the variables become assigned with their values in the initialize part. In the functionality part all routines and calculations are collected, which are executed during the simulation run as soon as the corresponding trigger conditions are fulfilled. The prototype contains several functionalities; some of them are generic and thus independent with respect to the specific MRF model under consideration. The functionalities for the *SyncReset_Behavior*, *AsyncReset_Behavior* and the *StartInit_Behavior* behavior are identical to all cell modules. These functions realize behavior, which takes place during a synchronous rest, during an asynchronous rest and during the start phase of the simulation. All together these behaviors guarantee that the simulation modules of this type are set in a well-defined and controlled overall simulation state. In contrast to the above mentioned functionalities the module call of the energy functional depends on the concrete MRF model and varies thus.

The following proposition establishes the formal guarantee that all simulation modules of single cells are qualified representations of specific architectural building blocks for simulation. We show that Prototype 4.1 for single cells is an admissible representation of particular building blocks $A \subset \mathcal{BB}^{TS}$. Hence it follows that all operational simulation modules of single cells, which are derive from this prototype are qualified representations of architectural building blocks for simulation, too.

Proposition 4.1 (Admissible $\subset \mathcal{BB}^{TS}$ Modeling)
The Prototype 4.1 for single cell modules is an admissible representation of building blocks $A \subset \mathcal{BB}^{TS}$ for simulation and hence all the operational simulation modules $c_z \in \mathbb{S}^C$ derived from it.

Proof: The proof is subdivided into two part. At first we show that Prototype 4.1 represents specific topology & structure defining building blocks and hence all the modules derived from it, i.e. Prototype 4.1 $\overset{\text{models}}{\longrightarrow} A \subset \mathcal{BB}^{TS} \Rightarrow \exists c_{\cdot}^{\mathbb{S}} \in \mathbb{S}^C : c_{\cdot}^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} A \subset \mathcal{BB}^{TS} \Rightarrow \forall c_{\cdot}^{\mathbb{S}} \in \mathbb{S}^C : c_{\cdot}^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} A \subset \mathcal{BB}^{TS}$. Secondly, we show that Prototype 4.1 did not model $a \notin \mathcal{BB}^{TS}$, which eventually change the represented architecture.
(1) The Prototype itself establishes a distinct and closed object with a COM_PORT structure for $\mathcal{N}^1 - \mathcal{N}^5$. Hence it is justified to say Prototype 4.1 $\overset{\text{models}}{\longrightarrow} s_{\cdot}^{Hull} \in S^{Hulls}$. For operational modules it follows $\exists c_{\cdot}^{\mathbb{S}} \in \mathbb{S}^C : c_{\cdot}^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} s_{\cdot}^{Hull} \in S^{Hulls} \subset \mathcal{BB}^{TS} \Rightarrow \forall c_{\cdot}^{\mathbb{S}} \in \mathbb{S}^C : c_{\cdot}^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} s_{\cdot}^{Hull} \in S^{Hulls} \subset \mathcal{BB}^{TS}$. Additionally, the structure $PORT\_MEM < ... >$ defines that Prototype 4.1 $\overset{\text{models}}{\longrightarrow} \{m_{\cdot}^{Port<\cdot,t>_x}\} \in M^{Ports}$. It follows $\exists c_{\cdot}^{\mathbb{S}} \in \mathbb{S}^C : c_{\cdot}^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} \{m_{\cdot}^{Port<\cdot,t>_x}\} \in W^{\mathcal{N}} \subset \mathcal{BB}^{TS} \Rightarrow \forall c_{\cdot}^{\mathbb{S}} \in \mathbb{S}^C : c_{\cdot}^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} \{m_{\cdot}^{Port<\cdot,t>_x}\} \in W^{\mathcal{N}} \subset \mathcal{BB}^{TS}$.
(2) Only the elements of the $STRUCTURE$ part contribute to the representation of the topology for simulation. It remains to clarify that $WATCH\_PORT \notin \mathcal{BB}^{TS} \wedge TRIGGER \notin \mathcal{BB}^{TS} \wedge INTERNAL\_VAR \notin \mathcal{BB}^{TS}$ do not change the represented architecture and thus the simulation outcome. $TRIGGER$ is a concept to guide the execution of specific functionalities and does not contribute to or change the represented structure of the architecture. $WATCH\_PORT$ and $INTERNAL\_VAR$ are passive structures, which do not alter or enhance the representation of $A \subset \mathcal{BB}^{TS}$ for simulation. Thus none of these elements add components $a \notin \mathcal{BB}^{TS}$.

Putting (1) and (2) together, we have: Prototype 4.1 $\overset{\text{models}}{\longrightarrow} s_{\cdot}^{Hull} \wedge \{m_{\cdot}^{Port<\cdot,t>_x}\} \subset \mathcal{BB}^{TS} \Rightarrow \exists c_z^{\mathbb{S}} \in \mathbb{S}^C : c_{\cdot}^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} s_{\cdot}^{Hull} \wedge \{m_{\cdot}^{Port<\cdot,t>_x}\} \subset \mathcal{BB}^{TS} \Rightarrow \forall c_z^{\mathbb{S}} \in \mathbb{S}^C : c_{\cdot}^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} s_{\cdot}^{Hull} \wedge \{m_{\cdot}^{Port<\cdot,t>_x}\} \subset \mathcal{BB}^{TS}$. This proves the Proposition.

□

Topology & Structure Prototypes - Cell-Cluster

The Prototype 4.2 for cell-cluster differs in many aspects with respect to the three functional parts compared to the single cell prototype. Because a cell-cluster encapsulates several cell modules or sub-cluster modules, it requires o-times the links for each neighborhood system, with o representing the number of cells respectively sub-cluster within the particular cell-cluster. Hence, Prototype 4.2 possesses a

Simulation Prototype 4.2 Cell-Cluster Modules

1: **BEGIN** *STRUCTURE*
2: TRIGGER$< a, b, ... >$
3: WATCH_PORT$< A, B, ... >$
4: LINK$< \{\mathcal{N}^1\}_{\times Cells}, ..., \{\mathcal{N}^5\}_{\times Cells}, >$
5: MEM_HIERARCHY$< M_{Parts}^{GMem} >$
6: **END** *STRUCTURE*
7: ———————————
8: **BEGIN** *INIT_RULES*
9: GENERATE$< CELLS \vee CELL_CLUSTERS >$
10: SORT$<$ w.r.t. to overall $CELL\_GRID\ \Omega >$
11: GENERATE$< WIRE_MODULES >$
12: WIRE$< CELLS \vee CELL_CLUSTERS >$
13: **END** *INIT_RULES*
14: ———————————
15: **BEGIN** *FUNCTIONALITY*
16: **EMPTY**
17: **END** *FUNCTIONALITY*

$LINK < ... >$ definition compared to the $COM\_PORT < ... >$ definition of the single cell prototype. Furthermore this prototype implements parts of the global memory hierarchy with $MEM\_HIERARCHY < M_{Parts}^{GMem} >$, which is completely realized by putting several cell-cluster together. Within the initialization rule part the following actions are performed: (1) The corresponding cells or sub-cell-cluster are generated. (2) The labeling of the generated cells or cell-sub-cluster is adjusted to match with the labeling of a regular two dimensional grid, i.e. the top leftmost cell has label one and the bottom rightmost cell has label $|\Omega|$. (3) The components of the prototype are finally wired to form a complete and correct part of the Ω cell grid. These actions of the cell-cluster prototype are of vital importance for the self-generating capabilities. We will explain and further discuss this in Section 4.3.2. The functionality part of cell-cluster prototype is empty, because all functionality is exclusively located within the cells.

Proposition 4.2 states the formal guarantee that all simulation modules of the cell-cluster type are qualified representations of architectural building blocks for sim-

ulation. The proposition proves that Prototype 4.2 is an admissible representation of a particular part b^* of a building block $b \in \mathcal{BB}^{TS}$, i.e. $b^* \subset b \in \mathcal{BB}^{TS}$. From this fact follows that all operational simulation modules of cell-cluster, which are derive from this prototype are qualified representations of architectural building blocks for simulation.

Proposition 4.2 (Admissible $\subset \mathcal{BB}^{TS}$ Modeling)
The Prototype 4.2 for cell-cluster modules is an admissible representation of building block parts $b^* \subset b \in \mathcal{BB}^{TS}$ for simulation and hence all operational simulation modules $cc_y^{\mathbb{S}} \in \mathbb{S}^C$ derived from it.

Proof: First, we show that Prototype 4.2 models topology & structure defining building blocks, i.e. Prototype 4.2 $\overset{\text{models}}{\longrightarrow} (B \supset A) \subset \mathcal{BB}^{TS} \Rightarrow \exists cc_\cdot^{\mathbb{S}} \in \mathbb{S}^C : cc_\cdot^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} (B \supset A) \subset \mathcal{BB}^{TS} \Rightarrow \forall cc_\cdot^{\mathbb{S}} \in \mathbb{S}^C : cc_\cdot^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} (B \supset A) \subset \mathcal{BB}^{TS}$. Secondly, we show that Prototype 4.2 models no additional topology & structure defining components $b \notin \mathcal{BB}^{TS}$.
(1) The Prototype of the cell-cluster incorporates the structure for a specific part M_{Part}^{GMem} of the global memory hierarchy $M^{GMem} = \bigcup_{Parts} M_{Part}^{GMem}$. Hence, it holds: Prototype 4.2 $\overset{\text{models}}{\longrightarrow} (M_{Part}^{GMem} \subset M^{GMem}) \in \mathcal{BB}^{TS}$. For operational modules derived from the prototype it holds $\exists cc_\cdot^{\mathbb{S}} \in \mathbb{S}^C : cc_\cdot^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} (M_{Part}^{GMem} \subset M^{GMem}) \in \mathcal{BB}^{TS} \Rightarrow \forall cc_\cdot^{\mathbb{S}} \in \mathbb{S}^C : cc_\cdot^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} (M_{Part}^{GMem} \subset M^{GMem}) \in \mathcal{BB}^{TS}$. Furthermore, the cell cluster prototype either encapsulates several cell modules or other cell cluster, i.e. Prototype 4.2 $\overset{\text{models}}{\longrightarrow} \left[\left(s_\cdot^{Hull} \wedge \{m_\cdot^{Port<\cdot,t>_x}\}\right) \vee \{cc_\cdot\}\right] \in \mathcal{BB}^{TS}$.
(2) Only the elements of the $STRUCTURE$ part contribute to the topology of the cell module. The encapsulated cell modules have been proved in Lemma 4.1 not to add components $a \notin \mathcal{BB}^{TS}$. It remains to clarify that $WATCH\_PORT \notin \mathcal{BB}^{TS} \wedge TRIGGER \notin \mathcal{BB}^{TS} \wedge LINK \notin \mathcal{BB}^{TS}$ do not change the represented architecture and thus the simulation outcome. $WATCH\_PORT$, $LINK$ and $TRIGGER$ are passive structures, which do not alter or enhance the simulation representation of $(B \supset A) \subset \mathcal{BB}^{TS}$. Hence none of these elements add components $b \notin \mathcal{BB}^{TS}$.
Putting (1) and (2) together we receive: Prototype 4.2 $\overset{\text{models}}{\longrightarrow} M_{Part}^{GMem} \wedge \left[\left(s_\cdot^{Hull} \wedge \{m_\cdot^{Port<\cdot,t>_x}\}\right) \vee \{cc_\cdot\}\right] \in \mathcal{BB}^{TS} \Rightarrow \exists cc_\cdot^{\mathbb{S}} \in \mathbb{S}^C : cc_\cdot^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} M_{Part}^{GMem} \wedge [\{c_\cdot\} \vee \{cc_\cdot\}] \in \mathcal{BB}^{TS} \Rightarrow \forall cc_\cdot^{\mathbb{S}} \in \mathbb{S}^C : cc_\cdot^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} M_{Part}^{GMem} \wedge [\{c_\cdot\} \vee \{cc_\cdot\}] \in \mathcal{BB}^{TS}$.
This proves the Proposition.

□

Remark 4.1 ($(B \supset A) \subset \mathcal{BB}^{TS}$)

1. *The statement $(B \supset A) \subset \mathcal{BB}^{TS}$ follows directly from the fact that cell-cluster modules either incorporate cell-modules or other cell-cluster modules. Consequently, if cell-modules are embedded in a cell-cluster also their representation is incorporated in the particular cell-cluster and hence $(B \supset A) \subset \mathcal{BB}^{TS}$ holds.*

A cell cluster module can generate and instantiate other cell cluster modules (cf. $INIT\_RULES$ of Prototype 4.2) in order to realize the global memory hierarchy as well as to implement a recursive method for the automatic expansion of the site-grid. With respect to the recursive process the questions comes up, whether the total number t of cell cluster modules a MRF simulation model is composed of is bounded from above. The following theorem states an upper bound.

Theorem 4.1 (Upper bound)
The number t of particular cell-cluster modules within a complete MRF simulation model of size $|\Omega|$ is bounded from above by

$$t \leq \sum_{i=1}^{2^i \leq |\Omega|} \left\lfloor \left(\frac{|\Omega|}{2^i} \right) \right\rfloor . \tag{4.2}$$

Constraints:

1. Site grids in accordance with Definition 2.10, i.e. regular and equally spaced two-dimensional site grids (see Figure 2.1).
2. Single cell modules as well as single cell cluster can not encapsulated into another cell cluster, again.
3. Always two cells respectively cell cluster are merged into a new cell cluster. Except Condition 2 is violated then moe than two cell cluster have to be merged into a new cell cluster.

□

We continue with the prototype of wiring. This prototype finalize the discussion of simulation prototypes, which represent the topology & structure defining architectural building blocks.

Topology & Structure Prototypes - Wiring

The corresponding Prototype 4.3 of wiring, connecting cell and cell-cluster, possesses only in the $STRUCTURE$ part information in the form of wire definitions, which will be used to connect the COM_PORT of cell modules respectively the $LINK$ of cell-cluster modules. In accordance with the considered class of MRFs the prototype supports wire structures for $\mathcal{N}^1$ - $\mathcal{N}^5$ neighborhood systems. Additionally, the IN and OUT wires of each neighborhood systems are pooled, i.e. the wires to the neighbors as well as the wires from the neighbors, defined by the corresponding neighborhood systems, are pooled in one wire structure.

Proposition 4.3 formally accounts that all simulation modules of the wiring type, which are derived from Prototype 4.3 are qualified representations of architectural building blocks for simulation. Precisely the Proposition proves that Prototype 4.2 is an admissible representation of particular building blocks $C \subset \mathcal{BB}^{TS} \wedge C \not\subset B$. From this fact directly follows that all operational simulation modules, which are derive from this prototype are qualified representations of architectural building blocks for simulation.

Simulation Prototype 4.3 Wiring Modules

1: **BEGIN** *STRUCTURE*
2: WIRES$< \{\mathcal{N}^1\}_{\times Cells} >^{IN \wedge OUT}$
3: WIRES$< \{\mathcal{N}^2\}_{\times Cells} >^{IN \wedge OUT}$
4: WIRES$< \{\mathcal{N}^3\}_{\times Cells} >^{IN \wedge OUT}$
5: WIRES$< \{\mathcal{N}^4\}_{\times Cells} >^{IN \wedge OUT}$
6: WIRES$< \{\mathcal{N}^5\}_{\times Cells} >^{IN \wedge OUT}$
7: **END** *STRUCTURE*
8: ———————————
9: **BEGIN** *INIT_RULES*
10: **EMPTY**
11: **END** *INIT_RULES*
12: ———————————
13: **BEGIN** *FUNCTIONALITY*
14: **EMPTY**
15: **END** *FUNCTIONALITY*

The proof is straightforward because this prototype contains only one type of information within its *STRUCTURE* part.

Proposition 4.3 (Admissible $\subset \mathcal{BB}^{TS}$ Modeling)
The Prototype 4.3 for wiring modules is an admissible representation of building block parts $C \subset \mathcal{BB}^{TS} \wedge C \not\subset B$ for simulation and hence all the operational simulation modules $w_u \in \mathbb{S}^C$ derived from it.

Proof: We show that Prototype 4.3 $\overset{\text{models}}{\longrightarrow} C \subset \mathcal{BB}^{TS} \Rightarrow \exists w_\cdot^{\mathbb{S}} \in \mathbb{S}^C : w_\cdot^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} C \subset \mathcal{BB}^{TS} \Rightarrow \forall w_\cdot^{\mathbb{S}} \in \mathbb{S}^C : w_\cdot^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} C \subset \mathcal{BB}^{TS}$. Additionally we show that Prototype 4.3 models no additional topology & structure defining components $c \notin \mathcal{BB}^{TS}$.
(1) The simulation module of the wirings solely provides structures to wire up cells or cell cluster among each other. Because the wiring module realizes both directions to and from a cell respectively cell cluster we have: Prototype 4.3 $\overset{\text{models}}{\longrightarrow} \{(w_k^{<k,t>} \wedge w_t^{<t,k>}) : k,t \subset \{1 \leq i \leq |\Omega|\}\} \in \mathcal{BB}^{TS}$. For operational modules, derived from this protoype, it follows: $\exists w_\cdot^{\mathbb{S}} \in \mathbb{S}^W : w_\cdot^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} \{(w_k^{<k,t>} \wedge w_t^{<t,k>}) : k,t \subset \{1 \leq i \leq |\Omega|\}\} \in \mathcal{BB}^{TS} \Rightarrow \forall w_\cdot^{\mathbb{S}} \in \mathbb{S}^W : w_\cdot^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} \{(w_k^{<k,t>} \wedge w_t^{<t,k>}) : k,t \subset \{1 \leq i \leq |\Omega|\}\} \in \mathcal{BB}^{TS}$.
(2) All information is located in the *STRUCTURE* part of this prototype and contributes solely to the first case. Thus none of these elements add components $c \notin \mathcal{BB}^{TS}$.
Consequently, putting (1) and (2) together, we have: Prototype 4.3 $\overset{\text{models}}{\longrightarrow} \{(w_k^{<k,t>} \wedge w_t^{<t,k>}) : k,t \subset \{1 \leq i \leq |\Omega|\}\} \in \mathcal{BB}^{TS} \Rightarrow \forall w_u^{\mathbb{S}} \in \mathbb{S}^W :: w_\cdot^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} \{(w_k^{<k,t>} \wedge w_t^{<t,k>}) : k,t \subset \{1 \leq i \leq |\Omega|\}\} \in \mathcal{BB}^{TS}$. This proves the Proposition.

□

The preceding simulation prototypes have been proved (Proposition 4.1, 4.2 and 4.3) to represent the functionality of specific topology & structure defining

architectural building blocks and no additional components. In order to guarantee that the complete set $\mathcal{BB}^{TS} = \{S^{Hull}, W^{\mathcal{N}}, M^{Ports}, M^{GMem}\}$ of building blocks is covered by the introduced prototypes, and hence the proposed simulation framework is able to completely image the topology & structure defining components of the derived architecture-template (cf. Chapter 3), the following Corollary is stated.

Corollary 4.1 (Complete $\mathcal{BB}^{TS}$ Coverage)
The simulation modules $\mathcal{S}^{TS}$ are able to completely represent the building blocks $\mathcal{BB}^{TS}$ of the architecture-template for simulation.

Proof: The proof falls back upon Proposition 4.1, 4.2 and 4.3. The Propositions guarantee the representation of specific building blocks $\mathcal{BB}^{TS}$ for simulation. It holds:

1) $\forall c_z^{\mathbb{S}} \in \mathbb{S}^C : c_z^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} \left[s_{\cdot}^{Hull} \in S^{Hulls} \wedge \{m_{\cdot}^{Port<\cdot,t>_x}\} \in M^{Ports} \right] \subset \mathcal{BB}^{TS}$.

2a) $\forall cc_y^{\mathbb{S}} \in \mathbb{S}^C : cc_y^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} \left[(M_{Part}^{GMem} \in \bigcup_{Parts} M_{Part}^{GMem}) \wedge [\{c.\} \vee \{cc.\}] \right] \subset \mathcal{BB}^{TS} \Rightarrow$

2b) $\exists \{cc_v^{\mathbb{S}} : v \leq y\} \in \mathbb{S}^C \overset{\text{models}}{\longrightarrow} \bigcup_{Parts} M_{Part}^{GMem} \equiv M^{GMem}$.

3) $\forall w_u^{\mathbb{S}} \in \mathbb{S}^W : w_u^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} \{(w_k^{<k,t>} \wedge w_t^{<t,k>}) : k,t \subset \{1 \leq i \leq |\Omega|\}\} \subset \mathcal{BB}^{TS}$.

$1 \Rightarrow S^{Hulls} = \{s_i^{Hull} : 1 \leq i \leq |\Omega|\} \wedge M^{Ports} = \{m_i^{Port<i,t>_x} : 1 \leq i \leq |\Omega|,\ \forall t \in \mathcal{N}_i,\ x = \#bits\}$ is covered by $\mathcal{S}^{TS}$.
$2\text{b} \Rightarrow M^{GMem}$ is covered by $\mathcal{S}^{TS}$.
$3 \Rightarrow W^{\mathcal{N}} = \{w_i^{<i,t>} : 1 \leq i \leq |\Omega|,\ \forall t \in \mathcal{N}_i\}$ is covered by $\mathcal{S}^{TS}$.
Hence, $\mathcal{BB}^{TS}$ is completely covered by $\mathcal{S}^{TS}$. This proves the Corollary.

□

Propositions 4.1, 4.2, 4.3 and Corollary 4.1 establish the specific relation between the topology & structure representing set of building blocks $\mathcal{BB}^{TS}$ and the topology & structure representing set of simulation modules $\mathcal{S}^{TS}$, which is formally expressed by the mapping Π^{TS}. The mapping of subsets of the architectural building blocks S^{Hulls}, M^{Ports} and M^{GMem} to the set of simulation modules $\mathbb{S}^C$ and the mapping of wiring blocks $W^{\mathcal{N}}$ to the set $\mathbb{S}^W$ is, with $1 \leq i,t \leq |\Omega|$, given by

$$\Pi^{TS} : \begin{cases} \bigcup\limits_{|z|+|y|} \left\{ \{s_{k\subset\{i\}}^{Hull}\}, \{m_{k\subset\{i\}}^{Port<i,t>_\cdot} \vee \emptyset\}, M_{Parts}^{GMem} \vee \emptyset \right\} \longrightarrow \mathbb{S}^C \\ \\ \bigcup\limits_{|u|} \left\{ \left\{ (w_k^{<k,t>} \wedge w_t^{<t,k>}) : k,t \subset \{1 \leq i \leq |\Omega|\} \right\} \right\} \longrightarrow \mathbb{S}^W \end{cases} \tag{4.3}$$

This completes the discussion of the topology & structure defining simulation modules of the framework. In the next section we continue with simulation modules and their prototypes, which represent the processing functionalities of the MRF architecture.

4.2.3 Processing Simulation Modules

The second category of the ordering scheme comprises the type of simulation modules, which model the set of processing functionality defining architectural building

blocks $\mathcal{BB}^{PC}$ (cf. Definition 3.2). The different kinds of processing building blocks are individually modeled by their corresponding simulation modules, i.e. the energy functional building block, the optimization as well as the parameter estimation block are modeled by different simulation modules.

The set of processing functionality representing simulation modules will be denoted by $\mathcal{S}^{PC}$ in the sequel.

Definition 4.2 (Processing Modules $\mathcal{S}^{PC}$)
The set $\mathcal{S}^{PC}$ of processing representing simulation modules reads

$$\mathcal{S}^{PC} = \{\mathbb{S}_{PC}^{EF}, \mathbb{S}_{PC}^{OPT}, \mathbb{S}_{PC}^{PAR}\} \tag{4.4}$$

with

- $\mathbb{S}_{PC}^{EF} = \{EF_k^{\mathbb{S}} : k \gg 1\}$, the collection of simulation modules representing energy functionals.
- $\mathbb{S}_{PC}^{OPT} = \{OPT_l^{\mathbb{S}} : l \gg 1\}$, the collection of simulation modules representing optimization methods.
- $\mathbb{S}_{PC}^{PAR} = \{PAR_m^{\mathbb{S}} : m \gg 1\}$, the collection of simulation modules representing parameter estimation tasks.

Normally it is assumed that $\mathcal{H}_i^{EF} \in PC_{EF}$, $1 \leq i \leq |\Omega|$, are functionally identical and thus only one simulation module, which is replicated $|\Omega|$-times, is required for a particular MRF architecture. If this assumption is not applicable, the proposed simulation framework offers the flexibility to generate other constellations of energy functional simulation modules. For the set of PC_{OPT} optimization processing building blocks the same arguments as for the energy functional building blocks are valid. Finally, the building block PC_{PAR} is modeled by simulation modules $\mathbb{S}_{PC}^{PAR}$. If the parameter estimation process is distributed within the site-grid structure and interlinked with the memory hierarchy, then several different simulation modules are required. Otherwise, and in case the parameter estimation is done at the top of the memory hierarchy, only one single simulation module is needed.

The set $\mathcal{S}^{PC}$ of simulation modules solely models those kinds of architectural building blocks, which are exclusively dedicated to numerical processing tasks within a massively parallel MRF architecture. Each processing module is an independent and functionally closed unit, embedded in the topology gantry established by the simulation modules $\mathcal{S}^{TS}$. Merely the neighborhood signals have to be transfered to the processing modules in order to conduct the local MRF specific calculations. The strict separation of the simulation sets $\mathcal{S}^{TS}$ and $\mathcal{S}^{PC}$ supports the study of particular variants of MRF simulation models, since each part can be separately changed, without affecting the other parts.

The prototypes of the processing functionality modules are illustrated in Prototype 4.4, 4.5 and 4.6. In the ongoing discussion it will be proved that these prototypes represent the architectural building blocks PC^{EF}, PC^{OPT} and PC^{PAR} for simulation.

Due to the clear separation of the different processing functionalities within the proposed architecture-template and also among each other it is convenient for the

ongoing discussion on prototypes to give the *"Admissible $\mathcal{S}^{PC}$ Modeling"* proof of all these prototypes together at the end of this section. Proposition 4.4 establishes the formal arguments for all prototypes of the processing functionality type to model the building blocks PC^{EF}, PC^{OPT} and PC^{PAR} for simulation.

Processing Prototypes - Energy-Functional

Prototype 4.4 represents the structure of energy functional simulation modules $EF^{\mathbb{S}}_{\cdot}$. This kind of modules possesses an empty $STRUCTURE$ as well as an $INIT\_RULES$ part. All functionality of the energy functional modules is located in the equally named part and starts with a reset-command, which clears or sets the complete local memory of the simulation module. The following command

Simulation Prototype 4.4 Energy-Functional Modules

```
1: BEGIN STRUCTURE
2:    EMPTY
3: END STRUCTURE
4: ————————————————
5: BEGIN INIT_RULES
6:    GENERATE_MODULE< OPT^S_{i,l} >
7: END INIT_RULES
8: ————————————————
9: BEGIN FUNCTIONALITY
10:   RESET< Local_Memories >
11:   CALCULATE< H_i >
12:   CALL_MODULE(OPT^S_{i,l}(H_i) )
13: END FUNCTIONALITY
```

calculates the local energy $\mathcal{H}_i$, and the last command finally calls the corresponding optimization module $OPT^{\mathbb{S}}_{i,l}$ (see Prototype 4.5) with the value of $\mathcal{H}_i$. Obviously, the calculation of a specific energy functional $\mathcal{H}_i$ depends on the MRF model and can be diverse and complex. In this discussion it is approximately expressed by the single line command $CALCULATE{<}\ \mathcal{H}_i\ {>}$.

Processing Prototypes - Optimization

The corresponding prototype of the optimization modules is depicted in Prototype 4.5. For these simulation modules the $STRUCTURE$ part as well as the $INIT\_RULES$ part contains no commands or signal assignments. All commands are located within the $FUNCTIONALITY$ part and they start with the resetting of the local memories of the module. In the following the energy functional $\mathcal{H}_i$ becomes optimized, to allow different optimization strategies, which were described in Section 2.3, to become possible. The outcome of this optimization process determines the actual value of the site state, which is set by the next command SET_STATE$<$ New_State $>$. Similar to the commands of the energy functional Prototype 4.4, the commands of the optimization prototype eventually also feign calculations with low complexity and numerical simplicity. However, this is not nec-

Simulation Prototype 4.5 Optimization Modules

```
1: BEGIN STRUCTURE
2:   EMPTY
3: END STRUCTURE
4: ————————————————
5: BEGIN INIT_RULES
6:   EMPTY
7: END INIT_RULES
8: ————————————————
9: BEGIN FUNCTIONALITY
10:  RESET< Local_Memories >
11:  OPTIMIZE< H_i >
12:  SET_STATE< New_State >
13: END FUNCTIONALITY
```

essarily correct and solely depends on the specific energy functional and the used optimization method for each particular MRF model.

Processing Prototypes - Parameter-Estimation

For Prototype 4.6 two different cases can occur, which are determined by index k. In the first case with $k = 1$ the complete parameter estimation becomes executed on the top level of the memory hierarchy, where the complete data is available for conducting the calculation of $\mathbf{\Theta}$ - the set of free parameters a MRF model possesses. In the second case with $k \neq 1$ the parameter estimation is executed - if the formulas

Simulation Prototype 4.6 Parameter Estimation Modules

```
1: BEGIN STRUCTURE
2:   EMPTY
3: END STRUCTURE
4: ————————————————
5: BEGIN INIT_RULES
6:   EMPTY
7: END INIT_RULES
8: ————————————————
9: BEGIN FUNCTIONALITY
10:  RESET< Local_Memories >
11: if k==1 then
12:      CALCULATE< Θ > at top of M^GMem hierarchy
13: else
14:      CALCULATE< Θ > along M^GMem hierarchy
15: end if
16: END FUNCTIONALITY
```

of the free parameters $\mathbf{\Theta}$ factorize appropriately - along the memory hierarchy. The second case additionally includes the setting where each site determines the free parameters only on the basis of their locally available information. Consequently,

only the bottom most level of the memory hierarchy is equipped with parameter estimation modules.

The following Proposition 4.3 establishes the formal justification that all simulation modules of the processing functionality type, which are derived from Prototype 4.4, 4.5 and 4.6 are qualified representations of $\mathcal{BB}^{PC}$ building blocks for simulation. In detail the Proposition proves that Prototype 4.4, 4.5 and 4.6 are admissible representations of $\mathcal{BB}^{PC}$. From this fact follows that all operational simulation modules, which are derive from these prototypes are qualified representations of architectural building blocks for simulation.

Proposition 4.4 (Admissible $\subset \mathcal{BB}^{PC}$ Modeling)
Prototype 4.4, 4.5 and 4.6 are admissible representations of building blocks $\mathcal{BB}^{PC}$ for simulation and hence all the operational simulation modules derived from them.

Proof: The proof contains two parts. In the first part we show that the Prototype 4.4, 4.5 and 4.6 $\overset{\text{models}}{\longrightarrow} D \subset \mathcal{BB}^{PC} \Rightarrow \exists EF_{\cdot}^{\mathbb{S}} \in \mathbb{S}_{PC}^{EF} \wedge \exists OPT_{\cdot}^{\mathbb{S}} \in \mathbb{S}_{PC}^{OPT} \wedge \exists PAR_{\cdot}^{\mathbb{S}} \in \mathbb{S}_{PC}^{PAR} : EF_{\cdot}^{\mathbb{S}}, OPT_{\cdot}^{\mathbb{S}}, PAR_{\cdot}^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} D \subset \mathcal{BB}^{PC}$. The second part shows that these Prototypes 4.4, 4.5 and 4.6 do not model $A \notin \mathcal{BB}^{PC}$.
(1) All Prototypes 4.4, 4.5 and 4.6 are distinct simulation modules with well separated processing functionalities. Prototype 4.4 is solely dedicated to building blocks $\mathcal{H}_{\cdot}^{EF}$, Prototype 4.5 is exclusively dedicated to *opt.* and finally Prototype 4.6 is made for PC^{PAR}. Due to this clear structure of the prototypes it directly follows: $\exists EF_{\cdot}^{\mathbb{S}} \in \mathbb{S}^{PC} : EF_{\cdot}^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} \mathcal{H}_{\cdot}^{EF} \in \mathcal{BB}^{PC} \wedge \exists OPT_{\cdot}^{\mathbb{S}} \in \mathbb{S}^{PC} : OPT_{\cdot}^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} opt. \in \mathcal{BB}^{PC} \wedge \exists PAR_{\cdot}^{\mathbb{S}} \in \mathbb{S}^{PC} : PAR_{\cdot}^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} PC^{PAR} \in \mathcal{BB}^{PC}$
(2) Only the elements of the $FUNCTIONALITY$ part realize processing tasks and because each prototype is dedicated to a specific MRF processing functionality it is straightforward that no functionality $d \notin \mathcal{BB}^{PC}$ is added.
Consequently, putting (1) and (2) together we have: Prototypes 4.4, 4.5 and 4.6 $\overset{\text{models}}{\longrightarrow} EF_{\cdot}^{\mathbb{S}} \wedge OPT_{\cdot}^{\mathbb{S}} \wedge PAR_{\cdot}^{\mathbb{S}} \Rightarrow \forall EF_{\cdot}^{\mathbb{S}} \in \mathbb{S}^{PC} : EF_{\cdot}^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} \mathcal{H}_{\cdot}^{EF} \in \mathcal{BB}^{PC} \wedge \forall OPT_{\cdot}^{\mathbb{S}} \in \mathbb{S}^{PC} : OPT_{\cdot}^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} opt. \in \mathcal{BB}^{PC} \wedge \forall PAR_{\cdot}^{\mathbb{S}} \in \mathbb{S}^{PC} : PAR_{\cdot}^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} PC^{PAR} \in \mathcal{BB}^{PC}$.
This proves the Proposition.

□

In the preceding discussion we have proved that the simulation Prototypes 4.4, 4.5 and 4.6 exclusively represent processing functionality defining architectural building blocks. To formally guarantee that the simulation framework is powerful enough to completely cover the set $\mathcal{BB}^{PC} = \{PC^{EF}, PC^{OPT}, PC^{PAR}\}$ of building blocks (see derivation in Chapter 3), Corollary 4.2 is stated.

Corollary 4.2 (Complete $\mathcal{BB}^{PC}$ Coverage)
The components of the simulation modules $\mathcal{S}^{PC}$ are able to completely represent the functionality of the architectural building blocks $\mathcal{BB}^{PC}$ for simulation.

Proof: The proof falls back upon Proposition 4.4, which guarantees
1) $\forall EF_{\cdot}^{\mathbb{S}} \in \mathbb{S}^{PC} : EF_{\cdot}^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} \mathcal{H}_{\cdot}^{EF} \in \mathcal{BB}^{PC}$

2) $\forall OPT_{\cdot}^{\mathbb{S}} \in \mathbb{S}^{PC} : OPT_{\cdot}^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} opt. \in \mathcal{BB}^{PC}$
3) $\forall PAR_{\cdot}^{\mathbb{S}} \in \mathbb{S}^{PC} : PAR_{\cdot}^{\mathbb{S}} \overset{\text{models}}{\longrightarrow} PC^{PAR} \in \mathcal{BB}^{PC}$
$1 \Rightarrow PC^{EF} = \{\mathcal{H}_i^{EF} : 1 \leq i \leq |\Omega|\}$ is covered by $\mathcal{S}^{PC}$.
$2 \Rightarrow PC^{OPT} = \{opt_i : 1 \leq i \leq |\Omega|\}$ is covered by $\mathcal{S}^{PC}$.
$3 \Rightarrow PC^{PAR}$ is covered by $\mathcal{S}^{PC}$.
Consequently, $\mathcal{BB}^{PC} = \{PC^{EF}, PC^{OPT}, PC^{PAR}\}$ is completely covered by $\mathcal{S}^{PC}$.

□

The specific relation between the processing functionality representing building blocks $\mathcal{BB}^{PC}$ and the processing functionality representing simulation modules is established by Proposition 4.6 and Corollary 4.2. This coherence between the building blocks $\mathcal{BB}^{PC}$ and the collection of simulation modules is given by the mappings $\mathbf{\Sigma}$, which read

$$\Sigma^{EF} : \bigcup_{|k|} \mathcal{H}^{EF} \longrightarrow \mathbb{S}_{PC}^{EF}. \tag{4.5}$$

$$\Sigma^{OPT} : \bigcup_{|l|} opt \longrightarrow \mathbb{S}_{PC}^{OPT}. \tag{4.6}$$

$$\Sigma^{PAR} : \bigcup_{|m|} PC^{PAR} \longrightarrow \mathbb{S}_{PC}^{PAR}. \tag{4.7}$$

This concludes the presentation of the set $\mathcal{S}^{PC}$ available within the simulation framework. Until now we have introduced different simulation module sets: One set for topology & structure modeling modules and the other set for processing modeling modules. The following paragraph describes the simulation modules for the architectural control building blocks. These control modules complete the simulation sets of the framework in order to model the massively parallel architecture of a particular Markov Random Field, which is covered by the considered MRF-class defined in Section 2.5.

4.2.4 Control Simulation Modules

The third and final category of the organization scheme comprises the type of simulation modules $\mathcal{S}^{CT}$, which exclusively model control functionalities. In view of the reasons for simplification the complete control-functionality of the architectural building blocks $\mathcal{BB}^{CT}$ are represented by one single simulation module. This specific simulation module realizes the functionality of the system control building block CT^{System} and the memory hierarchy building block CT^{GMem}, whereby both control units are only once represented within a massively parallel architecture. The situation looks slightly different with regard to the control building block sets $CT^{PortMem}$, CT^{EF} and CT^{OPT}. Not all members of these building block sets are represented by particular control modules; rather for all sets the same control module as for CT^{System} and CT^{GMem} is used. This can be justified as it is assumed that the components of $ctrl_i^{PortMem}$, $ctrl_i^{EF}$ and $ctrl_i^{OPT}$, with $1 \leq i \leq |\Omega|$, are required for functionally and structurally identical processing modules (see Section 4.2.3). For

the case that a differing setting becomes necessary, several control modules can be generated and used within one particular MRF model. The simulation framework also supports these settings.

The set of simulation modules representing the control-functionality will be denoted by $\mathcal{S}^{CT}$ in the sequel and reads as follows:

Definition 4.3 (Control Modules $\mathcal{S}^{CT}$)
The set $\mathcal{S}^{PC}$ of control representing simulation modules reads

$$\mathcal{S}^{CT} = \mathbb{S}^{CT} \tag{4.8}$$

with

- $\mathbb{S}^{CT} = \{CT_o^{\mathbb{S}} : o \gg 1\}$, the set of control modules, which determines the sequence of processing order during simulation runs.

This finalizes the description and formal definition of the simulation modules $\mathcal{S}^{CT}$ representing the control-functionality available within the proposed framework. We continue the discussion with the description of a prototypical control simulation module.

The complete control-functionality of a MRF simulation model, which mimics the massively parallel architecture compiled out of building blocks $\mathcal{BB}$, is represented by a single simulation module $CT_o^{\mathcal{S}} \in \mathcal{S}^{CT}$. Naturally, these modules are solely dedicated to take on control tasks within the simulation model. Each control-functionality is pooled within one single control simulation module for each MRF model. This module is not independent and functionally self-contained as it refers to the $TRIGGER$ signals of other modules. Thus it is guaranteed that the simulation model can be systematically generated step by step and module by module. The strict separation of all simulation module sets and their corresponding elements systematically structures the compilation and refinement of particular MRF simulation models as each module can be separately changed and improved.

The prototype of the control functionality representing simulation modules is depicted in Prototype 4.7. After a short description of this prototype we prove that Prototype 4.7 represents the architectural building blocks $\mathcal{BB}^{CT}$.

Control Prototypes

The Prototype 4.7 for control functionality contains commands only within the $INIT\_RULES$ part. The $STRUCTURE$ and $FUNCTIONALITY$ part is empty and thus no instructions become executed during the generation process of the simulation model or during the simulation run itself. Each $SET\_RULES^{\cdot} < ... >$ instruction assigns one ore several $TRIGGER$ signals to specific $RULES$, i.e. as soon as the $RULES$ are evaluated as true the corresponding $TRIGGER$ is activated.

In the next Proposition 4.7 we show that all simulation modules of the control type are qualified representations of architectural building blocks for simulation. In order to achieve this statement we prove that Prototype 4.7 is an admissible representation of a building blocks $E \subset \mathcal{BB}^{CT}$. Again it follows that all operational simulation modules, which are derive from this prototype are qualified representations of control building blocks for simulation.

Simulation Prototype 4.7 Control Modules

1: **BEGIN** *STRUCTURE*
2: **EMPTY**
3: **END** *STRUCTURE*
4: ———————————
5: **BEGIN** *INIT_RULES*
6: SET_RULES$^{System} < (a, Rule), (b, Rule), ... >$
7: SET_RULES$^{Cell} < (a, Rule), (b, Rule), ... >$
8: SET_RULES$^{EF} < (a, Rule), (b, Rule), ... >$
9: SET_RULES$^{OPT} < (a, Rule), (b, Rule), ... >$
10: SET_RULES$^{PAR} < (a, Rule), (b, Rule), ... >$
11: SET_RULES$^{GMem} < (a, Rule), (b, Rule), ... >$
12: **END** *INIT_RULES*
13: ———————————
14: **BEGIN** *FUNCTIONALITY*
15: **EMPTY**
16: **END** *FUNCTIONALITY*

Proposition 4.5 (Admissible $\subset \mathcal{BB}^{CT}$ Modeling)
The Prototype 4.7 for control modules is an admissible representation of building blocks $E \subset \mathcal{BB}^{CT}$ for simulation and hence all the operational simulation modules $CT_o^{\mathbb{S}} \in \mathbb{S}^{CT}$ derived from it.

Proof: We show that Prototype 4.7 $\stackrel{\text{models}}{\longrightarrow} E \subset \mathcal{BB}^{CT} \Rightarrow \exists CT_\cdot^{\mathbb{S}} \in \mathbb{S}^{CT} : CT_\cdot^{\mathbb{S}} \stackrel{\text{models}}{\longrightarrow} E \subset \mathcal{BB}^{CT} \Rightarrow \forall CT_\cdot^{\mathbb{S}} \in \mathbb{S}^{CT} : CT_\cdot^{\mathbb{S}} \stackrel{\text{models}}{\longrightarrow} E \subset \mathcal{BB}^{CT}$. Additionally, we show Prototype 4.7 $\stackrel{\neg\text{models}}{\longrightarrow} e \notin \mathcal{BB}^{CT}$.
(1) Within the $INIT\_RULES$ part $TRIGGER$ signals are affiliated to specific $RULES$. The rule assignment is separately done for the components $SYSTEM$, $Cell$, EF, OPT, PAR and $GMem$. Hence it is justified to conclude that Prototype 4.7 $\stackrel{\text{models}}{\longrightarrow} CT^{System} \wedge CT^{GMem} \wedge ctrl_\cdot^{PortMem} \wedge ctrl_\cdot^{EF} \wedge ctrl_\cdot^{OPT} \in \mathcal{BB}^{CT}$.
(2) Merely the $INIT\_RULES$ part contains instructions. But none of these elements add functionality $e \notin \mathcal{BB}^{CT}$.
Putting (1) and (2) together we receive: Prototype 4.7 $\stackrel{\text{models}}{\longrightarrow} CT^{System} \wedge CT^{GMem} \wedge ctrl_\cdot^{PortMem} \wedge ctrl_\cdot^{EF} \wedge ctrl_\cdot^{OPT} \in \mathcal{BB}^{CT} \Rightarrow \exists CT_\cdot^{\mathbb{S}} \in \mathbb{S}^{CT} : CT_\cdot^{\mathbb{S}} \stackrel{\text{models}}{\longrightarrow} CT^{System} \wedge CT^{GMem} \wedge ctrl_\cdot^{PortMem} \wedge ctrl_\cdot^{EF} \wedge ctrl_\cdot^{OPT} \in \mathcal{BB}^{CT} \Rightarrow \forall CT_\cdot^{\mathbb{S}} \in \mathbb{S}^{CT} \stackrel{\text{models}}{\longrightarrow} CT^{System} \wedge CT^{GMem} \wedge ctrl_\cdot^{PortMem} \wedge ctrl_\cdot^{EF} \wedge ctrl_\cdot^{OPT} \in \mathcal{BB}^{CT}$ This proves the Proposition.

□

The preceding Proposition gives rise to the following Corollary on the complete coverage of the building blocks $\mathcal{BB}^{CT}$ by the simulation modules $\mathcal{S}^{CT}$.

Corollary 4.3 (Complete $\mathcal{BB}^{CT}$ Coverage)
The components of the simulation modules $\mathcal{S}^{CT}$ completely represent the control functionality of the architectural building blocks $\mathcal{BB}^{CT}$ for simulation.

□

The specific relation between the building blocks $\mathcal{BB}^{CT}$ for control functionalities and the corresponding simulation modules $\mathbb{S}^{CT}$ has been established by Proposition 4.5 and Corollary 4.3. The mapping formally reads

$$\Delta^{CT} : \bigcup_{|k|} \{CT^{System}, CT^{GMem}, CT^{PortMem}, CT^{EF}, CT^{OPT}\} \longrightarrow \mathbb{S}^{CT}. \qquad (4.9)$$

Simulation Framework - $\mathcal{BB}^{System}$ Coverage

The previous discussion of this section finally cumulates in Theorem 4.2, which states that the proposed simulation framework with its different kind of simulation modules is powerful enough to represent the complete massively parallel MRF architecture template.

Theorem 4.2 (Complete $\mathcal{BB}^{System}$ Coverage)
The architecture template, which is composed of the building blocks $\mathcal{BB}^{System}$ is completely representable by the simulation sets $\mathbb{S}^{TS}$, $\mathbb{S}^{PC}$ and $\mathbb{S}^{CT}$ of the proposed simulation framework.

Proof: Follows immediately from: (1) Mapping 4.3 between all types of topology & structure defining building blocks $\mathcal{BB}^{TS}$ and representing simulation models from $\mathbb{S}^{TS}$.
(2) Mapping 4.5, 4.6 and 4.7 between all processing functionality representing building blocks $\mathcal{BB}^{PC}$ and simulation models from $\mathbb{S}^{PC}$.
(3) Mapping 4.9 between all building blocks for control $\mathcal{BB}^{CT}$ and representing simulation models from $\mathbb{S}^{CT}$.
Consequently, putting (1), (2) and (3) together it follows that the architecture template can be composed by the simulation modules of the proposed framework. This proves the Theorem.

□

The following remarks, which accentuate some features of particular MRF simulation models, finalize this section.

Remarks 4.1 (Particular MRF simulation model)

1. *Each MRF simulation model is composed of exactly $|\Omega|$ distinct simulation modules of single cells. Hence each cell module is a closed object, which is independent with respect to data and structure from the other cell modules.*
2. *Within each cell module are embedded distinct modules for the three different processing functionalities. Consequently, a MRF simulation model comprises $3 \times |\Omega|$ modules for the three processing functionalities of each cell module.*
3. *All other components of the architecture are also represented by distinct simulation modules to reflect the massively parallel architecture.*

4. *The number representation and calculation can be switched between float-point and fixed-point precision in order to study MRF models and their massively parallel architectures under hardware relevant conditions.*

5. *Different parallel processing sequences, which can arise due to the demands of the various MRF models of the considered class, can be systemically realized by defining and setting the corresponding trigger of each single cell module.*

4.3 Building MRF Simulation Models

The construction of a hardware-relevant simulation model with the help of the introduced framework, which represents the massively parallel architecture (cf. Chapter 3) of a specific MRF model, as from the user's perspective is executed in two phases. During the first building phase - further discussed in the section directly following - the user has to do some coding work by hand in order to provide modules, which are specific for the intended simulation setting and the MRF model. The second phase, described in Section 4.3.2, just needs to be triggered by the user. The following steps of this phase will automatically be executed by the simulation framework and the particular simulation modules to make a complete simulation model available at the end of the second phase.

4.3.1 Building the Simulation Model - Model Preparation

During the first phase of the simulation model building process the MRF simulation framework requires a close user-interaction in order to coordinate and realize the compilation of the model. At this stage the *Frame-Cell Module* represents the central module. This kind of module has to be coded by hand respectively has to be modified for each particular MRF simulation model and simulation setting. The second phase, which is automatically executed in order to expand the complete model, starts with information of this module. Within the Frame-Cell module several mechanisms are embedded, which will be sequentially executed in the second phase of the building process. Each triggered mechanism executes other processes on his part and so on and so forth, to allow the simulation model to be completely expanded within the framework environment at the end of this sequence.

One mechanism of a Frame-Cell module, which has to be adapted by the framework user, generates and instantiates the top-level Cell-Cluster module, i.e. that mechanism generates $cc^{x \times y} \in \mathbb{S}^C$ with the MRF grid size $x \times y$. Furthermore, another mechanism, which has to be adapted by the user, assigns predefined values to the COM_PORT (cf. Simulation Prototype 4.2) of the top-level cell-cluster. This step is required as the top-level cell-cluster does not possess neighboring cell-clusters, which are connected to the COM_PORT of the top level cell-cluster. Therefore the incomplete neighborhood system of the top level cell-cluster has to be completed by this mechanism. In view of digital hardware realizations and resources usage, it is useful to either assign logical 0 or logical 1 to the COM_PORT of the top level cell-cluster simulation module. Additionally, the user has to adapt a mechanism, which connects the $TRIGGER$ and $WATCH\_PORT$ signals of the top level cell-cluster. Moreover, the user can change the standard display functionalities of the framework

as well as the standard functionalities of the data-storage system. However, normally these changes are not necessary as the standard functionalities are sufficient to monitor the simulation run. The rich set of stored data is also appropriate for most off-line processing tasks and investigations.

Consequently, the number of required adaptations hand-made by the user is moderate for each particular MRF simulation model and, in addition to this, is essentially located within the *Frame-Cell* module. Larger volumes of user-made hand-coding are only needed, if the existing module collections have to be extended because they can not provide the required functionalities.

4.3.2 Building the Simulation Model - Model Generation

During the second phase of the model building process only a supporting user-interaction is required. The user of the Simulation-Framework merely has to trigger the generation of the particular MRF simulation model by means of a predefined graphical user-interface. Any other generation process of the particular simulation model is hidden from the user and executed automatically within and between the different module types. Starting with the information encoded by the *Frame-Cell* module, the *Simulation-Model Generator* triggers the first generation procedures in order to compile the simulation model. During this step the framework generates wires for the different clock domains, wires with predefined values in order to provide values for the incomplete neighborhood of the top-level site-cluster and special ports to observe and collect simulation data. After this is finished the framework generates the top-level site-cluster. By doing this additional mechanisms are triggered, which generate the sub-site-clusters, wire them up, sort the indexes of the sites generated until now and triggers the next mechanisms, which continue with this scheme until single-site simulation modules are generated. The single-site modules generate respectively instantiate their corresponding energy-functional and optimization module. Furthermore, the single-site modules set the trigger signals in order to make the control-module for the simulation run operational. After all these mechanisms have successfully been executed, the framework finally connects the wires and signals of the top-level module, a step which finalizes the automatic generation process of the corresponding MRF simulation model.

Provided that the generation process of the simulation model has been executed successfully, the complete simulation model of the massively parallel architecture for a specific MRF model is modeled and held within the memory of the computer, which runs the Simulation-Framework. At this point the user can make input data available to the model and start the simulation.

4.4 Results - Simulation Framework

The proposed simulation framework and its software-technical realization has been intensively tested by means of static tests of the particular simulation modules and the complete framework. Foremost the two exemplary MRF image processing models (Section 3.7.1, 3.7.2) have been used to prove the capabilities of the simulation framework. Since we can only present and discuss a few selected simulation results in this section without impairing the section's readability, we have moved additional

simulation results to Appendix A. We kindly advise the inclined reader to study the rich set of simulation results in Appendix A, which underpin the capabilities of the proposed simulation framework. In the following we will discuss simulation results, which have been achieved with the histogram segmentation model and its corresponding massively parallel hardware architecture. We have selected the RGB-encoded landscape image depicted in Figure 4.2 a and the gray-scaled balloon image shown in Figure 4.3 a as image data.

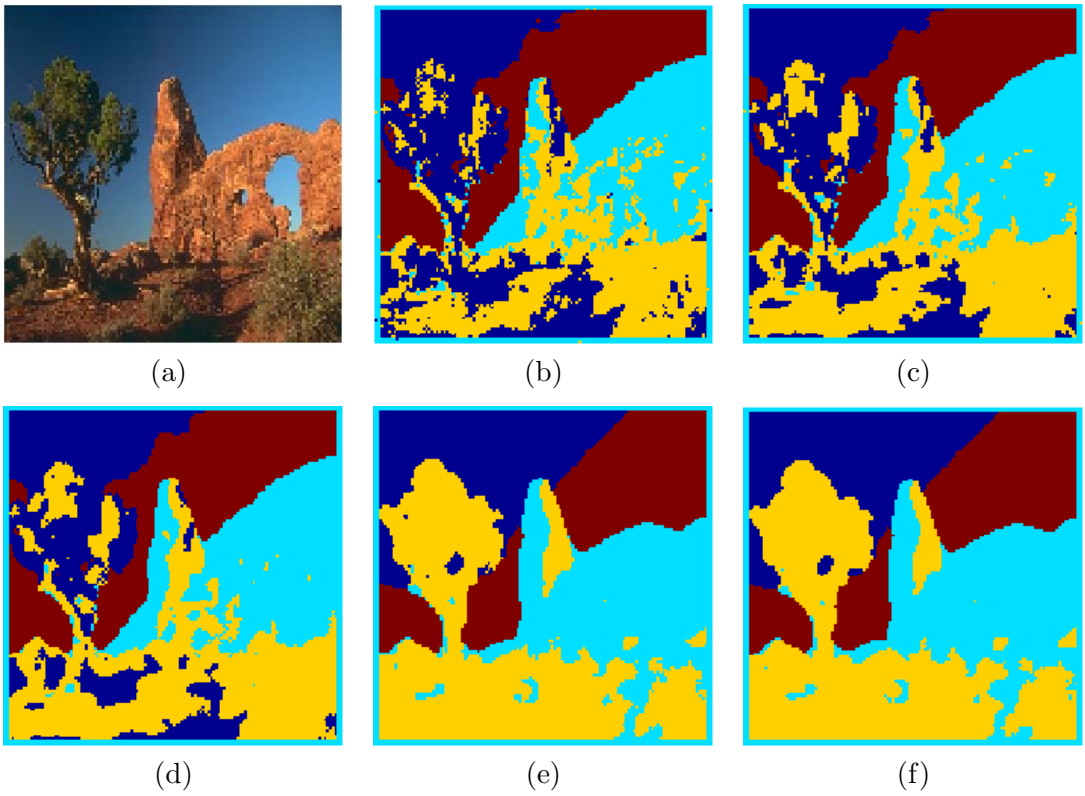


Figure 4.2: Ground-Truth Segmentation Results. (a) Original RGB image data. (b)-(f) Segmentation results for 1st to 5th order neighborhood system. Model/Architecture settings: 4 classes and 8 equally sized and spaced bins for each of the RGB channels. RGB channels: value range [0,255].

In order to study and assess the performance of a specific architecture/model combination it is indispensable to define the ground-truth architecture of the model, and thus to determine the ground-truth results of this specific architecture/model combination. With our advocated approach, put into practice by the architecture template and the Simulation-Framework, the realization becomes straightforward. Only one single simulation-structure is required for the ground-truth simulations as well as for the hardware relevant simulations. The simulation model for the ground-truth architecture/model combination differs only with respect to the number precision - double precision float-point versus fixed-point precision - from the simulation model of the hardware-relevant architecture/model combination. Already with respect to this task we significantly profit from the capabilities of the

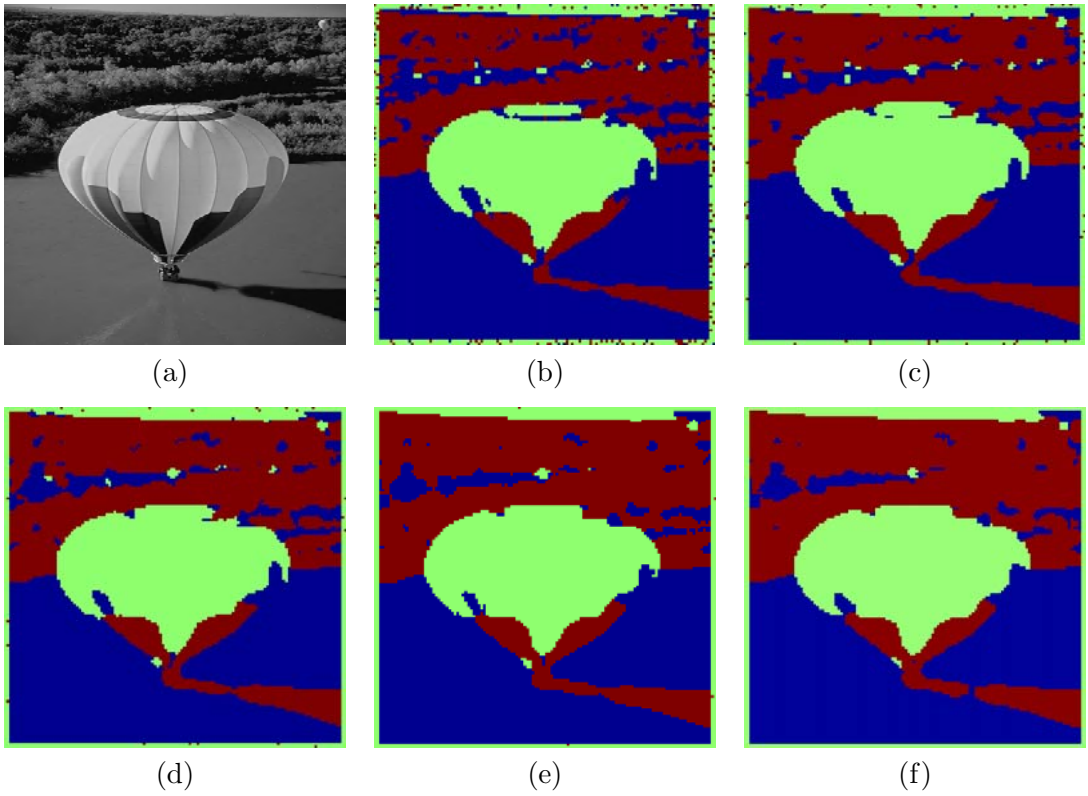


Figure 4.3: Ground-Truth Segmentation Results. (a) Original gray-scaled image data. (b)-(f) Segmentation results for 1st to 5th order neighborhood system. Model/Architecture settings: 3 classes and 4 equally sized and spaced bins for the gray-scale channel. Gray-scale channel: value range [0,255].

framework, because only one simulation model is required, which can be configured to handle double precision float-point numbers or alternatively fixed-point numbers. Consequently, it is automatically guaranteed by our proposed framework that errors, which are induced by structural and architectural mismatches between the float- and fixed-point architecture/model variant, are excluded right from the outset. Furthermore it is guaranteed that each architecture/model combination is consistent with MRF theory, because the simulation framework has been shown to represents the architecture template, which was derived with respect to the claim of strictly respecting Markov Random Field theory.

The simulation results of the float-point architecture/model combination and thus the ground-truth results are depicted in Figure 4.2 (b)-(f) (landscape image) and Figure 4.3 (b)-(f) (balloon image) for the first five neighborhood systems $\mathcal{N}^1 - \mathcal{N}^5$. To receive these different results it is sufficient to activate the corresponding neighborhood system (ports and wirings) and to adopt the histogram forming procedure. Due to the modular character of the framework and the strict separation of the simulation modules among each other we only have to parameterize the corresponding single cell module for histogram based segmentation. All other simulation modules are not affected. Again we profit from features the framework offers to

systematically compile variants of a specific architecture/model combination.

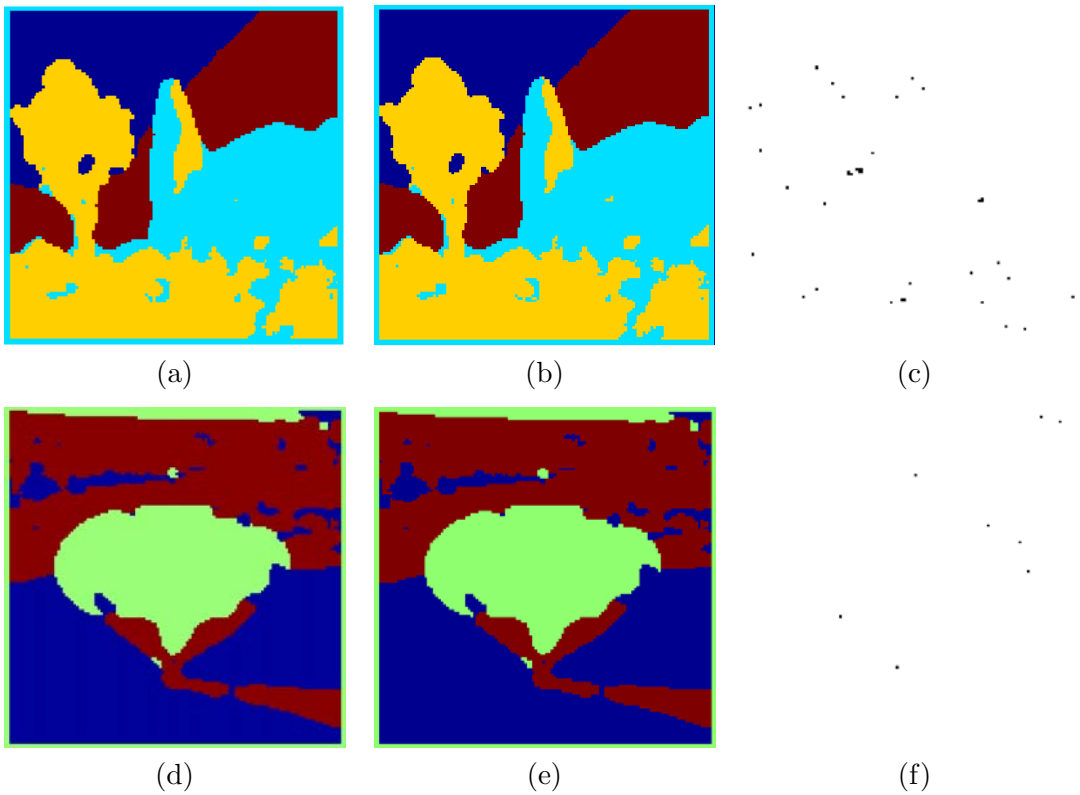


Figure 4.4: Comparison of ground-truth and 32 bit fixed-point precision architecture. (a,d) Shows the ground-truth of the segmentation model for the 5th order neighborhood system. (b,e) Shows the result for a 32bit fixed-point number representation and calculation. (c,f) Differences (black) between ground-truth and fixed-point model. The fixed-point is located in the middle of the 32bit. Over- and Underflow is covered by truncation respectively saturation.

Comparing the ground-truth results among each other it becomes obvious that the segmentation model with larger neighborhood systems (3rd, 4th and 5th order) produces more homogeneous segments and thus a smoother segmentation within unstructured image parts like the treetop of the landscape image. Even at image data with homogeneous portions like the balloon picture we observe this behavior (cf. Figure 4.3 b with f). This segmentation behavior is determined by the larger number of values forming the histograms when using the 3rd, 4th or 5th order neighborhood system as well as the larger spatial portion of the image. Both interrelation features cause a more "discriminative" empirical distribution, which is compared with the prototypes to determine the cluster-assignment. At smaller neighborhood systems (1st and 2nd) the bins are either empty or the masses of the bins concentrate at specific bins. Regard for instance the treetop of the landscape image: When using a 1st order neighborhood system it is mostly likely that some of the neighbors take the values of the sky, which shines through leaves of the treetop, and hence the masses are concentrated to this bin, following a cluster-assignment equally with the sky-

cluster. These expected and consistent results, received by the architecture/model combination for segmentation, manifest the capabilities of the proposed simulation framework to correctly represent the architecture template and the massively parallel processing dynamic for simulation. Several additional results, collected in Appendix A, further underpin this statement.

For the following discussion we will take the model/architecture combination with the 5th order neighborhood system as a basis and thus also the corresponding ground-truth result of this variant. Figure 4.4 illustrates the segmentation performance of the 32bit model/architecture variant in comparison with the ground-truth model/architecture variant. Picture 4.4a,d shows the ground-truth segmentation-result for the landscape respectively balloon image and picture 4.4b,e the segmentation result of the 32bit variant. At first look it is difficult to identify any differences between the segmentation results. The particular cluster-sizes, their separation as well as the overall cluster-structure seems to be identical for both model/architecture variants. A detailed site-by-site comparison confirms this visual observation. In Figure 4.4c,f are depicted the sites in black, which have a different cluster assignment, when comparing the cluster-assignments of the ground-truth result with the cluster-assignments of the 32bit model/architecture variant. For the landscape image we have in summary that the 32bit variant possesses 34 sites (see Table 4.1), which have different cluster-assignments. Consequently, the classification rate of the 32bit model/architecture variant is 99.79% compared with the ground-truth rate of 100%.

In Figure 4.5 we have opposed the segmentation result of the 16bit model/architecture variant to the segmentation result of the ground-truth model/architecture variant. The ground-truth segmentation-results are depicted in Figure 4.5a,d and Figure 4.5b,e shows the segmentation result of the model/architecture variant with 16bit fixed-point precision. Although the different cluster-assignments of particular sites are evident, the 16bit model/architecture variant is still able to reproduce very similar cluster-sizes, their separation as well as the overall cluster-structure compared with the ground-truth model. Figure 4.5c,f shows the particular sites in black, whose cluster-assignment differs from the cluster-assignment of the ground-truth result. In detail, for the landscape image data, the 16bit variant possesses 1222 sites respectively 965 sites without counting the border sites (see Table 4.1), which have different cluster-assignments. The classification rate of the 16bit model/architecture variant is thus 94.11%, when neglecting the results of the border sites, compared with the ground-truth rate of 100%. These results of different fixed-point architecture/model variants demonstrate the capabilities of the proposed simulation framework to represent massively parallel MRF architectures in a hardware-relevant setting. To stress this point again, the simulation framework offers the unique opportunity to simulate and hence study complete MRF hardware-architectures, their parallel processing dynamics, their numerical sensitiveness and their overall convergence behavior with respect to hardware constraints.

Figure 4.6 shows different results of the 12bit segmentation model/architecture variant, with Fig. 4.6a depicting the ground-truth segmentation result, Fig. 4.6b depicting the segmentation result of the 12bit variant and Fig. 4.6c-d showing intermediate results with a significant worse segmentation compared with the segmentation result of the previous relaxation step. The 12bit model/architecture variant is the first variant that possesses obvious segmentation problems caused by the lim-

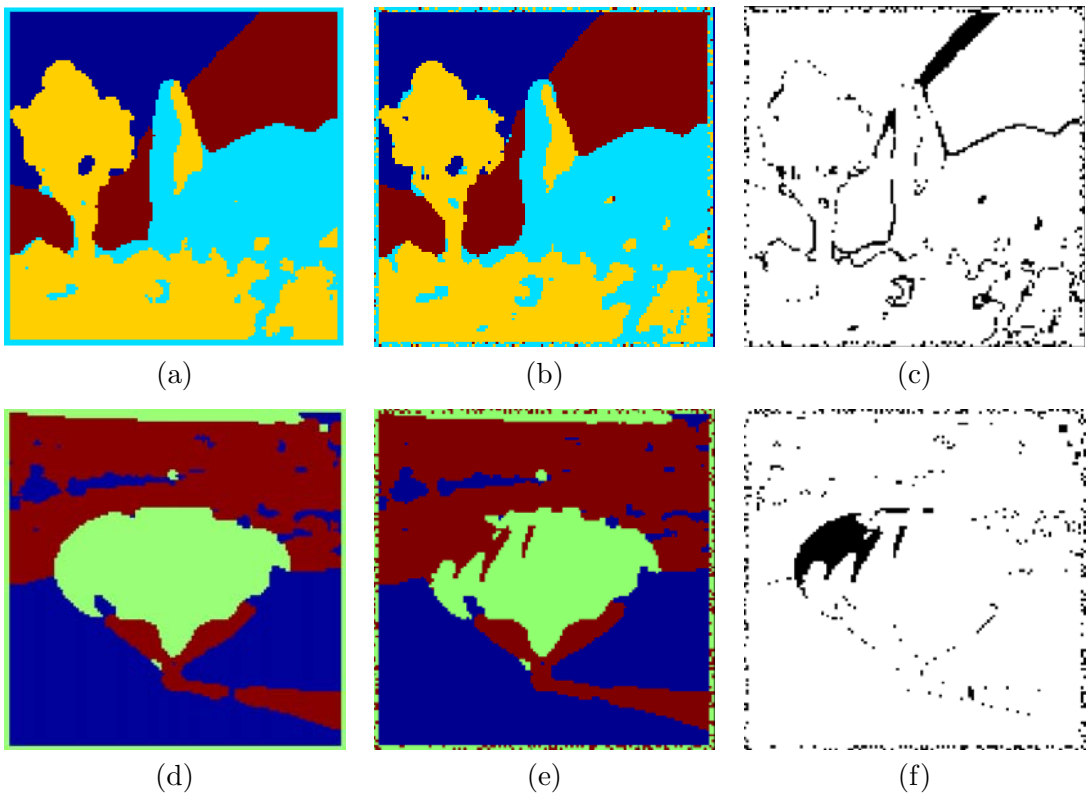
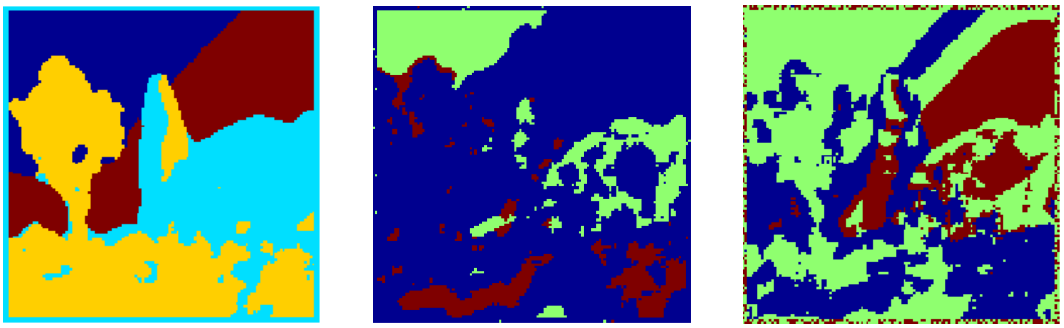


Figure 4.5: Comparison of ground-truth and 16bit fixed-point precision architecture. (a,d) Shows the ground-truth of the segmentation model for the 5th order neighborhood system. (b,e) Shows the result for a 16bit fixed-point number representation and calculation. (c,f) Differences (black) between ground-truth and fixed-point model. The fixed-point is located in the middle of the 16bit. Over- and Underflow covered by truncation respectively saturation.

ited precision of the 12bit fixed-point number representation. When comparing the ground-truth Fig. 4.6b and the segmentation result of the 12bit model/architecture variant depicted in Fig. 4.6b it becomes evident that the 12bit variant generates only a three cluster segmentation although four clusters are encoded within the MRF model and its corresponding 12bit architecture. This specific behavior is caused by the limited numerical 12bit fixed-point precision and the situation that two estimated prototypes q_ν have became identical. Because the model performs the comparison of the site-local empirical distributions (histograms) with the different prototypes sequentially it always assigns an appropriate site to the cluster, which is represented by the first of the identical prototypes. The comparison with the second of the identical prototypes will not generate a new comparison scoring and thus als not a change of the cluster assignment. This is the reason why only three different cluster marks are used by the 12bit model/architecture variant although four different clusters are principally available. Figure 4.7 shows the estimated prototypes from the simulation run, which has calculated the images of Figure 4.6. From this illustration of the prototypes it becomes apparent that class 3 and class 4 have

Architecture/Model Combination	Classification Rate [%]	[%] w. b.	Misclassification [# of sites]	Misclassification [# of sites] w. b.
Ground Truth	100	100	0	0
32bit	99.79	99.79	34	34
30bit	99.77	99.77	37	37
26bit	99.75	99.75	40	40
22bit	99.63	99.67	60	53
20bit	99.10	99.16	147	137
18bit	97.39	97.50	427	409
16bit	92.54	94.11	1222	965
14bit	87.07	92.98	2117	1149

Table 4.1: Segmentation Performance of different Architecture/Model combinations with respect to landscape image data. Color segmentation architecture with 5th order neighborhood system and 8 equally spaced bins. Abbreviation *w.b.* means without boarder sites.



(a) (b) (c)

Figure 4.6: Comparison of ground-truth and 12bit fixed-point precision architecture. (a) Shows the ground-truth of the segmentation model for the 5th order neighborhood system. (b) Shows the result for a 12bit fixed-point number representation and calculation. The fixed-point is located in the middle of the 12bit. Over- and Underflow covered by truncation respectively saturation. (c)-(d) Exemplary model-convergence defects.

identical prototypes.

Figure 4.8a-d shows different intermediate results of the 8bit model/architecture variant. The numerical precision of 8bit is no more sufficient for the unsupervised segmentation model and its massively parallel processing architecture to calculate usefulness segmentation results. Furthermore the limited 8bit fixed-point precision causes the model/architecture variant to oscillate between completely different cluster assignments of consecutive relaxation steps, i.e. the intermediate segmentation results of this specific model/architecture variant did not converge to a plausible

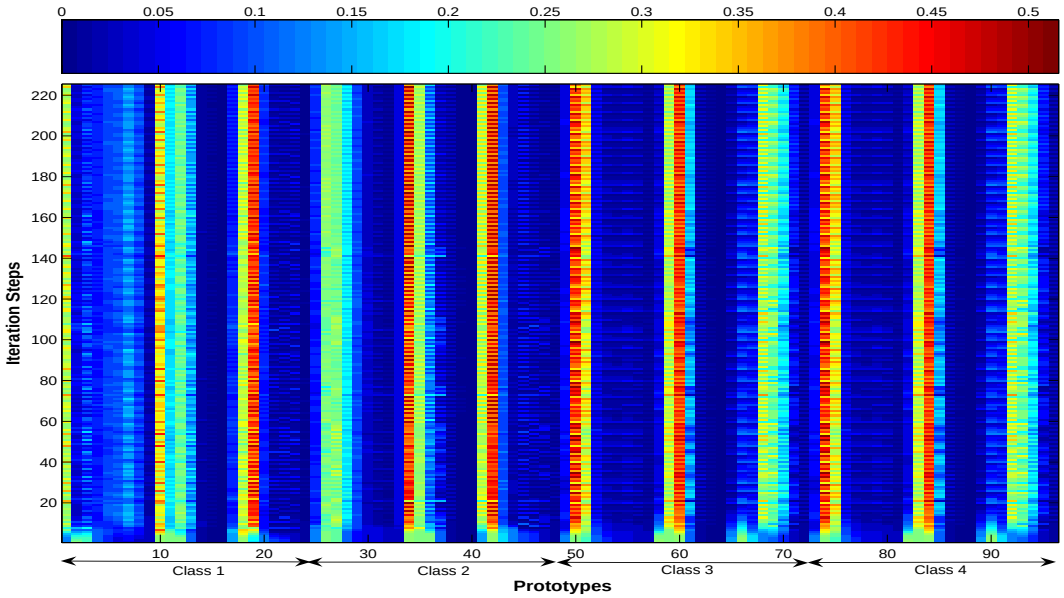
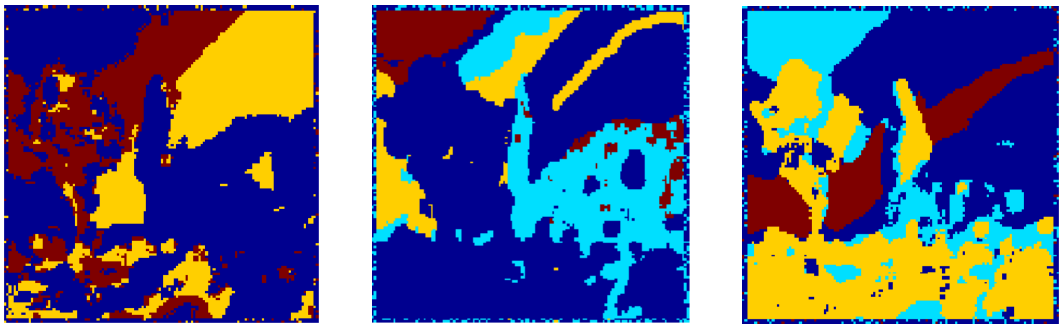


Figure 4.7: Estimated prototypes q_ν. Prototypes of the architecture/model variant with 12bit fixed point precision and with respect to the landscape image. Each class prototype is composed of probability distributions for the RGB channels. The probability values are encoded corresponding to the colorbar.



(a) (b) (c)

Figure 4.8: Model/Architecture divergence. Results of the segmentation model for the 5th order neighborhood system. The result are received for a 8bit fixed-point number representation. The fixed-point is located in the middle of the 8bit. Over- and Underflow covered by truncation respectively saturation. (a)-(c) Exemplary results of the model, which does not converge.

segmentation at all (see Figure 4.8). We notice that the segmentation behavior respectively the segmentation performance of the 8bit model/architecture variant is not surprising because we have to realize that only 4bit are available for the integer part of the fixed-point number and 4bit are available for the fractional part. Hence the model can utilize a max value of 15 for the integer part and a min

value of 1/16 for the fractional part of the 8bit fixed-point number representation ($2^3 2^2 2^1 2^0 . 2^{-1} 2^{-2} 2^{-3} 2^{-4}$).

During each simulation run of a specific architecture/model variant the simulation framework is able to display and store the complete state of the simulation model. Hence the simulation model and its intermediate results are completely observable and analyzable. The MSE and PSNR analysis depicted in Figure 4.9 is one possibility to further analyze the states of the simulation model and to receive insights in the convergence dynamic of a particular architecture/model variant. In this case the PSNR curve of the 16bit architecture/model variant gives rise to question how the PSNR curve-progression is explainable. A detailed analysis of the

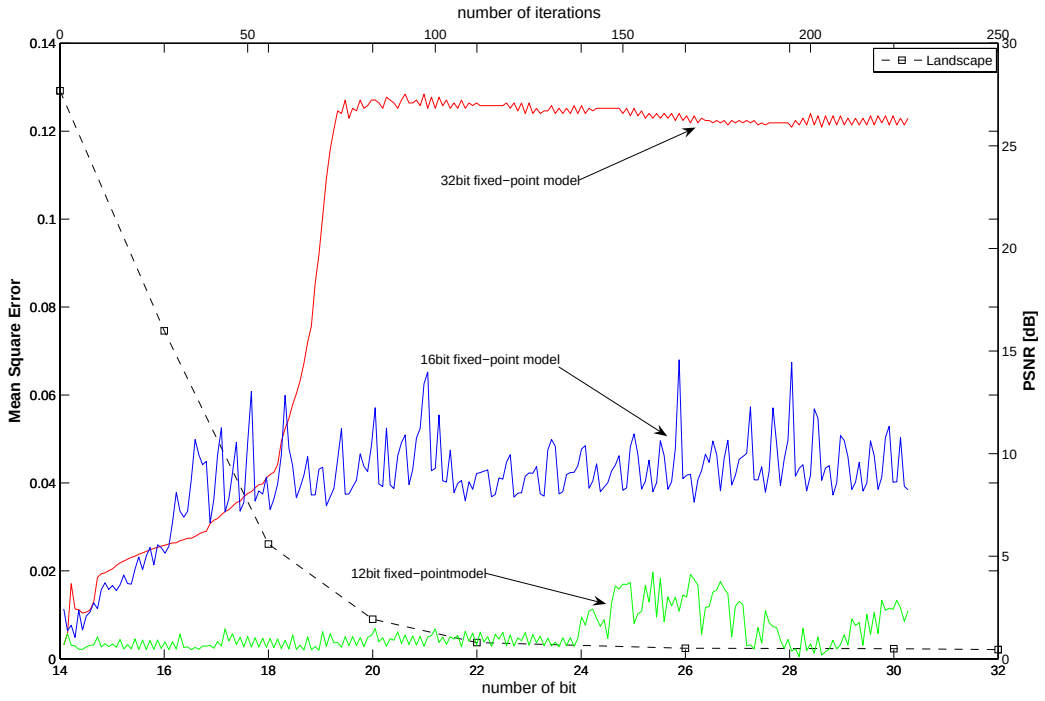


Figure 4.9: Means Square Error (MSE) and Peak signal-to-noise ratio (PSNR) analysis of fixed-point simulation runs (landscape image, cf. Figure 4.2a) of the segmentation model. MSE versus number of bit used for the model. PSNR versus number of iterations.

estimated prototypes (see. Figure 4.10) reveals that the limited numerical precision with truncation and saturation for over- and underflow causes the prototypes to change significantly between particular iterations steps. This induces several sites to change their cluster assignment. Exactly this behavior can be observed for the segmentation results at each iteration step. In contrast to the changing prototypes of the 16bit architecture/model variant the prototypes of the 32bit architecture/model variant (see Figure 4.11) are stable and hence in accordance with the PSNR curve.

Because the contemporary literature has not been investigated and reported on the fixed-point precision behavior respectively parallel processing performance of

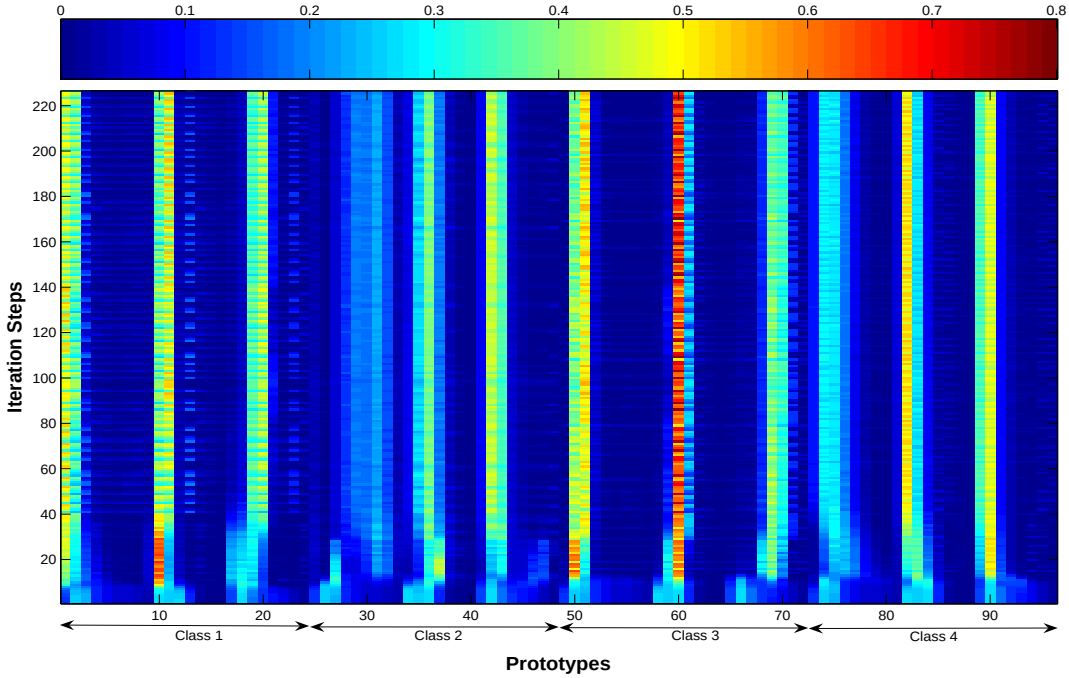


Figure 4.10: Estimated prototypes q_ν. Prototypes of the architecture/model variant with 16bit fixed point precision and with respect to the landscape image. Each class prototype is composed of probability distributions for the RGB channels. The probability values are encoded corresponding to the colorbar.

MRF based models, the herein presented insights and results are of far reaching relevance for any kind of fixed-point MRF model implementation. Based on the currently gained insights it can be expected that the required fixed-point bit-width becomes moderate - ranging from 16bit to 32bit - for low-level MRF image processing models formulated on regular site-grids, with a neighborhood system of 1st to 5th order.

4.5 Implementation Issues

Obviously, the software technical realization of the proposed simulation framework, whose structure is summarized and illustrated in Figure 4.1, raises various problems respectively questions. It is mandatorily necessary to identify and adequately deal with them, in order to realize a flexible and modular simulation framework for massively parallel MRF based processing architectures. We focus the discussion on the major concerns, which have been considered during the conception-phase and the following iterations of the implementation. The major concerns, which have been identified are:

- *Extensibility & Flexibility.* The current software-technical realization of the proposed MRF simulation framework has been operationally approved by

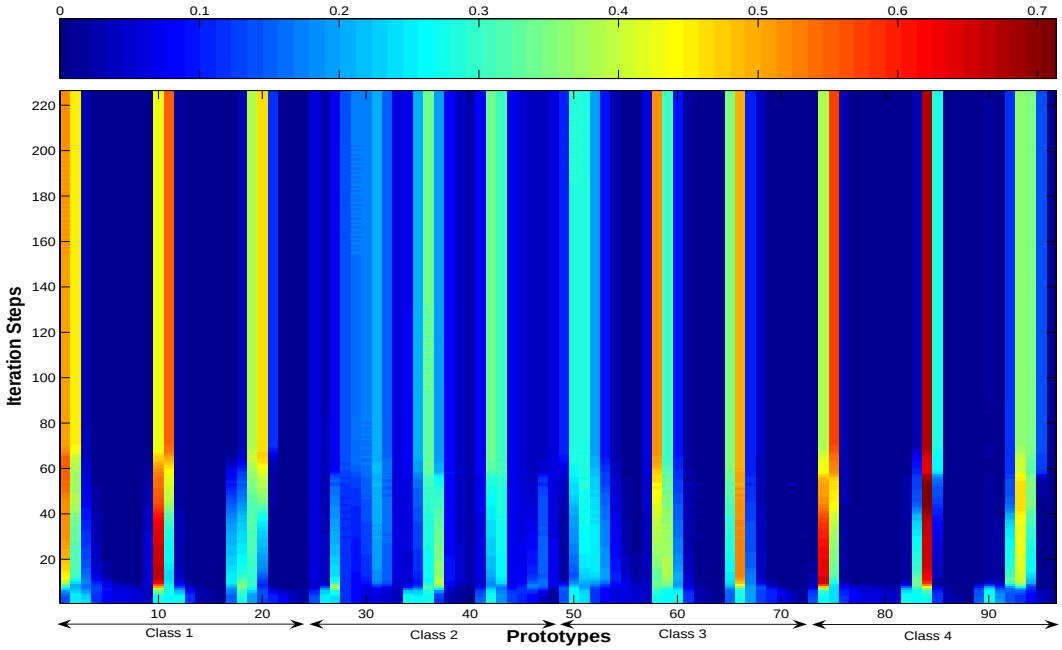


Figure 4.11: Estimated prototypes q_ν. Prototypes of the architecture/model variant with 32bit fixed point precision and with respect to the landscape image. Each class prototype is composed of probability distributions for the RGB channels. The probability values are encoded corresponding to the colorbar.

means of several artificial testing simulations and above all also by the intensive simulations of the two regarded image processing models. In spite of the status of the simulation framework reached until now, further improvements and enhancements, eventually caused by new MRF models, should still systematically be supported. Thus the software structure of the framework as well as the particular module collections have to be open and changeable. Furthermore all enhancement should be possible without changing large portions of the framework software and their overall conception or even changing the SystemC simulation kernel.

- *Efficiency & Scalability.* The MRF models, which already have been investigated and are intended to be investigated and simulated by the framework, possess distended site-grids, neighborhood systems of up to fifth order and energy functions. This leads to specific demands within the simulation framework in order to be able to generate and simulate such models within the framework. The demands are split up into pure memory resource-usages and processing-time resource-usages. Both demands have to be equally addressed by the simulation framework. Obviously, the memory resource-usages as well as the required processing-time for a particular simulation model directly affect the scalability capabilities of the simulation framework. Additionally, it is assumed that standard and cost-efficient personal computers can be used to

simulate the contemplated class of MRF models and no expensive supercomputers.

- *Compatibility.* The proposed simulation framework is built up on the open-source SystemC kernel and environment, which is constantly evolved by the improvements of the community and its developers. Consequently, it has to be ensured that the simulation framework is compatible with new versions of SystemC or at least adaptable to either obsolete or new features. In view of this constraint it is important that all necessary changes are kept small without affecting fundamental software-structures of the framework.

- *Self-Checking.* The MRF simulation models under consideration as well as the models, which are intended to be addressed by the simulation framework, are generally large site-grids with neighborhood systems of up to the fifth order. This leads to complex site-grid constellations, which are not obvious to overlook and debug. Thus it is mandatorily required to integrate a self-checking functionality into the simulation framework as well as into its particular modules.

With regard to the proposed simulation framework the above mentioned concerns become realized by means of the following implementation approaches and features. They are described in exactly the same chronological order as mentioned above.

- The extensibility and flexibility of the MRF simulation framework significantly profits from the modular structure and the pooling of functionally similar modules into distinct libraries. Furthermore the *Simulation-Model Generator* as central unit and exclusive instance to generate simulation models additionally supports the extensibility and flexibility of the framework. This specific arrangement guarantees that only one single interface toward the simulation-model generator has to be fostered to keep the framework operational, i.e. any kind of extension and change only has to be compatible with this specific interface. Thus a new module can systematically be added to the corresponding collection just by guaranteeing a compatible interface; if, in any case, it should not be possible to establish a compatible interface for such a new model, an interface-wrapper around this module can solve all kinds of problems. In addition to this appropriate wrapper, which establishes the interface to the simulation-model generator any programming language can be chosen, which together with C++ is compilable in all the current development environments. This offers a far-reaching flexibility to implement new simulation modules.

- The efficiency and scalability of the simulation framework is mainly realized by the specifically tuned simulation modules. Each single simulation module becomes conceptualized and implemented with respect to an optimized memory- and processing-time usage. Furthermore the pooling of several modules with an identical functionality into larger modules - e.g. cell-cluster modules and wiring modules - increases the event-driven simulation performance because the dynamic event-list data-structure is kept small to largely prevent memory fragmentation. In contrast to that, a flat simulation model structure with its large number of ungrouped modules (cf. Table 3.5, 3.4) would even more lead

to a tremendous data transfer between the CPU and the external memory without efficiently using the CPU-cache. Additionally, each module generates its internal sub-modules and variables, dependent on its own concrete functionality within the MRF simulation model. Hence, the memory usage is optimized for each specific MRF model. The implementation of the central components in C++ also supports the efficiency and scalability of the simulation framework. The complete simulation framework likewise profits from the fact that it is built up on open-source SystemC, which becomes steadily improved by the community.

- The compatibility of the simulation framework with respect to the eventually changing SystemC part, to its large extent is ensured by the modular organization. If a significant SystemC change will take place to become certain features obsolete and new features are introduced, only specific modules are affected. This is caused by the fact that the module collections are functionally disjunct and a SystemC change will not equally involve all modules. It is therefore possible to adequately deal with SystemC changes as only small adoptions of the framework are necessary at a certain time.

- The self-checking capabilities of the MRF specific simulation framework, which are focused on the site-grid structure and the neighborhood wiring, are realized by the simulation modules as well as by the simulation kernel during initialization. Within the cell simulation modules algorithms are embedded, which sort the particular cells to assign a unique number to ech cell. This number equals the value, which is received by the standard row-by-column scanning. The consistency of the numbers as well as the maximum value correctly indicates the generated cell modules, constituting the grid gantry. Furthermore each cell cluster ensures a correct instantiation and wiring of the corresponding sub-cell cluster. Finally the simulation kernel scans all ports during the instantiation of the model to identify unconnected ports of the modules. These mechanisms all together guarantee a completely generated and wired cell-grid.

4.6 Relation of Thesis Parts

The main parts of this thesis, including the fundamentals of Markov Random Field theory (Chapter 2), the building blocks of the system-architecture template (Chapter 3), the simulation-modules of the simulation framework and the graph-theoretical representation of the MRF device in the design framework (Chapter 5), are closely linked to each other, are interdependent and together form a seamless simulation- and design-environment for massively parallel hardware architectures of Markov Random Field based image processing systems this thesis is dealing with. The interplay of the different thesis parts is, as already introduced in Chapter 3, represented by the *relation of the thesis parts*, which is shown in Figure 4.12. So far we have established the part of the relation diagram, which represents the coherence of the universal constituents and the different building blocks derived out of them. This has defined the step toward the system-architecture template, for massively parallel MRF-based processing devices (see Figure 4.12). Additionally, this chapter deal-

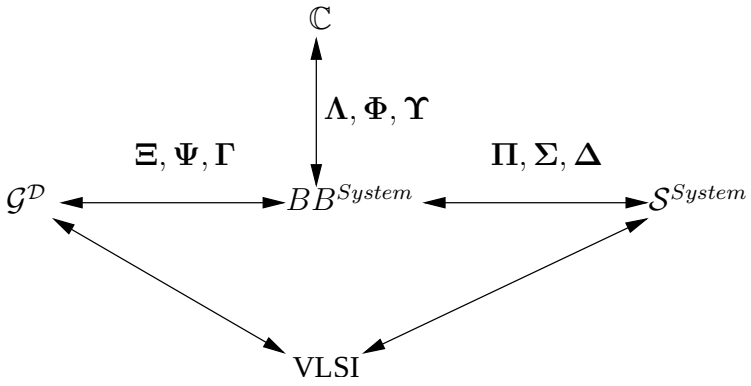


Figure 4.12: Established congruence relation.

ing with the simulation framework has established the relation of the architectural building blocks to their corresponding simulation modules.

Recapitulating, we have up to now established the following parts and relations of particular thesis parts: At first the fundamentals of Markov Random Field theory and a structure of models, which are formulated on regular two-dimensional site-grids. Based on this MRF model structure, we have defined the architecturally uncommitted universal constituents, which represent the first part of the relation. The second part comprises the building blocks $\mathcal{BB}^{System}$ of the system-architecture template. This part is connected with the universal constituents and vice versa by means of the corresponding mappings. The third part is represented by the simulation modules, which are linked to the architectural building blocks by the corresponding mappings. This forms the second relation. The consecutive chapter will present the final part and third relation.

4.7 Summary

This chapter has presented a novel simulation framework for massively parallel image processing architectures, which are based on the processing principles of Markov Random Fields. The simulation framework has been conceptualized and tuned to support the specific simulation requirements that are prevalent in hardware-relevant and massively parallel MRF processing architectures. Furthermore, the overall framework has been designed to systematically simplify, structure and centralize the compilation procedure of different MRF simulation models. Hence, the set-up procedure, the model refinement and the generation of the simulation model has been automated in large parts by the introduced simulation framework. In order to prevent a time-consuming and proprietary development from scratch of an event-driven simulation kernel as well as from elementary simulation capabilities, we have decided to built our simulation framework up on the open-source SystemC [81] class-hierarchy. This offers the advantage of allowing our simulation framework to profit from further developing the simulation kernel within the open-source community as well as our simulation framework to be easily and cost efficiently used within the community.

In Section 4.1 we have introduced the proposed simulation framework. The modular and open software structure as well as the principal steps through the framework in order to generate and simulate a MRF model have been presented. Additionally, each of the framework components has been shortly described and qualified either as a core module component or as a supporting module component. Within the following sections of this chapter only the core modules have been further discussed. All core modules of the simulation framework have been classified in Section 4.2 according to the scheme of Chapter 3, which discriminates between *Topology & Structure*, *Processing* and *Control* representing simulation modules. This section has furthermore formally defined the different simulation module collections and also formally established the coherence between the building blocks $\mathcal{BB}^{System}$ and the particular simulation module collections. Thus we have mapped the architectural building blocks to corresponding simulation modules to model and simulate the massively parallel MRF architectures, which have previously been derived in Chapter 3. For each kind of simulation module, we have also presented prototypes to illustrate the principle structure of the modules, which is divided up in a structure part, an initialization part and a functionality part (cf. e.g. Prototype 4.1). Section 4.3 has presented the particular steps, which have to be executed to trigger the simulation framework for generating a MRF model and for starting the simulation. These steps have been divided up into two major phases: During the first phase the user has to do some manual coding and arranging of modules. In contrast to this the simulation framework executes its operation automatically the second phase, i.e. the defined MRF model is represented by the corresponding simulation model, which becomes executed to receive the simulation results.

The relation diagram of the different thesis parts, which illustrates the interdependencies of the central thesis parts as well as it gives an overview of the main thesis-topics has been presented in Section 4.6. This section has discussed the branch of the relation diagram, which represents the mapping of the architectural building blocks to its corresponding simulation modules. The following Section 4.5 has commented on essential implementation issues of the proposed and software-technically realized simulation framework. In summary, the MRF simulation framework features itself by means of a flexible and modular framework arrangement, which rests upon the SystemC kernel, parametrized modules, extensible module collections and a self-generating capability of the simulation models. Various simulation results, presented and discussed in Section 4.4, have demonstrated the capabilities of the proposed simulation system. These results have underpinned our claim for a flexible simulation framework for massively parallel and hardware-relevant processing architectures, which are based on MRF processing principles. The selected results of Section 4.4 are completed with additional simulation results, collected in Appendix A.

4.8 Bibliographical Comments

The topic hardware-relevant simulation of massively parallel processing structures, which are based on MRF/CNN principles until today is only partially addressed within the CNN community. As the CNN community has its origin [35] [34] in

biologically motivated models, it was the logical deduction to only look at analogue implementation technologies [18]. Additionally, the research focus is currently set to mimic the biological signal shape by means of analogue signals. Thus only single cell models or very small cell-cluster structures have been simulated and investigated. Consequently, the demand for supporting simulation frameworks was not present, as the simulation models of the particular cells or small cell-structures have been coded by hand and adjusted to the specific analogue semiconductor technology.

Disregarding the current situation, Krieger and Chua [100] conceptualized in *ASIM* a supporting simulation environment 1990, which was specialized to work with analogue Cellular Neural Networks. The ASIM simulation environment provided an efficient simulation approach compared with traditional analogue simulation approaches, which are very slow and cumbersome. The required computations performed by ASIM are directly derived from the nonlinear differential equations, describing the processing at each node. Cell-cluster and small systems are merely represented by the internal data structure of ASIM. The modeling and simulation of large CNN/MRF systems is not supported by this system. In addition, a direct link to the complete architecture and the implementation technology is missing. This is a typical feature and at the same time represents the major drawback of ASIM and other introduced simulation environments [104]. The recently presented simulation environment *SCNN* [108] [109] does also not focus on detailed and hardware-relevant architecture modeling capabilities. Furthermore SCNN does not address the massively parallel processing dynamic of CNN/MRF based processing devices. The same applies for the SIRENA [30] simulation system. The massively parallel processing dynamic of CNN/MRF systems is systematically investigated and visualized by Hanggi et al. [66]. Whereas Matei and Goras [117] regarded a time-discrete evolving simulation setting, similar to event driven approaches, for 1D CNN models. Already Schikuta [145] followed up a data-parallel simulation approach, which tries to implement event-driven simulation capabilities. But again detailed architectural models are not systematically supported, so that the direct link to VLSI implementations and the massively processing dynamic is missing. The combined CAD system of Carmona et al. [41] only supports a behavioral modeling and simulation approach, too. Consequently, the link to hardware-relevant architecture variants and the VLSI implementation technology is once again missing.

Chapter 5

VLSI Design Framework

This chapter introduces a novel, completely semiconductor-technology independent VLSI design framework for Markov Random Field based processing devices, which is specifically tailored to address the distinct system design needs of these massively parallel hardware architectures. The proposed VLSI design framework rests upon two essential and uniquely combined design principles in order to fulfill the demand for a completely semiconductor-technology independent VLSI design framework for massively parallel Markov Random Field based processing devices. The first design principle claims that the level of abstraction for the design descriptions has been shifted from the currently predominant VLSI specific register-transfer-level (RTL) to a hardware-neutral behavioral and specification-like level. This first design principle systematically supports the semiconductor-technology independent description of MRF devices as well as the complexity handling of MRF architecture descriptions. The second design principle claims that the synthesis of the global MRF device topology is addressed as a completely independent design step. Furthermore this topology-synthesis procedure represents the very first step toward any MRF specific hardware architecture in the advocated VLSI design approach introduced in this chapter. This second design principle organizes and simplifies the architecture synthesis process insofar as global structures, which establish the overall architecture-topology, did not mix either with local control blocks or with local data-path blocks.

The continuously impressive progress of semiconductor-technology [82] [84] and the resulting availability of advanced digital design technology families pave the way for highly complex and densely integrated System-on-Chips (SoCs), and thus in a similar way also for massively parallel Markov Random Field based processing devices. This optimistic statement at least seems correct when viewed from an exclusive semiconductor technology specific point of view. However, with respect to the field of well-founded and systematic VLSI design methodologies this promising prospect soon ceases to be valid. The design methodologies established [111], which are qualified to support the new and advanced technologies, are not able at all to adequately handle the design of highly complex and distributed processing architectures. Additionally, a technology-independent design entry is not supported. Thus, a suitable VLSI design methodology for Markov Random Field based processing devices is still missing, too. The reasons for this deficit cropping up in the

field of VLSI design methodologies are multifaceted and multi-layered. However, the essence of the problem lies in the abstraction-level of the design descriptions [83]. Today the design description is realized by one of the two IEEE standardized hardware description languages (VHDL, Verilog), whose customized capabilities to represent hardware designs are mainly limited to take place on the VLSI specific register-transfer-level [3]. Exactly this abstraction-level is not adequate and efficient enough to describe large, complex and distributed systems, which are principally possible with todays' multi-million gate VLSI semiconductor technologies. This situation is well known within the EDA-community as *design productivity gap* [84]. Due to this fact a paradigm change with respect to design descriptions and representations [28] [51] is mandatorily needed to overcome this limitation and to establish new design methodologies. In general VLSI design approaches, which start from hardware-neutral or behavioral descriptions are called *High-Level Synthesis (HLS)* or sometimes *Behavioral Synthesis (BS)* [51].

The VLSI design approach we propose for massively parallel MRF hardware architectures and which we realize by means of the novel VLSI design framework, performs exactly this paradigm change within the design methodology. The design representations are shifted from the pure RT-level to hardware-neutral and specification-like design descriptions. Foremost this step allows it to systematically capture and design MRF devices with an appropriate size and to realize them with different semiconductor-technologies. Due to this paradigm change the established RTL-based design flow becomes altered. A direct technology synthesis and mapping is no longer possible as we have left the RT-level, which only allows a straight technology-synthesis and mapping. Consequently, the hardware-neutral and specification-like MRF descriptions first have to be parsed and analyzed to extract the relevant information. Secondly, this information becomes further processed in order to expand graph-structures, which represent the architecture of the MRF model in a hardware-neutral and technology-independent manner. Finally these graphs are compiled into synthesizeable RTL-representations to drive established and approved RT-level VLSI design flows. In contrast to already proposed High-Level design approaches, where the overall design topology is implicitly defined by the inherent structure of the data- and control-paths, we advocate a design approach, in which the graph representation of the architecture topology is generated at first and as part of a separate design step. All other architectural representations, generated in the following design steps, are linked with and embedded in this topology.

The following Section 5.1 describes the structure, the different components and the arrangement of the components among each other characterizing the novel VLSI design framework. Section 5.2 defines the guidelines, the definitions of the parts and the relevant features of the canonical design representation for Markov Random Fields. Section 5.3 represents the complete design flow of Markov Random Fields supported by the VLSI design framework. In Section 5.4 we describe the algorithms, which generate a IEEE-conform hardware description from the abstract graph representation. Section 5.5 demonstrates the capabilities and the technology independence by means of different prototypical FPGA implementations. Section 5.6 comments on concrete implementation issues of the design framework. Section 5.7 finally classes the design framework within the relation-diagram of thesis parts.

5.1 Design Framework Overview

The proposed technology independent VLSI design framework for massively parallel hardware architectures of Markov Random Field based signal- and image-processing models is software-technically realized as a modular and thus extensible design environment. This modular and tool-box like environment is arranged around one single data structure, called *canonical design representation*, which serves as the only central data-basis. Thus the canonical design representation apparently stands for the integrative design and information container of the MRF specific design flow.

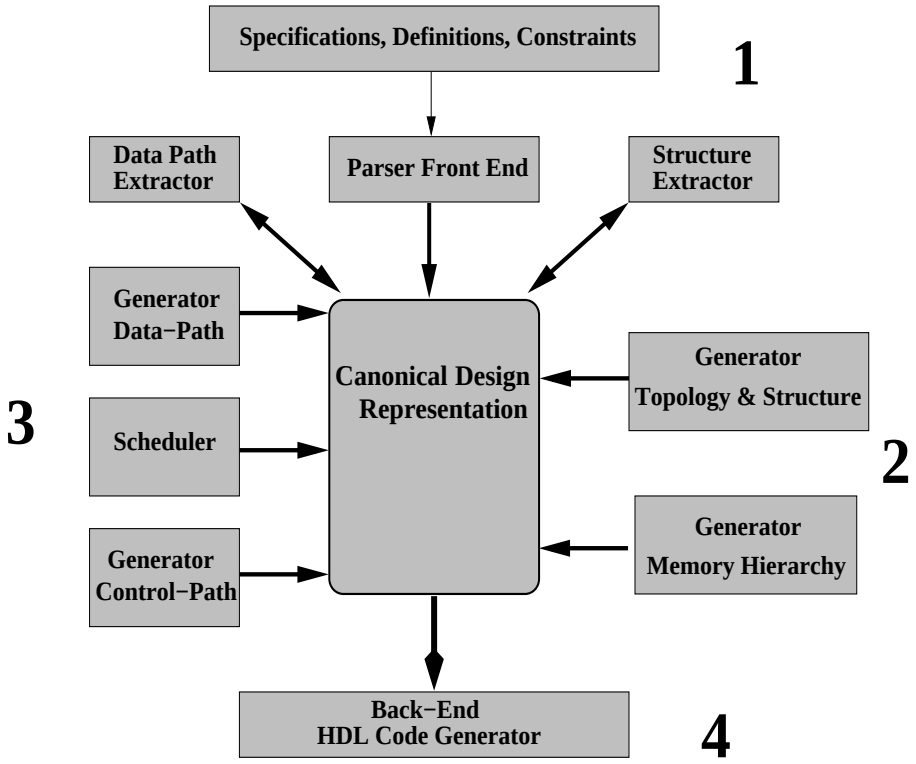


Figure 5.1: Arrangement and components of the proposed VLSI Design Framework.

Two different data-type groups form the foundation of the canonical design representation. The first data-type group of the VLSI development process contains the complete specification of the Markov Random Field, which has to be realized. This specific information has been parsed and extracted from several descriptions and specifications at the beginning of the design flow. Whereas the second group of data-types covers the actual version of the generated massively parallel MRF hardware architecture. Obviously merely some parts of the architecture are represented at the beginning of the generation process by the second group of data-types. The architecture is successively completed with the progress of the generation process. Several different modules are arranged around the central canonical design representation and each of these modules either generates or systematically manipulates the MRF design representation in order to finally result in the intended massively parallel VLSI architecture. This design framework configuration and all its single

components is diagrammatically depicted in Figure 5.1.

The modular arrangement of the proposed VLSI design framework offers several vital advantages for future extensions of the framework. Due to the modular characteristic of the VLSI design framework, arranged around the central and integrative design data basis, subjoining additional modules for the design environment, which generate, manipulate, restructure and optimize the canonical design representation, are systematically supported and were included in the conception of that VLSI design framework. Consequently, the design framework available is at the outset not restricted to the sub-class of MRF models, which is investigated in this thesis (cf. Definition 2.10 and Section 2.5), but is rather open, including the corresponding extensions, to other MRF classes. With a corresponding parsing front end, every grammatically well-defined specification-, programming- and modeling language is suitable as input language for the design framework. Apparently the proposed VLSI design framework is principally not limited to one specific input language. The same flexibility applies to the back-end code generation part. Currently the focus of intention is exclusively set on IEEE standardized HDL languages such as Verilog or VHDL, because only these two languages systematically support later technology-specific VLSI design implementation steps and industrially-approved tape-out procedures.

Essentially, the MRF specific design flow by means of the proposed VLSI design framework passes through four main steps (cf. Figure 5.1) and is set-up as follows: Step **1** parses the specifications, descriptions and constraints of the Markov Random Field model to be developed. In the following this information is processed and stored in the first data-type group within the canonical design representation. Step **2** analyzes and processes the stored data of step **1** with regard to structure and topology information in order to synthesize the topology defining gantry of the MRF architecture. The MRF architecture-gantry is set up of the building blocks $\mathcal{BB}^{TS}$, which have been described in Section 3.2.1. Step **3** generates the control-blocks $\mathcal{BB}^{CT}$, defined in Section 3.2.3 as well as the data-paths $\mathcal{BB}^{PC}$, defined in Section 3.2.2 of the MRF architecture. As a next step these parts are embedded in the MRF topology earlier defined, i.e. the building blocks $\mathcal{BB}^{CT}$ and $\mathcal{BB}^{PC}$ are embedded in their corresponding distinct site hulls. This third step finally completes the MRF architecture, which is now completely represented by the canonical design representation. In step **4** the canonical design representation of the compiled MRF architecture is analyzed in order to generate synthesizeable HDL source-code, which will be used for the following standard VLSI implementation process.

So far a basic set of modules has been implemented and integrated together with the canonical design representation to form a functioning realization of our proposed VLSI design framework with the previously described main features and an overall structure as shown in Figure 5.1. Each part of the design environment will be shortly described in the following, listed exactly in the chronological order they are used in during the design flow. This will conclude the introduction and overview of the proposed VLSI design framework for massively parallel VLSI architectures of Markov Random Fields. The details of the design flow, with all different steps, will be exhaustively explained and further commented on in Section 5.3, after having finalized the formal presentation of the required graph-theoretical definitions and foundations in Section 5.2.

Overview Framework Modules

The following discussion shortly introduces the central components of the proposed design framework for massively parallel architectures of MRF based signal- and image processing models. This presentation concludes the overview of the proposed design framework and prepares for the following discussion on the graph-theoretical representation.

Parser Front End

The parser front end pass reads and analyzes the different specifications and descriptions of the MRF image processing model, which is intended to be realized as a VLSI device. Several parameters have to be made available to sufficiently describe and define a MRF device. These parameters comprise the size of the site set, its arrangement, the neighborhood, the memory hierarchy and the processing functionalities, to name just the essential ones. After a first syntax- and consistency check was successfully performed on the different input sources, the MRF defining parameters are stored in the data container $\mathbb{C}^P$ (cf. Definition 5.1) within the canonical design representation. The following design steps further process these information to step by step compile a graph-theoretical representation of the massively parallel MRF architecture.

Structure & Topology Extractor

The structure & topology extractor component processes the MRF parameters in $\mathbb{C}^P$ stored before and collects all structure- and topology-relevant data, works this data up and finally embeds the resulting information as structure-core data container $\mathbb{C}^{ST}$ (cf. Definition 5.2) into the canonical design representation. This advocated approach, where generic structure respectively topology information is preprocessed and stored in a structure-core data container, ensures an advantageous flexibility for the subsequent topology generator stage of the proposed design flow. Different topology generator modules, producing different structure and topology characteristics, can consequently fall back upon a common data structure with a well-defined data interface.

Data- and Control Path Extractor

Likewise the previously described *Structure & Topology Extractor*, also the *Data- and Control Path Extractor* component systematically scans the information stored in $\mathbb{C}^P$. But in contrast to the structure & topology extractor pass, this process solely collects and pre-processes all information on data- and control-flow definitions and dependencies. With respect to MRFs data- and control-flow dependencies are essentially encoded by the energy-functional, the parameter estimation procedure as well as by the optimization method. The worked up data- and control-flow information is stored back in $\mathbb{C}^P$. Again, this information storing scheme allows different data- and control-path generators to fall back upon common data structures with well-defined data interfaces.

Structure & Topology Generator

The structure and topology generator pass represents the very first step toward a systematic synthesis approach of massively parallel MRF hardware architectures. Hitherto this component scans the structure and topology information filed in the structure-core container $\mathbb{C}^{ST}$ and expands the corresponding topology representing graph. Therefore the concrete expanding sequence and thus the outcome of this processing step - the topology representing graph - obviously depends on two factors. Firstly on the specifications, which have been extracted and stored, and secondly on the strategies to generate the representing graph.

Memory Hierarchy Generator

This component of the proposed framework creates a graph representation of the required distributed memory hierarchy (cf. Section 3.1). This specific generator also makes use of the stored information of the structure-core data container to synthesize a memory hierarchy representation. Different memory hierarchy generators are obviously able to synthesize memory hierarchy representations with certain peculiarities to flexibly match with numerous requirements and VLSI technologies.

Data-Path Generator

The data-path generator pass represents the processing step within the design flow, which systematically creates the corresponding graph theoretical representation of the MRF data-paths as part of the canonical design representation. During this process the container, which holds the data-path relevant information is scanned to receive the required data-path information and its dependencies. The additional sub processing steps completely resolve all existent data dependencies to form a connected graph, which completely encodes the data-path. Furthermore several specific features establish essentially required links to the control path structures in order to form a well-defined and hardware-relevant processing scheme at the end of the design flow.

Control-Path Generator

The control-path generator pass represents the processing step as part of the proposed design flow, which systematically creates the corresponding graph representation of the control-paths necessary for the previously generated MRF data-paths within the canonical design representation. This processing step also establishes the links to the data-path structure in order to embed these operations into the control-flow.

Scheduler

So far the generated data-paths as well as the generated control-paths both solely model data and control dependencies but not a time discrete processing scheme required for clocked digital devices. Therefore the scheduler pass of the design flow assigns each operation of the data- and control flow to discrete time slots respectively steps, simultaneously respecting all data and control dependencies. As a result a

time-discrete processing sequence is finally defined, which is suitable for purely digital and clocked VLSI architectures.

FSM Embedding

Besides the control structures for the data path several additional control schemes, realized by finite state machines, are required to guarantee a correct and deterministic overall processing sequence of the MRF device. These finite state machines are generated in this pass and finally all control structures, defining the processing sequence of the MRF device, are embedded into the canonical design representation together with the other architecture representing parts.

Canonical Design Representation

The canonical design representation is the basis and above all the central integrative data container of our proposed VLSI design framework. As already partly mentioned when describing the framework components, during the first phase of the design flow different data containers are instantiated within the canonical design representation and filled with the parsed Markov Random Field specifications. Thus a complete specification of the specific Markov Random Field, which is intended to be realized as a VLSI device, is available in an uniform data representation and with well-defined data interfaces after the first design phase. These data containers are exclusively designed to hold specifications of the Markov Random Field device. Therefore they represent by no means parts of the massively parallel VLSI architecture. The second design phase is based on these data containers and their stored information. During this second design phase the different components generate - sequentially and step by step - a graph-theoretical design representation $\mathbb{G}^D$ (see Section 5.4 and Definition 5.6) of the massively parallel Markov Random Field processing device. Such a central and uniquely arranged data representation offers some fundamental advantages regarding flexibility, extensibility and portability. This is what characterizes the software implementation of the VLSI design framework and the proposed design flow itself. This main concept structures the definition and inheritances of the software classes, their methods, design patterns and polymorphisms. In Section 5.2 directly following the features and formal definitions of the canonical design representation as well as exhaustive comments on the requirements and principles, which lead to these definitions, are presented.

Back-End - HDL Code Generator

Finally the back-end code generation component analyzes the graphs within canonical design representation and generates the source code of this architecture in a standardized hardware description language. In order to be compatible and consistent with advanced VLSI processing steps like synthesis, technology mapping, placing & routing and design rule checking (DRC) the back-end module currently supports the IEEE standardized HDL languages VHDL. However, not the complete HDL language complexity is supported by the back-end code generator, because merely the synthesizeable language subset is required for VLSI implementations. The simulation-specific HDL language ingredients are currently of no importance

for the back-end code generation pass and have been excluded.

The next section provides a detailed introduction of the canonical design representation. Starting with fundamental principles of a MRF specific design representation suitable for high-level design flows, the discussion continues with the presentation of elementary data container and formal graph definitions. Furthermore several qualities of the different graphs will be derived. These particular graph properties support specific VLSI design steps, which are indispensable to realize a massively parallel processing device for MRF based image processing models.

5.2 Canonical Design Representation

A flexible and extensible *Canonical Design Representation* (CDR) has to fulfill some general requirements to tackle abstract design representation issues and concrete VLSI implementation problems of massively parallel processing architectures, which are derived from a Markov Random Field based statistical image- and signal processing approach. So far neither any systematic VLSI design approach or high-level synthesis flow has been proposed for this kind of highly complex and massively parallel digital systems, nor has this design automation problem at all been addressed within the communities. This is why we have specified and developed from scratch a new canonical design representation matching exactly this purpose. Some of these guidelines are generally valid for any high-level design approach [51] [29] and other principles are specifically formulated to represent of massively parallel Markov Random Field architectures. Our advocated novel canonical design representation is established on the basis of the following fundamental guidelines and principles:

Principles 5.1 (MRF specific CDR)

- The canonical design representation serves as an integrative container, which at any time of the design flow contains the complete design specification and furthermore the generated abstract, massively parallel architecture. *The design representation is said to be design-integrative and complete.*
- The canonical design representation is completely independent of the concrete languages and their grammatical forms, which are used for the design specifications and design descriptions. *The design representation is said to be language-neutral.*
- The canonical design representation possesses the capabilities to model and represent all language constructs, which are used to specify and describe the Markov Random Filed processing device. *The design representation is said to be language-expressive.*
- The canonical design representation by itself does not directly or indirectly constrain the structure and the outcome of the massively parallel architecture. *The design representation is said to be architecturally-neutral.*
- The canonical design representation possesses a form and data-structure implementation, which allows the back-end task of automatic HDL code generation

to systematically be conducted, whereas the HDL code generation process is focused on the synthesizeable HDL language subset. *The design representation is said to be HDL-code-generation-appropriate.*

- The canonical design representation and its underlying data-structure is designed and implemented to meet the requirement of scaling with large MRF systems. *The design representation is said to be design-complexity-appropriate.*

The first guideline states that the initial design specification as well as the current version of the massively parallel MRF system architecture is at any time of the design flow represented within this novel canonical design representation. Thereby the encapsulated information among others comprises the initial design specification, generated design representations of the different architectural building blocks, all kinds of dependencies between different design parts and various embedded constraints to guide the VLSI back-end process of synthesis and placing & routing.

The second guideline implies that the design representation is completely decoupled from specific language features and specification verbalizations. Merely the the specification information and elementary definitions are extracted from the input specifications. Thus the parsing module has to ensure that these qualities are totally extracted, regardless of any concrete specification and description language.

The following guideline expresses that the canonical design representation is expressive and also abstract enough to model and represent the extracted qualities of the MRF specifications. Thus the canonical design representation must at least have the same expressiveness as the input specification language.

Finally the next guideline states the important fact that only the generation and transformation algorithms, systematically applied to the canonical design representation, determine the concrete structure and form of the massively parallel MRF processing architecture and definitely not any inherent features and constraints of the canonical design representation. The fact that all different modules, arranged around the design representation, influence and define the architecture further stresses this point.

The second to last guideline describes another essential feature namely the concrete internal realization of the design representation. The underlying data-structure of the canonical design representation is implemented in such a way that the procedure of hardware-description code-generation in VHDL can systematically be conducted and automated by appropriate algorithms.

The last guideline finally states that the canonical design representation and its underlying data-structure is designed to meet complexity constraints of ultra-large massively parallel MRF processing architectures. To substantiate this statement, the canonical design representation has been designed to handle complete MRF architectures of realistic and industrially relevant size - a quadratic size of 1024 by 1204 processing elements has been successfully generated. But this size does not represent the inherent limitation of the canonical design representation and its underlying algorithms or data-structures. Additional features of the data-structure such as the partial collapsibility and also expandability of the design representation and the principle capability of the algorithms to work on these structures, in the future allow larger MRF architectures.

In the following paragraph, we define the set of graph theoretical representations $\mathcal{G}^{System}$ for the set of architectural building blocks $\mathcal{BB}^{System}$. The graph representations $\mathcal{G}^{System}$ are also organized along the organization scheme, introduced in Chapter 3, and thus we distinguish between *topology & structure* representing graphs $\mathcal{G}^{TS}$, *processing functionality* representing graphs $\mathcal{G}^{PC}$ and finally *control functionality* representing graphs $\mathcal{G}^{CT}$. This specific organization scheme, applied to the various graph representations $\mathcal{G}^{System}$, systematically supports and simplifies the graph generation process itself as well as the back-end HDL code generation process because topology, processing and control graphs are clearly separated and can thus be treated independently of each other by using specific graph-expansion and code-generation algorithms.

At first the elementary data containers of the canonical design representation will be shortly introduced to explain the algorithms of the VLSI design framework in the upcoming discussions of this chapter.

5.2.1 Elementary Data Container

In order to establish a flexible and expandable data-interface between the parser front-end and the different graph-generation components of the VLSI design framework, two elementary data container, which store the complete design specification of the MRF, have been defined and incorporated into the canonical design representation. This arrangement allows it to change the specific parser front-end and thus the specification language without altering any other consecutive steps of the design flow, since solely the parser front-end process has to ensure that the correct data is stored within the corresponding elementary data container.

The parsing process stores its results in the MRF parameter container $\mathbb{C}^P$ for further processing by other components (see Figure 5.1) of the VLSI design framework. Generally, the MRF parameter container $\mathbb{C}^P$ holds several different string- and integer-values.

Definition 5.1 (MRF Parameter Container $\mathbb{C}^P$)
The data container for Markov Random Field specific parameters is denoted by $\mathbb{C}^P$ and comprises several different sets of string and integer data-types.

However, without detailing and listing all elements of $\mathbb{C}^P$, the MRF parameter container for instance comprises the extracted data- and control path information, the absolute directory-path location for the generated files, various naming rules, the $x \times y$ size of the MRF field, the absolute port bit-width and the port wiring bit-width, to name just the most important one.

In the first phase of the topology and structure generation process, i.e. before the corresponding graph is fully expanded, information from the MRF parameter container $\mathbb{C}^P$ becomes edited by the *structure & topology extractor* and transfered to the structure-core container $\mathbb{C}^{ST}$ in order to establish a well-defined data interface with the other components of the VLSI design framework (see Figure 5.1). The structure-core container $\mathbb{C}^{ST}$ stores different string- and integer-values.

Definition 5.2 (Structure-Core Container $\mathbb{C}^{ST}$)
The structure-core container is denoted by $\mathbb{C}^{ST}$ and comprises sets of string and integer data-types.

Again, without further detailing and listing all components of the string set as well as the integer set, the container $\mathbb{C}^{ST}$ essentially stores values regarding the grid-size of the MRF, wiring-specific parameters and some user-defined naming conventions to enhance the overall readability of the compiled HDL code.

5.2.2 Topology & Structure Graphs

In the first category of the organization scheme we find collected graphs, which represent architectural building blocks for topology & structure, i.e. the set $\mathcal{BB}^{TS}$ of topology & structure defining architectural building blocks is completely covered by the set $\mathcal{G}^{TS}$ of topology & structure graphs. But there is no strict one to one linkage between the members of the two sets $\mathcal{BB}^{TS}$ and $\mathcal{G}^{TS}$. Consequently, several architectural building blocks of the set $\mathcal{BB}^{TS}$ are covered by one graph of the set $\mathcal{G}^{TS}$. The topology & structure defining architectural building block sets S^{Hulls}, $W^{\mathcal{N}}$ and M^{Ports} (cf. Definition 3.1) are all represented in a semiconductor-technology independent manner by one single graph, which is denoted by $\mathbb{G}^{GT}$.

Unchangeable constraints of the two hardware description languages VHDL or Verilog, into which the graphs will be compiled, alone justify to the decision of representing the architectural building block sets S^{Hulls} and $W^{\mathcal{N}}$ by one single graph. The creation of the site hulls and their ports, to which the wiring blocks will connect, has to be done in one single step and in textual unity, whereby the precise port definition can solely be derived from the wiring blocks. Furthermore, the M^{Ports} are also included in the graph $\mathbb{G}^{GT}$ because the precise definition of these parts is determined by the wiring blocks, too. All these arguments justify the merging of these three parts in one single graph representation in order to allow and simplify the graph compilation and HDL generation process.

The distributed global memory hierarchy building block M^{GMem} is represented by a particular graph, to allow the distinctiveness of this large memory dominant structure to be addressed separately and to be well-controlled within the VLSI design framework.

In summary we formally define the set of topology & structure representing graphs, denoted by $\mathcal{G}^{TS}$ in the sequel, as follows:

Definition 5.3 (Topology & Structure Graphs $\mathcal{G}^{TS}$)
The set $\mathcal{G}^{TS}$ of topology & structure representing graphs reads

$$\mathcal{G}^{TS} = \{\mathbb{G}^{GT}, \mathbb{G}^{MH}\} \tag{5.1}$$

with

- $\mathbb{G}^{GT}$, the representing graph for the architectural building block sets S^{Hulls}, $W^{\mathcal{N}}$ and M^{Ports}.
- $\mathbb{G}^{MH}$, the representing graph for the building block M^{GMem}.

This finalizes the formal definition of the topology & structure representing graphs $\mathcal{G}^{TS}$ within the canonical design representation. The discussion continues with a detailed presentation of the particular $\mathcal{G}^{TS}$ graphs $\mathbb{G}^{GT}$ (cf. Definition 5.4) and $\mathbb{G}^{MH}$ (cf. Definition 5.5), and their features relevant for VLSI implementation steps.

Topology & Structure Graph Set $\mathcal{G}^{TS}$

The specific graph-set $\mathcal{G}^{TS}$ of the canonical design representation represents the most fundamental graph-set as it completely models the massively parallel architecture gantry in a semiconductor-technology independent manner. Exactly this pooling of the topology- and structure-relevant building blocks within one graph-set, where the graph $\mathbb{G}^{GT}$ models the architectural building blocks S^{Hulls}, $W^{\mathcal{N}}$, M^{Ports} and the graph $\mathbb{G}^{MH}$ models the distributed global memory hierarchy M^{GMem}, realizes one basic design principle of the VLSI design framework, namely that the generation of the MRF device topology is addressed as a completely independent design step. The two topology and structure representing graphs are realized separately and thus can also be treated completely independent without mixing up with the processing- and control-graphs during the advocated MRF specific design flow.

The graph $\mathbb{G}^{GT}$ itself, which represents the architectural building blocks S^{Hulls}, $W^{\mathcal{N}}$ and M^{Ports} in an hardware-abstract and semiconductor-technology independent manner, is realized as a directed acyclic graph (DAG) and formally reads

Definition 5.4 (Topology & Structure Graph $\mathbb{G}^{GT}$)
The directed acyclic graph for the design's topology and structure is denoted by $\mathbb{G}^{GT} = (V_{GT}, E_{GT})$. *The vertex set* $V_{GT} = \{V_{GT_{TOP}} \cup V_{GT} \cup V_{PhysConGT}\}$ *consists of three entirely different node sets and comprises:*
(1) $V_{GT_{TOP}} = \{GT_{TOP}\}$ *the root at graph-level* 0,
(2) $V_{GT} = \{cluster_{i,k} | i = 1, ..., m_{cluster}; k = 1, ..., n_{level}\}$ *representing the sub-cluster set of its split predecessor or a singl cell and*
(3) $V_{PhysCon} = \{Const_{i,(k+1)} \mid i = 1, ..., m_{cluster} \ ; \ k = 1, ..., n_{level}\}$ *representing the physical constraint set associated with each single node of* V_{GT}.
The edge set $E_{GT} = \{E_{TOP} \cup E_{GT} \cup E_{PhysCon}\}$ *comprises:*
(1) the directed edges $E_{TOP} = \{e_{i,1} | e = (GT_{TOP}, cluster_{i,1}); i = 1, ..., m_{cluster}\}$ *to indicate the connection between the node* $GT_{TOP} \in V_{GT_{TOP}}$ *and the grid-split blocks* $V_{GT} = \{cluster_{i,1} | i = 1, ..., m_{cluster}\}$ *of level 1,*
(2) $E_{GT} = \{e_{i,k} | e = (cluster_{i,k}, cluster_{i,k+2})\}$ *representing directed edges between a* $cluster_{i,k}$ *on level* k *and its successors on level* $k + 2$ *and*
(3) $E_{PhysConGT} = \{e_{i,k} = (cluster_{i,k}, Const_{i,k})\}$ *representing directed edges between the blocks on level* k *and their physical constraints on level* $k + 1$.

Graph Description - In summary an arbitrarily expanded graph $\mathbb{G}^{GT}$ first of all consists of the root node $V_{GT_{TOP}}$ at graph-level 0, which serves as an entry point for algorithms as well as a link to the data container $\mathbb{C}^{P}$ and $\mathbb{C}^{ST}$ in order to pass specification data forward to algorithms. The root node $V_{GT_{TOP}}$ is connected with site-cluster nodes, denoted by $cluster_{.,1} \in V_{GT}$. These nodes represent split site-clusters of the original MRF site-grid, i.e. the MRF site-grid with size $x \times y$ is either

divided into several smaller site clusters or, if the site-grid is small enough into its particular sites. Each node $cluster_{\cdot,1} \in V_{GT}$ carries information about its sub-grid size, the inter-cluster wiring and several other parameters, which are required for the graph compilation procedures. Eventually these nodes $cluster_{\cdot,1} \in V_{GT}$ possess children-nodes representing an additional splitting of the parent site-cluster. This scheme can be continued to expand a graph. Additionally, each node $cluster_{\cdot,\cdot} \in V_{GT}$ of the graph owns a child node $Const_{\cdot,\cdot}$, which can carry very specific constraints for the VLSI design realization steps.

With the previously given definition of $\mathbb{G}^{GT}$ at hand, several fundamentally different graphs can be constructed, which completely represent the architectural building blocks $S^{Hulls}, W^{\mathcal{N}}, M^{Ports}$ in an abstract and semiconductor-technology independent manner. But not all principally possible graphs $\mathbb{G}^{GT}$ are equally suitable for the design flow and later stages of the physical device implementation. Therefore Section 5.3.1 presents Algorithm 5.1, which systematically expands appropriate graphs $\mathbb{G}^{GT}$.

Next, we will establish the basic graph-properties of $\mathbb{G}^{GT}$, which are essential for different VLSI design phases and realization steps. Naturally these properties of $\mathbb{G}^{GT}$ have been in mind when defining $\mathbb{G}^{GT}$ and thus have been explicitly constructed into this graph-structure and are not incidental properties, which were discovered later on.

Graph Features

We continue with the proof of any topology and structure graph $\mathbb{G}^{GT}$, which is expanded with respect to Definition 5.4, being acyclic. This specific graph feature will be discussed with respect to the support of other $\mathbb{G}^{GT}$ graph features and the occurring effects regarding different VLSI implementation issues directly after the following lemma.

Lemma 5.1 *($\mathbb{G}^{GT}$* **is acyclic)**
Any topology and structure graph $\mathbb{G}^{GT}$, which is generated according to Definition 5.4, is acyclic.

Proof: From the root node GT_{TOP} on graph-level 0 only directed edges $e_{\cdot,\cdot} \in E_{TOP}$ run to site(s) respectively site cluster $cluster_{\cdot,1} \in V_{GT}$ on graph-level 1. Between V_{GT} nodes on graph-level 1 as well as between V_{GT} nodes on the same k graph-level there are no edges and consequently no cycles. Each $cluster_{\cdot,1} \in V_{GT}$ is source of no more than two types of directed edges. The first type of directed edges is derived from the set $E_{PhysConGT}$ and connects nodes $cluster_{\cdot,1} \in V_{GT}$ with nodes $Const_{\cdot,2} \in V_{PhysCon}$ without forming cycles. Generally the set $E_{PhysConGT}$ connects nodes $cluster_{\cdot,k} \in V_{GT}$ with nodes $Const_{\cdot,k+1} \in V_{PhysCon}$. Also between $V_{PhysCon}$ nodes on graph-level 2 as well as between $V_{PhysCon}$ nodes on the same graph-level $k+1$ there are no edges and thus no cycles. The second type of directed edges derives from the set E_{GT} and connects nodes $cluster_{\cdot,1} \in V_{GT}$ with nodes $cluster_{\cdot,3} \in V_{GT}$. Generally the edge set E_{GT} connects nodes $cluster_{\cdot,k} \in V_{GT}$ with nodes $cluster_{\cdot,k+2} \in V_{GT}$ without generating cycles. Furthermore there are no edges

between nodes $cluster_{\cdot,k} \in V_{GT}$ and $Const._{\cdot,k-1} \in V_{PhysCon}$. Thus any topology and structure graph $\mathbb{G}^{GT}$, which is expanded according to Definition 5.4, is acyclic. This proves the Lemma.

□

The feature that any topology and structure graph $\mathbb{G}^{GT}$, which is generated according to Definition 5.4, is acyclic has various useful consequences. On the one hand the acyclic feature of $\mathbb{G}^{GT}$ causes the graph to be topologically sortable [40], on the other hand it significantly simplifies the following proof of graph planarity [164] (see Theorem 5.2). Consequently, the acyclic feature of $\mathbb{G}^{GT}$ represents a very fundamental quality, explicitly encoded into Definition 5.4, which causes and supports more profound $\mathbb{G}^{GT}$ graph features. But the acyclic feature of graph $\mathbb{G}^{GT}$ neither complicates the overall data-structure and its implementation nor the graph-expanding- and graph-processing procedures.

Next Theorem 5.1 proofs the $\mathbb{G}^{GT}$ feature that a hierarchical ordering on the node set $V_{GT} = \{cluster_{\cdot,\cdot}\}$ exists.

Theorem 5.1 (Hierarchical $cluster_{i,k}$ Order)
The graph $\mathbb{G}^{GT}$ is topologically sortable and a list of the topologically sorted $\mathbb{G}^{GT}$ nodes with $TOPOL_{Sort} = \{v_1 \leq v_2 \leq ... \leq v_u | v_u \in V_{GT}, 1 \leq u \leq |V_{GT}|\}$ does exist. Furthermore subsets of $TOPOL_{Sort}$ are uniquely identified by their graph-level index k. Within each k-indexed subset the index i defines an ordering and consequently for each odd k an ordering on the $cluster_{\cdot,\cdot}$ nodes.

Proof: The proof is divided up into two parts: At first we show that $\mathbb{G}^{GT}$ is topologically sortable and secondly we show that on each k-indexed subset there exists an order. (1) The graph $\mathbb{G}^{GT}$ is directed, according to Definition 5.4, as well as acyclic shown by means of Lemma 5.1, and is thus a directed acyclic graph (DAG). Each DAG is topologically sortable, which proves the first part of the Theorem. (2) The k-indexed subsets are ordered among each others by means of the topological sorting and further sortable within each subset by means of the unique index i. This proves the Theorem.

□

Design Step Support - This property of the topology & structure representing graph $\mathbb{G}^{GT}$ reveals that this graph possesses an inherent structure and arrangement, which follows well-defined rules and establishes useful coherences between the site cluster or sites essentially modeled by the graph nodes V_{GT}. These coherences are used in conjunction with the planarity feature of $\mathbb{G}^{GT}$, which will be proved in Theorem 5.2 directly following, during the chip floor-planning procedure.

In order to elegantly prove Theorem 5.2 dealing with the planarity of $\mathbb{G}^{GT_1}$ by induction, we have to introduce K. Kuratowskis [103] [162] criterion of graph planarity [114].

Lemma 5.2 (Graph Planarity Criterion)
Each finite graph is planar if and only if it does not contain a subgraph, which can be generated by extensions of K^5 or $K_{3,3}$.

□

Remark 5.1 (K^5 **or** $K_{3,3}$)

1. *K^5 denotes the complete graph with five nodes.*
2. *$K_{3,3}$ denotes the bipartite graph with two node sets of cardinality three each.*
3. *K^5 and $K_{3,3}$ are the smallest non-planar graphs, which follow directly from Kuratowskis Corollary.*

Now we are able to formulate and prove the following Theorem 5.2 on the planarity of the $\mathbb{G}^{GT_1}$ graph type, which is a specialized variant of the topology & structure graph $\mathbb{G}^{GT}$ without the nodes and edges of the physical constraints.

Theorem 5.2 ($\mathbb{G}^{GT_1} = (V_{GT} \backslash V_{PhysCon}, E_{GT} \backslash E_{PhysCon})$ **Planarity)**
The topology and structure modeling graph $\mathbb{G}^{GT_1}$ is planar.

Proof: We apply induction on $|\mathbb{G}^{GT_1}|$. Obviously, for $|\mathbb{G}^{GT_1}| \leq 4$ neither K^5 nor $K_{3,3}$ can be contained as subgraph and thus every $\mathbb{G}^{GT_1}_{\leq 4}$ is planar. In the following we regard $|\mathbb{G}^{GT_1}| \gg 4$, and we additionally assume that neither K^5 nor $K_{3,3}$ is included as subgraph. If we extend this graph $\mathbb{G}^{GT_1}$ it is only possible to add nodes $cluster_{\cdot,k+2}$ and edges $e = (cluster_{\cdot,k}, cluster_{\cdot,k+2})$ according to Definition 5.4. Because $\mathbb{G}^{GT}$ is acyclic, which is obviously also valid for $\mathbb{G}^{GT_1}$, every extension by nodes V_{GT} and edges E_{GT} will not generate a cycle. Thus it is impossible that the extension of $\mathbb{G}^{GT_1}_{\gg 4}$ contains either K^5 or $K_{3,3}$ as subgraph. Consequently, each $\mathbb{G}^{GT_1}$ is K^5 and $K_{3,3}$ subgraph-free and thus planar. This proves the theorem.

□

Design Step Support - This planarity feature of the graph $\mathbb{G}^{GT_1}$ is of far-reaching relevance for the chip floor-planning. The chip floor-planning itself is required for the expected extremely large MRF hardware architectures in order to realize the technology placing and routing at all. A placing and routing of these hardware architectures without any kind of floor-planning is completely unrealistic. The established relations of the $cluster_{\cdot,\cdot}$ nodes among each other by the graph $\mathbb{G}^{GT_1}$ are conserved during the graph compilation process and translated into a corresponding hierarchy, encoded in the hardware descriptions. Consequently, the planarity feature is also valid for the hierarchy of site-clusters respectively sites represented by the hardware descriptions, which have been compiled from the $cluster_{\cdot,\cdot}$ nodes of $\mathbb{G}^{GT_1}$. Hence the chip floor-planning, conducted along the planar site-cluster respectively site hierarchy, becomes well-organized and above all is significantly simplified. Additionally, the hierarchical ordering feature of Theorem 5.1 further structures the chip floor-planning as all children site-cluster, compiled from each $cluster_{\cdot,\cdot}$ node, can be arranged in a particular rectangular chip area according to their ordering.

The following Theorem 5.3 about the combinational-logic neutrality of $\mathbb{G}^{GT}$ during graph compilation affects the synthesis step for this part of the architecture and reads:

Theorem 5.3 ($\mathbb{G}^{GT}$ is combinational-logic neutral)
The topology and structure representing graph $\mathbb{G}^{GT}$ is combinational-logic neutral at compilation.

Proof: We assume that the compilation process of graphs into hardware descriptions itself will not generate any combination-logic. The root node GT_{TOP} does not model any part of the system's topology & structure, but rather serves as an unique entry point for algorithms and a link to specifications. Consequently, GT_{TOP} will not introduce any combinational-logic during the compilation process. Furthermore the node set V_{GT} only represents the structure definition of sites respectively site-clusters and wiring definitions between sites or site-clusters. Hence these nodes will not translate into combinational-logic during compilation. By definition of physical constraints in VLSI design flows, these constraints cannot produce combinational logic by themselves and this is equally valid for each $V_{PhysCon}$ of $\mathbb{G}^{GT}$. Finally each single edge of $\mathbb{G}^{GT}$ establishes an ordering and linkage of the nodes and does not represent topology & structure components of the architecture. Thus the edges will not be translated into hardware description parts at all. This proves the Theorem.

□

Design Step Support - Each topology & structure representing graph $\mathbb{G}^{GT}$, which is expanded in accordance to Definition 5.4, fulfills Theorem 5.3. This specific $\mathbb{G}^{GT}$ quality ensures that every hardware description compiled of $\mathbb{G}^{GT}$ is combinational-logic free. Thus the VLSI synthesis step can be conducted without finite state machine (FSM) extraction, without FSM state encoding, without FSM optimization as well as without any kind of logic optimization. Consequently, the synthesis step for $\mathbb{G}^{GT}$ compiled hardware descriptions reduces to the technology-independent generation of wires for $W^{\mathcal{N}}$ and block structures for S^{Hulls} as well as to the technology-dependent generation of storage elements for M^{Ports}.

So far the topology and structure defining building block sets S^{Hulls}, $W^{\mathcal{N}}$ and M^{Ports} of $\mathcal{BB}^{TS}$ have been captured and represented by its graph $\mathbb{G}^{GT}$. Consequently, the graph theoretical representation of the distributed memory hierarchy building block M^{GMem} has to be defined.

The graph $\mathbb{G}^{MH}$ itself, which represents the architectural building block M^{GMem}, in a hardware-abstract and semiconductor-technology independent manner, is realized by a directed acyclic graph. $\mathbb{G}^{MH}$ is formally defined as follows

Definition 5.5 (Memory Hierarchy Graph $\mathbb{G}^{MH}$)
The directed acyclic graph for the distributed memory hierarchy is denoted with $\mathbb{G}^{MH} = (V_{MH}, E_{MH})$. The vertex set $V_{MH} = \{V_{MH_{TOP}} \cup V_{MH} \cup V_{PhysConMH}\}$ consists of the three entirely different node types and comprises:
(1) $V_{MH_{TOP}} = \{MH_{TOP}\}$ a single unique node, the root of the memory hierarchy graph,
(2) $V_{MH} = \{MEMblock_{i_k,k} | i = 1, ..., m_{blocks}; k = 1, ..., n_{levels}\}$ representing the split sub-grid blocks of its predecessor and

(3) $V_{PhysConMH} = \{Const_{i,(k+1)} \mid i = 1, ..., m_{blocks} \ ; \ k = 1, ..., n_{levels}\}$ *representing the physical constraints associated with each single node of* V_{MH}*.*

The edge set $E_{MH} = \{E_{TOP} \cup E_{MH} \cup E_{PhysConMH}\}$ *comprises:*
(1) the directed edges $E_{TOP} = \{e_{i,1} | e = (MH_{TOP}, block_{i,1}); i = 1, ..., m_{blocks}\}$ *to indicate the connection between the node* $MH_{TOP} \in V_{MH_{TOP}}$ *and the grid-split blocks* $V_{MH} = \{block_{i,1} | i = 1, ..., m_{blocks}\}$ *of level 1,*
(2) $E_{MH} = \{e_{i,k} | e = (block_{i,k}, block_{i,k+2})\}$ *representing directed edges between a* $block_{i,k}$ *on level* k *and its successors on level* $k + 2$ *and*
(3) $E_{PhysConMH} = \{e_{i,k} = (block_{i,k}, Const_{i,k})\}$ *representing directed edges between the blocks on level* k *and their physical constraints on level* $k + 1$*.*

Graph Description - Every arbitrary graph $\mathbb{G}^{MH}$, which is generated according to Definition 5.5, consists of the root node MH_{TOP} at graph-level 0. The root node serves as starting point for various algorithms and also as a link to the elementary data containers $\mathbb{C}^{P}$ and $\mathbb{C}^{ST}$ so that required parameters can easily be provided. Each node $MEMblock_{.,1} \in V_{MH}$ at graph-level 1 represents a memory-block, which stores specific parts of the input- respectively output-data during the transmission process to and from particular site-clusters. Consequently, the overall input- and output-data is distributed and stored within memory-blocks represented by the nodes $MEMblock_{.,1} \in V_{MH}$. The nodes $MEMblock_{.,1} \in V_{MH}$ themselves carry information about the required storage size derived from the overall storage size and the distribution to the nodes. This information will be used during the HDL-code generation process. If the particular memory-blocks represented by the nodes $MEMblock_{.,1} \in V_{MH}$ are too large, it is principally possible to further split each memory-block and thus to generate children nodes $MEMblock_{.,3} \in V_{MH}$. This split-procedure can be continued several times until a specific graph $\mathbb{G}^{MH}$ becomes expanded. Obviously the actual shape of the graph $\mathbb{G}^{MH}$ depends on the split- respectively expanding-procedure.

The definition of $\mathbb{G}^{MH}$ is - similar to the Definition 5.5 of $\mathbb{G}^{GT}$ - generic enough to allow many different graphs to principally be generated according to this definition. Obviously, not all graph variants are equally qualified for further processing procedures within the VLSI design framework and are also not properly tuned with respect to their structure to support back-end technology implementation tasks. Hence Section 5.3.1 introduces Algorithm 5.2, which systematically expands appropriate graphs $\mathbb{G}^{MH}$ for the design framework and further implementation steps according to Definition 5.5.

Next, we will prove essential features of the graph $\mathbb{G}^{MH}$, which are important for different steps of the VLSI design flow and the physical device implementation.

Graph Features

Each expanded and distributed memory hierarchy graph $\mathbb{G}^{MH}$, irrespective of its expanding-procedure, possesses fundamental features, which are advantageous for different processing steps along the complete VLSI realization flow. First we will prove the characteristic feature of the graph $\mathbb{G}^{MH}$ being acyclic.

Lemma 5.3 ($\mathbb{G}^{MH}$ is acyclic)
Any distributed memory hierarchy graph $\mathbb{G}^{MH}$, which is generated according to Definition 5.5, is acyclic.

Proof: The argumentation follows that of Lemma 5.1. Essentially the root node MH_{TOP} and its children $MEMblock_{.,1} \in V_{MH}$ do not form a cycle. Furthermore there are no edges and consequently also no cycles between V_{MH} nodes at the same k graph-level. Every node $MEMblock_{.,.} \in V_{MH}$ has no more than two different children on the two directly following graph-levels. First these are nodes $Const._{.,.} \in V_{PhysConMH}$, which are connected by a directed edge and thus will not generate any cycles. Additionally, the nodes $Const._{.,.} \in V_{PhysConMH}$ possess no edges between nodes on the same k graph-level and thus no cycles. The second group contains the nodes $MEMblock_{.,.} \in V_{MH}$, which are connected by a directed edge. Again this will not form a cycle. This node pattern can be continued several times without forming any cycle. Thus any distributed memory hierarchy graph $\mathbb{G}^{MH}$, which is expanded according to Definition 5.5, is acyclic. This proves the lemma.

□

Design Step Support - The acyclic quality of $\mathbb{G}^{MH}$ implies the essential and, for the place and route back-end VLSI task, important feature of planarity. Thus the acyclic feature of the abstract and semiconductor-technology independent representation simplifies technology specific implementation-tasks and their underlying algorithms.

The following Theorem 5.4 proves the planarity of any graph $\mathbb{G}_1^{MH}$, which is expanded according to Definition 5.5. Each graph $\mathbb{G}_1^{MH}$ is a specialized variant of the memory hierarchy graph $\mathbb{G}^{MH}$, which excludes all nodes and edges representing the physical constraints. As for the topology and structure representing graph, we also use Kuratowskis [103] planarity criterion of Lemma 5.2 to prove the planarity of $\mathbb{G}_1^{MH}$.

Theorem 5.4 ($\mathbb{G}_1^{MH} = (V_{MH} \backslash V_{PhysConMH}, E_{MH} \backslash E_{PhysConMH})$ Planarity)
The memory hierarchy graph $\mathbb{G}_1^{MH}$ is planar.

Proof: The proof runs by induction on $|\mathbb{G}^{MH_1}|$. For every graph $|\mathbb{G}^{MH_1}| \leq 4$ it becomes obvious by Definition 5.5 that neither K^5 nor $K_{3,3}$ is contained as a subgraph. Consequently, each $\mathbb{G}_{\leq 4}^{MH_1}$ is planar. Next we regard $|\mathbb{G}^{MH_1}| \gg 4$ with graph level k and furthermore we assume that neither K^5 nor $K_{3,3}$ is included as a subgraph. It directly follows that $\mathbb{G}^{MH_1}$ is planar for $|\mathbb{G}^{MH_1}| \gg 4$. Extending this graph $\mathbb{G}^{MH_1}$ it is only possible to add nodes $MEMblock_{i_k,k+2}$ and edges $e = (block_{i,k}, block_{i,k+2})$ in accordance with Definition 5.5. Because $\mathbb{G}^{MH}$ is acyclic (cf. Lemma 5.3), which is obviously also valid for $\mathbb{G}^{MH_1}$, none of the graph extension will generate a cycle. Consequently, it is not possible that the extension of $\mathbb{G}_{\gg 4}^{MH_1}$ contains a K^5 or $K_{3,3}$ as a subgraph. Hence, every $\mathbb{G}^{MH_1}$ is planar. This proves the theorem.

□

Design Step Support - As already discussed for the topology and structure graph $\mathbb{G}^{GT_1}$, the property of planarity of the graph $\mathbb{G}^{MH_1}$ is also of vital relevance for the chip floor-planning of the architectural topology, which the memory hierarchy belongs to (see Section 3.2.1). The planarity of $\mathbb{G}^{MH_1}$ is conserved during the graph compilation process and encoded by the corresponding hardware descriptions of the memory hierarchy. Hence, the planarity of $\mathbb{G}^{MH_1}$ also remains valid for the memory blocks and their interdependencies represented by the hardware descriptions, which have been compiled from the $MEMblock_{\cdot,\cdot}$ nodes of $\mathbb{G}^{MH_1}$. Consequently, the chip floor-planning for the memory hierarchy becomes a well-organized task. Furthermore, the unique indexing i on each level of graph $\mathbb{G}^{MH_1}$, also embedded into the hardware descriptions, further organizes the chip floor-planning procedure as all children of a memory-block can be assigned to their particular rectangular chip area - according to the imposed ordering of index i.

The next Theorem 5.5 establishes the combinational-logic neutrality of $\mathbb{G}^{MH}$ during compilation.

Theorem 5.5 ($\mathbb{G}^{MH}$ is combinational-logic neutral)
The memory hierarchy representing graph $\mathbb{G}^{MH}$ is combinational-logic neutral during compilation.

Proof: We presuppose that the graph compilation process itself will not generate any kind of combinational-logic. When considering the root node MH_{TOP}, we remark that this node does not model any part of the memory hierarchy. This specific node rather serves as starting point for algorithms and at the same time as small data container for parameters. Hence, MH_{TOP} will not be translated into combinational-logic during the compilation process. Furthermore, the node set V_{MH} of $\mathbb{G}^{MH}$ only represents the memory-blocks. Obviously these nodes will not translate into combinational-logic while being compiled. Physical constraints of VLSI design flows normally cannot produce combinational-logic by themselves and thus each $V_{PhysCon} \in \mathbb{G}^{MH}$ will not generate combinational-logic. Furthermore the edge set between the nodes V_{MH} of $\mathbb{G}^{MH}$ forms the shape of the hierarchy and the connections of the memory blocks among each other. Consequently, these edges only contribute to wiring structures of the memory hierarchy and by no means to combinational-logic structures. This proves the Theorem.

□

Design Step Support - Each memory hierarchy graph $\mathbb{G}^{MH}$ is combinational-logic free during compilation, which has been shown by the previous Theorem 5.5. This property of $\mathbb{G}^{MH}$ simplifies the synthesis- and technology-mapping step, because it is not necessary to extract finite state machines (FSM), to encode and optimize the states of FSMs or to optimize any kind of combinational logic. Therefore, the synthesis- and technology-mapping step for the memory hierarchy is reduced to the technology-independent generation of hierarchy structures and connecting wires as well as to the technology-dependent mapping onto memory blocks. In summary the

synthesis process of the hardware descriptions of the memory hierarchy only covers the mapping to special and technology-dependent memory blocks.

Based on Definition 5.4, 5.5 and the preceding discussion the coherence between the building block set $\mathcal{BB}^{TS}$ and the representing graph set $\mathcal{G}^{TS}$ for VLSI realizations is formally given by the mappings

$$\Xi^{TS} : \{S^{Hulls}, W^{\mathcal{N}}, M^{Ports}\} \longrightarrow \mathbb{G}^{GT}. \tag{5.2}$$

and

$$\Xi^{GMem} : M^{GMem} \longrightarrow \mathbb{G}^{MH}. \tag{5.3}$$

The mappings 5.2 and 5.3 give rise to the following corollary on the complete representation of $\mathcal{BB}^{TS}$ in a semiconductor technology independent manner.

Corollary 5.1 *($\mathcal{BB}^{TS}$* **Representation)**
The building block set $\mathcal{BB}^{TS}$, which represents the topology & structure of the proposed architecture template, is completely modeled by the graph set $\mathcal{G}^{TS}$ in a semiconductor technology independent manner.

□

This finalizes the discussion on the graph theoretical representation of topology and structure defining building blocks of the proposed architecture template. We continue with the introduction of specific graphs, which model the processing functionality of the architecture template.

5.2.3 Processing Graphs

The second category of the ordering scheme comprises graphs, which represent architectural building blocks $\mathcal{BB}^{PC}$ defining processing functionality (cf. Definition 3.2). Consequently, the set $\mathcal{BB}^{PC}$ of processing building blocks is completely covered by the graph set $\mathcal{G}^{PC}$. In contrast to the topology and structure building blocks each processing building block of Definition 3.2 is represented by exactly one graph. The set PC_{EF} of energy functional processing building blocks is represented in a semiconductor-technology independent manner by the single graph $\mathbb{G}^{EF}_{PC}$, whereas the graph prototypically models only one energy functional processing building block $\mathcal{H}^{EF} \in PC_{EF}$, as it is assumed that all $\mathcal{H}^{EF}_{i\leq n} \in PC_{EF}$ are functionally and structurally identical. If this assumption cannot be applied then functionally and structurally differentiating energy functional building blocks have to be modeled simultaneously by different graphs $\mathbb{G}^{EF}_{PC_{z\leq k}}$. Likewise, the set of PC_{OPT} optimization processing building blocks is represented by the single graph $\mathbb{G}^{OPT}_{PC}$, because all optimization processing building blocks $\mathcal{H}^{OPT}_{i\leq n} \in PC_{OPT}$ are functionally and structurally identical. The last processing building block PC_{PAR}, which represents the parameter estimation process is modeled by the graph $\mathbb{G}^{PAR}_{PC}$.

Precisely this set of processing building blocks representing graphs, denoted by $\mathcal{G}^{PC}$ in the sequel, is formally defined as follows:

Definition 5.6 (Processing Graphs $\mathcal{G}^{PC}$)
The set $\mathcal{G}^{PC}$ of architectural processing graphs reads

$$\mathcal{G}^{PC} = \{\mathbb{G}_{PC}^{EF}, \mathbb{G}_{PC}^{OPT}, \mathbb{G}_{PC}^{PAR}\} \tag{5.4}$$

with

- $\mathbb{G}_{PC}^{EF}$, the representing graph for the prototype of the functionally and structurally identical elements of the building block set $\{\mathcal{H}_i^{EF} : 1 \leq i \leq n\}$.
- $\mathbb{G}_{PC}^{OPT}$, the representing graph for the prototype of the functionally and structurally identical elements of the building block set $\{opt_i : 1 \leq i \leq n\}$.
- $\mathbb{G}_{PC}^{PAR}$, the representing graph for the architectural building block PC_{PAR}.

This finalizes the introduction of the graphs $\mathcal{G}^{PC}$, which represent processing building-blocks, as part of the canonical design representation. The discussion continues with a detailed presentation and definition of the particular $\mathcal{G}^{PC}$ graphs $\mathbb{G}^{EF}$ (cf. Definition 5.7), $\mathbb{G}^{OPT}$ (cf. Definition 5.8) and $\mathbb{G}^{PAR}$ (cf. Definition 5.9), following the chronological order of Definition 5.6.

Although the processing graphs $\mathbb{G}^{EF}$, $\mathbb{G}^{OPT}$ and $\mathbb{G}^{PAR}$ resemble each other with respect to their global graph structure in the following, we will further detail the presentation for each processing graph and will provide a separate definition. This detailed presentation simplifies the discussion as well as the referencing in later sections of this chapter.

Processing Graph Set $\mathcal{G}^{PC}$

The graph set $\mathcal{G}^{PC}$ of the canonical design representation solely models architectural building blocks, which are dedicated to numerical processing tasks within the massively parallel architecture. These processing blocks are independent and distributed units, embedded into the architectural gantry modeled by the graph set $\mathcal{G}^{TS}$. Consequently, none of these processing blocks and their abstract graph representation $\mathcal{G}^{TS}$ alters the architectural topology gantry and their corresponding abstract graph representation $\mathcal{G}^{PC}$. Hence the design principle of the proposed VLSI design framework, claiming the topology and structure generation process to be independent, and the first design step also remain valid for the introduction of the graph set $\mathcal{G}^{PC}$. Furthermore, the strict separation of the graph sets $\mathcal{G}^{TS}$, $\mathcal{G}^{PC}$ and their particular elements simplify all further design steps because very specific processing-strategies can be applied to distinct parts of the system architecture.

The representing graph $\mathbb{G}_{PC}^{EF}$ itself is realized as a directed acyclic graph, which prototypically captures the data flow dependencies of the energy functional building blocks. This is why such a graph is called *Data Flow Graph (DFG)*. The data flow graph $\mathbb{G}_{PC}^{EF}$ formally reads

Definition 5.7 (DFG Energy Functional $\mathbb{G}_{PC}^{EF}$)
The data flow graph $\mathbb{G}_{PC}^{EF} = \{V_{PC}^{EF}, E_{PC}^{EF}\}$ of the energy functional is a directed acyclic graph. The node set $V_{PC}^{EF} = \{op_i | i = 1, ..., n_{ops}\}$ describes the different operators, and the edge set $E_{PC}^{EF} = \{e_{i,j} | (op_i, op_j)\}$ represents the data dependencies

and flow of information between the corresponding operators, i.e. $\exists e_{i,j} \in E_{PC}^{EF}$ *if the result of operator* op_i *is used by operator* op_j*. Some of the nodes op.* $\in V_{PC}^{EF}$ *are specialized because they model the data-inputs and data-outputs of the energy functional. Furthermore the root node is represented by* EF_{TOP}*.*

Graph Description - The previous definition reveals that a data flow dependency determines, when a specific operator can start its processing. Consequently, if operator $op_i \in V_{PC}^{EF}$ establishes a data flow dependency with operator $op_j \in V_{PC}^{EF}$, then operator $op_j \in V_{PC}^{EF}$ can start its processing only after operator $op_i \in V_{PC}^{EF}$ has finished its processing. As already mentioned the following graphs that represent processing-functionality $\mathbb{G}_{OPT}^{EF}$ and $\mathbb{G}_{PAR}^{EF}$, are also data flow graphs and thus, compared with graph $\mathbb{G}_{PC}^{EF}$, only differ with respect to the node- and edge-set.

The graph $\mathbb{G}_{PC}^{OPT}$ is also realized as a directed and acyclic graph, which prototypically models the data flow dependencies of an arbitrary optimization building block *opt.*. A single prototypical graph representation is justified, because it is assumed that all optimization building blocks are functionally identical. The formal definition of the data flow graph $\mathbb{G}_{PC}^{OPT}$ is given by

Definition 5.8 (DFG Optimization $\mathbb{G}_{PC}^{OPT}$)
The model's optimization data flow graph $\mathbb{G}_{PC}^{OPT} = \{V_{PC}^{OPT}, E_{PC}^{OPT}\}$ *is a directed acyclic graph. The node set* $V_{DFG}^{OPT} = \{op_i | i = 1, ..., n_{ops}\}$ *describes operators and the edge set* $E_{DFG}^{OPT} = \{e_{i,j} | (op_i, op_j)\}$ *represents the data dependencies and flow of information between the operators. Some of the nodes op.* $\in V_{DFG}^{OPT}$ *are specialized because they model the data-inputs and data-outputs of the energy functional. Furthermore the root node is represented by* OPT_{TOP}*.*

What was already stated for the data flow graph $\mathbb{G}_{PC}^{EF}$ of Definition 5.7, also applies to the data flow graph $\mathbb{G}_{PC}^{OPT}$. Each single data flow dependency of an optimization building block is prototypically modeled by this graph of the canonical design representation.

Definition 5.9 (DFG Parameter Estimation $\mathbb{G}_{PC}^{PAR}$)
The model's parameter estimation data flow graph $\mathbb{G}_{PC}^{PAR} = \{V_{PC}^{PAR}, E_{PC}^{PAR}\}$ *is a directed acyclic graph. The node set* $V_{PC}^{PAR} = \{op_i | i = 1, ..., n_{ops}\}$ *describes the operators and the edge set* $E_{PC}^{PAR} = \{e_{i,j} | (op_i, op_j)\}$ *represents the data dependencies and flow of information between the operators. Some of the nodes op.* $\in V_{DFG}^{PAR}$ *are specialized because they model the data-inputs and data-outputs of the energy functional. Furthermore the root node is represented by* PAR_{TOP}*.*

The description and definition of the parameter estimation graph $\mathbb{G}_{PC}^{PAR}$ concluded the discussion of the processing graph set $\mathcal{G}^{PC}$ as part of the canonical design representation. So far abstract and semiconductor-independent graph representations for topology & structure defining building blocks as well as for processing-functionality defining building blocks have been introduced in the preceding subsections. The following paragraph defines the graph representations still missing

to completely model the architectural control building blocks. These graphs representing control-functionalities will complete the graph set of the canonical design representation in order to model a particular Markov Random Field, which is covered by the contemplated MRF-class defined in Section 2.5.

Based on Definition 5.7, 5.8, 5.9 and the discussion the coherence between the processing building blocks $\mathcal{BB}^{PC}$ and the representing graphs $\mathcal{G}^{PC}$ are given by the mappings:

$$\Psi^{EF} : \{\mathcal{H}_i^{EF} : 1 \leq i \leq n\} \longrightarrow \mathbb{G}_{PC}^{EF}. \tag{5.5}$$

$$\Psi^{OPT} : \{opt_i : 1 \leq i \leq n\} \longrightarrow \mathbb{G}_{PC}^{OPT}. \tag{5.6}$$

$$\Psi^{EF} : PC_{PAR} \longrightarrow \mathbb{G}_{PC}^{PAR}. \tag{5.7}$$

The previous mappings 5.5, 5.6 and 5.7 give rise to the following corollary:

Corollary 5.2 *($\mathcal{BB}^{PC}$* **Representation)**
The processing functionality defining building block set $\mathcal{BB}^{PC}$ is completely represented by the graph set $\mathcal{G}^{PC}$ in a semiconductor technology independent manner.

□

We continue the ongoing discussion with the description of specific graphs, which model the control functionality of the architecture template.

5.2.4 Control Graphs

The third and last category of the organization scheme comprises a set of graphs $\mathcal{G}^{CT}$ relevant for control functionalities. These graphs model the architectural control building block set $\mathcal{BB}^{CT}$ (cf. Definition 3.3) in an abstract and semiconductor-technology independent manner. Between the control-functionality of the architectural building blocks $\mathcal{BB}^{CT}$ and the representing graphs $\mathcal{G}^{CT}$ exists a one-to-one correspondence, i.e. each particular control-functionality of $\mathcal{BB}^{CT}$ is represented by exactly one distinct graph of $\mathcal{G}^{CT}$. The architectural control building blocks CT^{System} and CT^{GMem} are at the time modeled by the distinct graphs $\mathbb{G}^{System}$ and $\mathbb{G}^{GMem}$, respectively. With regard to the control building block sets $CT^{PortMem}$, CT^{EF} and CT^{OPT} the situation looks slightly different: Not all members of these building block sets are represented by particular graphs; rather for each particular set exactly one graph ($\mathbb{G}^{PortMem}$, $\mathbb{G}^{EF}$ and $\mathbb{G}^{OPT}$) prototypically models the components of the corresponding architectural building block sets. This procedure can be justified, as it is assumed that the components $ctrl_{i\leq n}^{PortMem}$, $ctrl_{i\leq n}^{EF}$ and $ctrl_{i\leq n}^{OPT}$ are functionally and structurally identical for a specific MRF model. Should a differing setting be necessary, then several graphs per set have to be generated to represent this setting. Such settings are principally possible within the proposed novel VLSI design framework, but until now software-technically have not been realized.

Formally, the control graphs will be denoted by $\mathcal{G}^{CT}$ in the sequel and are defined as follows:

Definition 5.10 (Control Graphs $\mathcal{G}^{CT}$)
The set $\mathcal{G}^{CT}$ of architectural control building blocks reads

$$\mathcal{G}^{CT} = \{\mathbb{G}_{CT}^{System}, \mathbb{G}_{CT}^{GMem}, \mathbb{G}_{CT}^{PortMem}, \mathbb{G}_{CT}^{EF}, \mathbb{G}_{CT}^{OPT}\} \tag{5.8}$$

with

- $\mathbb{G}_{CT}^{System}$, the graph representing the overall system control building block CT^{System}.
- $\mathbb{G}_{CT}^{GMem}$, the graph representing the distributed memory hierarchy control building block CT^{GMem}.
- $\mathbb{G}_{CT}^{PortMem}$, the graph prototypically representing the set port memory control building blocks $CT^{PortMem} = \{ctrl_{i\leq n}^{PortMem}\}$.
- $\mathbb{G}_{CT}^{EF}$, the graph prototypically representing the set of energy functional control building blocks $CT^{EF} = \{ctrl_{i\leq n}^{EF}\}$.
- $\mathbb{G}_{CT}^{OPT}$, the graph prototypically representing the set of control building blocks $CT^{OPT} = \{ctrl_{\leq n}^{OPT}\}$ for the optimization procedure.

The previous Definition 5.10 finalizes the formal definition of the control-functionality representing graph set $\mathcal{G}^{CT}$ within the canonical design representation. We continue the discussion with the presentation of a generic control graph. All graphs of $\mathcal{G}^{CT}$ can be derived from a customized version of this generic control graph, which will be introduced in the upcoming section. This kind of presentation tightens the section of graphs for control building blocks. Additionally, it allows us to sum up the essential features of control graphs in the single Definition 5.11 on a generic control graph.

The strict separation of topology & structure graphs $\mathcal{G}^{TS}$, processing graphs $\mathcal{G}^{PC}$ and control graphs $\mathcal{G}^{CT}$ leads to the fact that none of these graph sets can alter components of the other ones. In addition, specifically adjusted procedures can be applied to these graph sets in a controlled manner, as the graph sets $\mathcal{G}^{TS}$, $\mathcal{G}^{PC}$ and $\mathcal{G}^{CT}$ are strictly separated and thus can be addressed individually. The different mappings between building blocks respectively building block sets will be described in the following.

All previously mentioned graphs are directed graph variants, which capture the control-flow-dependencies of the architectural control building blocks $\mathcal{BB}^{CT}$ in an abstract an semiconductor-technology independent manner. Because these graph types model the flow of control [48], such graphs are called *Control Flow Graphs (CFG)*. As already mentioned, in order to tighten the following text-paragraph and the discussion on control building-block representing graphs, we will define one generic control flow graph instead of several detailed variants for the set $\mathcal{BB}^{CT}$. The generic control flow graph is formally given by

Definition 5.11 (Generic Control Flow Graph $\mathbb{G}_{CT}^{Generic}$)
The generic control flow graph is denoted with $\mathbb{G}_{CT}^{Generic} = (V_{CT}, E_{CT})$. $\mathbb{G}_{Generic}^{CT}$ is a directed graph. The node set $V_{CT} = v_{CT}^{TOP} \cup \{sb_i | i = 1, ..., n_{sb}\}$ describes the

scope blocks of the control statements and the edge set $E_{CT} = \{e_{i,j} = (sb_i, sb_j) | i, j = 1, ..., n_{sb}\}$ *the control flow between different scope blocks. Furthermore each edge is marked by conditions which determine the scope block transitions.*

Graph Description - Every control representing graph out of $\mathcal{G}^{CT}$ is a customized version of the generic control graph $\mathbb{G}^{CT}_{Generic}$, i.e. the canonical design representation at the same time contains a customized version of $\mathbb{G}^{CT}_{Generic}$ for $\mathbb{G}^{System}_{CT}$, $\mathbb{G}^{GMem}_{CT}$, $\mathbb{G}^{PortMem}_{CT}$, $\mathbb{G}^{EF}_{CT}$ as well as of $\mathbb{G}^{System}_{OPT}$. For each of these control graphs $\mathbb{G}^{\cdot}_{CT}$ the node-sets and the edge-sets are separated and the graphs themselves are completely independent from each other. The graph of Definition 5.11 determines the flow of control between the scope blocks. These scope blocks $sb_{i \leq n_{sb}} \in V_{CT}$ are wildcards for the operations, signal assignments etc., which will take place between control transitions. Consequently, if the scope block sb_i is connected with the scope block sb_j by the edge $e_{i,j} = (sb_i, sb_j)$, a transition from sb_i to sb_j takes place, if and only if the condition of $e_{i,j}$ is valid. The formulation with scope blocks is advantageous and more flexible compared with a pure state formulation, because conditional branches and loops can be represented in a much clear manner.

Obviously, the previously introduced data-flow-graphs as well as the control-flow-graphs alone cannot totally and generally represent a well-defined sequence of operations and control-conditions. Only a linkage of a data-flow-graph and its appropriate control-flow-graph can achieve this. The next text paragraph explains the DFG- and CFG-linkage in detail, which is mandatorily necessary to form a correct sequence of operations and control-conditions.

Linkage between DFG and CFG

A special case exists, when a control flow graph can be view in isolation without its corresponding data flow graph. This case is given when a pure control task has to be modeled in a graph-theoretical manner. The system control graph $\mathbb{G}^{System}_{CT}$ models such a pure control task and thus $\mathbb{G}^{System}_{CT}$ as a standalone control flow graph, becomes a valid representation of pure control-functionalities within the massively parallel MRF system architecture. All other control-flow-graphs of the set $\mathcal{G}^{CT}$ possess corresponding data-flow-graphs. The following corollary formalizes these specific DFG-CFG linkages within the canonical design representation and the assignment of data-flow operators to control-flow scope blocks.

Corollary 5.3 DFG and CFG Linkage
For each of the control flow graph and data flow graph tuple $(\mathbb{G}^{GMem}_{PC}, \mathbb{G}^{GMem}_{CT})$, $(\mathbb{G}^{PortMem}_{PC}, \mathbb{G}^{PortMem}_{CT})$, $(\mathbb{G}^{EF}_{PC}, \mathbb{G}^{EF}_{CT})$ *and* $(\mathbb{G}^{OPT}_{PC}, \mathbb{G}^{OPT}_{CT})$, *a many-to-one mapping of the corresponding operators* $op. \in V^{\cdot}_{DFG}$ *to its scope blocks* $sb. \in V^{\cdot}_{CT}$ *exist.*

The pairing of control-flow-graphs and data-flow-graphs is given for the canonical design representation of the proposed design framework. The assignment of data-flow operators *op.* to their control-flow scope blocks *sb.* is just as straightforward. The operators *op.* are encapsulated by control statements (conditional- and loop-constructs), whose beginning- and end-labels uniquely define a scope block. Thus

all operators *op.* can uniquely be assigned to their corresponding scope blocks to form a correct control-flow-graph and data-flow-graph linkage [118] [63].

Based on the generic Definition 5.11 on control graphs and the corresponding discussion the coherence between the building blocks $\mathcal{BB}^{CT}$ and the representing graphs $\mathcal{G}^{CT}$ are given by

$$\Gamma^{System} : CT^{System} \longrightarrow \mathbb{G}_{CT}^{System}, \tag{5.9}$$

$$\Gamma^{GMem} : CT^{GMem} \longrightarrow \mathbb{G}_{CT}^{GMem}, \tag{5.10}$$

$$\Gamma^{PortMem} : CT^{PortMem} \longrightarrow \mathbb{G}_{CT}^{PortMem}, \tag{5.11}$$

$$\Gamma^{EF} : CT^{EF} \longrightarrow \mathbb{G}_{CT}^{EF}, \tag{5.12}$$

$$\Gamma^{OPT} : CT^{OPT} \longrightarrow \mathbb{G}_{CT}^{OPT}. \tag{5.13}$$

The previous discussion and the above stated mappings give rise to Corollary 5.4.

Corollary 5.4 *($\mathcal{BB}^{CT}$* **Representation)**
The processing functionality defining building block set $\mathcal{BB}^{CT}$ is completely represented by the graph set $\mathcal{G}^{CT}$ in a semiconductor technology independent manner.

□

MRF-System Design Representation

The discussion and derivations of Section 5.2 have established important graph theoretical fundamentals to represent the architectural building blocks $\mathcal{BB}^{System}$ (cf. Section 3.2) and thus a massively parallel MRF architecture in an abstract and technology-independent manner. Hence, we can formulate the following essential theorem regarding the complete MRF design representation by system graph $\mathcal{G}^{\mathcal{D}}$.

Theorem 5.6 (Design Graph $\mathcal{G}^{\mathcal{D}}$)
For each massively parallel MRF architecture, which is exclusively compiled out of the architectural building blocks $\mathcal{BB}^{System}$ and which is also conformable with Definition 2.10, an abstract and semiconductor-technology independent graph representation $\mathcal{G}^{\mathcal{D}}$ does exists. This design graph $\mathcal{G}^{\mathcal{D}}$ is composed out of the topology and structure graph set $\mathcal{G}^{TS}$, the processing graph set $\mathcal{G}^{PC}$ and the control graph set $\mathcal{G}^{CT}$ and thus reads

$$\mathcal{G}^{\mathcal{D}} = \{\mathcal{G}^{\mathcal{TS}} \cup \mathcal{G}^{\mathcal{PC}} \cup \mathcal{G}^{\mathcal{CT}}\}.$$

□

The next sections present the entire design flow by means of the proposed novel VLSI design framework for massively parallel MRF architectures. The discussion starts with the parsing- and analysis-steps and continues with the central graph expanding procedures, which generate the design graph $\mathcal{G}^{\mathcal{D}}$. Finally Section 5.4 discusses the HDL code generation process based on the design graph representation $\mathcal{G}^{\mathcal{D}}$, which finalizes the presentation of the proposed design flow (cf. Figure 5.1).

5.3 High-Level Design Flow

The design flow traverses through several steps in order to generate a HDL code for the massively parallel architecture, introduced in Chapter 3, of a specific MRF model from hardware-abstract specifications and definitions. It starts with the reading and analysis of specifications and definitions, which describe the MRF model and fundamental features of the massively parallel architecture. Essentially each specific MRF model is represented by its corresponding energy-functional and the neighborhood system, whereas the neighborhood system represents a vital structural component of the architecture. Furthermore, the optimization method (cf. Section 2.3) determines the MRF model as well as its performance. The features of the massively parallel architecture are mainly determined by the already mentioned neighborhood system, the port memories and the memory hierarchy. This parsed information is processed and stored in the elementary data container $\mathbb{C}^P$ (cf. Definition 5.1). In the following design flow step the content of the data container $\mathbb{C}^P$ is further processed with respect to topology- and structure defining architectural information and these values are finally stored in $\mathbb{C}^{ST}$. The content of $\mathbb{C}^{ST}$ is used to expand the graph set $\mathcal{G}^{\mathcal{TS}}$ defined in Definition 5.3. The next design step generates the graph sets $\mathcal{G}^{\mathcal{PC}}$ as well as $\mathcal{G}^{\mathcal{CT}}$, which represent the processing and control parts of the architecture. Finally the last design step generates a HDL code from the design graph set $\mathcal{G}^{\mathcal{D}} = \{\mathcal{G}^{\mathcal{TS}} \cup \mathcal{G}^{\mathcal{PC}} \cup \mathcal{G}^{\mathcal{CT}}\}$.

The corresponding software-technical realization of the proposed VLSI design framework, depicted in Figure 5.1, in its current version possesses a XML 1.0 parsing front end [168]. Consequently, all specifications and descriptions of MRFs have to be realized in a special XML template. We decided to use XML as input language for the first version of the VLSI design framework, because of its flexibility as well as the availability of XML parser front-ends free of charge. The parsed information is processed and stored in the data container $\mathbb{C}^P$. This data container is the exclusive data-source $\mathbb{C}^P$ for the rest of the design flow. It therefore becomes obvious that the XML parsing front-end can be exchanged for or mixed up with any parsing front-end of any well-defined language. Based on the stored information in $\mathbb{C}^P$, which comprises the operands and operators of the energy-functional, their dependencies among each other, the optimization method, the kind of the neighborhood system and naming conventions, the graph-generation and graph-expansion of the topology & structure representing graphs $\mathcal{G}^{TS}$ can begin. This process will be introduced and explained in the following section.

5.3.1 Expanding the Topology & Structure Graphs

According to Definition 5.3 the topology and structure representing graph set $\mathcal{G}^{TS}$ first comprises the graph $\mathbb{G}^{GT}$, which models the site hulls, the neighborhood wiring and the port memory building blocks of the massively parallel architecture. Secondly it comprises the graph $\mathbb{G}^{MH}$, which models the globally distributed memory hierarchy building block. The representing graph $\mathbb{G}^{GT}$ of topology and structure components of a specific Markov Random Field model is expanded respectively generated by the Algorithm 5.1 based on the stored information of $\mathbb{C}^P$ and $\mathbb{C}^{ST}$.

Algorithm Description - The algorithm of generating graph $\mathbb{G}^{GT}$ works as follows: Firstly, the algorithm requires the elementary data containers $\mathbb{C}^{P}$ and $\mathbb{C}^{ST}$, which store fundamental information. Furthermore the algorithm needs a data-structure *Predecessor_List* with stack-functionalities and the two values $SPLIT_x = 2$ and $SPLIT_y = 2$, which define the x-direction and y-direction division-factors of the site-grids. These values have been set to the value two, since this will always

Algorithm 5.1 $\mathbb{G}^{GT}$ Generator

Require: $\mathbb{C}^{P}$, $\mathbb{C}^{ST}$, $SPLIT_x = 2$, $SPLIT_y = 2$, Predecessor_List
1: EXTRACT($Grid_x$, $Grid_y$, $\min_x$, $\min_y$) $\in \mathbb{C}^{ST}$;
2: GENERATE($GT_{TOP}(Grid_x, Grid_y)$); // Graph-level 0 per Def. 5.4
3: PUSH_BACK(GT_{TOP});
4: **while** HEAD[$Grid_x$] $\leq \min_x \vee$ HEAD[$Grid_y$] $\leq \min_y$ **do**
5: **for** $k = 1$; $k <= \infty$; $2 * (k++)$ **do**
6: **for** $i = 1$; $i <= (SPLIT_x * SPLIT_y)$; $i++$ **do**
7: $\lfloor$ X $\rfloor$ =HEAD[$Grid_x$]/$SPLIT_x$; $\lceil$ Y $\rceil$ =HEAD[$Grid_y$]/$SPLIT_y$;
8: GENERATE($cluster_{i,k}(X, Y) \in V_{MH}$);
9: GENERATE($Const_{i,k+1} \in V_{PhysConMH}$);
10: **if** $k = 1$ **then**
11: GENERATE($e_{i,1} \in E_{TOP}$);
12: CONNECT(HEAD(Predecessor_List), $e_{i,1}$, $cluster_{i,k}$);
13: GENERATE($e_{i,k} \in E_{PhysConMH}$);
14: CONNECT($cluster_{i,k}$, $e_{i,1}$, $Const_{i,k+1}$);
15: **else**
16: GENERATE($e_{i,k} \in E_{MH}$);
17: CONNECT(HEAD(Predecessor_List), $e_{i,k}$, $cluster_{i,k}$);
18: GENERATE($e_{i,k} \in E_{PhysConMH}$);
19: **end if**
20: PUSH_BACK($cluster_{i,k}$);
21: **end for**
22: POP_FRONT();
23: **end for**
24: **end while**

generate four children of each parent node. This organizes the chip-floorplanning and placing task with respect to the four quadrants in the chip-plane. In line 1 of Algorithm 5.1 the data necessary to run the algorithm becomes extracted from $\mathbb{C}^{ST}$. This comprises the overall MRF grid-size in x- and y-direction ($Grid_x$, $Grid_y$) as well as the smallest size of site-clusters ($\min_x$ and $\min_y$) within the expanded graph $\mathbb{G}^{GT}$. Lines 2-3 generate the top node GT_{TOP} of $\mathbb{G}^{GT}$.

Line 4 is the while-loop, which checks whether the smallest size - defined by $\min_x$ and $\min_y$ - of each site-cluster in the *Predecessor_List* has been reached. Line 5 denotes the counter for the k graph levels, which doubles with each iteration because the graph possesses $cluster_{\cdot,\cdot}$ nodes at graph level k and the corresponding physical constraints at level $k+1$. The next $cluster_{\cdot,\cdot}$ nodes are located at graph level $k + 2$. Line 6 realizes the counter for the particular $cluster_{\cdot,\cdot}$ nodes and its physical constraints $Const_{\cdot,\cdot}$. The counter value is calculated by means of the product of

the x- and y-direction split factors $SPLIT_x$ and $SPLIT_y$. Lines 7-22 generate the different nodes and edges of the graph $\mathbb{G}^{GT}$ and connect them among each other in order to expand the complete graph $\mathbb{G}^{GT}$. Line 7 extracts the x and y grid-size of the first element of the *Predecessor_List* and divides these values with the corresponding predefined split factors $SPLIT_x$ and $SPLIT_y$ in order to calculate the two values X and Y. These two values serve as name-components of the $cluster_{\cdot,\cdot}$ node-generation process in line 8. Line 8 generates particular $cluster_{\cdot,\cdot} \in V_{MH}$ nodes, which are named by a predefined string extended by the values X and Y as well as uniquely indexed by i and k. Furthermore, line 9 generates the corresponding physical-constraint nodes $Const_{\cdot,\cdot} \in V_{PhysConMH}$, which are indexed by i and $k+1$.

Lines 10-15 generate the corresponding edges and connect the $cluster_{\cdot,\cdot}$ and $Const_{\cdot,\cdot}$ nodes, which have already been generated in lines 8-9, with the GT_{TOP} node respectively the physical constraints with their cluster nodes. The graph level check $k = 1$ (line 10) within the algorithm and the following processing steps of lines 11-14 become necessary as all $cluster_{\cdot,1}$ nodes at graph level $k = 1$ have to be connected to the GT_{TOP} node by means of special edges $e_{i,1} \in E_{TOP}$ (cf. Definition 5.4). These specific connections between the top-node GT_{TOP} and the cluster nodes at the next graph level are mainly substantiated by software-technical implementation requirements. Lines 16-18 connect all generated nodes (lines 8-9) with a graph level $k >= 3$.

Line 20 pushes all newly generated cluster nodes to the end of the *Predecessor_List* for further processing. Line 22, which lies outside of the for-loop of the index i, pops the top element from the *Predecessor_List* and thus removes this element since its processing has been completed. If the conditions of the while-loop (line 4) are evaluated as being true, Algorithm 5.1 terminates.

After a successful termination of the algorithm, the canonical design representation contains a fully expanded graph $\mathbb{G}^{GT}$. This graph can be specifically selected and processed within the canonical design representation by means of its top node GT_{TOP}, which serves as central and only entry point into this graph structure.

Obviously, the introduced Algorithm 5.1 can produce different graphs $\mathbb{G}^{GT}$ with various shapes, depending on the parameter pairs $SPLIT_x$, $SPLIT_y$ and $\min_x$, $\min_y$. Let us for instance consider the following extreme case: If the values of $SPLIT_x$ and $SPLIT_y$ are equal to the overall site-grid size $Grid_x$ and $Grid_y$, then the algorithm expands a graph $\mathbb{G}^{GT}$, which at graph level 1 possesses a total number of $Grid_x \times Grid_y$ $cluster_{i,1}(1,1)$ nodes. Each of these $cluster_{i,1}(1,1)$ nodes model a particular site of the grid.

The second graph of the topology and structure representing graph set $\mathcal{G}^{TS}$ is graph $\mathbb{G}^{MH}$, which models the global and distributed memory hierarchy of the massively parallel system architecture. This specific graph $\mathbb{G}^{MH}$ is expanded respectively generated by Algorithm 5.2. Obviously, the algorithm can generate differently shaped $\mathbb{G}^{MH}$ graphs, according to its formal Definition 5.5. The expanding Algorithm 5.2 of graph $\mathbb{G}^{MH}$ operates in the same way as the graph $\mathbb{G}^{GT}$ expanding Algorithm 5.1. However, the graph $\mathbb{G}^{MH}$ is completely separated from $\mathbb{G}^{GT}$ within the canonical design representation, and $\mathbb{G}^{MH}$ models a fundamentally different part of the massively parallel hardware architecture.

Algorithm Description - Without detailing all steps of Algorithm 5.2, which have already been exhaustively explained for Algorithm 5.1, the generator procedure of graph $\mathbb{G}^{MH}$ works as follows: The algorithm requires the elementary data containers, the split parameter and a list-data structure, which possesses stack functionalities. Lines 1-3 cover the initialization sequence of the algorithm. Firstly, important parameters are extracted from the elementary data container (cf. Section 5.2.1) and stored in internal variables of the algorithm. Secondly, the top node MH_{TOP} of

Algorithm 5.2 $\mathbb{G}^{MH}$ Generator

Require: $\mathbb{C}^{P}$, $\mathbb{C}^{ST}$, $SPLIT_{x*y}$, Predecessor_List
1: EXTRACT($MemSize_{x*y}$, $\min_{x*y}$) $\in \mathbb{C}^{ST}$;
2: GENERATE($MH_{TOP}(MemSize_{x*y})$); // Graph-level 0 per Def. 5.5
3: PUSH_BACK(MH_{TOP});
4: **while** HEAD[$MemSize_{x*y}$] $\leq \min_{x*y}$ **do**
5: **for** $k = 1$; $2 * (k++)$ **do**
6: **for** $i = 1$; $i <= (SPLIT_{x*y})$; $i++$ **do**
7: MEMSIZE=HEAD[$MemSize_{x*y}$]/$SPLIT_{x*y}$;
8: GENERATE($block_{i,k}(MEMSIZE) \in V_{MH}$);
9: GENERATE($Const_{i,k+1} \in V_{PhysCon}$);
10: **if** $k = 1$ **then**
11: GENERATE($e_{i,1} \in E_{TOP}$);
12: CONNECT(HEAD(Predecessor_List), $e_{i,1}$, $block_{i,k}$);
13: GENERATE($e_{i,k} \in E_{PhysConGT}$);
14: CONNECT($block_{i,k}$, $e_{i,1}$, $Const_{i,k+1}$);
15: **else**
16: GENERATE($e_{i,k} \in E_{GT}$);
17: CONNECT(HEAD(Predecessor_List), $e_{i,k}$, $block_{i,k}$);
18: GENERATE($e_{i,k} \in E_{PhysConGT}$);
19: **end if**
20: PUSH_BACK($block_{i,k}$);
21: **end for**
22: POP_FRONT();
23: **end for**
24: **end while**

the global and distributed memory hierarchy graph $\mathbb{G}^{MH}$ is generated and pushed onto the *Predecessor_List* for further processing by the following algorithmic steps.

Lines 4-24 represent the complete graph expansion procedure in which all nodes and edges of the graph $\mathbb{G}^{MH}$ are generated and connected with each other. Line 4 represents the outermost while-loop of the algorithm, which terminates as soon as the top node of the *Predecessor_List* has reached the predefined minimum of the represented memory block. Lines 5-6 realize the two increasing parameters k and i, where k represents the graph level index and i the index of the nodes on each graph level k. Lines 7-9 determine one part of the node name (line 7) and generate the corresponding node and its physical constraint node.

Lines 10-14 check whether the previously generated nodes have to be connected with the top node MH_{TOP}. If this is true, the corresponding edges are generated and

connected with the top node as well as with the already generated nodes. If this is not true, the algorithm continues with lines 16-18, which generate the corresponding edges and connect them with the nodes located within the graph, i.e. nodes with a graph level $k > 2$.

Line 20 adds the newly generated nodes $block_{.,.}$ to the end of the $Predecessor\_List$. Line 22 finally removes the top element of $Predecessor\_List$ after the processing is finished.

From the viewpoint of VLSI implementations, where the particular functional units are composed of transistors, which are arranged in the plane, the split-factor $SPLIT_{x*y} = 4$ of Algorithm 5.2 is advantageous and preferable compared to other split-factors. A split-factor of 4 results in a sub-graph $\mathbb{G}^{MH}/V_{PhysCon} \cup E_{PhysConGT}$, which possesses a quad-tree structure, i.e. each $block_{.,.}$ node has exactly four $block_{.,.}$ children-nodes. As each $block_{.,.}$ node represents a memory-unit of a specific size within the graph $\mathbb{G}^{MH}$, it follows from a quad-tree structure of $\mathbb{G}^{MH}$ that the memory-unit size of each node is always divided up by four and represented by four $block_{.,.}$ children-nodes.

Consequently, this quad-tree structure determines the overall data-flow speed all the way through the global and distributed memory hierarchy. When abstracting from physical clock-cycles as a processing-step measure, which is prevalent in digital semiconductor technologies, we use δ-steps as a measure. A single δ-step comprises all tasks, which have to be executed to transfer a single datum from one level of the memory hierarchy to the next level.

For a quad-tree structure with M and N denoting the x and y size of the MRF respectively, the following theorem holds true:

Theorem 5.7 (δ-steps for a quad-tree $\mathbb{G}^{MH}$)
The δ-steps of the data flow within the global and distributed memory hierarchy, which is represented by the graph $\mathbb{G}^{MH}$, can be approximated by $\frac{3}{2}M \times N$ δ-steps upper-bound for one direction (completely up or down the memory hierarchy) if $\mathbb{G}^{MH}/V_{PhysCon} \cup E_{PhysConGT}$ is a quad-tree.

Proof: At the top level of the memory hierarchy all data has to be stored in one single memory-unit and serially extracted from this memory-unit, a process which requires $M \times N$ δ-steps. At the next level four particular memory-units - represented by the $block_{.,.}$ nodes of the quad-tree graph $\mathbb{G}^{MH}$ - are available in order to distribute the data in $\frac{1}{4}M \times N$ δ-steps to the next level of the memory hierarchy. This scheme continues with $\frac{1}{8}M \times N$ δ-steps, $\frac{1}{16}M \times N$ δ-steps and $\frac{1}{2^n}M \times N$ δ-steps for the consecutive levels of the memory hierarchy. The numerical series

$$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots + \frac{1}{2^n} + \ldots = 2, \tag{5.14}$$

describes this scheme, except for the second term $\frac{1}{2}$. We have thus reached the upper-bound of $\frac{3}{2}M \times N$ δ-steps, which proves the Theorem.

□

The result of $\frac{3}{2}M \times N$ δ-steps for one direction, either completely down or up the memory hierarchy represents only a rough upper-bound of the required steps.

A more precise δ-step-approximation and upper-bound is given by

$$\sum_{i=1}^{2} \left\lfloor \frac{(M \times N)}{(i * 4) - 4} \right\rfloor . \tag{5.15}$$

The particular VLSI results of the global memory hierarchy, presented in Section 5.5, and Appendix B, are based on quad-tree graphs $\mathbb{G}^{MH}/V_{PhysCon} \cup E_{PhysConGT}$, due to the resulting advantageous features for the processing steps of the VLSI-implementation chain.

5.3.2 Expanding the Processing Graphs

The graph set $\mathcal{G}^{PC}$, which comprises the graphs representing processing-functionality, consists of three particular graphs in accordance with Definition 5.6. This is graph $\mathbb{G}^{EF}_{PC} \in \mathcal{G}^{PC}$, which represents the prototype of the energy functional embedded in each particular processing cell. As it is assumed that the energy functionals of all processing cells are functionally and structurally identical, only a single modeling graph $\mathbb{G}^{EF}_{PC}$ is needed. Furthermore the graph $\mathbb{G}^{OPT}_{PC} \in \mathcal{G}^{PC}$ models the optimization method, which is used to determine a solution of the image processing problem. Finally the graph $\mathbb{G}^{PAR}_{PC} \in \mathcal{G}^{PC}$ models the processing-functionality of the parameter estimation task within the MRF system. The generation procedure for the afore mentioned graphs $\mathbb{G}^{\cdot}_{PC} \in \mathcal{G}^{PC}$ will be presented and discussed in the upcoming paragraph.

Algorithm Description - Algorithm 5.3, which generates the graphs $\mathbb{G}^{\cdot}_{PC} \in \mathcal{G}^{PC}$, executes as follows: The algorithm requires the elementary data container $\mathbb{C}^{P}$, which stores the parsed specifications of the MRF processing-functionalities. Additionally, two dynamical list data-structures, *Spec_List* and *Node_List*, are needed for operating of Algorithm 5.3. The naming of the lists already makes clear that *Spec_List* stores specifications extracted from $\mathbb{C}^{P}$ and *Node_List* stores nodes of the graphs $\mathbb{G}^{\cdot}_{PC} \in \mathcal{G}^{PC}$. Line 1 is the outermost $for\ all - loop$, which makes sure that all graphs of $\mathcal{G}^{PC}$ will be expanded. Next, line 2 generates the specific root-node of the graph $\mathbb{G}^{\cdot}_{PC} \in \mathcal{G}^{PC}$, which is currently generated. The following line 3 extracts the specifications of the graph $\mathbb{G}^{\cdot}_{PC}$ from $\mathbb{C}^{P}$, works them up and stores them in *Spec_List* for additional processing.

Within lines 4 - 9 the algorithm generates the first nodes of the graphs $\mathbb{G}^{\cdot}_{PC}$, which model the inputs (signals as well as data) of the corresponding processing building block. Line 4 scans the *Spec_List* and extracts all specifications, which define the $INPUTS$ of the building block. For these $INPUTS'$ parts line 5 generates a special *op.* node. The following line 6 pushes this node onto the *Node_List* for further processing. Line 7 generates an edge $e_{\cdot,\cdot}$ to connect the generated node with its predecessor, in this case with the root-node. The connection is established in line 8 of the algorithm. So far the explained algorithm generates the portion of the graphs, which consists of the root-nodes as well as the nodes of the input signals and data.

Lines 10 - 25 generate the rest of the graphs. The $for\ all - loop$ in line 10 selects all nodes within the *Node_List*. Line 11, directly following, starts another

Algorithm 5.3 $\mathbb{G}^{\cdot}_{PC} \in \mathcal{G}^{\mathcal{PC}}$ Generator

Require: $\mathbb{C}^{P}$, Spec_List, Node_List

```
 1: for all G^·_PC ∈ G^PC do
 2:   GENERATE(Root_Node of G^·_PC);
 3:   Spec_List = EXTRACT(∀ Specs of G^·_PC ∈ C^P);
 4:   for all Specs[INPUTS] ∈ Spec_List do
 5:     GENERATE(op.);
 6:     PUSH_BACK_NL(op.)
 7:     GENERATE(e.,.);
 8:     CONNECT(Root_Node, e.,., op.);
 9:   end for
10:   for all op. ∈ Node_List do
11:     for all Specs ∈ Spec_List do
12:       if op.[Result] = Specs[Operand] then
13:         if op.[Specs[Operand]] ∈ G^·_PC then
14:           GENERATE(e.,.);
15:           CONNECT(op., e.,., op.[Specs[Operand]]);
16:         else
17:           GENERATE(op.[Specs[Operand]]);
18:           PUSH_BACK_NL(op.)
19:           GENERATE(e.,.);
20:           CONNECT(op., e.,., op.[Specs[Operand]]);
21:         end if
22:       end if
23:     end for
24:   end for
25: end for
```

$for\ all-loop$, which runs over all $Specs \in Spec\_List$. These two $for\ all-loops$ ensure that elements of $Node\_List$ and $Spec\_List$ can be processed among each other. This process is executed in the following line 12. The $if-case$ of line 12 checks whether the result of the operation, represented by the node *op.*, is used as an operand within any other operation, which will be searched for in the $Spec\_List$ (Specs[Operand]). As soon as a match between $op.[Result]$ and $Specs[Operand]$ has been found, two different cases have to be considered. In the first case (line 13 - line 15) the graph $\mathbb{G}^{\cdot}_{PC}$ already contains the node $op.[Specs[Operand]]$ so that only one edge and its connection has to be established. In the second case (lines 16 - 21) also the node $op.[Specs[Operand]]$ itself has to be generated. The generation of the unique indexes i, j for the nodes op_i and the edges $e_{i,j}$ have been omitted in Algorithm 5.3 and the preceding discussion to tighten the overall presentation.

So far all graphs $\mathbb{G}^{\cdot}_{PC} \in \mathcal{G}^{\mathcal{PC}}$ - expanded by Algorithm 5.3 - just model the different operations, operands and data-dependencies of the building blocks representing processing-functionality. None of these graphs contains any directly usable information, which determines the execution order of the different operations [51]. The data-dependencies of the graphs contain the required information, but not in

a directly usable form. Hence, these data-dependencies have to be systematically exploited in order to extract parallel processing capabilities as well as to establish a well-defined, deterministic and overall processing order for the operations of each particular graph. This is done by scheduling-algorithms, which assign to each node of the graph $\mathbb{G}_{PC}^{\cdot} \in \mathcal{G}^{\mathcal{PC}}$ an integer-valued mark, which defines the scheduling step of this operation. Several nodes can possess the same mark (scheduling-step) and consequently will be executed in parallel. All these marks form a strict, monotonously increasing sequence of integer values and thus a well-defined order of execution.

Scheduling steps and scheduling sequences are formally presented in Definition 5.12, which reads

Definition 5.12 Scheduling Step & Scheduling Sequence
A scheduling step denoted by $step_j = \{op_k | k = 1, ..., n; j = 1, ..., m\}$ *is an agglomeration of n different operators op., which can be executed in parallel in this scheduling step without violating the data dependencies and the flow of information between the operators defined by any of the data flow graphs* $\mathbb{G}_{PC}^{\cdot} \in \mathcal{G}^{\mathcal{PC}}$.

A scheduling sequence is a strict, monotonously increasing sequence of integer values $\{j | j = 1, ..., m\}$, *which define the partial ordering* $\{j \leq j + 1 | \forall j = 1, ..., m\}$ *on the scheduling steps.*

As soon as each of the graphs $\mathbb{G}_{PC}^{\cdot} \in \mathcal{G}^{\mathcal{PC}}$ have been enriched with the information of *Scheduling-Steps*, which together define the strict, monotonously increasing and unique *Scheduling Sequence*, it becomes possible to expand the corresponding control-graphs (see Section 5.3.3) for those processing parts of the MRF architecture. A processing graph and a corresponding control graph together form a tuple $(\mathbb{G}_{PC}^{\cdot}, \mathbb{G}_{CT}^{\cdot})$ (cf. Corollary 5.3), which completely models the processing-functionality in a digital-electronic appropriate kind. Each of the algorithms, described in the following and realized within the VLSI design framework, enrich the graphs $\mathbb{G}_{PC}^{\cdot} \in \mathcal{G}^{\mathcal{PC}}$ with scheduling step marks by means of different strategies. At first the so called ASAP (As-Soon-As-Possible) scheduling algorithm [60] will be presented. This algorithm assigns an operation *op.* to a specific scheduling step as soon as the data-dependencies allow this action to be performed.

Algorithm Description - The As-Soon-As-Possible (ASAP) [51] scheduling algorithm is outlined in Algorithm 5.4 and works as follows: This scheduling method requires only the graph set $\mathcal{G}^{\mathcal{PC}}$ to be executed correctly. Any additional information is already embedded within the structure of the graphs $\mathbb{G}_{PC}^{\cdot} \in \mathcal{G}^{\mathcal{PC}}$ or will be added by the algorithm step-by-step and re-used later on.

Line 1 is the outermost $for\ all - loop$, which runs over all graphs $\mathbb{G}_{PC}^{\cdot} \in \mathcal{G}^{\mathcal{PC}}$. The scheduling algorithm ensures that to each node $op_i \in V_{PC}^{\cdot}$ an integer valued mark $Index_i^{step}$ is assigned, which represents the single time steps operations are executed in. More than one node $op_i \in V_{PC}^{\cdot}$ can be assigned by the same mark $Index_i^{step}$, which means that these operations are executed parallel during this time step. Naturally it is assumed that the scheduling algorithm respects the data dependencies between the single operations op_i. The expression of $Predecessors_{op_i}$ denotes the list of direct predecessor operations of the node op_i. These predecessor operations are uniquely determined by the structure of the corresponding $\mathbb{G}_{PC}^{\cdot}$

graph. The help function $\max_{Index}(Predecessors_{op_i})$ returns the maximum value $Index$ of all predecessors of operation op_i.

Algorithm 5.4 ASAP Schedule $\forall \dot{\mathbb{G}}_{PC} \in \mathcal{G}^{\mathcal{PC}}$

Require: $\mathcal{G}^{\mathcal{PC}}$

1: **for all** $\dot{\mathbb{G}}_{PC} \in \mathcal{G}^{\mathcal{PC}}$ **do**
2: **for all** $op_i \in \dot{V}_{PC}$ **do**
3: **if** $Predecessors_{op_i} = \emptyset$ **then**
4: $Index_i^{step} = 1$;
5: **else**
6: $Index_i^{step} = 0$;
7: **end if**
8: **end for**
9: **while** $\dot{V}_{PC} \neq \emptyset$ **do**
10: **for all** $op_i \in \dot{V}_{PC}$ **do**
11: **if** $\forall Predecessors_{op_i}$ holds $Index^{step} \neq 0$ **then**
12: $Index_i^{step} = \max_{Index}(Predecessors_{op_i}) + OperationSteps(op_i)$;
13: $\dot{V}_{PC} = \dot{V}_{PC} - \{op_i\}$;
14: **end if**
15: **end for**
16: **end while**
17: **end for**

Lines 2 - 8 of Algorithm 5.4 represent the initializing sequence of the ASAP scheduling-method. Here to all nodes op_i, which have no direct predecessors, are assigned the value $Index_i^{step} = 1$ and are thus scheduled to the first time step. All other nodes obtain the initializing value $Index_i^{step} = 0$. Lines 9 - 16 describe the crucial part of the scheduling method, where each node is assigned to a time step represented by the value $Index^{step}$. The assignment process of $Index^{step}$ continues as long as to all nodes op_i have been assigned a valid index (line 9). The following $for\ all - loop$ of line 10 ensures that those nodes previously not scheduled are processed later, as long as a valid schedule is possible. Lines 11 - 14 assign to the node op_i a valid index as soon as to all its predecessors have before been assigned a valid index. The current index value of node op_i is derived from the largest index value of its predecessors.

Another scheduling method is called ALAP (As-Late-As-Possible) scheduling and will be presented in the paragraph directly following. This method assigns each operation *op.* to the latest scheduling-step this specific operation can be executed [60]. ALAP is regularized by a given constraint T of maximum scheduling steps.

Algorithm Description - The As-Late-As-Possible (ALAP) [51] scheduling method is depicted in Algorithm 5.5 and works as follows: This scheduling algorithm requires the graph set $\mathcal{G}^{\mathcal{PC}}$ as well as the afore mentioned constraint T of maximum scheduling steps in order to run correctly. Any kind of additional information is already represented by the structure of the graphs $\dot{\mathbb{G}}_{PC} \in \mathcal{G}^{\mathcal{PC}}$ or will be added by the algorithm.

Line 1 is the outermost $for\ all-loop$, which runs over all graphs $\mathbb{G}_{PC}^{\cdot} \in \mathcal{G}^{\mathcal{PC}}$ to allow all these graphs to become equipped with the ALAP mark after the successful execution of this scheduling algorithm. In contrast to the ASAP scheduling method it is possible that the ALAP scheduling method completely fails because the constraint T has been improperly chosen. To each node $op_i \in V_{PC}^{\cdot}$ is assigned an integer valued mark $Index_i^{step}$, which represents the scheduling step this operation becomes executed in. Obviously, to more than one node $op_i \in V_{DFG_{EF}}$ can be assigned the same mark $Index_i^{step}$, i.e. the operations with identical marks execute in parallel during this scheduling step. Of course it is assumed that the ALAP scheduling algorithm respects the data dependencies, which are defined by the structure of the corresponding graph $\mathbb{G}_{PC}^{\cdot} \in \mathcal{G}^{\mathcal{PC}}$. The expression $Successors_{op_i}$ denotes

Algorithm 5.5 ALAP Schedule $\forall \mathbb{G}_{PC}^{\cdot} \in \mathcal{G}^{\mathcal{PC}}$

Require: $\mathcal{G}^{\mathcal{PC}}$, T
1: **for all** $\mathbb{G}_{PC}^{\cdot} \in \mathcal{G}^{\mathcal{PC}}$ **do**
2: **for all** $op_i \in V_{PC}^{\cdot}$ **do**
3: **if** $Successors_{op_i} = \emptyset$ **then**
4: $Index_i^{ALAP} = T$;
5: $V_{PC}^{\cdot} = V_{PC}^{\cdot} - \{op_i\}$;
6: **else**
7: $Index_i^{ALAP} = 0$;
8: **end if**
9: **end for**
10: **while** $V_{PC}^{\cdot} \neq \emptyset$ **do**
11: **for all** $op_i \in V_{PC}^{\cdot}$ **do**
12: **if** $\forall Predecessors_{op_i}$ holds $Index^{ALAP} \neq 0$ **then**
13: **if** $\min_{Index}(Successors_{op_i}) - OperationSteps(op_i) < 0$ **then**
14: $Index_i^{ALAP} = \min_{Index}(Successors_{op_i}) - OperationSteps(op_i)$;
15: **else**
16: break with ERROR;
17: **end if**
18: $V_{PC}^{\cdot} = V_{PC}^{\cdot} - \{op_i\}$;
19: **end if**
20: **end for**
21: **end while**
22: **end for**

the list of the direct successor nodes with respect to the node op_i. The successors are uniquely determined by the graph structure of $\mathbb{G}^{DFG_{EF}}$. The help function $\min_{Index}(Successors_{op_i})$ returns the minimum value $Index$ of all successors of the contemplated node op_i.

Within lines 2 - 8 of Algorithm 5.5 the currently processed graph $\mathbb{G}_{PC}^{\cdot} \in \mathcal{G}^{\mathcal{PC}}$ is initialized, i.e. all nodes, which do not possess successors are assigned to the last possible state (line 4), defined by the constraint T and the other nodes are assigned to the first state (line 6). Lines 10 - 21 describe the crucial part of the ALAP scheduling method where to each node is assigned a time step represented by the value of $Index^{step}$. The $Index^{step}$ assignment process continues as long as to

all nodes op_i are assigned a valid index (line 14). In the ALAP scheduling method it becomes possible that negative values of $Index^{step}$ are generated, in case the constraint T was chosen improperly. This case is checked in line 13 to allow the complete algorithm to be terminated in line 16 as soon as the value of $Index^{step}$ becomes negative. The $for-loop$ of line 11 ensures that all previously not scheduled nodes will be processed during a later process-pass of the scheduling algorithm. Line 14 assigns to the node op_i a valid index, when to all its successors were assigned a valid index before. The current index value of node op_i is derived from the smallest index value of its predecessors.

5.3.3 Expanding the Control Graphs

The set of control-unit representing graphs $\mathcal{G}^{\mathcal{CT}}$ (cf. Definition 5.10) comprises the system control-unit graph $\mathbb{G}_{CT}^{System}$, which handles the overall sequence of operations and data-flows within the MRF device. This control unit triggers and stops the different parts of the architecture to allow a well-defined processing sequence to be always guaranteed. The control-unit graph $\mathbb{G}_{CT}^{GMem}$ manages the operation of the globally distributed memory hierarchy and is triggered and stopped by the system control-unit. Furthermore, the port memory control-unit graph $\mathbb{G}_{CT}^{PortMem}$ models the data transfer between the particular processing elements. Depending on the data transfer strategy, which can be vary from being fully parallel to being fully serial, the graph $\mathbb{G}_{CT}^{PortMem}$ is generated to cover the corresponding data-transfer functionality. $\mathbb{G}_{CT}^{EF}$ is the graph, which models the control-unit of the energy-functional. Finally $\mathbb{G}_{CT}^{System}$ comprises, the control-unit representing graph $\mathbb{G}_{CT}^{OPT}$ that handles the execution of the optimization method. In the following we present the particular graph-generation algorithms one after the other for all control-unit representing graphs $\mathcal{G}^{\mathcal{CT}}$.

Algorithm Description - Algorithm 5.6 describes the generation process of the system control graph $\mathbb{G}_{CT}^{System}$, which the proposed design framework is currently executing. The algorithm works as follows: Two dynamic list data-structures named $Predecessor\_List\_1$, $Successor\_List\_1$ with $PUSH$, $CLEAR$ and copying functionalities next to the generated graph data structure are the only additional algorithmic conditions for Algorithm 5.6. As the names of the list data-structures already suggest, one list contains all direct predecessors and the other list all direct successors.

Line 1 generates the root-node of $\mathbb{G}_{CT}^{System}$, which serves as an unique identification-node for this graph and pushes it onto the predecessor list. Line 2 fills the successor list with the $Sync\_Reset$ and $ASync\_Reset$ nodes, which have to be connected with the top root-node of the predecessor list.

The connection procedure runs in lines 3 - 7. Obviously, line 5 represents an essentially condensed instruction, which executes several different tasks. At first the corresponding edge between the nodes Pre_Nodes and Suc_Nodes is generated and marked with the state transition conditions encoded by $CONDITIONS$.

Secondly, this previously generated edge will then be connected to its nodes to form a valid sub-graph. If the $for-loops$ are finished, line 8 copies the successor list to the predecessor list and in line 9 completely clears the successor list. Thus new nodes can be generated and pushed into the successor list and in the following

Algorithm 5.6 Synthesizer FSM for $\mathbb{G}_{CT}^{System}$

Require: Predecessor_List_1, Successor_List_1

```
 1: PUSH_FRONT_PL1(GENERATE(Root_Node of G_CT^System));
 2: PUSH_BACK_SL1(GENERATE(Sync_Reset),GENERATE(ASync_Reset));
 3: for all Pre_Nodes ∈ Predecessor_List_1 do
 4:    for all Suc_Nodes ∈ Successor_List_1 do
 5:       CONNECT(Pre_Nodes, Suc_Nodes, CONDITIONS);
 6:    end for
 7: end for
 8: Predecessor_List_1 = Successor_List_1;
 9: CLEAR(Successor_List_1);
10:    PUSH_BACK_SL1(GENERATE(IDLE));
11:       EXECUTE Line 3 - Line 9;
12:    PUSH_BACK_SL1(GENERATE(Start_Cell),GENERATE(Start_Mem));
13:    PUSH_BACK_SL1(GENERATE(Start_MemCell));
14:       EXECUTE Line 3 - Line 9;
15:    PUSH_BACK_SL1(GENERATE(RETURN_STATE));
16:       EXECUTE Line 3 - Line 9;
17:    CONNECT(RETURN_STATE, IDLE, CONDITIONS)
```

can be connected with the nodes of the predecessor list. The processing of lines 3 - 9 is repeated in the current algorithm as well as in all other control graph generation algorithms. In order to tighten the presentation we will use the statement *EXECUTE Line 3 - Line 9 of Algorithm 5.6* to refer to the repeated processing just described. Lines 10 - 16 complete the graph $\mathbb{G}_{CT}^{System}$ by means of the described repeating scheme. Algorithm 5.6 closes with line 17, which generates a feedback-loop between the $RETURN\_STATE$ node and the $IDLE$ node.

Algorithm Description - The graph expansion procedure for $\mathbb{G}_{CT}^{GMem}$, which models the control-unit of the globally distributed memory hierarchy is explained in Algorithm 5.7. This algorithm is executed within the following steps: The algorithm requires the two elementary data-containers $\mathbb{C}^P$, $\mathbb{C}^{ST}$ (cf. Definition 5.1, 5.2) as well as the two dynamic list data-structures $Predecessor\_List\_1$, $Successor\_List\_1$ as a precondition to be able to operate.

Line 1 extracts two values from the elementary data container $\mathbb{C}^P$ and $\mathbb{C}^{ST}$, which together are necessary to generate the corresponding control graph of the global memory hierarchy for a specific MRF architecture. At first the overall grid size becomes extracted from $\mathbb{C}^{ST}$, which determines the maximum number of input/output values that have to be stored and transported within the memory hierarchy. Secondly, the number of parameters is extracted from $\mathbb{C}^P$, which, together with the input/output values, has to be stored and transported within the memory hierarchy, too. If necessary the value of $Parameter\_Size$ already covers the case in which parameters with bit-sizes larger than the memory hierarchy up- and down path have to be considered; i.e. a 16-bit parameter value is broken up into two parts in a 8-bit memory hierarchy setting and thus with a value of two contributes to the

overall *Parameter_Size* value.

In Line 2 both values *Grid_Size* and *Parameter_Size* are simply added to determine the overall size of *GMem* and thus the maximum number of the address range the control-unit counts up to. In Line 3 the root node of this control graph is generated and pushed into the predecessor list to allow Line 4 to generate the standard reset sequence and connect it with the root node of $\mathbb{G}_{CT}^{GMem}$.

Algorithm 5.7 Synthesizer FSM for $\mathbb{G}_{CT}^{GMem}$

Require: $\mathbb{C}^P$, $\mathbb{C}^{ST}$, Predecessor_List_1, Successor_List_1

1: EXTRACT(Grid_Size) $\in \mathbb{C}^{ST}$; EXTRACT(Parameter_Size) $\in \mathbb{C}^P$
2: GMem_Size = Grid_Size + Parameter_Size;
3: PUSH_FRONT$_{PL1}$(GENERATE(Root_Node of $\mathbb{G}_{CT}^{GMem}$));
4: EXECUTE Line 2 - Line 9 of Algorithm 5.6;
5: PUSH_BACK$_{SL1}$(GENERATE(IDLE));
6: EXECUTE Line 3 - Line 9 of Algorithm 5.6;
7: PUSH_BACK$_{SL1}$(GENERATE(ADDR_STATE));
8: EXECUTE Line 3 - Line 9 of Algorithm 5.6;
9: CONNECT(ADDR_STATE, ADDR_STATE, Addr $\leq$ GMem_Size);
10: PUSH_BACK$_{SL1}$(GENERATE(RETURN_STATE));
11: EXECUTE Line 3 - Line 9 of Algorithm 5.6;
12: CONNECT(RETURN_STATE, IDLE, CONDITIONS);

The following lines 5 - 6 generate the $IDLE$ node of this control graph and connect it with the previously added reset nodes. This finalizes the generation of the trunk $\mathbb{G}_{CT}^{GMem}$ graph. Lines 7 - 8 add the $ADDR\_STATE$ node to the graph, which models the address generation for the global memory hierarchy. This is why line 9 establishes a special feedback edge for the $ADDR\_STATE$ node, which controls the feedback. As long as the generated address *Addr* is smaller than or equals $GMem\_Size$, the feedback path will be used within the $\mathbb{G}_{CT}^{GMem}$ graph. If the $Addr \leq GMem\_Size$ condition becomes false, the path generated in Line 10-12 will be used. Line 10 generates the $RETURN\_STATE$ node and in Line 11 connects it to the $ADDR\_STATE$ node. Finally line 12 generates a feedback edge between the $RETURN\_STATE$ node and the $IDLE$ node.

Algorithm Description - An additional control-unit representing graph is $\mathbb{G}_{CT}^{PortMem} \in \mathcal{G}^{\mathcal{CT}}$. This specific graph models the port memories of the cells and the corresponding send- and receive-actions in order to coordinate the data transfer between the cells. Algorithm 5.8 describes the generation procedure for this control-unit representing graph $\mathbb{G}_{CT}^{PortMem}$ and works as follows: Both elementary data containers $\mathbb{C}^P$ and $\mathbb{C}^{ST}$ (cf. Definition 5.1, 5.2) as well as two list data structures are required as a precondition for this algorithm.

At first Line 1 extracts the port length respectively the port bit-width of the cells. As the bit-width of the cell ports is equal for all cells $n \in \Omega$ and also for all ports of the cells by definition of the architecture template, it is merely necessary to evaluate an arbitrary port specification in $\mathbb{C}^{ST}$. Furthermore Line 1 extracts the value of DIV, which determines the factor by which the port bit-width is divided from $\mathbb{C}^P$.

Algorithm 5.8 Synthesizer FSM for $\mathbb{G}_{CT}^{PortMem}$

Require: $\mathbb{C}^P$, $\mathbb{C}^{ST}$, Predecessor_List_1, Successor_List_1

```
 1: EXTRACT(Port_Length) ∈ C^ST; EXTRACT(DIV) ∈ C^P
 2: PUSH_FRONT_PL1(GENERATE(Root_Node of G_CT^PortMem));
 3:    EXECUTE Line 2 - Line 9 of Algorithm 5.6
 4: PUSH_BACK_SL1(GENERATE(IDLE_Send),GENERATE(IDLE_Receive));
 5:    EXECUTE Line 3 - Line 7 of Algorithm 5.6;
 6: CLEAR(Successor_List_1); CLEAR(Predecessor_List_1);
 7: PUSH_BACK_PL1(SEARCH(Root_Node, IDLE_Send));
 8: if MOD(Port_Length,DIV)==0 then
 9:    for (i=1; i=< (Port_Length/DIV); i++) do
10:       PUSH_BACK_SL1(GENERATE(Send_State(i)));
11:       EXECUTE Line 3 - Line 9 of Algorithm 5.6
12:    end for
13:    CONNECT(Pre_Nodes, IDLE_Send, CONDITIONS)
14:    CLEAR(Successor_List_1); CLEAR(Predecessor_List_1);
15:    PUSH_BACK_PL1(SEARCH(Root_Node, IDLE_Receive));
16:    for (i=1; i=< (Port_Length/DIV); i++) do
17:       PUSH_BACK_SL1(GENERATE(Receive_State(i)));
18:       EXECUTE Line 3 - Line 9 of Algorithm 5.6
19:    end for
20:    CONNECT(Pre_Nodes, IDLE_Receive, CONDITIONS)
21: else
22:    DIV = DIV+1;
23:    if Port_Length > DIV then
24:       GOTO Line 8;
25:    else
26:       BREAK;
27:    end if
28: end if
```

Lines 2 - 3 generate the root-node as well as the reset nodes and connect them in order to form the standard reset sequence all control units possess. Following lines 4 - 5 generate the $IDLE$ nodes for the send- and receive-part of the control-unit and connect them to the reset nodes. After this both list data-structures in Line 6 are cleared since the algorithm needs a new initialization to separately generate the state-nodes for the send part and the receive part.

Line 7 traverses the so-far generated graph, starting from the *Root_Node*, until the *IDLE_Send* node is found. This node is pushed into the predecessor list, to allow the additional send nodes to be appended. The outermost $if - condition$ checks whether the partitioning of the port length by DIV is correct, i.e. expression $mod(Port\_Length, DIV) == 0$ is valid. If this condition evaluates to be false, the algorithm increases DIV by one and checks whether *Port_Length* is still bigger than DIV (Lines 22 - 27), in order to have a valid port-length partitioning, and repeated Line 8. Lines 9 - 12 generate the send-nodes and connects them. The last send-node is connected with the *IDLE_Send* node by means of a feedback edge (Line 13). The

same procedure takes place with regard to the receive-nodes and becomes realized in lines 14 - 20. This finalizes the algorithm under normal conditions without disturbing abnormally the processing with any kind of fatal errors.

Algorithm Description - The graphs $\mathbb{G}_{CT}^{EF}$, $\mathbb{G}_{CT}^{OPT}$ and $\mathbb{G}_{CT}^{PAR}$ representing control-units are synthesized by Algorithm 5.9, which processes the graphs $\mathbb{G}_{PC}^{EF}$, $\mathbb{G}_{PC}^{OPT}$, $\mathbb{G}_{PC}^{PAR}$ and especially the embedded scheduling index in order to expand the corresponding control graphs. In detail this algorithm works as follows: Besides the processing graphs $\mathbb{G}_{PC}^{EF}$, $\mathbb{G}_{PC}^{OPT}$ and $\mathbb{G}_{PC}^{PAR}$, whose control graphs should be synthesized by Algorithm 5.9, three dynamic list data-structures $Predecessor\_List\_1$, $Successor\_List\_1$, and $Index\_List$ as well as the scheduling type $Schedule\_Type$ are required to execute this algorithm. The scheduling type determines, which indexes (ASAP, ALAP etc.) of the graphs $\mathbb{G}_{PC}^{\cdot}$ will be used during processing.

Line 1 starts the outermost $for-loop$, which selects the graphs $\mathbb{G}_{PC}^{EF}$, $\mathbb{G}_{PC}^{OPT}$ and $\mathbb{G}_{PC}^{PAR}$ one after the other for further processing. Lines 2 - 6 parse $\mathbb{G}_{PC}^{\cdot}$ and extract all scheduling indexes with respect to the $Scheduling\_Type$. But only indexes,

Algorithm 5.9 Synthesizer FSM for $\mathbb{G}_{CT}^{EF}$, $\mathbb{G}_{CT}^{OPT}$, $\mathbb{G}_{CT}^{PAR}$

Require: $\mathbb{G}_{PC}^{EF}$, $\mathbb{G}_{PC}^{OPT}$, $\mathbb{G}_{PC}^{PAR}$, Predecessor_List_1, Successor_List_1, Index_List, Schedule_Type

```
 1: for (G_PC^EF, G_PC^OPT, G_PC^PAR) do
 2:   for all (nodes ∈ G_PC^·) do
 3:     if (EXTRACT(Schedule_Type, Index) ∉ Index_List) then
 4:       PUSH_BACK_IL(EXTRACT(Schedule_Type, Index) );
 5:     end if
 6:   end for
 7:   SORT_≤(Index_List);
 8:   PUSH_FRONT_PL1(GENERATE(Root_Node of G_CT^·));
 9:     EXECUTE Line 2 - Line 9 of Algorithm 5.6
10:   PUSH_BACK_SL1(GENERATE(IDLE));
11:     EXECUTE Line 3 - Line 9;
12:   while (Index_List≠ ∅) do
13:     PUSH_BACK_SL1(GENERATE(POP_FRONT(Index_List)));
14:     EXECUTE Line 3 - Line 9 of Algorithm 5.6
15:   end while
16:   CONNECT(Pre_Nodes, IDLE, CONDITIONS)
17: end for
```

which are not already stored within the $Index\_List$ (line 3) are pushed (line 4) onto this list.

The following line 7 sorts the elements of the $Index\_List$ to allow the elements to be stored in an ascending order within the list. Consequently, the $Index\_List$ contains a sequence of indexes, which are arranged in accordance with Definition 5.12 about scheduling sequences. Lines 8 - 11 perform the generation of the standard sub-graph, which consists of the root node of the corresponding graph $\mathbb{G}_{CT}^{\cdot}$, the reset nodes, the $IDLE$ node as well as the connecting edges.

Lines 12 - 15 generate the nodes for the scheduling steps and connect them among each other. Therefore the first element of the *Index_List* always becomes processed and is popped from the list until the list is empty. Finally line 16 connects the last node of the generated graph, which models the last scheduling step, with the $IDLE$ node in order to start the control sequence again.

5.4 Compiling - From Abstract Graphs to Circuit Descriptions

After the graph generation process, which is specific for different MRF models, has successfully and completely been finished, the canonical design representation (CDR) contains a graph theoretical representation $\mathcal{G}^{\mathcal{D}}$ (cf. Corollary 5.6) of the corresponding massively parallel processing architecture. The different graphs of the CDR will be compiled, described in the next steps, into well-defined hardware descriptions, which can be handed over to industrially approved design flows for synthesis and technology specific placing & routing. Two IEEE standardized languages - VHDL and Verilog - are available today to describe hardware systems and to conduct industrially approved synthesis and technology back-end processes.

The language back-end part of our proposed VLSI design framework (see Figure 5.1) currently supports VHDL as output description and can be extended to support Verilog as well. The Verilog language back-end extension is possible because the representing graphs of $\mathcal{G}^{\mathcal{D}}$ are completely neutral with regard to hardware-description languages and the compilation process and its algorithms follow up deterministic procedures to allow the concrete language dependent constructs of VHDL and Verilog to be systematically exchanged. Because of the strict separation of the graph sets $\mathcal{G}^{TS}$, $\mathcal{G}^{PC}$ and $\mathcal{G}^{CT}$, which model the architectural functionalities of topology & structure, processing and control, respectively, the synthesis algorithms for each graph set can be specifically adjusted and tuned to match with the requirements the particular graph sets pose due to their different architecture-functionalities. The next three sections will describe the general synthesis approaches for each graph set as well as discuss the corresponding algorithms.

5.4.1 Compiling Topology & Structure Parts

In summary, the graph set $\mathcal{G}^{TS}$, which represents the topology & structure modeling graphs within the canonical design representation, in particular covers the graphs $\mathbb{G}^{GT}$ (cf. Definition 5.4) and $\mathbb{G}^{MH}$ (cf. Definition 5.5). Each graph of $\mathcal{G}^{TS}$ is processed by Algorithm 5.10 and the Function 5.11, which is called in line 4 and line 9. The procedure of compiling the HDL-code for all graphs $\mathbb{G}^{\cdot}$ of $\mathcal{G}^{\mathcal{TS}}$ has been split up into its two parts (Algorithm 5.10 and Function 5.11) in order to separate two principally different functionalities and also to improve the overall presentation of the topology&structure specific compilation procedure. Within the first part, represented by Algorithm 5.10, the different graphs of $\mathcal{G}^{TS}$ are selected, files are opened and closed, a top-level module wrapper becomes written and the nodes of $\mathbb{G}^{\cdot} \in \mathcal{G}^{\mathcal{TS}}$ become further processed by Function 5.11. Within the second part, represented by Function 5.11, the hardware description of a module, which is modeled by a partic-

ular node and its sub-graph, is synthesized. This function is executed by Algorithm 5.10, when traversing the topology & structure defining graphs of $\mathcal{G}^{TS}$.

Algorithm Description - The complete HDL-compilation procedure for the topology & structure defining architecture-parts in detail works as follows: Algorithm 5.10 requires the topology & structure representing graph set $\mathcal{G}^{TS}$ as well as a dynamic list data-structure with stack functionalities. As the name of the list data structure already suggests, this dynamic list stores nodes of the graph $\mathbb{G}^{\cdot} \in \mathcal{G}^{\mathcal{TS}}$, which is currently processed.

Line 1 is the outermost loop, which one after the other selects all graphs of $\mathcal{G}^{TS}$ for further processing. The selected graph $\mathbb{G}^{\cdot} \in \mathcal{G}^{\mathcal{TS}}$ is referenced by its root-node, which is not required for the algorithm, so that line 2 pushes the child-nodes to the *Node_List*. The information of the first element in the *Node_List* is used to build up a unique file-name in order to open the file in line 3 of the Algorithm 5.10. The following steps will write the hardware description of the module, which is represented by the node, to this file. Line 4 calls the function $SYNTHESIZE^{\mathcal{G}^{\mathcal{TS}}}_{HDL}(\mathbb{G}^{\cdot}, Node\_List)$ (see Function 5.11) with the two parameters $\mathbb{G}^{\cdot}$ and *Node_List*, which synthesizes the virtual hardware description. After returning from the function $SYNTHESIZE^{\mathcal{G}^{\mathcal{TS}}}_{HDL}(\cdot,\cdot)$ line 5 removes the first element of the *Node_List*, which has been processed. Finally line 6 closes the corresponding file. Within the function $SYNTHESIZE^{\mathcal{G}^{\mathcal{TS}}}_{HDL}(\cdot,\cdot)$ additional nodes of the

Algorithm 5.10 Compiling HDL-code from $\mathcal{G}^{TS}$

Require: $\mathcal{G}^{TS}$, Node_List
1: **for all** $\mathbb{G}^{\cdot} \in \mathcal{G}^{\mathcal{TS}}$ **do**
2: PUSH_FRONT$_{NL}$(CHILDREN(Root_Node of $\mathbb{G}^{\cdot}$)
3: OPEN$_{FILE}$(FRONT(Node_List));
4: SYNTHESIZE$^{\mathcal{G}^{\mathcal{TS}}}_{HDL}$($\mathbb{G}^{\cdot}$, Node_List); // Function 5.11
5: POP_FRONT(Node_List);
6: CLOSE$_{FILE}$();
7: **while** Node_List $\neq \emptyset$ **do**
8: OPEN$_{FILE}$(FRONT(Node_List));
9: SYNTHESIZE$^{\mathcal{G}^{\mathcal{TS}}}_{HDL}$($\mathbb{G}^{\cdot}$, Node_List); // Function 5.11
10: POP_FRONT(Node_List);
11: CLOSE$_{FILE}$();
12: **end while**
13: **end for**

processed graph are pushed to the end of the *Node_List* for additional processing. The details will be explained directly during the discussion of Function 5.11. All elements of the *Node_List* are regarded in the $while-loop$ of line 7. Within the $while-loop$ line 8 opens a file for the first element of the *Node_List*. In line 9 the function $SYNTHESIZE^{\mathcal{G}^{\mathcal{TS}}}_{HDL}(\cdot,\cdot)$ is called, which synthesizes the hardware description model of the processed node. Line 10 removes the processed node from *Node_List* and line 11 finally closes the file. Either the algorithm will continue with the $while-loop$ of line 7 or the $for-all-loop$ of line 1 or the algorithm will

completely terminate.

The function $SYNTHESIZE^{\mathcal{GTS}}_{HDL}(\cdot,\cdot)$, which is called in line 4 and line 9 of Algorithm 5.10 is illustrated in Function 5.11 and runs as follows: Two parameters are required to execute the function $SYNTHESIZE^{\mathcal{GTS}}_{HDL}(\cdot,\cdot)$. Firstly, the *Node_List*, which represents the common data-structure of Algorithm 5.10 and Function 5.11 and thus the connection between them. Secondly, the actual graph $\mathbb{G}^{\cdot} \in \mathcal{GTS}$, which is currently processed. This function is divided into two parts by the outermost $if-else$ case. The if-branch runs from line 1 - line 11 and exclusively processes the root-node of each $\mathbb{G}^{\cdot} \in \mathcal{GTS}$, whereas the *else*-branch runs from line 12 - line 25 and processes all other nodes of the graphs $\mathbb{G}^{\cdot} \in \mathcal{GTS}$. The if-branch synthesizes a specific wrapper for the top-components of $\mathbb{G}^{\cdot} \in \mathcal{GTS}$ to address special issues, which will be explained in the following.

Function 5.11 SYNTHESIZE$^{\mathcal{GTS}}_{HDL}(\cdot,\cdot)$

Require: Node_List and actual $\mathbb{G}^{\cdot} \in \mathcal{GTS}$ from Algorithm 5.10

```
 1: if FRONT(Node_List) == (Root_Node of G·) then
 2:   OPEN_FILE(Wrapper);
 3:   WRITE_FILE(STD_Header);
 4:   WRITE_FILE(Constraints);
 5:   while Port_List(FRONT(Node_List)) ≠ ∅ do
 6:     WRITE_FILE(Port_List[Definition]);
 7:   end while
 8:   WRITE_FILE(Wire_Definition);
 9:   WRITE_FILE(WRAPPER(FRONT(Node_List)));
10:   CLOSE_FILE(Wrapper);
11:   CONTINUE;
12: else
13:   WRITE_FILE(STD_Header);
14:   WRITE_FILE(Constraints);
15:   while Port_List(FRONT(Node_List)) ≠ ∅ do
16:     WRITE_FILE(Port_List[Definition]);
17:   end while
18:   WRITE_FILE(Wire_Definition);
19:   for all Child_Node of FRONT(Node_List) do
20:     WRITE_FILE(COMPONENT(Child_Node));
21:     if Child_Node_{x,y} ∉ Node_List_{x,y} then
22:       PUSH_BACK_NL(Child_Node);
23:     end if
24:   end for
25: end if
```

Line 1 checks whether the first node of *Node_List* is a root-node of $\mathbb{G}^{\cdot}$. If evaluated as true, line 2 opens an additional file, which will be written until it closes in line 10. In line 3 a standard HDL-header becomes synthesized, which includes standard IEEE library definitions and also user specific definitions. The content of this header is partially predefined by the design-framework and can be extended by

user-definitions included in the MRF-specifications (cf. Figure 5.1). Line 4 adds a section of physical constraints to the file, which will be used in later processing-steps of the VLSI implementation flow. Within the $while-loop$ of lines 5 - 7 the specific $Port\_List$ data-structure of the top node is scanned and the corresponding port-definitions become synthesized and finally written to the file. Each port-definition is composed of a port-name, which has to be unique for that component, a specification of the signal transmission direction, the signal type of the port and eventually the bit-length of the port, if not already implicitly defined by the signal type. Furthermore, the complete port-definition part has to be surrounded by marks, which define the beginning as well as the end of the port-definitions. The details are language dependent and vary depending on VHDL or Verilog is generated. We again remark, that the design-framework currently synthesizes VHDL.

Line 8 synthesizes different wire definitions, which are required to connect ports and parts among each other. The wire definitions differ in detail with regard to this part is synthesized for the graph $\mathbb{G}^{GT}$ or the graph $\mathbb{G}^{MH}$. Essentially for the graph $\mathbb{G}^{GT}$ wires with a predefined logic-value are required, which fix the input ports of the neighborhood system located at the boarder of the MRF grid, where no more neighboring processing elements are available. Additionally, wires are required for $\mathbb{G}^{GT}$, which collect the outputs of the neighborhood system at the boarder of the MRF grid and assign them to a single dummy output-port. For the graph $\mathbb{G}^{MH}$ wire definitions are required, which connect sub-components to a particular memory component. The following line 9 synthesizes the encapsulation of the component, which represents the central part of the wrapper component. In detail the $WRAPPER(\cdot)$ function in line 9 instantiates the component, which is represented by the root-node itself and connects either the component ports with the port-definitions or with the wire definitions, depending on which graph $\mathbb{G}^{\cdot}$ currently becomes processed. Obviously, the $WRAPPER(\cdot)$ function has to establish functional correct connections among the particular ports and wires. The technical details are hidden by the $WRAPPER(\cdot)$ function in order to tighten the presentation. Finally line 10 closes the file, where the synthesized wrapper has been written to. The following line 11 forces the function to continue with the *else*-branch. This procedure is necessary, since a component description, which has been instantiated within the wrapper, also has to be synthesized for the root-node and this is done within the *else*-branch.

Before the description is continued with the *else*-branch of Function 5.11, we justify the synthesis of the wrapper components. The synthesis of the wrapper components for the graphs $\mathbb{G}^{\cdot} \in \mathcal{G}^{\mathcal{TS}}$ has been included in the design-framework due to requirements of the back-end place & route implementation process. Without the wrapper components it becomes possible that the design flow will fail or becomes difficult to manage. When regarding the graph $\mathbb{G}^{GT}$ and its root-node, which represents the complete MRF grid and its neighborhood wiring, it becomes obvious that the neighborhood system on the MRF grid boarder is incomplete. Consequently, there are missing input-sources and output-destinations for particular neighborhood-connections of sites on or near the grid-boarder. Together with the fact that this component is the top-, respectively the main-component of the MRF system, this leads to the situation that all inputs and outputs of this component are normally treated as physical chip I/Os. Hence, the technology mapper will generate for each of these I/Os special I/O buffers, which are linked with the pads of the chip-package

pins. But the physical chip-package pins are limited and furthermore the pin assignment is one of the very first back-end implementation steps cause the design process to fail before placing and routing takes place. If one decides to prevent the technology mapper from generating I/O pads, the design will contain dangling nets, which cause the following design steps to abort, because electrical consistency is no longer guaranteed. For the graph $\mathbb{G}^{GM}$ the wrapper encapsulates the top stage of the up- and down-path within the memory hierarchy as well as the instantiations of the next level in one component. This simplifies the floor-planning procedure because all components can be referenced to by the wrapper component. Thus the wrapper components are mandatorily required to resolve different practical issues regarding design implementation.

The *else*-branch runs from lines 12 - 25 and works up to line 19 similar with to if-branch, i.e. the standard header is written (line 13), the constraints are generated (line 14), the port definitions are synthesized (lines 15 - 17) and finally line 18 writes the wire definitions. The following lines synthesize the sub-components and additionally decide, which sub-component has to be further processed in order to iteratively traverse the graph. Line 19 starts the $for\ all-loop$, which picks all children of the first node stored in the $Node\_List$ data-structure. For each of the children Function 5.11 writes a component description to the already opened file, i.e. the component represented by the first node in $Node\_List$ is composed out of sub-components, which are represented by their children nodes. The command $WRITE_{FILE}(COMPONENT(Child\_Node))$ also hides several technical details and processing steps that have to be conducted in order to synthesize the corresponding components and their wirings. These processing details have been omitted to stress the essential algorithmic steps. Line 21 checks whether the currently regarded type of child node, which is defined by its x, y size, is already stored in the $Node\_List$. If this is not the case, line 22 pushes this node type onto the list. When the $for\ all-loop$ of line 19 finishes its processing, the function returns to the calling line of Algorithm 5.10 and the processing of the nodes, stored in the data structure $Node\_List$, continues.

The lines 20, 21 and 22 of Function 5.11 are central with respect to several aspects. Firstly, these lines ensure that only those sub-graphs are processed, whose parent-nodes are different with respect to their x, y sizes. Consequently, not the complete graph is traversed but only a single part of the graph, which improves the processing-performance of Algorithm 5.10. This algorithmic-feature becomes significant, when the processing of the modeling-graphs $\mathbb{G}^{\cdot} \in \mathcal{G}^{\mathcal{TS}}$ of MRF systems with grid sizes that are larger than 100 sites in x as well as y direction takes place. Furthermore this approach results in an effective and optimal synthesis procedure of hardware descriptions for components with respect to the generated number of different components and files. Only components with different x, y sizes become synthesized and equal components are just instantiated. This limits the data, which has to be passed over to the technology synthesis and mapping tools, and likewise reduces the overall technology synthesis and mapping effort, since instantiations are merely renaming and copying tasks (cf. particular discussions of Chapter 3). The run-time improvements of the 3rd party technology synthesis and mapping tools achieved by this kind of hardware description synthesis are tremendous and result in a substantial reduction of the processing time for for even small MRF grid sizes.

5.4.2 Compiling Processing Parts

The graph set $\mathcal{G}^{PC}$, which represents the processing functionalities within the canonical design representation, consists of the graphs $\mathbb{G}_{PC}^{EF}$, $\mathbb{G}_{PC}^{OPT}$ and $\mathbb{G}_{PC}^{PAR}$ (cf. Definition 5.5). All these graphs of $\mathcal{G}^{PC}$ will be systematically processed by Algorithm 5.12 and the Function 5.13 in order to synthesize a hardware description for the processing functionalities of the corresponding MRF architecture. Again the overall procedure of compiling the HDL-code for the graphs of $\mathcal{G}^{\mathcal{PC}}$ has been split up into two parts (Algorithm 5.12 and Function 5.13) in order to separate the two principally different functionalities and also to improve the overall presentation. The Algorithm 5.12 selects each graph of $\mathcal{G}^{PC}$ for processing, opens and closes files and calls the Function 5.13, which synthesizes the hardware descriptions by means of the information of the modeling graphs.

Algorithm 5.12 Compiling HDL-code from $\mathcal{G}^{PC}$

Require: $\mathcal{G}^{PC}$, Node_List, Schedule_Type
1: **for all** ($\mathbb{G}^{\cdot} \in \mathcal{G}^{\mathcal{PC}}$) **do**
2: OPEN$_{FILE}$(FRONT(Node_List));
3: SYNTHESIZE$_{HDL}^{\mathcal{G}^{\mathcal{PC}}}$($\mathbb{G}^{\cdot}$, Schedule_Type); // Function 5.13
4: CLOSE$_{FILE}$();
5: **end for**

Algorithm Description - The complete HDL-code compilation procedure for the parts representing processing-functionality of the massively parallel architecture runs as follows: Algorithm 5.12 requires the graph set $\mathcal{G}^{PC}$, a dynamic list data-structure and the type of the scheduling method. Within the compilation process the scheduling type determines which kind of time-partitioning has to be used during the generation of hardware descriptions for processing functionalities. As the name $Node\_List$ of the data-structure already suggest, this dynamic list stores the nodes of the currently processed graph $\mathbb{G}^{\cdot} \in \mathcal{G}^{\mathcal{PC}}$. Line 1 is the outermost $for\ all - loop$, which selects each graph of $\mathcal{G}^{\mathcal{PC}}$ for further processing steps. The following line 2 opens the specific file into which the synthesized hardware description is written. The actual synthesis of the hardware description is realized by Function 5.13, which in line 3 is called with its required parameters. Finally, line 4 closes the file into which Function 5.13 has written the synthesized hardware description. Either the algorithm continues with another graph $\mathbb{G}^{\cdot} \in \mathcal{G}^{\mathcal{PC}}$, or if all graphs have been processed, the algorithm terminates normally.

The function $SYNTHESIZE_{HDL}^{\mathcal{G}^{\mathcal{PC}}}(\cdot,\cdot)$, which is called in line 3 of Algorithm 5.12, is illustrated in Function 5.13 and runs as follows: $SYNTHESIZE_{HDL}^{\mathcal{G}^{\mathcal{PC}}}(\cdot,\cdot)$ requires two parameters to work correctly. At first the actual graph $\mathbb{G}^{\cdot} \in \mathcal{G}^{\mathcal{PC}}$ is selected, which has to be processed and secondly the information about the scheduling type, whose results will be used during the synthesis of the hardware description. Furthermore two dynamic list data-structures are needed, which store the nodes of sub-graphs of $\mathbb{G}^{\cdot}$. At the beginning (line 1) a standard header is written to the file, which references IEEE libraries and sub-packages with different basic functionalities. Line 2 eventually adds a section of constraints to the file to allow later steps of

the VLSI implementation flow to be evaluated and uses them. The $while - loop$ of lines 3 - 5 scans the root-node of the actual graph and synthesized the corresponding port-definitions, which are written to the file in the following. Each particular port-definition is composed of several parts as it has already been explained in Section 5.4.1. Furthermore the port-definitions are surrounded by special marks, which define the beginning as well as the end of this part within the hardware description.

Function 5.13 SYNTHESIZE$^{\mathcal{GPC}}_{HDL}(\cdot,\cdot)$

Require: Node_List1, Node_List2, Schedule_Type and actual $\mathbb{G}^{\cdot} \in \mathcal{G}^{\mathcal{PC}}$ from Algorithm 5.12

```
 1: WRITE_FILE(STD_Header);
 2: WRITE_FILE(Constraints);
 3: while Port_List( G·) ≠ ∅ do
 4:    WRITE_FILE(Port_Definition);
 5: end while
 6: WRITE_FILE(Signal_Definition);
 7: for all RESET_Nodes ∈ G· do
 8:    for all CHILDREN do
 9:       Node_List1 = DFS(CHILDREN);
10:       for all Nodes ∈ Node_List1 do
11:          if STATE(Node(Schedule_Type,Index)) ∉ State_List then
12:             PUSH_BACK_SL(STATE(Node, Schedule_Type));
13:          end if
14:          WRITE_FILE(State_List);
15:       end for
16:    end for
17: end for
18: WRITE_FILE(Reset_Sequence);
19: Schedule_Index = any RESET_Node(Schedule_Type,Index)+1
20: while Node(Schedule_Type,Schedule_Index) ∈ G· do
21:    WRITE_FILE(STATE(Schedule_Type,Schedule_Index))
22:    for all Nodes ∈ G· do
23:       if Node(Schedule_Type,Index) = Schedule_Index then
24:          WRITE_FILE(Node(Op));
25:       end if
26:    end for
27:    Schedule_Index++;
28: end while
```

Line 6 synthesizes wire-definitions, which are eventually used to connect ports and different parts of boolean-logic within the component. Within lines 7 - 17 the discrete states operations are defined with respect to the used scheduling type and index, i.e. the different discrete states to which the operations will be assigned to later on, become synthesized in these lines. As the reset nodes and their hardware description synthesis are treated differently, only the sub-graphs of the reset nodes

are relevant for the synthesis of the states. Hence line 7 starts a $for\ all-loop$ with respect to the reset-nodes, and the following line 8 starts an additional $for\ all-loop$, however in this case with respect to the children of the reset nodes.

Line 9 executes a depth-first-search with these children nodes and stores the visited nodes in $Node\_List1$. All nodes within $Node\_List1$ become examined with respect to their Index of the $Schedule\_Type$ and the resulting state and afterward compared with all elements of $State\_List$. If the state is not yet stored within this list, line 12 pushes the new state to the end of the $State\_List$. This comparison-procedure is mandatorily required since nodes that have been scheduled to the same processing-step, as they can be executed in parallel, will have identical index values and thus also identical state descriptions. In order to prevent multiple identical states, the comparison-procedure has to be conducted. Finally line 14 writes the state definitions to the file. After that line 18 synthesizes the hardware description for the reset-sequence and writes it to the file. When preparing for the assignment of the operations to their states, line 19 extracts the index value with respect to the $Schedule\_Type$ of an arbitrary reset-node and adds 1 to receive the next state after the reset-node state. The $while-loop$ of line 20 checks whether a node with the actual Schedule_Index lies within the processed graph $\mathbb{G}^{\cdot}$. If this is not the case, the graph $\mathbb{G}^{\cdot}$ has to be processed completely and Function 5.13 returns to its calling line. Otherwise line 21 writes a STATE marker, which matches with the defined and generated elements of $State\_List$. This state is assigned to all operations of nodes with the same $Schedule\_Index$. Therefore line 22 starts a $for\ all-loop$, which processes all nodes of $\mathbb{G}^{\cdot}$. If the node's Index, with respect to the $Schedule\_Index$, matches with the $Schedule\_Index$, the operations become written to this state. Line 27 increases the $Schedule\_Index$ by one each time until the $forall-loop$ of line 22 has finished.

5.4.3 Compiling Control Parts

All graphs, which represent control functionalities within the canonical design representation, are formally pooled in the graph set $\mathcal{G}^{CT}$. This particularly graph set consists of the graphs $\mathbb{G}_{CT}^{System}$, $\mathbb{G}_{CT}^{GMem}$, $\mathbb{G}_{CT}^{PortMem}$, $\mathbb{G}_{CT}^{EF}$, $\mathbb{G}_{CT}^{OPT}$ and $\mathbb{G}_{CT}^{PAR}$ (cf. Definition 5.10). The particular graphs of $\mathcal{G}^{CT}$ are processed by Algorithm 5.14 and the Function 5.15, which is called within the main algorithm. For this compilation procedure of HDL-code we have also split up the algorithm into two part (Algorithm 5.14 and Function 5.15) to structure both the presentation and the corresponding discussion. The Algorithm 5.14 traverses each graph of $\mathcal{G}^{CT}$ in order to extract the graph nodes for further processing, opens and closes files and calls the Function 5.15, which synthesizes the hardware description for the different control units of the massively parallel MRF hardware architecture.

Algorithm Description - The HDL-code compilation procedure for the control-functionalities within the massively parallel architecture runs as follows: Algorithm 5.14 requires only the graph set $\mathcal{G}^{CT}$ as input data. Line 1 is the outermost $for\ all-loop$, which selects each graph of $\mathcal{G}^{\mathcal{CT}}$ in order to systematically analyze and process these graph structure to allow the HDL-code describing the corresponding control-functionality to be synthesized. The following line 2 opens the specific file

Algorithm 5.14 Compiling HDL-code from $\mathcal{G}^{CT}$

Require: $\mathcal{G}^{CT}$

```
1: for all (𝔾· ∈ 𝒢^𝒞𝒯) do
2:    OPEN_FILE(Root_Node(𝔾·));
3:    SYNTHESIZE^𝒢^𝒞𝒯_HDL(𝔾·); // Function 5.15
4:    CLOSE_FILE();
5: end for
```

to which the synthesized hardware description is written. The necessary information for compiling an unique file name becomes extracted from the root-node of the graph $\mathbb{G}^{\cdot}$. The actual synthesis of the hardware description is realized by Function 5.13, which in line 3 is called with its required parameters. Finally line 4 closes the file to which Function 5.13 has written the synthesized hardware description. The algorithm either continues with another graph $\mathbb{G}^{\cdot} \in \mathcal{G}^{\mathcal{PC}}$ or, if all graphs of $\mathcal{G}^{\mathcal{PC}}$ have been processed, it terminates.

Function 5.15 SYNTHESIZE$^{\mathcal{G}^{\mathcal{CT}}}_{HDL}(\cdot)$

Require: Node_List and actual $\mathbb{G}^{\cdot} \in \mathcal{G}^{\mathcal{PC}}$ from Algorithm 5.14

```
1: if 𝔾·_𝕋 ≠ 𝔾^EF_CT & 𝔾^OPT_CT & 𝔾^PAR_CT then
2:    WRITE_FILE(STD_Header);
3:    WRITE_FILE(Constraints);
4:    while Port_List( 𝔾· ) ≠ ∅ do
5:       WRITE_FILE(Port_Definition);
6:    end while
7: end if
8: WRITE_FILE(Signal_Definition);
9: for all IDLE_Node ∈ 𝔾· do
10:   WRITE_FILE(FSM_STATES(BREADTH_FIRST_SEARCH(IDLE_Node));
11: end for
12: WRITE_FILE(Reset_Sequence);
13: for all IDLE_Node ∈ 𝔾· do
14:   Node_List=BREADTH_FIRST_SEARCH(IDLE_Node);
15:   for all Nodes ∈ Node_List do
16:      WRITE_FILE(FSM_State(Node_List));
17:      WRITE_FILE(Signal_Assignment(Node_List));
18:      WRITE_FILE(State_Transition(Node_List));
19:   end for
20: end for
```

The function SYNTHESIZE$^{\mathcal{G}^{\mathcal{CT}}}_{HDL}(\cdot)$, which is called in line 3 of Algorithm 5.14 for all graphs $\mathbb{G}^{\cdot} \in \mathcal{G}^{\mathcal{CT}}$, is outlined in Function 5.15. SYNTHESIZE$^{\mathcal{G}^{\mathcal{CT}}}_{HDL}(\cdot)$ takes a graph $\mathbb{G}^{\cdot} \in \mathcal{G}^{\mathcal{CT}}$ as input parameter and additionally requires the dynamic list data-structure *Node_List* to work properly. Within the $if - case$ of lines 1 - 7 only control-functionality graphs are processed, which did not possess a corresponding processing-functionality graph, and hence did not form a tuple $(\mathbb{G}^{\cdot}_{PC}, \mathbb{G}^{\cdot}_{CT})$

(cf. Corollary 5.3). These graphs require a hardware description, which contains a standard header (line 2), constraints (line 3) and above all port-definitions. In contrast to these graphs those graphs representing control-functionality, which form a $(\mathbb{G}_{PC}^{\cdot}, \mathbb{G}_{CT}^{\cdot})$, did not require these hardware description parts, because these specific parts have already been synthesized during the HDL-code generation of the processing graphs. Line 8 generates various signal-definitions. The $for\ all - loop$ of lines 9 - 11 searches all $IDLE\_Nodes$ of $\mathbb{G}^{\cdot}$ and for the sub-graphs - identified by breath-first-search (line 10) - synthesizes the state definitions of the control unit. In line 12 the reset sequence, defined by a sub-graph of $\mathbb{G}^{\cdot}$, is synthesized and written to the file. Finally lines 13 - 20 synthesize the complete states of the control unit, which are composed of the state-label, the affiliated signal-assignments and the rule of the state-transition. Line 13 extracts all $IDLE\_Node$ of $\mathbb{G}^{\cdot}$ to allow line 14 to generate a list $Node\_List$ by means of breadth-first-search, which contains all nodes of the corresponding $IDLE\_Node$ sub-graph. The following $for\ all - loop$ synthesizes for each of the nodes within $Node\_List$ the control-unit state (line 16), the signal assignments (line 17) and the state transition (line 18).

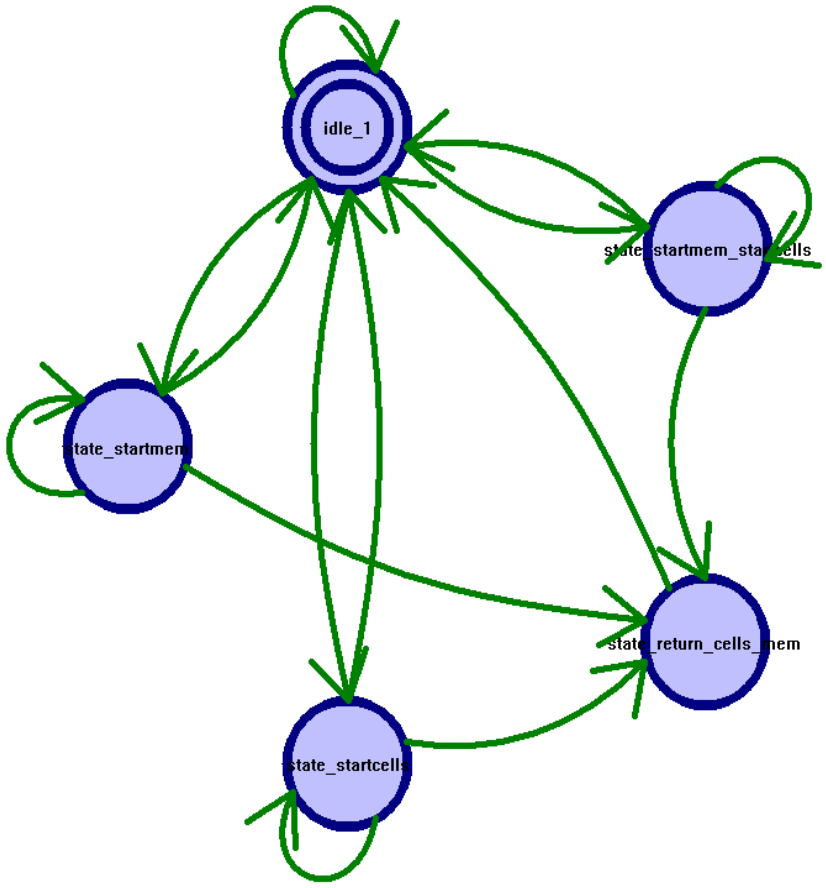


Figure 5.2: Modeling of system's control unit. State transition scheme extracted from generated HDL code by third party synthesis engine.

5.5 Results - Design Framework

The previously introduced novel VLSI design methodology for the massively parallel architectures of the defined MRF model class has software-technically been put into practice by the Design-Framework. This framework has been intensively tested and steadily improved in several software revisions. Two stages have been applied to systematically test the framework, its output and to disclose the framework's current limitations. At first numerous artificial test cases have been used to check the complete framework and all its particular modules. During the second stage we have used the two exemplary MRF image processing models (Section 3.7.1, 3.7.2) to further check and prove the capabilities of the framework. In the following discussion we omit the artificial test cases and the corresponding results and only present selected results, which are convincing to underpin the capabilities of the proposed design framework. Additional results have been moved to Appendix B.

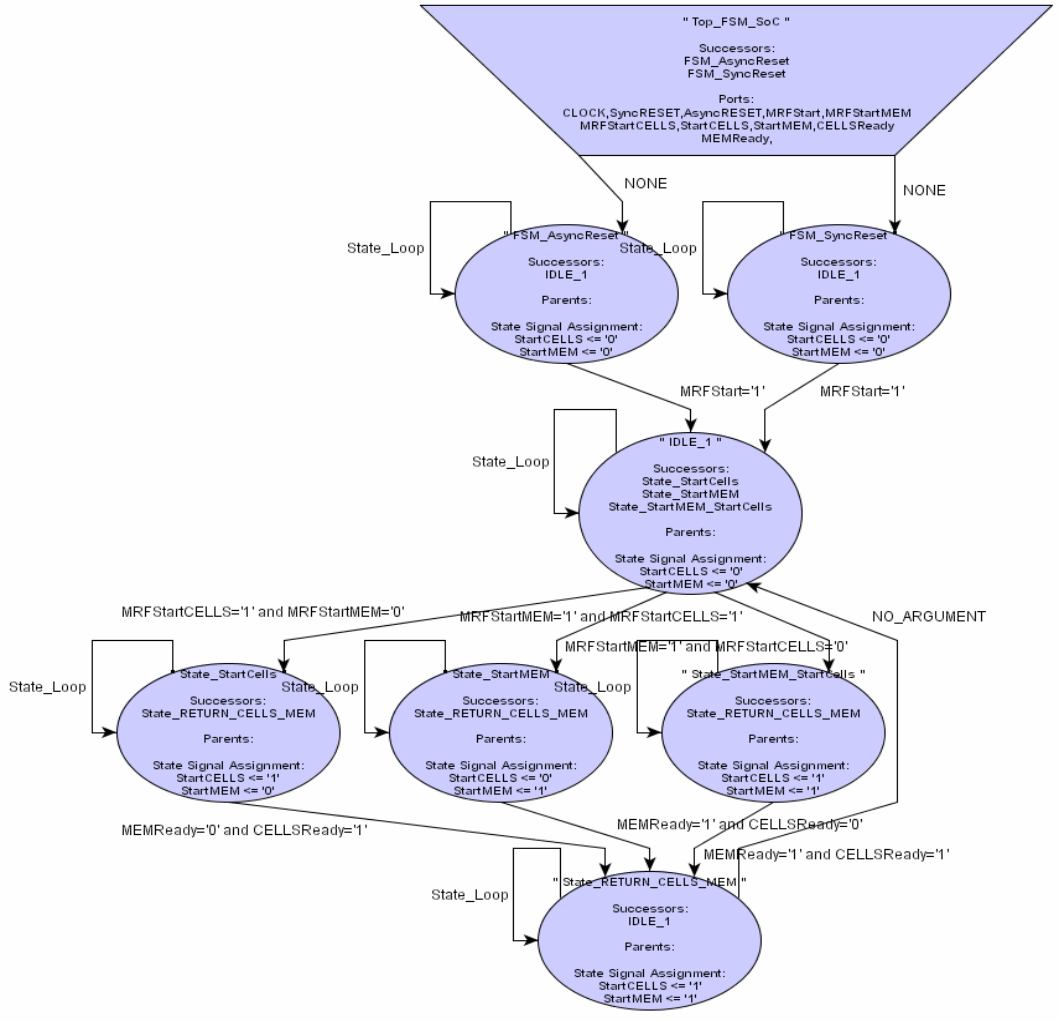


Figure 5.3: Modeling of MRF system's control unit. Detailed graph representation $\mathbb{G}_{CT}^{System}$ within design framework.

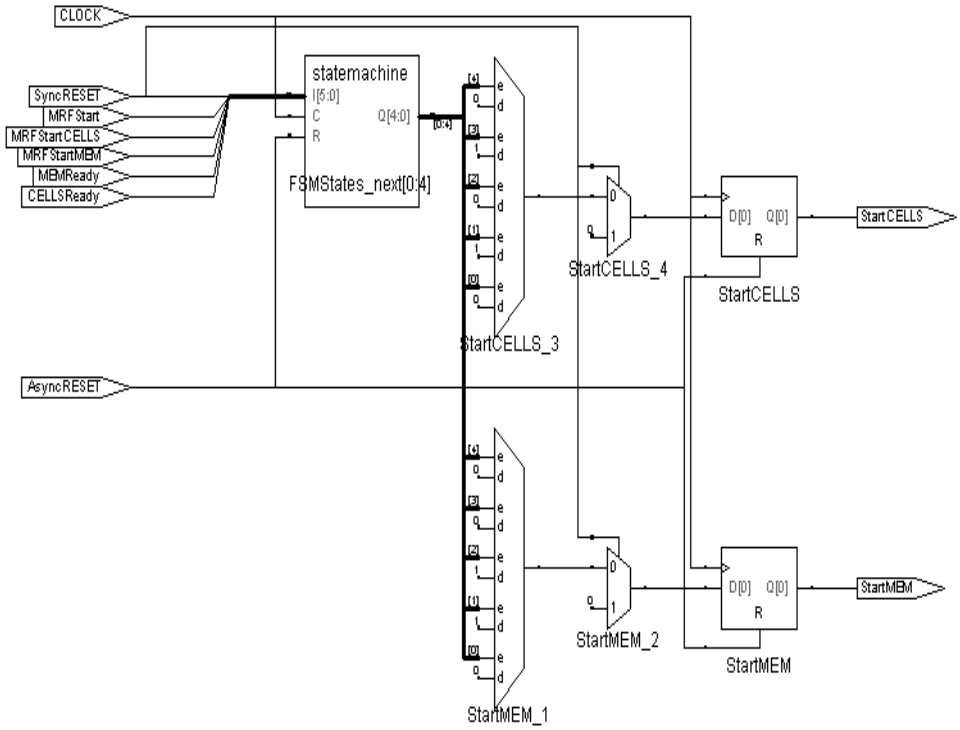


Figure 5.4: VLSI Implementation of system's control unit. RTL schematic synthesized from generated HDL code.

The control unit of a particular massively parallel architecture represents an ideal part of the architecture to illustrate and discuss aspects of the design framework. Within the canonical design representation the system control unit is modeled by the graph $\mathbb{G}_{CT}^{System}$. Figure 5.3 depicts the original graph structure of $\mathbb{G}_{CT}^{System}$ as it is expanded by the framework and stored in the canonical design representation. The entire data, each node holds within the canonical design representation is not completely shown in the nodes of Figure 5.3. The graph $\mathbb{G}_{CT}^{System}$ possesses the root node "TOP_FSM_SoC" and its child nodes are the "SyncReset"- and "ASyncReset" node in accordance with Definition 5.11 and Algorithm 5.6 (Line 1-7). Both reset nodes assign the logic value 0 to the output signals "StartCELLS" and "StartMem". The child of the reset nodes is the "IDLE_1" node and this node can be reach from the reset nodes when the condition "MRFStart='1'" is true as is indicated by the marked edges connecting these nodes. The "IDLE_1" node is connected to three nodes, which represent the states to start the processing of the cells, the memory hierarchy or alternative to start the processing of the cells and the memory hierarchy concurrently. Each of these nodes can be reached from the "IDLE_1" node if the condition of the corresponding marked edge is true. Furthermore each of these nodes assign different values to the output signals "StartCELLS" and "StartMem" (cf. Figure 5.3; node data). As soon as the cells, the memory hierarchy or both indicate

the end of their operation (conditions of marked edges evaluate true) it is possible to reach the last node at the bottom of Figure 5.3. This return node is automatically left without evaluating any conditions and goes back to the "IDLE_1" node. Starting from the "IDLE_1" node the process can execute again. The described functionality is illustrated by Figures 5.6, 5.7 and 5.8, which show particular simulation sequences of the generated VHDL code of the system's control unit

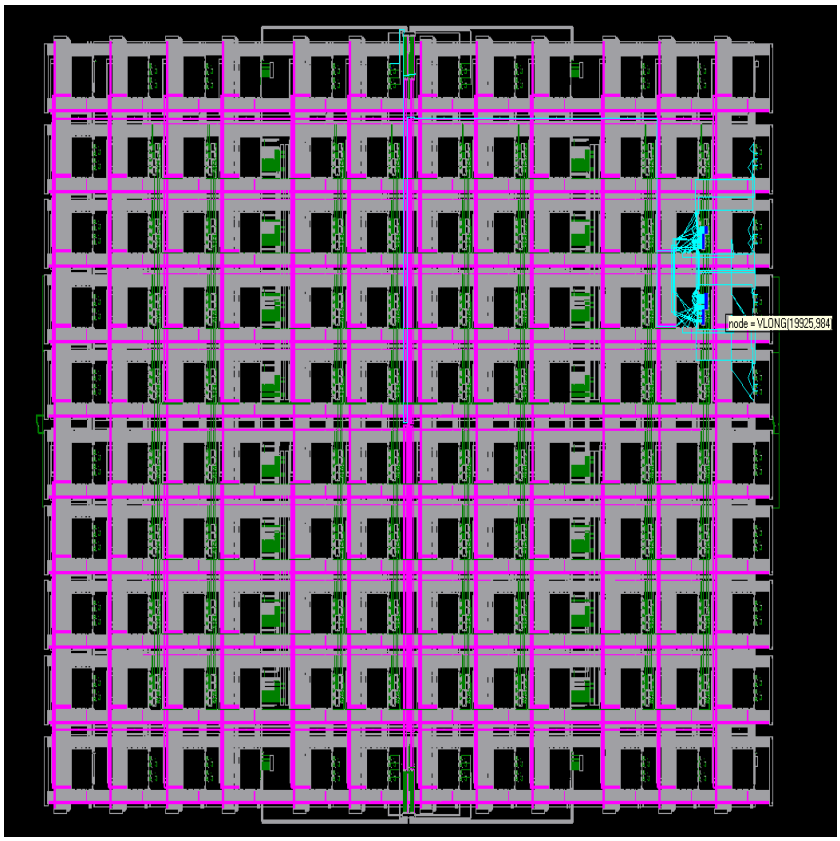
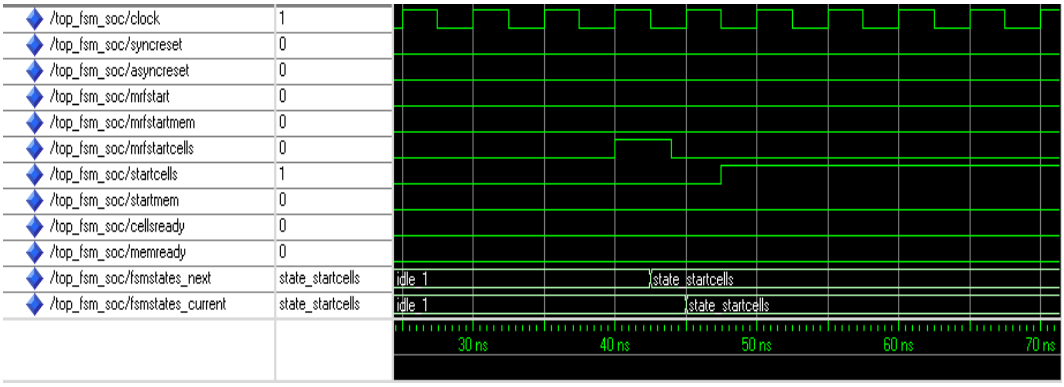


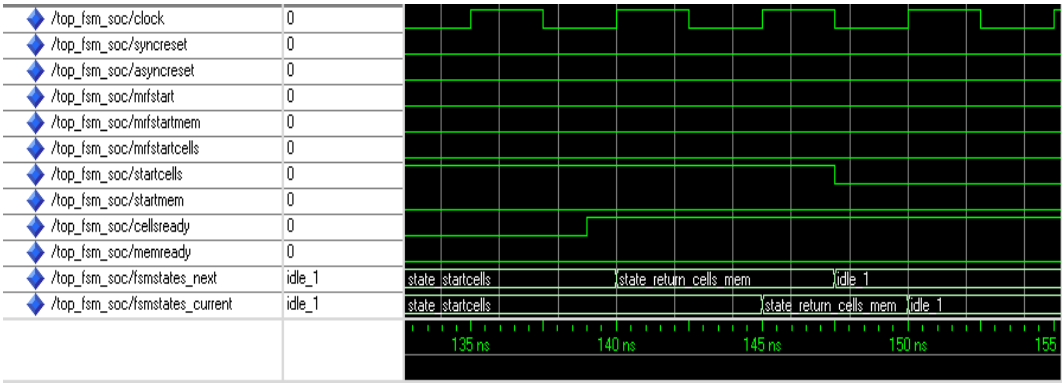
Figure 5.5: FPGA implementation (Xilinx Virtex 2 family) of system's control unit.

The graph representation of $\mathbb{G}_{CT}^{System}$, illustrated in Figure 5.3 undoubtedly demonstrates the technology independence and hardware abstract modeling capabilities of the design framework for this part of the architecture.

Algorithm 5.14 and Function 5.15 process this graph to synthesize a hardware description of this control unit. Figure 5.2 shows the transition scheme of the finite state machine, which was extracted by a commercial third party hardware-development tool from the generated HDL code of the design framework. First of all, this state transition scheme underpins that the generated HDL code was syntactically and logically correct because it has been processed by the third party hardware-development tool. The functional correctness of the HDL code generation process can be systematically verified by means of the two Figures 5.3 and 5.2. First of all, it is remarkable that the number of vertexes (top node of $\mathbb{G}_{CT}^{System}$ not considered) of Figure 5.3 and Figure 5.2 are not identical. This fact is caused by



(a)



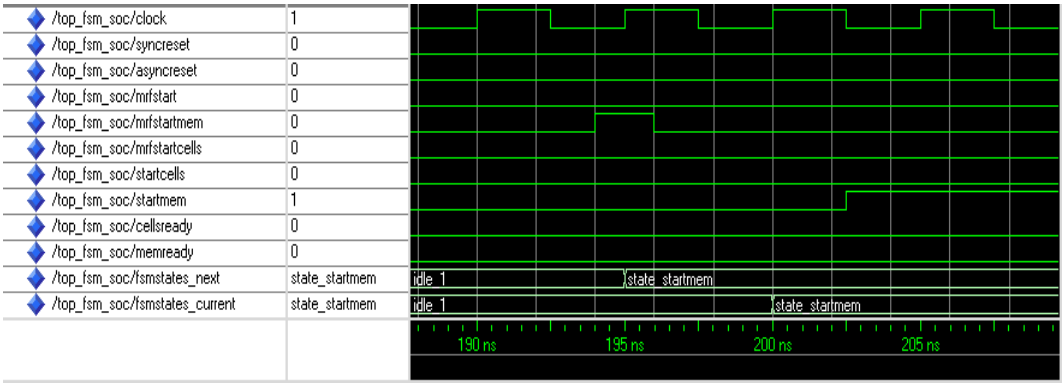
(b)

Figure 5.6: Wave-Diagram of HDL-Code Simulations. (a) Impulse sequence for starting the cell-processing of the complete site-grid. (b) Impulse sequence for stopping the cell-processing (complete grid). Signals (Top-Down): Clock, Synchronous Reset, Asynchronous Reset, MRF-Start, MRF-StartMem, MRF-StartCells, Start-Cells (internal signal, Start-Mem (internal signal), Cells-Ready (internal signal), Mem-Ready (internal signal), FSM State next, FSM State current.

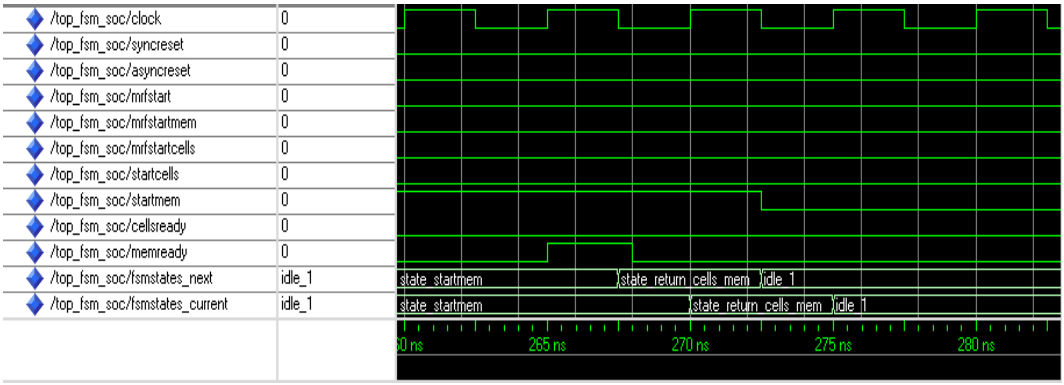
the special role of the reset nodes. These nodes will not translate to a particular state within the transition scheme depicted in Figure 5.2. Rather the reset nodes of $\mathbb{G}_{CT}^{System}$ resolve to edges of the states "state_startmem", "state_startcell" and "state_startmem_startcell" back to the "idle_1" state in the transition scheme (see Figure 5.2). These back-edges of the state transition scheme are sufficient to model the reset behavior, because a synchronous- as well as asynchronous reset is no more than a state transition to the initial state.

The detailed discussion on the reset nodes of $\mathbb{G}_{CT}^{System}$ and their resolving within the transition scheme clearly demonstrates the capabilities of the framework to generated qualified HDL code from control functionality representing graphs; exemplarily verified by means of the system's control unit.

Our statement that the design framework generates qualified HDL code from control graphs is further substantiated by analyzing the RTL schematic of the system's control unit, which is shown in Figure 5.4. The schematic shows a clear and



(a)

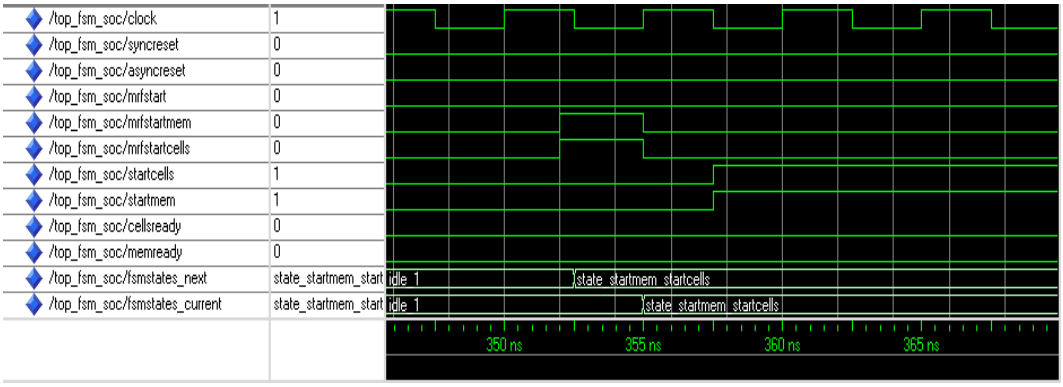


(b)

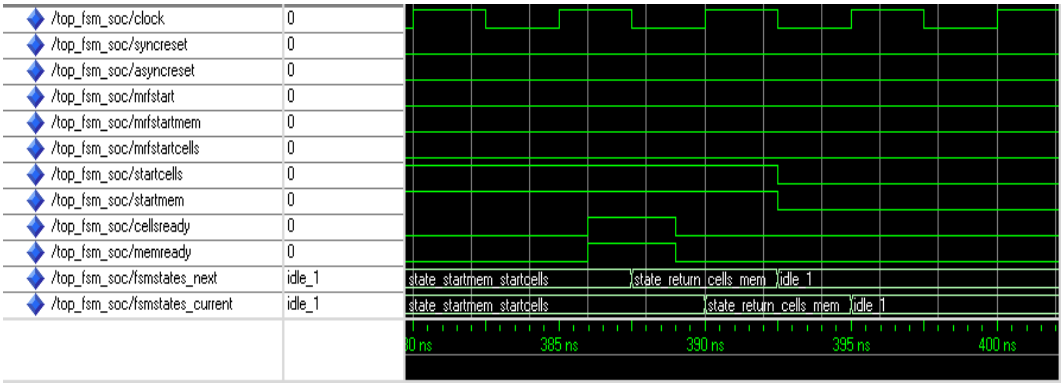
Figure 5.7: Wave-Diagram of HDL-Code Simulations. (a) Impulse sequence for starting the activity of the memory hierarchy. (b) Impulse sequence for stopping the activity of the memory hierarchy. Signals (Top-Down): Clock, Synchronous Reset, Asynchronous Reset, MRF-Start, MRF-StartMem, MRF-StartCells, Start-Cells (internal signal, Start-Mem (internal signal), Cells-Ready (internal signal), Mem-Ready (internal signal), FSM State next, FSM State current.

well separated structure of particular elements from left to right, i.e. we can identify input ports, an encapsulated state-machine, muxes, flip-flops and finally output ports. Furthermore, the schematic is free of feedback signals and this fact complies with the expected input-output signal flow for this control unit. Additionally, the two storing elements of the outputs are realized as flip-flops and not as latches, which shows the complete and unique output assignment at each state. Figure 5.5 shows the complete place&route of the system control unit in FPGA technology. Summarizing, the discussion has demonstrated that the design framework is able to generate qualified HDL-code for control units from abstract graphs.

The control unit for the port memories and the corresponding connection with the memory elements, which are associated with the cell hull, is another insightful example. Figure 5.9 shows a graph $\mathbb{G}_{CT}^{PortMem}$ from the canonical design representation, which realizes the send and receive of data in two steps, i.e. the input and



(a)



(b)

Figure 5.8: Wave-Diagram of HDL-Code Simulations. (a) Impulse sequence for starting the cell-processing of the complete site-grid and the activity of the memory hierarchy. (b) Impulse sequence for stopping the cell-processing (complete grid) and the activity of the memory hierarchy. Signals (Top-Down): Clock, Synchronous Reset, Asynchronous Reset, MRF-Start, MRF-StartMem, MRF-StartCells, Start-Cells (internal signal, Start-Mem (internal signal), Cells-Ready (internal signal), Mem-Ready (internal signal), FSM State next, FSM State current.

output data of a cell is transferred in two portions. Again this graph possesses a top node with the two children representing the reset behavior. Each of the reset nodes are connected to sub-trees, which model the send or the receive procedure, respectively. Due to the separation of the send and receive sub-trees the graph offers the option that this control unit can execute the send and receive process in parallel.

Figure 5.10 depicts the state transition schemes, which have been extracted from the generated HDL-code. The HDL-code was generated by the design framework with respect to the graph shown in Figure 5.9. As expected two distinct transition schemes have been extracted from the generated code, one scheme for the send part and another scheme for the receive part. A short simulation sequence of the generated VHDL code is shown in the wave diagrams of Figure 5.11 a-b. The reset nodes have, as already explained, been resolved to edges, which go back to the initial (idle) state (cf. Figure 5.10). Figure 5.12 a-b shows the wave diagrams of the

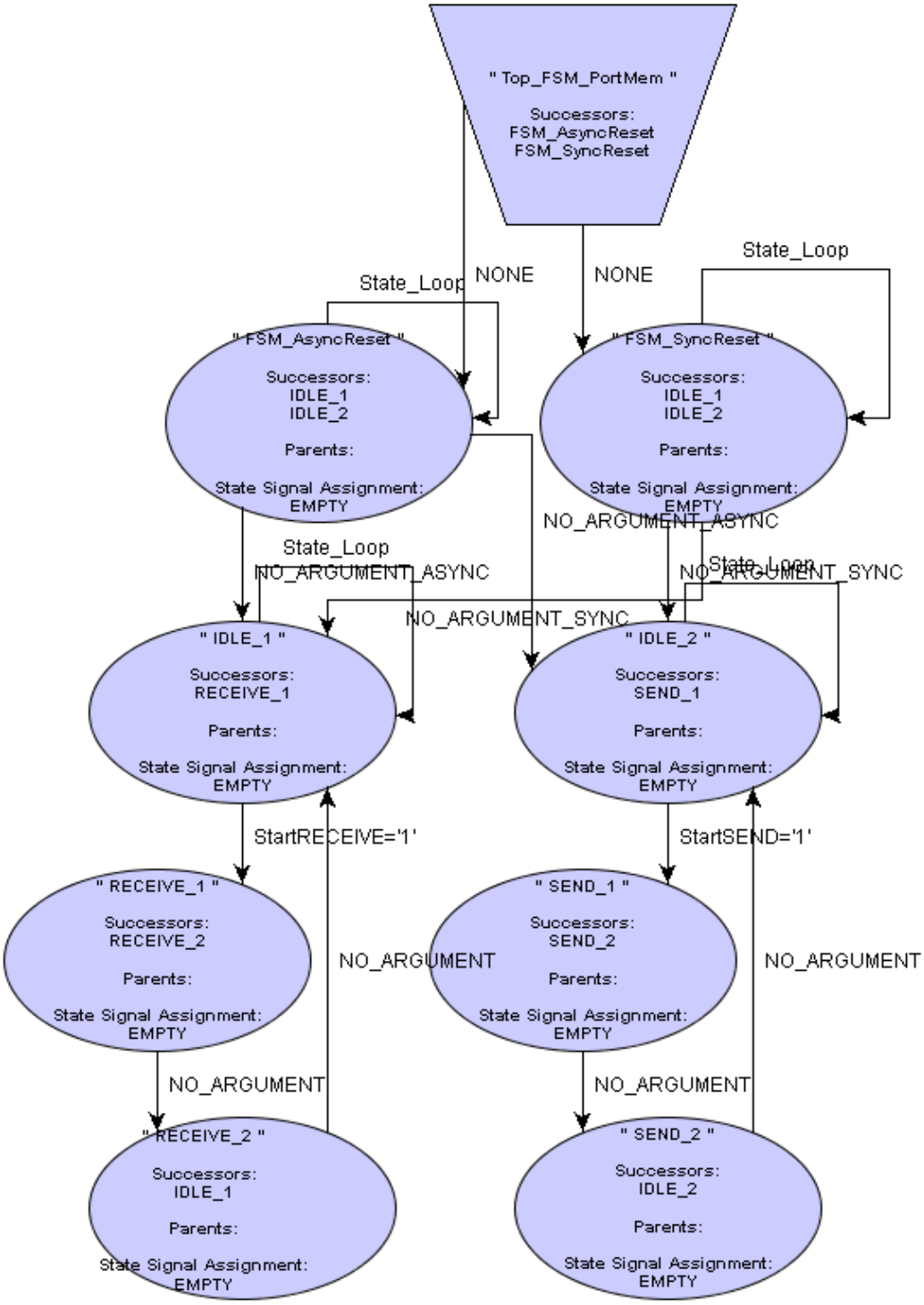
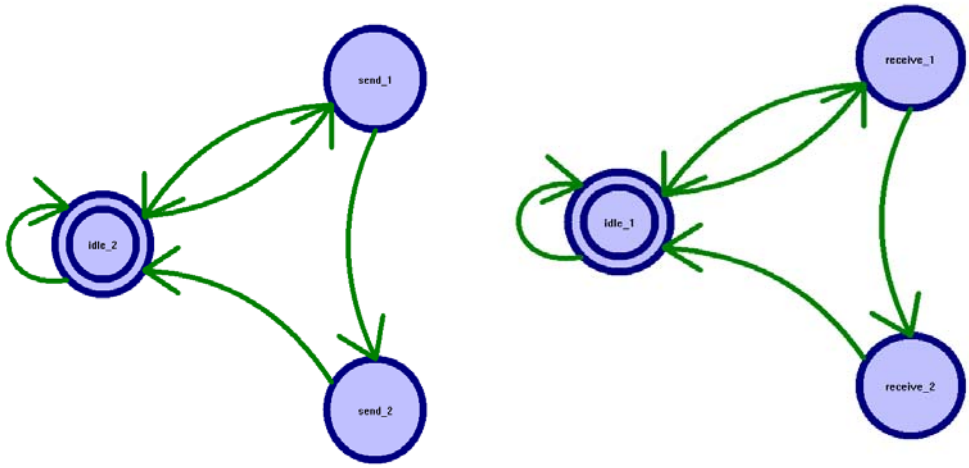


Figure 5.9: Modeling of port memory control unit. Graph representation $\mathbb{G}_{CT}^{PortMem}$ within design framework.



(a) (b)

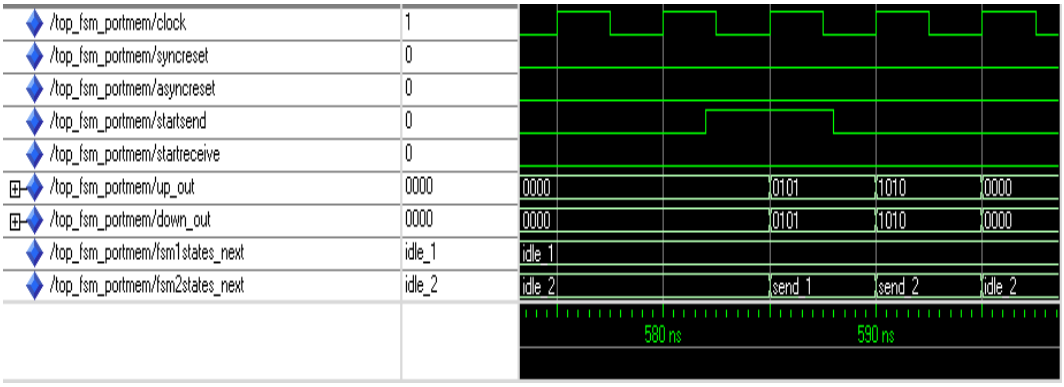
Figure 5.10: Modeling of port memory control unit. State transition schemes extracted from generated HDL code by third party synthesis engine. (a) State transition scheme for *SEND* functionality. (b) State transition scheme for *RECEIVE* functionality.

simulated VHDL code for the synchronous and asynchronous behavior, respectively.

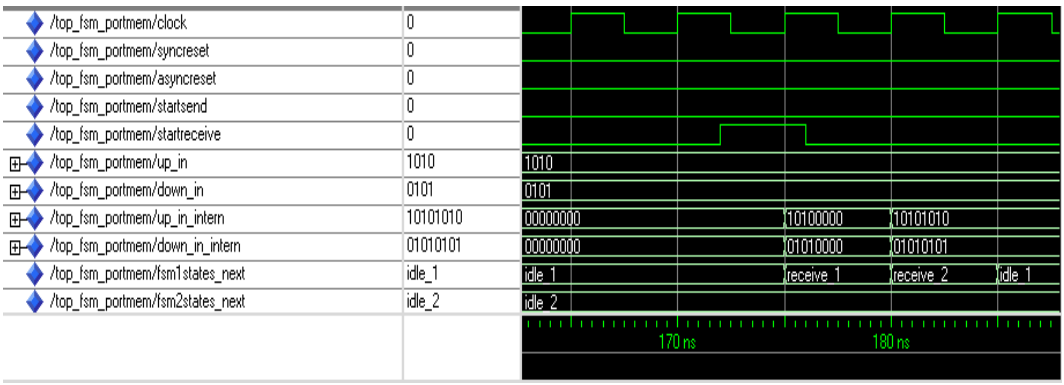
The port memory control unit is closely linked with single cells, the used neighborhood system and the port memories of the cell. Hence, the generated HDL-code for $\mathbb{G}_{CT}^{PortMem}$ can not regarded in isolation; rather it has to be discussed in conjunction with the cell component. Figure 5.13 shows a RTL schematic - synthesized from generated HDL-code, which comprises the port memory control unit, the port memory elements and the input and output ports for a $\mathcal{N}^1$ neighborhood system. The RTL schematic of Figure 5.13 illustrates the seamless integration of the port memory control unit with the corresponding port memory elements. We further identify two separated state-machines within the schematic, which represent the parallel send and receive part in accordance with graph $\mathbb{G}_{CT}^{PortMem}$. An FPGA implementation of the schematic in Figure 5.13 is shown in Figure 5.14.

The discussion on the results of the port memory control unit has again demonstrated the capabilities of the design framework to generate qualified HDL-code from an abstract graph representation. In addition to the system control unit it has been manifested that the generated HDL-code fits together with other parts of HDL-code, which was generated from different graphs.

A challenging architectural part of each massively parallel MRF architecture is the memory hierarchy, which is represented by graph $\mathbb{G}^{MH}$. Figure 5.15 shows the graph $\mathbb{G}^{MH}$ of a memory hierarchy, which is suitable for a 64×64 sized MRF. Obviously, the graph $\mathbb{G}^{MH}$ in Figure 5.15 hides any kind of detailed information. But the important structural feature of planarity as well as of a well defined hierarchy is clearly demonstrated for this type of graph in Figure 5.15. This essential structural feature is conserved by the design framework during the HDL-code generation pro-



(a)



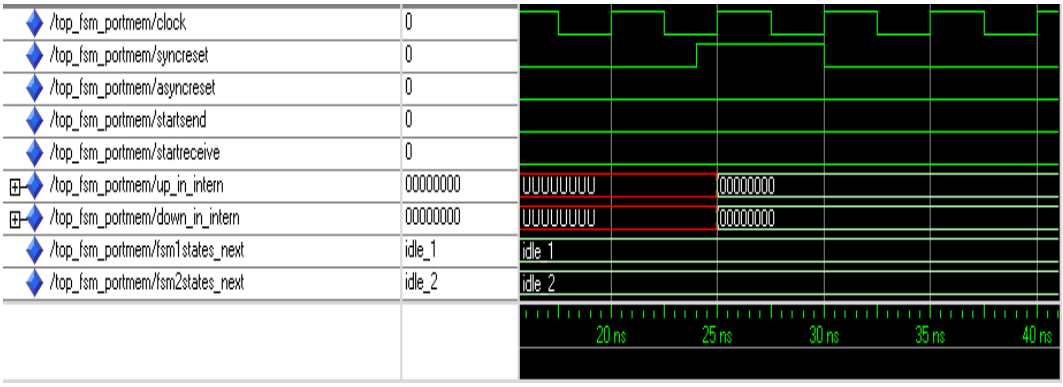
(b)

Figure 5.11: Wave-Diagram of HDL-Code Simulations. (a) Sending 8bit datum 01011010 as two 4bit data packages. (b) Receiving 8bit data 10101010 (UP input) 01010101 (DOWN input) as two 4bit data packages, at a time. Signals (Top-Down): Clock, Synchronous Reset, Asynchronous Reset, Start Sending, Start Receiving, UP-Input (only image b), DOWN-Input (only image b), Register Bank for UP-Input, Register-Bank for DOWN-Input, State Receive-FSM, State Send-FSM.

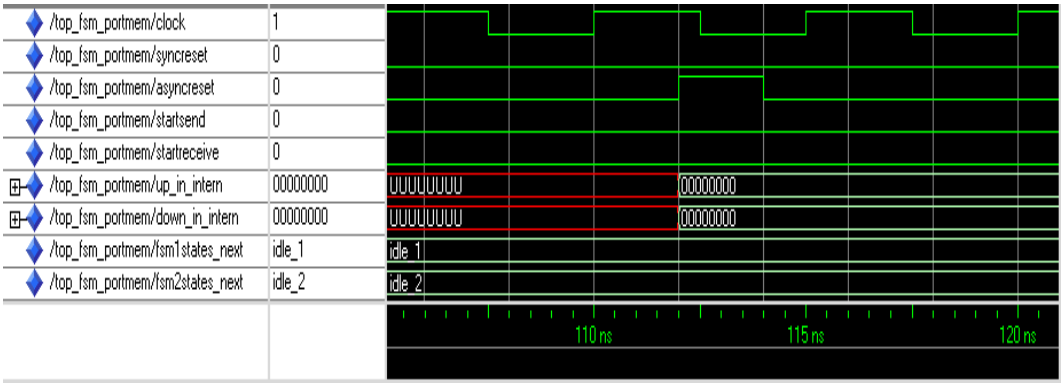
cess and hence available for floor-planning and place&route implementation steps.

The ongoing discussion and the following Figures 5.16, 5.17 and 5.18 further substantiate the previous statement. Graph $\mathbb{G}^{MH}$ is processed by the design framework and translated to HDL-code by Algorithm 5.10 and Function 5.11. Figure 5.16 shows the top level of the RTL schematic, which has been synthesized from the generated HDL-code. The complete memory hierarchy, which is represented by the graph $\mathbb{G}^{MH}$ in Figure 5.15 is recursively folded within the middle box of Figure 5.16. The boxes left and right represent the memory wrapper modules (see discussion in Section 5.4.1) for the up and down path of the memory hierarchy.

In Figure 5.17 parts of the memory hierarchy have been resolved to illustrate the hierarchical structure and the particular contents. The part shown in Figure 5.17 was before folded within the middle box of Figure 5.16. The schematic of the memory hierarchy, depicted in Figure 5.17, demonstrates that the proposed design framework preserves the hierarchical structure of graph $\mathbb{G}^{MH}$ during the HDL-code



(a)



(b)

Figure 5.12: Wave-Diagram of HDL-Code Simulations. (a) Synchronous (rising clock edge) reset behavior. Resetting (zeros) register banks of UP and DOWN input. (b) Asynchronous reset behavior. Resetting (zeros) register banks of UP and DOWN input. Signals (Top-Down): Clock, Synchronous Reset, Asynchronous Reset, Start Sending, Start Receiving, Register Bank for UP-Input, Register-Bank for DOWN-Input, State Receive-FSM, State Send-FSM.

generation process for this part of the architecture.

Additionally, the planarity of the hierarchy elements as shown at graph $\mathbb{G}^{MH}$ in Figure 5.15 is also preserved by the algorithms of the design framework. The relevant information of planarity is encoded within the generated HDL-code by means of the relations among the different RTL blocks and their mutual embedding.

Figure 5.18a-b shows a floor-plan and the corresponding FPGA implementation of the discussed memory hierarchy. The planarity- and hierarchy features have been utilized for the creation of this floor-plan. As can be seen at Figure 5.18a the floor-plan reflects the planarity feature of graph $\mathbb{G}^{MH}$ (cf. Figure 5.15) and hence demonstrates that the design framework preserves this central feature. Without the encoding of the planarity- and hierarchy feature within the generated HDL-code of the memory hierarchy the floor-planning would become extremely difficult for this large and distributed part of the architecture.

Summarizing, it has been shown that the proposed design framework can repre-

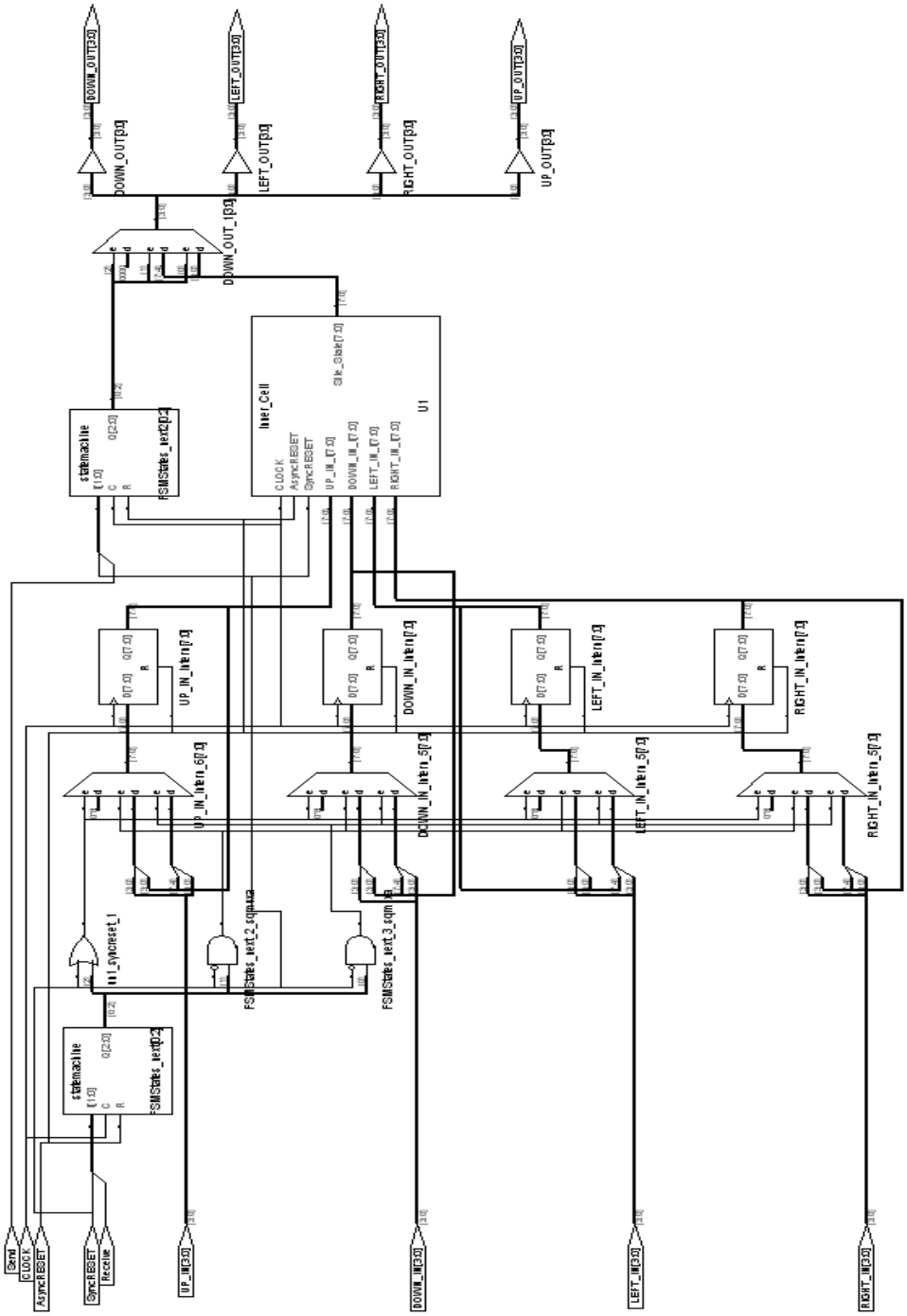


Figure 5.13: VLSI Implementation of combined single cell, port memory control unit and corresponding memory elements. RTL schematic synthesized from generated HDL code.

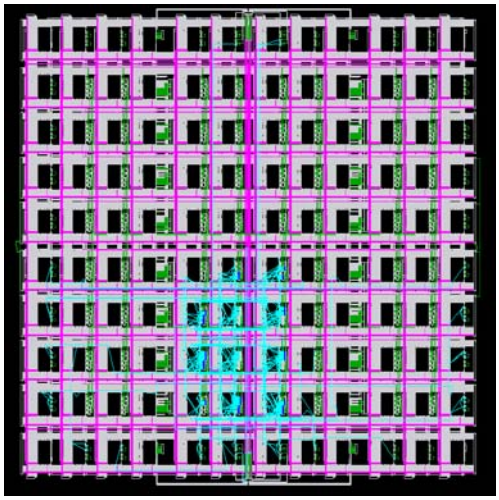


Figure 5.14: VLSI Implementation of combined single cell, port memory control unit and corresponding memory elements. FPGA implementation (Xilinx Virtex 2 family).

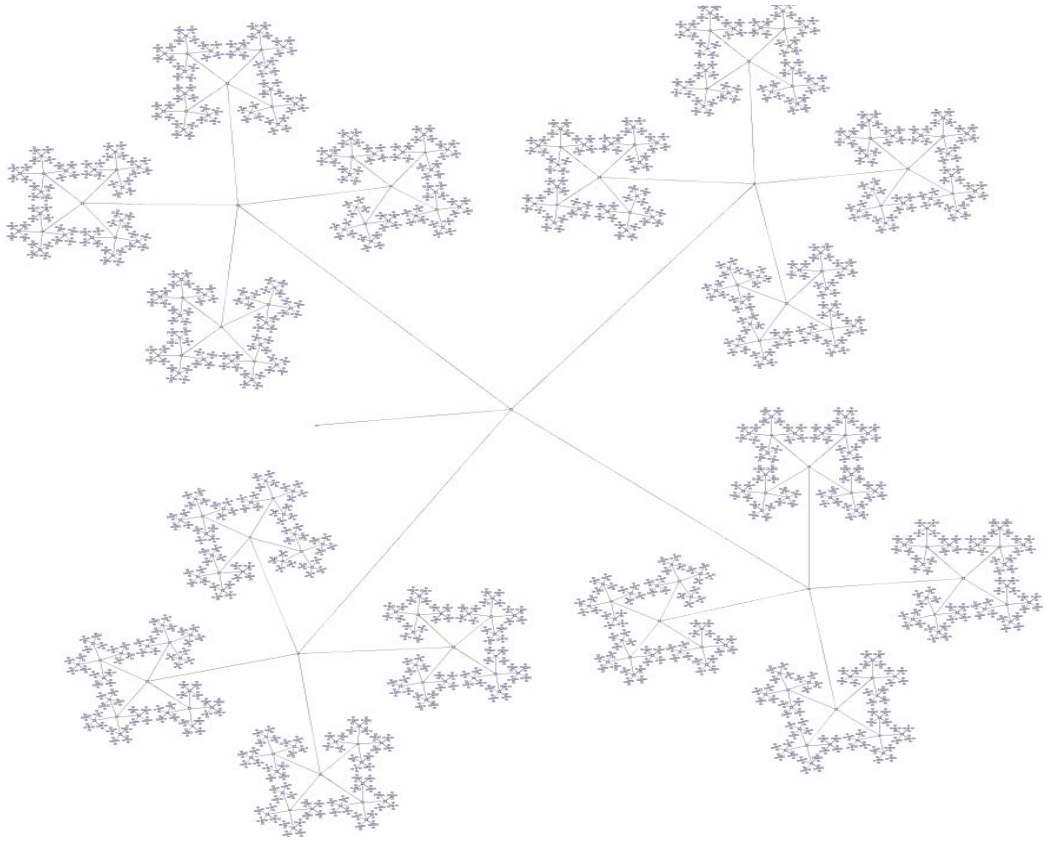


Figure 5.15: Modeling of memory hierarchy. Graph representation $\mathbb{G}^{MH}$ within design framework. Suitable for a 64×64 sized MRF.

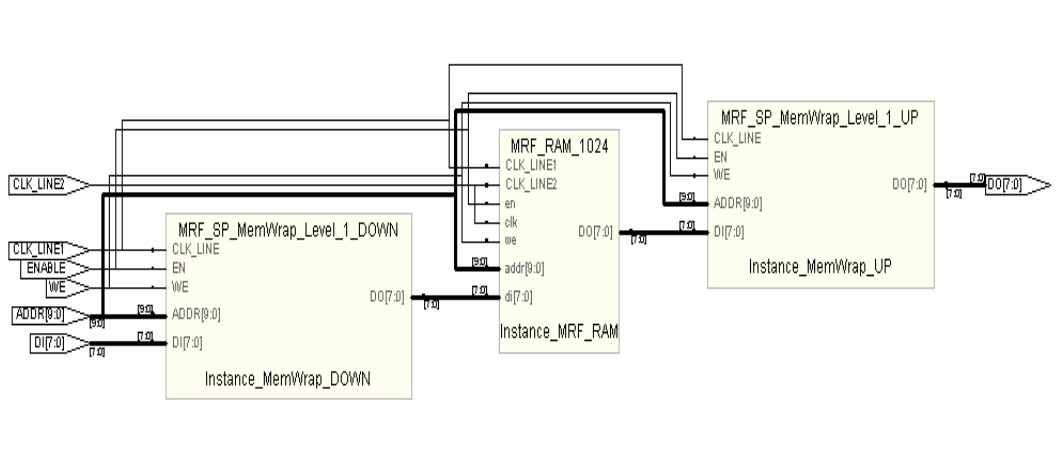


Figure 5.16: RTL schematic of memory hierarchy. Schematic synthesized from generated HDL-code.

sent the large graph of the memory hierarchy within the canonical design representation and is able to generate qualified HDL-code for this structure. Additionally, it has been demonstrated by means of the memory hierarchy that the algorithms of the design framework preserve elementary graph features within the generated HDL-code, which are exploited for instance during the floor-planning process.

5.6 Implementation Issues

The software technical realization of the introduced design framework (cf. Figure 5.1) generates some issues, which have to be respected and adequately dealt with. Without describing all software-implementation details exhaustively, the following vital concerns have been considered during implementation:

- *Efficiency & Scalability.* The graph data-structures of the framework, which will be generated and processed during the particular steps of the design flow are huge and complex, i.e. these structures are extremely memory- and processing-intensive. Consequently, the concrete software implementation should use different strategies in order to adequately cope with the memory and CPU-time usage during the design flow.

- *Extensibility.* The proposed and implemented MRF specific VLSI design framework represents a software-version, which operationally has been approved with respect to the claimed architectural requirements and the contemplated MRF-class. Despite the current state of the framework, further enhancements and improvements should be possible and systematically supported by the structure and arrangement of the software implementation without changing larger portions of the software or even changing the complete data-structure concept.

- *Consistency & Correctness.* The canonical design representation (CDR) is a diverse and complex data-structure, which consists of data-container and

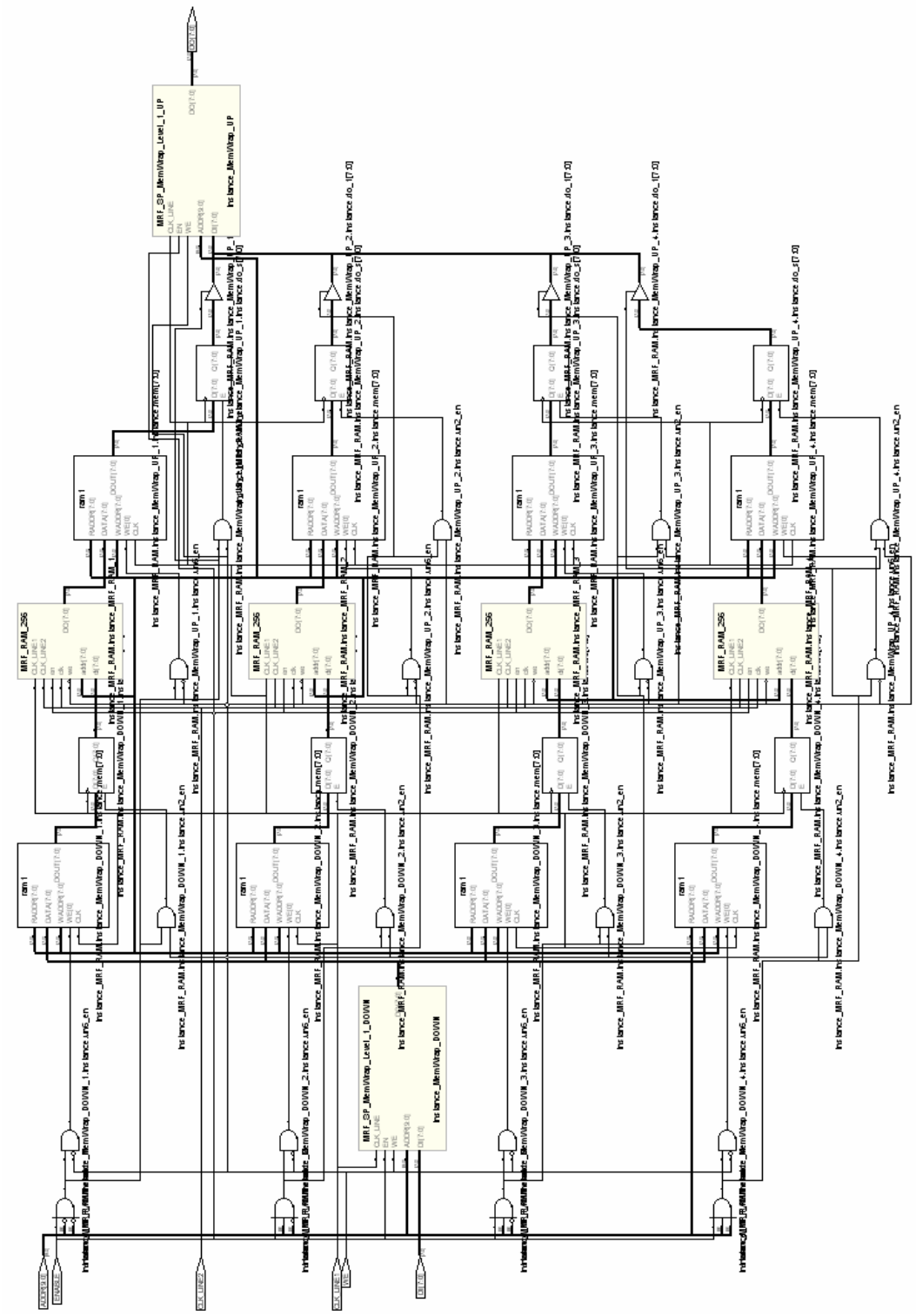
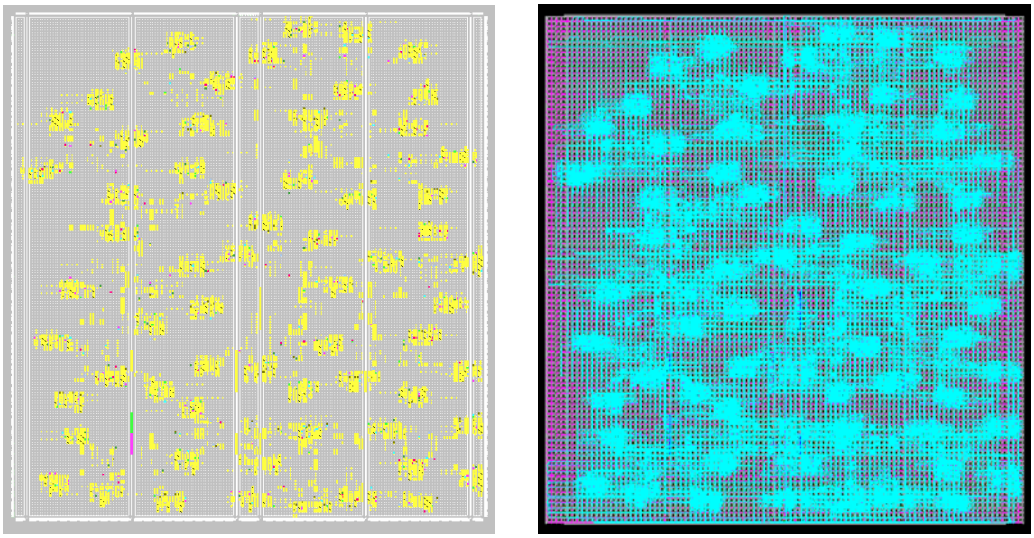


Figure 5.17: Partially collapsed RTL schematic of memory hierarchy. Schematic synthesized from generated HDL-code.



(a) (b)

Figure 5.18: FPGA Implementation of memory hierarchy. (a) Floor-plan of memory hierarchy. (b) Complete Place&Route based on created floor-plan.

different separated graph-structures. Consequently, the design flow, the particular graph generation procedures as well as the CDR itself should offer mechanisms, which ensure a guided and deterministic design flow, consistent graph-structures and an overall consistent CDR.

- *Platform Independence.* The outputs of the proposed MRF specific VLSI design framework are further processed by HDL simulation-tools and back-end tool chains, which are partially semiconductor technology dependent. All these additional and sometimes semiconductor-technology dependent tools are available for different computing and operation system platforms. Consequently, it would be of far-reaching practical relevance that the proposed design framework is platform-independent in order to establish a seamless MRF implementation flow on one single computing and operating system platform.

Concretely, the following implementation strategies and features of the design framework put into practice the above mentioned concerns. They are presented in the chronological order as above.

- The efficiency and scalability of the design framework, especially in view of the canonical design representation and its underlying graph data-structures, is realized by a fully customized and from scratch developed C++ class hierarchy. This graph-class solely rests on standardized C++ language constructs without applying any commercial or open-source library extensions. This approach of developing the basic data-structures from scratch gives us the optimum degree of control over the data-structures we need to put the claim of efficiency with regard to memory and CPU-time usage into practice. We are able to

tune the memory usage of each distinct node and edge within the canonical design representation till particular bytes. Each single byte saved is definitely an advantage as the canonical design representation typically contains several ten-thousands of nodes and edges for the representation of a massively parallel MRF architecture. Likewise we are able to tune the algorithms to allow them to be well-adjusted to the customized graph data-structures. By using open-source or commercial libraries we have never been in the position to receive the efficiency with respect to memory and CPU-time usage we are currently receiving with our fully customized basic data-structures and algorithms developed from scratch. Obviously, the overall scalability of the implemented design framework is a direct consequence of the tuned and adjusted data-structures and their algorithms.

- The extensibility of the MRF specific VLSI design framework is basically enabled by our fully customized data-structures as well as their algorithms and practically realized by the modular implementation (cf. Figure 5.1), which is arranged around the central data basis - the canonical design representation (CDR). The modular character of the framework allows it to easily add and replace specific modules, e.g. the parsing front-end, particular graph generators or the HDL back-end, which will improve or alter the capabilities of the framework. As all modules will always work on the CDR, which is fully customized and thus to its full extent open to us, it is a straightforward task to provide the interfaces between the CDR and new or modified modules.

- The consistency and correctness of the design flow as well as the canonical design representation at each time of the framework-internal development process is essentially realized by means of a strictly guided graphical user interface and checking procedures of the graph generation methods, the particular graphs and the complete canonical design representation. The graphical user interface is conceptualized to allow the user to only proceed with appropriate design steps without having the opportunity to execute misleading design actions. Furthermore, the design flow and CDR-inherent mechanisms additionally ensure the consistency and correctness of the design flow data. Each graph generation method is enhanced by procedures, which traverse the generated graph and at the same time check against typical features of that graph in order to identify profound structural errors. Likewise the complete design graph $\mathcal{G}^{\mathcal{D}}$ (cf. Section 5.4 and Corollary 5.6) is checked. Additionally, each single graph of the canonical design representation as well as the complete design graph can be written out in the standardized *GraphML* format for further detailed visual inspections with appropriate tools.

- The computing platform and operating system independence of the proposed design framework is put into practice by two strategies. At first we have completely forgone open-source or commercial libraries in order to realize data-structures and algorithms. All data-structures and algorithms are fully customized and developed from scratch with standardized C++, whereas the C++ standard today includes the STL. Furthermore the platform independence with respect to the graphical user interface is realized by using Qt [139],

which supports Windows, MAC and X11 platforms.

5.7 Relation of Thesis Parts

As already mentioned, all parts of this thesis are closely related to each other and together form a seamless development-environment for massively parallel hardware architectures of Markov Random Fields. In summary, the environment rests on the theoretical fundamentals of Markov Random Field theory (Chapter 2) and the building blocks of the system-architecture template (Chapter 3) derived out of it. Essentially, the MRF specific development-environment comprises the simulation framework (Chapter 4) and the design framework (Chapter 5). The interplay of the different thesis parts is formally expressed by the *congruence relation*already introduced, which is repeatedly shown in Figure 5.19.

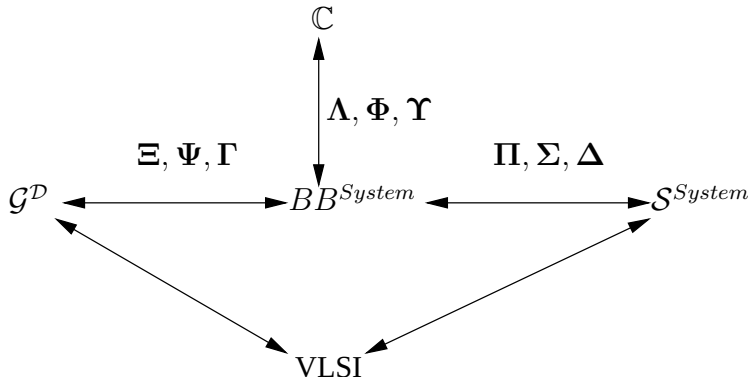


Figure 5.19: Established congruence relation.

This chapter has presented the formal fundamentals, that allows us to establish the last open branch of the congruence relation, which describes the relation of the building blocks and the corresponding representing graphs (see left branch in Figure 5.19). The building blocks and the graphs are related to each other by the set of mappings $\mathbf{\Psi}$, $\mathbf{\Gamma}$ and $\mathbf{\Xi}$, which have been defined and commented on in the preceding sections of this chapter. This finalizes the description of the congruence relation, which represents the interrelation of the different thesis parts and the overall line of thought this thesis pursues.

5.8 Summary

In this chapter we have presented a semiconductor-technology independent VLSI design framework for massively parallel Markov Random Field based processing devices, which rests upon a graph theoretical architecture representation and graph compilation techniques in order to transform graphs into synthesizeable hardware descriptions. Particular MRF model definitions are handed over to the design framework by means of hardware abstract specifications. The language front-end of the framework currently supports XML as input for the hardware abstract specifications of MRF models. Due to the modular structure of the framework, which has

been described in Section 5.1, other input languages can be used, in case the XML front-end is replaced by the corresponding front-end of the regarded language.

Many different graphs are required to completely model the massively parallel architecture of a specific MRF in accordance with the architecture template (see Chapter 3) in a semiconductor-technology independent manner. All these different graphs, which are collected in the *Canonical Design Representation* (CDR) of the framework, have been formally defined in Section 5.2. Furthermore each single graph has been classified with respect to a VLSI appropriate scheme, which distinguishes between *Topology & Structure*, *Processing* and *Control* representing graphs. Exactly this graph-classification scheme simplifies the structure, the definition and compilation to HDLs of the particular graphs, as they are specially tuned to model specific architectural components of the massively parallel architecture. Important and for consecutive VLSI implementations steps - especially place&route steps - far-reaching inherent features of the graphs have been proved in Section 5.2.

The overall MRF-specific VLSI design flow, which parses and analyzes the input specification, generates the different representing graphs and in the following compiles these graphs into synthesizeable HDL representations, has been presented in Section 5.3 respectively in Section 5.4. In Section 5.3 we have described the different algorithms, which expand all graphs of the canonical design representation in accordance with the definitions and features of the particular graphs given in Section 5.2. The graph generation algorithms also have been presented in the chronological order of the scheme, which distinguishes between *Topology & Structure*, *Processing* and *Control* representing graphs. Finally, Section 5.4 has presented the strategies and procedures to translate hardware-abstract graphs into hardware-concrete and synthesizeable HDL (hardware description language) representations, which are suitable for industrially relevant synthesis and technology back-end steps.

The congruence relation, which illustrates the interdependencies of the different thesis parts and simultaneously gives an overview of the thesis theme, has been presented in the following section. This Section 5.7 has explained the last branch of the congruence relation, which is given by the mappings of the building blocks to hardware-abstract graph representations. Some essential implementation issues of the proposed and software-technical realized VLSI design framework for massively parallel MRF-based processing devices have been identified and discussed in Section 5.6. In summary, the VLSI design framework distinguishes itself by a completely customized data- and algorithm structure, a modular and thus extensible software structure, a computing resources (memory and computing-time) tuned processing procedure and inherent self-checking capabilities of the data-structures and the design flow. Exemplary graphs, RTL schematics and prototypical FPGA implementations, which have been generated by means of the proposed design framework, have been presented and described in Section 5.5. These selected results are completed by numerous VLSI implementation results, collected in Appendix B.

5.9 Bibliographical Comments

The basic idea to raise the level of abstraction with respect to the description respectively representation of technical systems in order to cope with complexity, is

neither new within the computer science- and engineering-community nor in the electronic design automation community [51] [118]. All these disciplines have undergone a steady process of raising the description-level of abstraction in the past. Obviously, the computer science and software community has made the greatest and fastest progress in this field [50] [1] and this community is rather prepared to adopt new approaches [46] [8], whereas the electronic design automation community is traditionally more conservative in this field [83] [98].

This is caused by the fact that firstly new methods and design approaches must work absolutely correct from the beginning as a failure of the method, which leads to a bug in a produced semiconductor device, is seldom fixable and thus extremely costly. Secondly, the next level of abstraction will shift hardware descriptions to an algorithmic and behavioral level (currently discussed under the topic *electronic system level* (ESL) design [111]). This abstraction level mostly neglects hardware specific aspects - e.g. clocks, ports, fix-point number representation and finite state machines [28] [9] - and approaches toward software-like descriptions. This fact and situation is hardly to accept for VLSI designers as their familiar description kind changes fundamentally [22] [81].

The origin of High-Level Synthesis in the electronic design automation community goes back to R. Camposano and his early research work conducted at the German National Research Center for Computer Science (GMD Sankt Augustin, Germany, now Fraunhofer-Gesellschaft). This fundamental research cumulated in the first operational silicon compiler, called Yorktown [22] [28] [27] [29], owned by IBM, where R. Camposano led the project on high-level synthesis at the IBM T.J. Watson Research Center, between 1986 and 1991.

Based on the successful technology demonstration of High-Level Synthesis by means of IBM's operational approved silicon compiler a few textbooks respectively collections, e.g. [3], [22] in D.D. Gajski, Editor, Silicon Compilation and [51], on this topic appear in this time period. But none of these textbooks rigorously covers the fundamental themes of High-Level Synthesis and a comprehensive presentation of the High-Level Synthesis topic is up to now still missing.

Hence, several suggestions exists and are still proposing in the contemporary literature for each of the representation and processing steps of High-Level Synthesis. In [20] Borriello proposed a design representation with combined data- and control-flow graphs in one unified data structure in order to simplify the overall processing of the representation. Curatelli extended the unification approach to general systems in [38] and enhanced the graph structure with various information and constraints. In contrast to that Ge [53] proposed a control flow graph centered representation with a linkage to the relevant data-flow information. Summarizing, the picture of design representations is diverse and not consistent. Furthermore it is difficult to compare and assesses the different approaches among each other. A well-defined categorization of the different approaches to represent a design respectively a system is still missing.

A similar situation can be found with respect to the processing steps *scheduling* and *binding* of High-Level Synthesis. Over the last decade several different scheduling approaches have been investigated and reported on [27] [163] [32] [51] [73]. Recently Chabini and Wolf [32] discussed a combined scheduling, binding and ritiming approach under constraints in a systematic optimization setting. The re-

ported results are promising but it remains to clarify whether this approach can be efficiently generalized for other application domains and systems.

Chapter 6

Conclusion

"Alles Gescheidte ist schon gedacht worden,
man muss nur versuchen es noch einmal zu denken."

Johann Wolfgang von Goethe (1749-1832)
Wilhelm Meisters Wanderjahren 1829
(Betrachtungen im Sinne der Wanderer. Kunst, Ethisches, Natur)

It was the aim of this work to introduce a novel and seamless approach to the hardware-relevant simulation and semiconductor-independent VLSI design of Markov Random Field based processing-grids. The Markovian processing architectures are characterized by their topology of processing elements, their neighborhood-system support, their massively parallel processing capabilities and their specification for purely, digital semiconductor implementation technologies. Without anticipating the readers opinion about how far this dissertation has succeeded in presenting appropriate approaches and results for the above stated claim, we will summarize the content and the results of this work from our point of view. Finally, we will close this thesis with a realistic outlook on how we intend to proceed in the next months as well as on what we have in mind for the future with respect to the presented research-direction.

Each single decision-making within this thesis is rigorously justified and realized in accordance with established theoretical foundations as well as with respect to the central requirements of algorithmic robustness in real world scenarios, real-time processing capabilities and the physical compact realizations this work is built-up. Therefore it was the main purpose of Chapter 2 to attain a self-contained and independent basis the work can refer to. In particular, a condensed synopsis of Bayesian image analysis and the related general processing structures of Markov Random Fields have been presented as well as the theoretically well-founded criteria for massively parallel processing approaches. Furthermore we have defined the contemplated class of MRF models so that a well-defined delimitation of the models this thesis deals with, is given. Additionally, we have presented an independent analysis, which likewise regards semiconductor-technologies variants and their design methodologies. Our analysis has revealed that purely digital implementation technologies offer profound advantageous compared with other technology variants. This discussion as well as the presented approach and the result of this thesis re-

futes the prevalent claim made by the CNN community, which advocates design approaches and implementations based on purely analog technologies for more than a decade. Finally two realistic and industrially relevant image processing models, which serve as test-cases throughout the thesis, have been derived.

Within Chapter 3 we have systematically derived the different building blocks, which together define the architecture-template of the proposed massively parallel processing topology. Our advocated derivation process of these architectural building blocks is characterized by two features worth mentioning. At first we have exclusively based the derivation process on architectural uncommitted constituents, which are solely justified by well-founded theoretical fundamentals, i.e. we have defined a basis for the derivation process of the building blocks, which is architecturally neutral and strictly in accordance with theory. Secondly, the actual derivation process is conducted with respect to the general architectural constraints of (1) massively parallel processing and (2) VLSI implementation capability. These constraints impose several structural features to allow the architectural uncommitted constituents to take shape of architectural building blocks. Finally we have demonstrated how the proposed architecture-template can be used by applying it to the two test-case models.

Our simulation approach for the defined class of Markov Random Fields and the derived architecture-template, which is put into practice by the Simulation-Framework, has been introduced in Chapter 4. In particular, we have presented the overall structure of the framework and the different essential components, which pool functional similar building blocks. Furthermore, we have presented the principal structure of each building block by means of prototypes. In addition the mechanisms of centralizing the compilation procedure of specific MRF simulation-models has been explained as well as the automated set-up process of MRF models within the proposed simulation-framework. The capabilities and performance of the proposed Simulation-Framework has been intensively studied by means of numerous simulation-runs and result analyzes, which undoubtedly confirm the stated features of the software-technical realized simulation-framework.

The VLSI design approach described in Chapter 5 introduces a novel graph-theoretical methodology, which adequately covers the defined model class as well as the complexity of the particular massively parallel architectures. In particular, this advocated methodology maps the different building blocks to special graph-structures, which represent the functionality in a hardware-neutral, a semiconductor-technology and a coding-language neutral manner. This overcomes the problem of providing the complete hardware details of these large and complex architectures already at the beginning of the design-flow and thus of fixing the architecture at the outset without exploring variants. Architecture variants are generated by adjusting the corresponding graph appropriately. The following algorithms analyze the particular graphs and synthesize a HDL-code in accordance with the IEEE VHDL language specification. Several different FPGA-implementations of building blocks have demonstrated the capabilities of the proposed design methodology. To further underpin the advocated approach and illustrate the intermediate steps, we have also shown particular graph structures, RTL schematics, gate netlists and the prototypical FPGA place & route results.

The short term outlook of the so-far pursued research covers the following topics:

Although the already implemented scheduling methods have proved to be adequate, we want to adapted scheduling approaches in order to further improve the datapaths of the architecture. As the debugging and analysis of the generated graphs becomes unmanageable by means of visual inspection, we plan to add a scalable and flexible graph-checking mechanism, which replaces the currently more rudimentary realized checking-routines for graph-consistency.

As a long term research and development perspective for the two introduced frameworks, we plan to extend them to allow irregular site-topologies with more irregular neighborhood systems to be supported. This will offer the option to systematically address various applications in the fields of speech processing, control-systems, communication-systems and of course again in the domain of image processing by means of differently shaped graphical models. It is to be expected that the resulting processing devices with their physically compact realizations and advantageous packaging, their power consumption characteristic and their better processing speed realized by parallelism, will make novel systems for completely new application scenarios possible.

I personally believe that the discussed architectures and their resulting processing devices are a key-enabling element for novel systems and applications. Hopefully, I have convinced the reader that the presented approach, which has been put into practice by the two frameworks, adequately addresses the specific simulation- and design needs of these high-performance processing architectures.

Appendix A

Simulation Framework Results

This appendix recapitulating presents various simulation results, which have been relocated from the main text and the result Section 4.4 to this appendix to enhance the overall readability of this thesis. Furthermore several additional results have been included, especially complete simulation series to illustrate the processing dynamics of the models and thus also of the proposed hardware architecture.

The illustrated simulation results of this appendix are based on the collection of raw image data, which is composed of artificial pictures and real world pictures. In order to receive a realistic assessment of the processing architecture and the MRF model combination, we have consciously chosen image data with different fitness for the models. Central features of the raw image data are summarized in Table A.1.

Figure	**Description**
Figure A.1a-b	Grayscaled raw image data. Image size: 128×128. Grayscale channel: 8bit; value range [0,255]. Image features: Artificial image scenes. Advantageously grayscale-object relation for segmentation.
Figure A.1c-f	Grayscaled raw image data. Image size: 128×128. Grayscale channel: 8bit; value range [0,255]. Image features: Real-world image scenes. Figure A.1c is difficult to segment on the basis of purely local grayscale information.
Figure A.2a-d	Colored raw image data. Image size: 128×128. RGB channels: each 8bit; value range [0,255]. Image features: Real-world image scenes. Advantageously color-object relation for segmentation.
Figure A.2e-f	Colored raw image data. Image size: 128×128. RGB channels: each 8bit; value range [0,255]. Image features: Real-world image scenes. Unfavorably color-object relation for segmentation.

Table A.1: Overview of raw image data features.

Selected simulation sequences have been chosen to illustrated the processing dynamic of the segmentation model in a massively parallel calculation setting. Table A.2 summarizes the features of the simulation sequences.

Figure	Description
All Figures	5th order neighborhoods. Massively parallel processing dynamic. Float-point precision.
Figure A.3	Simulation sequence for grayscaled image A.1a.
Figure A.4	Simulation sequence for color image A.2c.

Table A.2: Overview of simulation sequence features.

In Table A.3 all segmentations are listed, which have been calculated for the grayscaled raw image data with 1st-5th order neighborhood systems and full float-point precision.

Figure	Description
All Figures	1st-5th neighborhoods. Float-point precision. Massively parallel processing dynamic. Relaxation steps: 225.
Figure A.5	Segmentations for grayscaled image A.1a. 3 Classes and 4 Bins.
Figure A.6	Segmentations for grayscaled image A.1b. 3 Classes and 3 Bins.
Figure A.7	Segmentations for grayscaled image A.1c. 4 Classes and 8 Bins.
Figure A.8	Segmentations for grayscaled image A.1d. 3 Classes and 6 Bins.
Figure A.9	Segmentations for grayscaled image A.1e. 3 Classes and 6 Bins.
Figure A.10	Segmentations for grayscaled image A.1f. 3 Classes and 4 Bins.

Table A.3: Overview of segmentation results for grayscaled images.

Table A.4 summarizes all segmentations, which have been calculated for the colored raw image data with 1st-5th order neighborhood systems and full float-point precision.

Figure	Description
All Figures	1st-5th order neighborhoods. Float-point precision. Massively parallel processing dynamic. Relaxation steps: 225.
Figure A.15	Segmentations for color image A.2a. 4 Classes and 8 Bins.
Figure A.16	Segmentations for color image A.2b. 3 Classes and 4 Bins.
Figure A.17	Segmentations for color image A.2c. 2 Classes and 5 Bins.
Figure A.18	Segmentations for color image A.2d. 3 Classes and 5 Bins.
Figure A.19	Segmentations for color image A.2e. 3 Classes and 5 Bins.
Figure A.19	Segmentations for color image A.2f. 3 Classes and 10 Bins.

Table A.4: Overview of segmentation results for color images.

Table A.5 summarizes all segmentations, which have been calculated for the colored raw image data with 5th order neighborhood systems and fixed-point precision (typically 26bit, 20bit and 16bit).

Figure	**Description**
Figure A.20	Segmentations for color image A.2a. 5th order neighborhood. 30bit, 26bit and 22bit fixed-point precision.
Figure A.21	Segmentations for color image A.2a. 5th order neighborhood. 20bit, 18bit and 14bit fixed-point precision.
Figure A.24	Segmentations for color image A.2b. 5th order neighborhood. 26bit, 20bit and 16bit fixed-point precision.
Figure A.25	Segmentations for color image A.2c. 5th order neighborhood. 26bit, 20bit and 16bit fixed-point precision.
Figure A.27	Segmentations for color image A.2d. 5th order neighborhood. 26bit, 20bit and 16bit fixed-point precision.
Figure A.32	Segmentations for color image A.2e. 5th order neighborhood. 26bit, 20bit and 16bit fixed-point precision.
Figure A.35	Segmentations for color image A.2f. 5th order neighborhood. 26bit, 20bit and 16bit fixed-point precision.
Figure A.29	Analysis of PSNR and corresponding sequence of iteration steps for color image A.2d.

Table A.5: Overview of segmentation results with fixed-point precision.

Table A.6 summarizes particular segmentations, which have been calculated for selected grayscaled images with 5th order neighborhood systems and fixed-point precision (26bit, 20bit and 16bit).

Figure	**Description**
Figure A.11	Segmentations for grayscaled image A.1c. 5th order neighborhood. 26bit, 20bit and 16bit fixed-point precision.
Figure A.12	Segmentations for grayscaled image A.1d. 5th order neighborhood. 26bit, 20bit and 16bit fixed-point precision.
Figure A.13	Segmentations for grayscaled image A.1e. 5th order neighborhood. 26bit, 20bit and 16bit fixed-point precision.
Figure A.14	Segmentations for grayscaled image A.1f. 5th order neighborhood. 26bit, 20bit and 16bit fixed-point precision.

Table A.6: Overview of segmentation results with fixed-point precision.

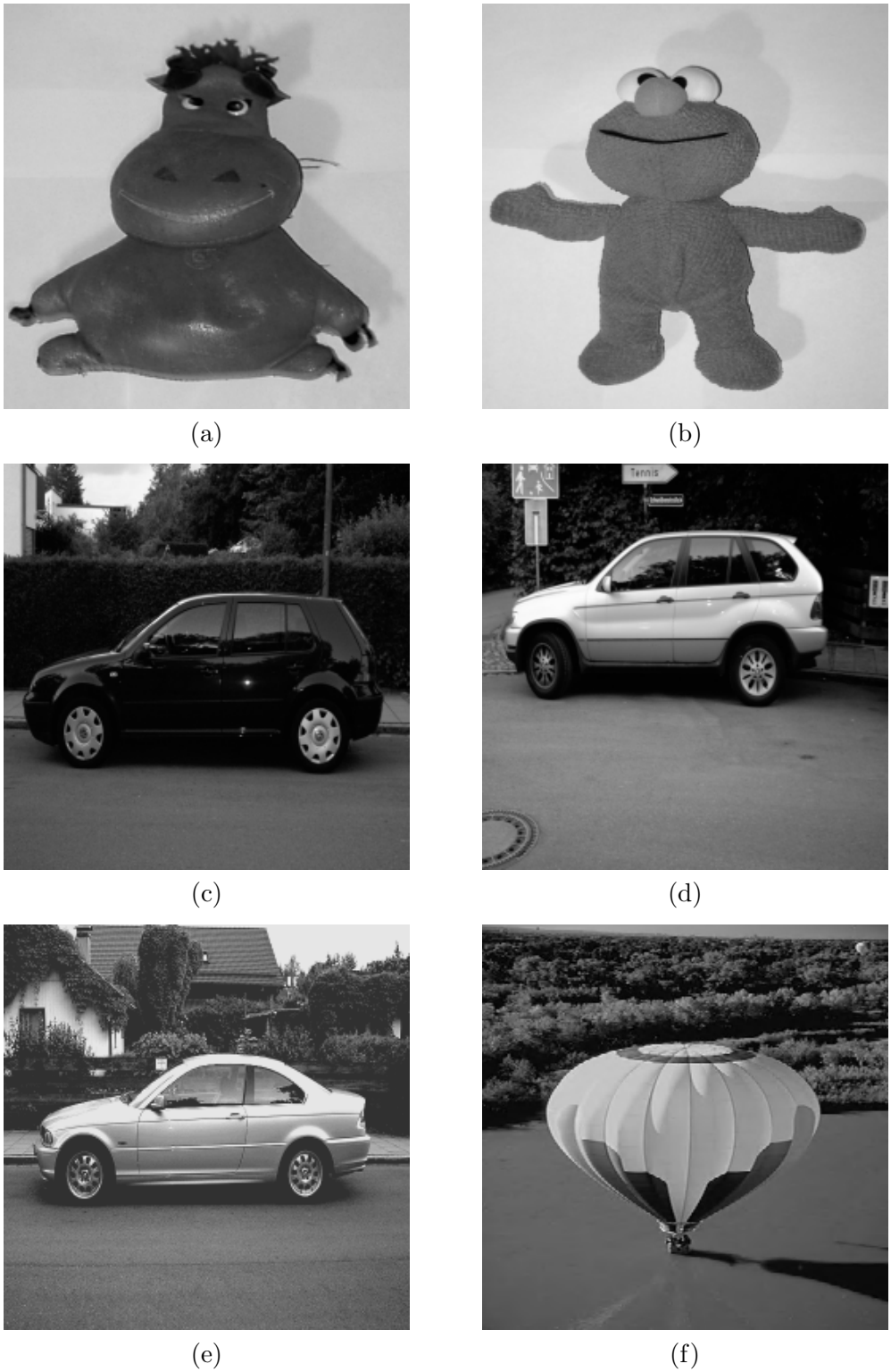
Table A.7 summarizes the estimated prototypes of selected architecture/model variants and image data and Table A.8 summarizes MSE and PSNR analysis.

Figure	Description
All Figures	5th order neighborhood. Massively parallel processing dynamic. Relaxation steps: 225.
Figure A.30	Class prototypes for color image A.2d. 3 Classes and 5 Bins for each RGB channel. Fixed-point precision: 26bit.
Figure A.31	Class prototypes for color image A.2d. 3 Classes and 5 Bins for each RGB channel. Fixed-point precision: 20bit.
Figure A.34	Class prototypes for color image A.2e. 3 Classes and 5 Bins for each RGB channel. Fixed-point precision: 20bit.
Figure A.37	Class prototypes for color image A.2f. 3 Classes and 10 Bins for each RGB channel. Fixed-point precision: 16bit.

Table A.7: Overview of estimated class prototypes.

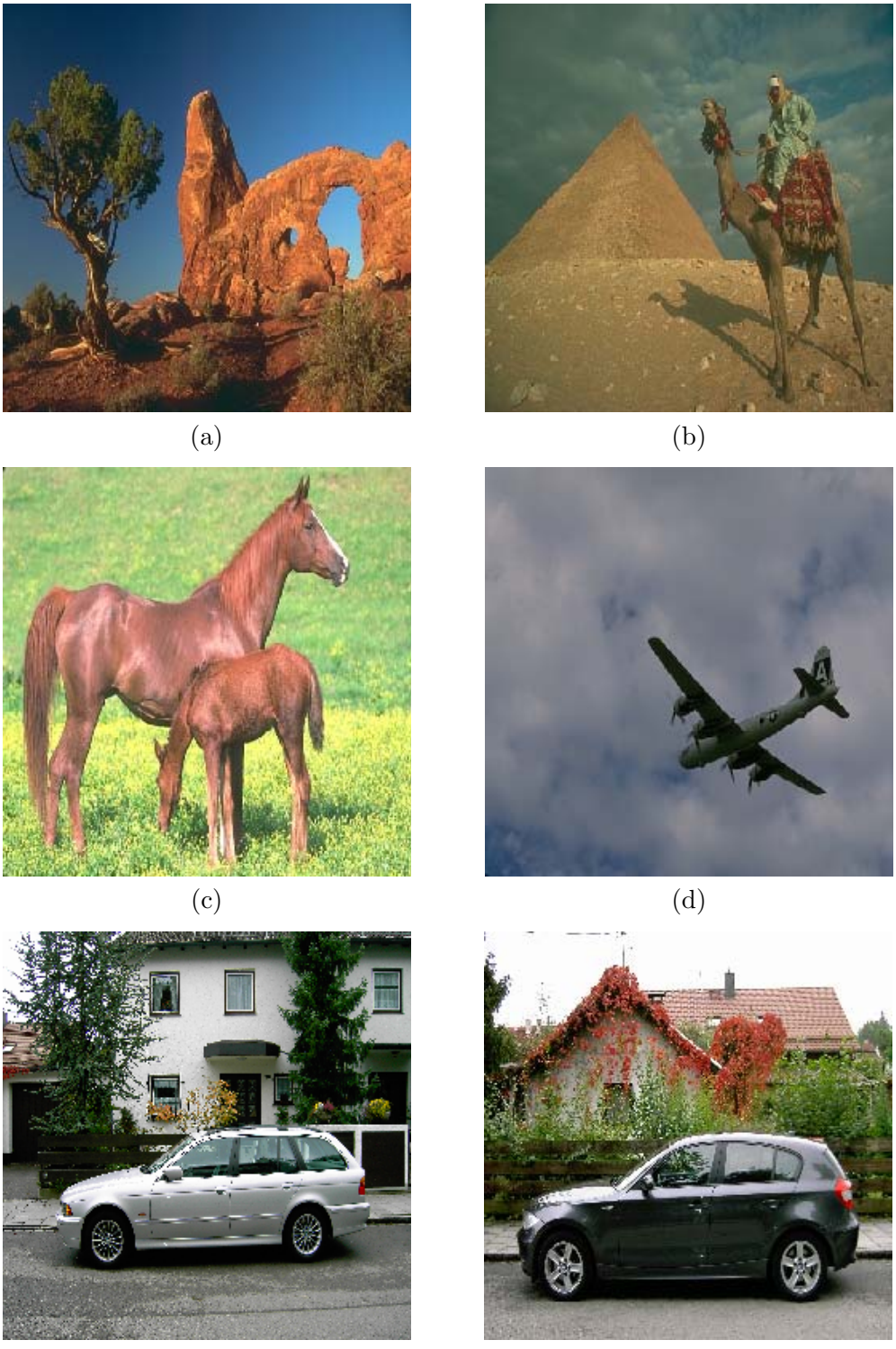
Figure	Description
All Figures	MSE and PSNR taken with respect to number of misclassification between float-point precision segmentation (ground-truth) and corresponding fixed-point precision segmentation. 5th order neighborhood. Massively parallel processing dynamic. Relaxation steps: 225.
Figure A.22	MSE and PSNR for segmentations of color image A.2a. MSE for fixed-point precisions: 32bit, 30bit, 26bit, 22bit, 20bit, 18bit, 16bit and 14bit. PSNR for fixed-point precisions: 32bit, 20bit and 14bit.
Figure A.23	MSE and PSNR for segmentations of color image A.2a. MSE for fixed-point precisions: 32bit, 30bit, 26bit, 22bit, 20bit, 18bit, 16bit and 14bit. PSNR for fixed-point precisions: 32bit, 26bit and 22bit.
Figure A.26	MSE and PSNR for segmentations of color image A.2c. MSE for fixed-point precisions: 32bit, 26bit, 20bit and 16bit. PSNR for fixed-point precisions: 32bit, 26bit, 20bit and 16bit.
Figure A.28	MSE and PSNR for segmentations of color image A.2d. MSE for fixed-point precisions: 26bit, 20bit and 16bit. PSNR for fixed-point precisions: 26bit, 20bit and 16bit.
Figure A.33	MSE and PSNR for segmentations of color image A.2e. MSE for fixed-point precisions: 26bit, 20bit and 16bit. PSNR for fixed-point precisions: 26bit, 20bit and 16bit.
Figure A.36	MSE and PSNR for segmentations of color image A.2f. MSE for fixed-point precisions: 26bit, 20bit and 16bit. PSNR for fixed-point precisions: 26bit, 20bit and 16bit.

Table A.8: Overview of MSE and PSNR plots.



(a) (b) (c) (d) (e) (f)

Figure A.1: Collection of raw image data used during experiments.



(a) (b) (c) (d) (e) (f)

Figure A.2: Collection of colored raw image data used during experiments.

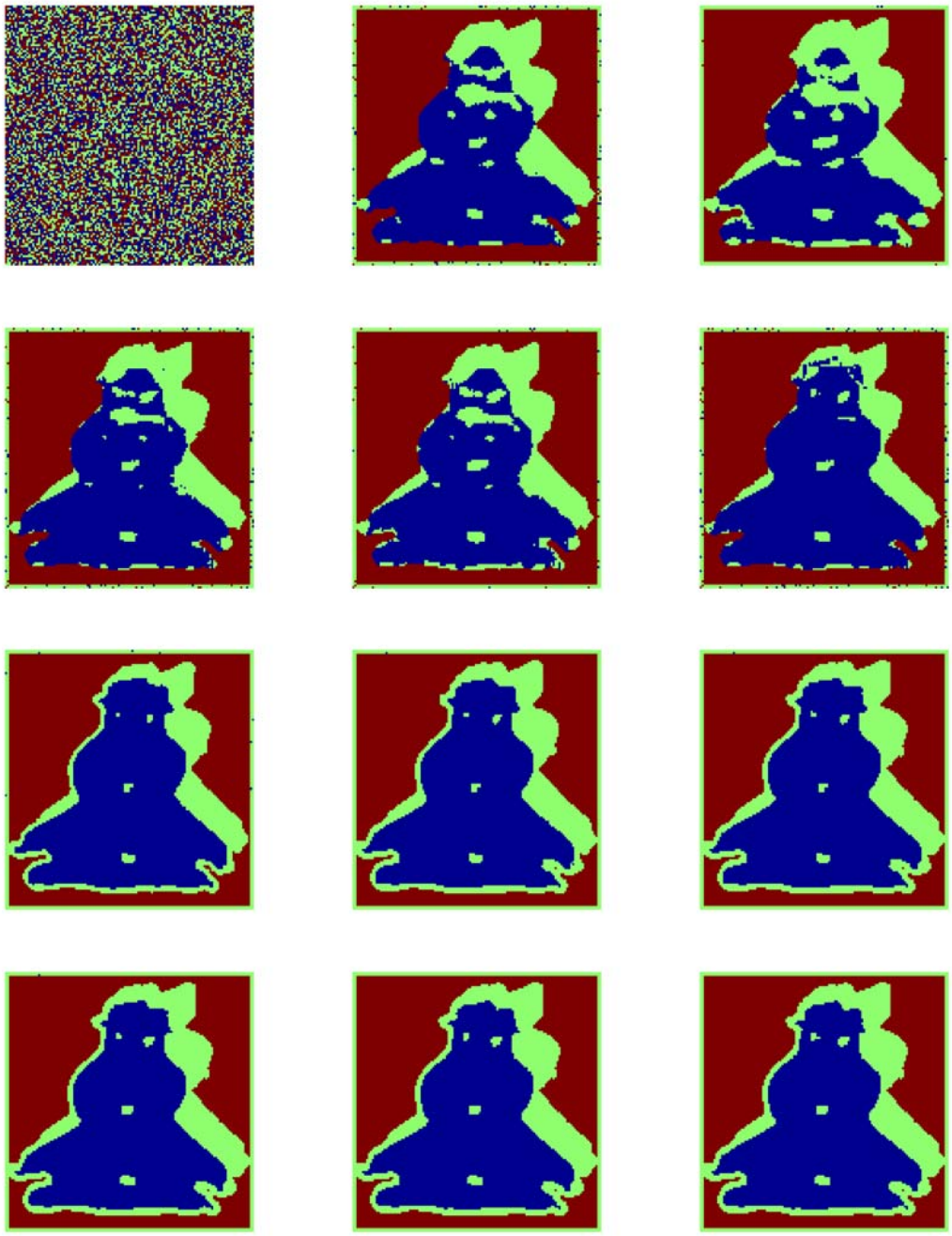


Figure A.3: Simulation sequence (left to right and up to down). Processing dynamic of unsupervised segmentation model. Half of the pictures show the first processing steps and the remaining pictures show processing steps with larger spacings. The sequence reads: $1, 2, 3, 4, 5, 6, 50, 60, 70, 80, 90, 100$. Model/architecture parameters: 3 classes, 4 equally spaced bins, 5th order neighborhoods, massively parallel processing dynamic and float-point precision.

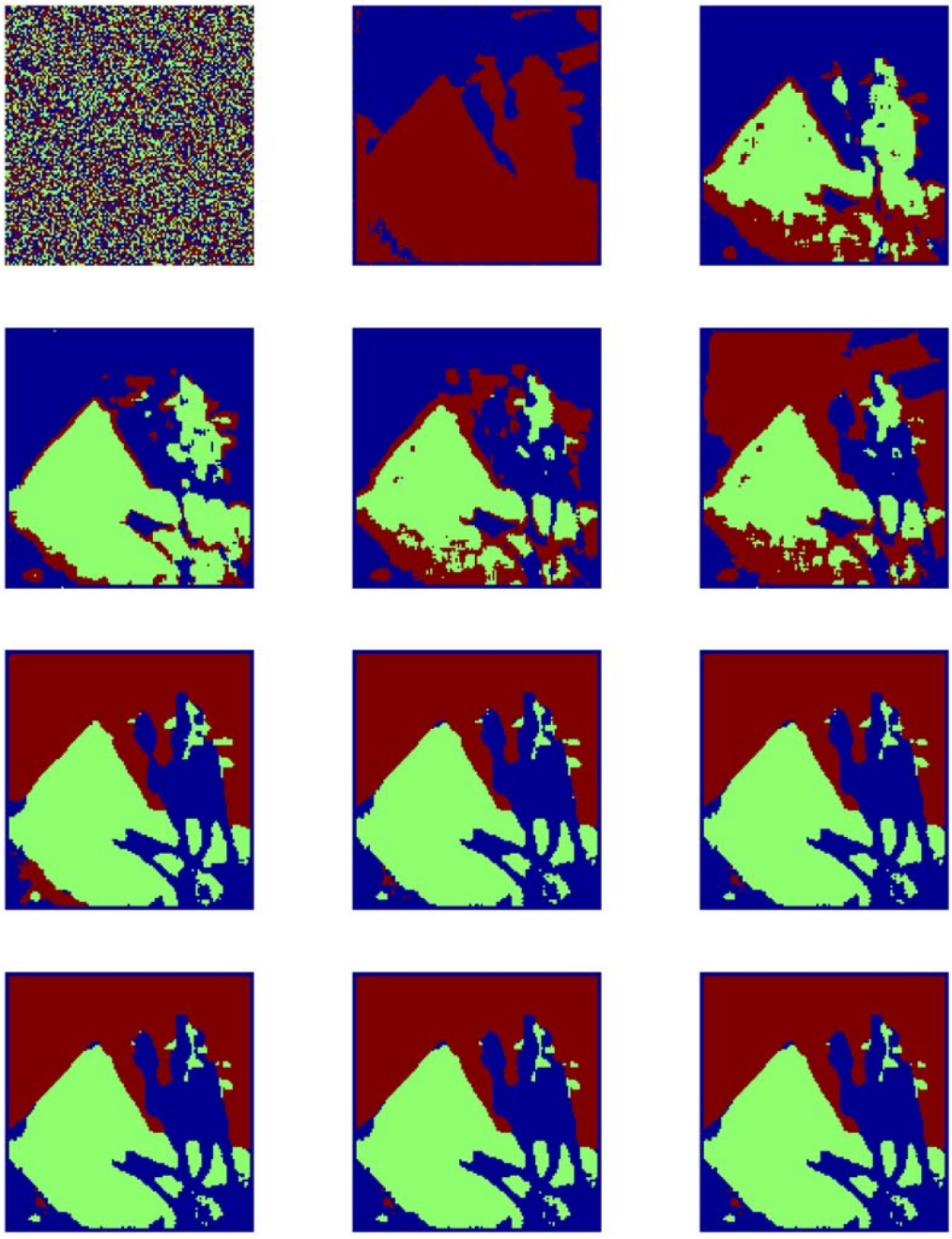


Figure A.4: Simulation sequence (left to right and up to down). Processing dynamic of unsupervised segmentation model. Half of the pictures show the first processing steps and the remaining pictures show processing steps with larger spacings. The sequence reads: $1, 2, 3, 4, 5, 6, 50, 60, 70, 80, 90, 100$. Model/architecture parameters: 3 classes, 4 equally spaced bins, 5th order neighborhoods, massively parallel processing dynamic and float-point precision.

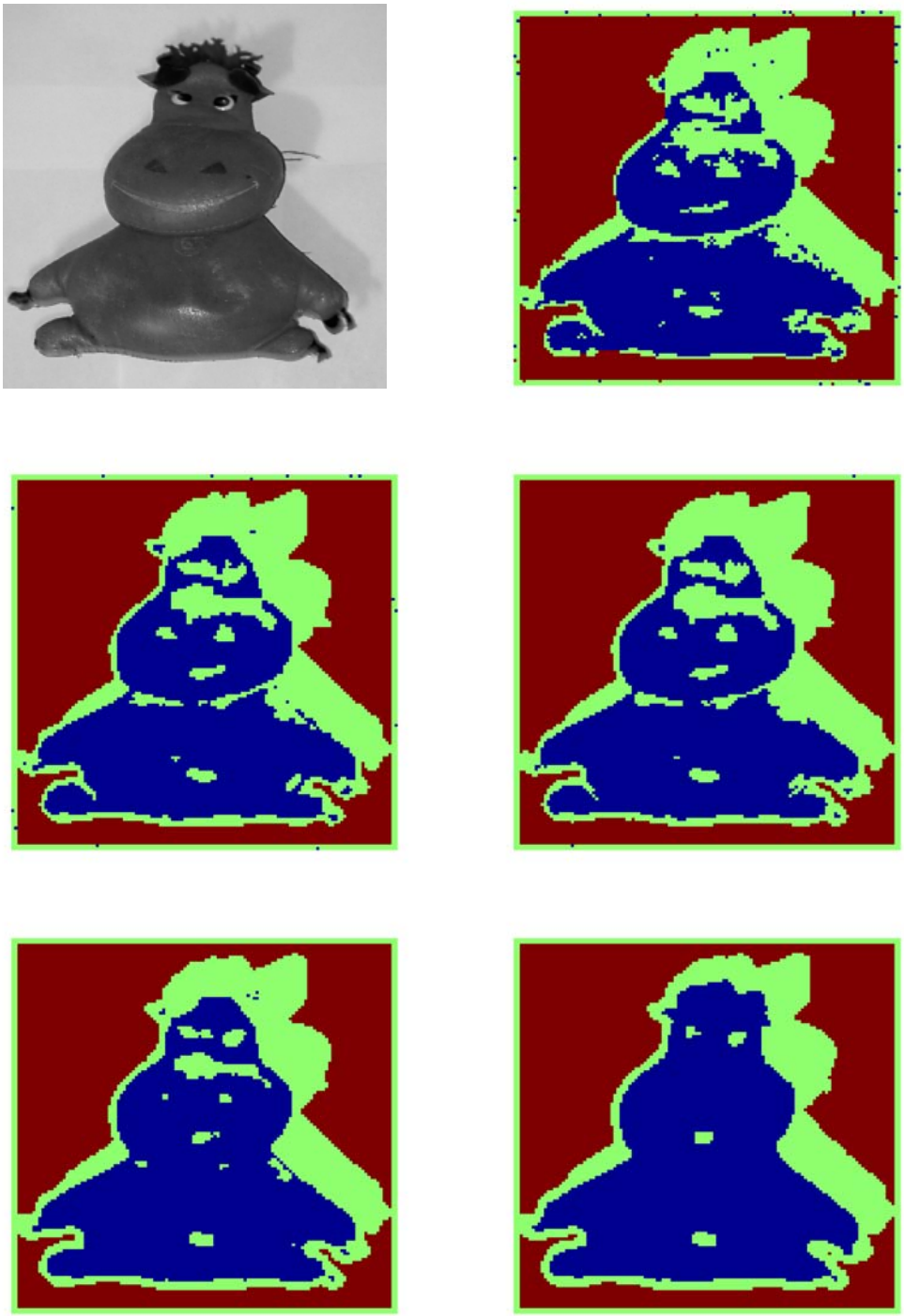


Figure A.5: Grayscale image segmentation. Segmentation results with 1st to 5th order neighborhood system (left to right and up to down; topmost left: Original image). Model/architecture settings: 3 classes, 4 equally sized and spaced bins, float-point precision and massively parallel processing dynamic. Grayscale channel: 8bit; value range [0,255].

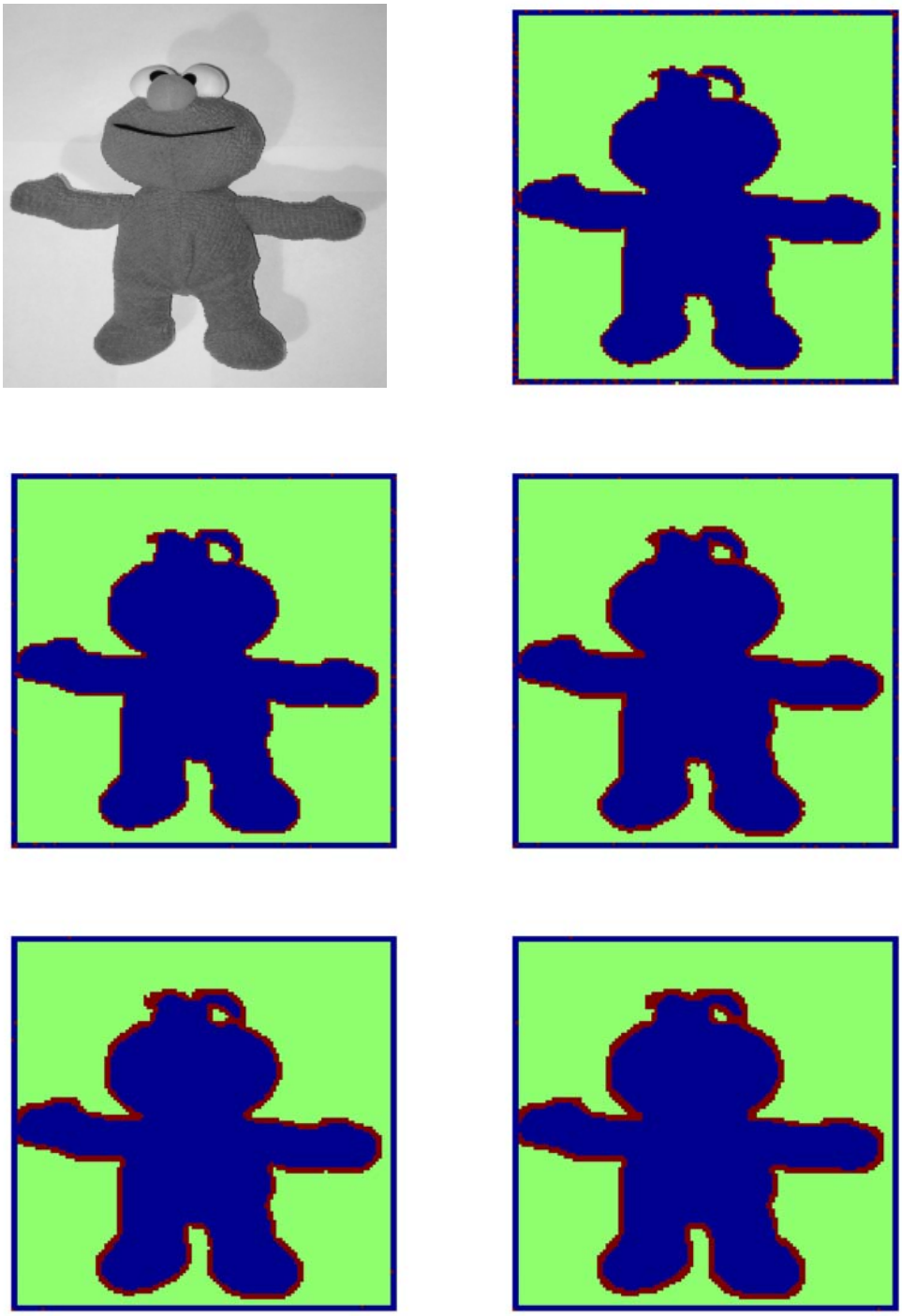


Figure A.6: Grayscale image segmentation. Segmentation results with 1st to 5th order neighborhood system (left to right and up to down; topmost left: Original image). Model/architecture settings: 3 classes, 3 equally sized and spaced bins, float-point precision and massively parallel processing dynamic. Grayscale channel: 8bit; value range [0,255].

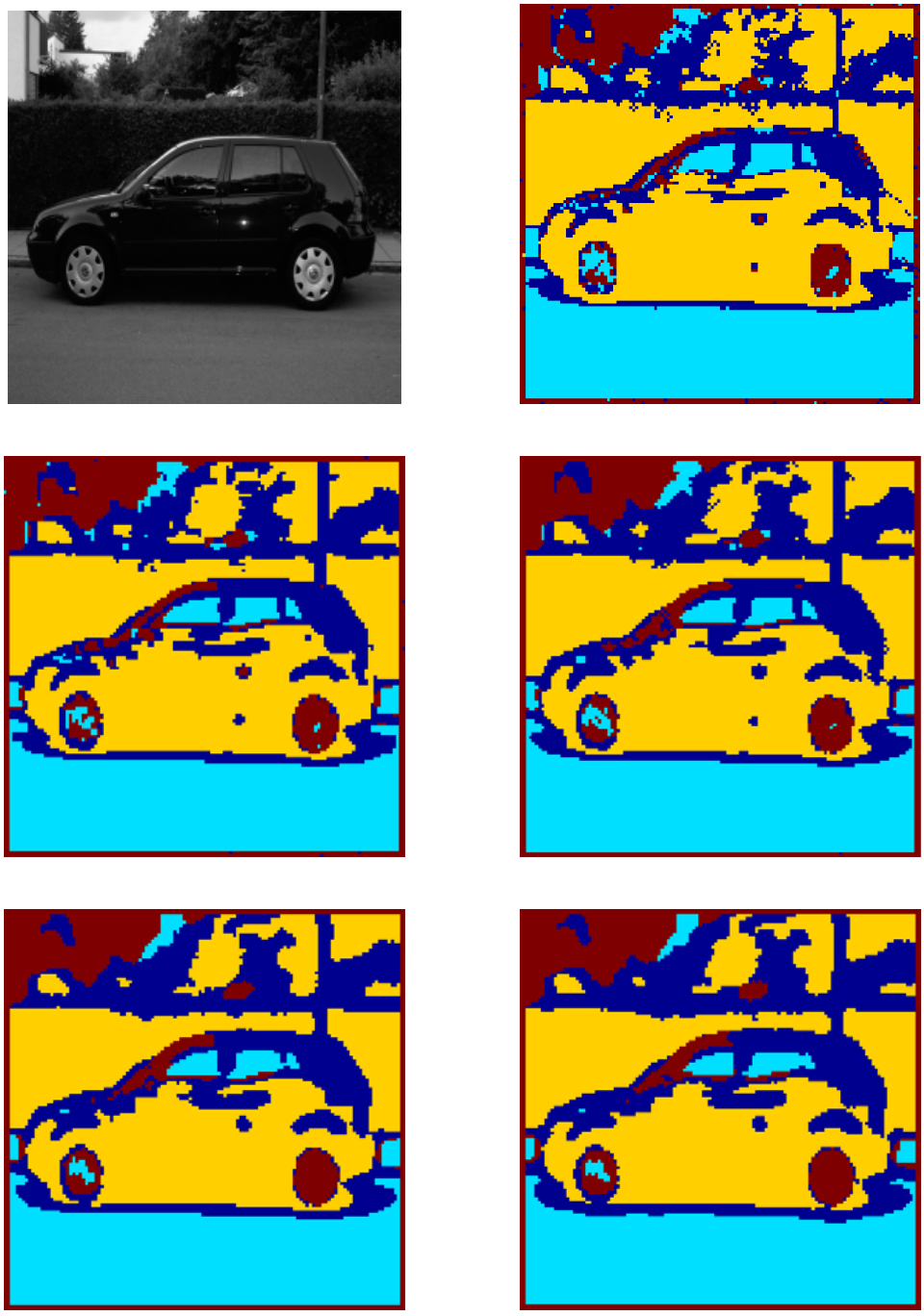


Figure A.7: Grayscale image segmentation. Segmentation results with 1st to 5th order neighborhood system (left to right and up to down; topmost left: Original image). Model/architecture settings: 4 classes, 8 equally sized and spaced bins, float-point precision and massively parallel processing dynamic. Grayscale channel: 8bit; value range [0,255].

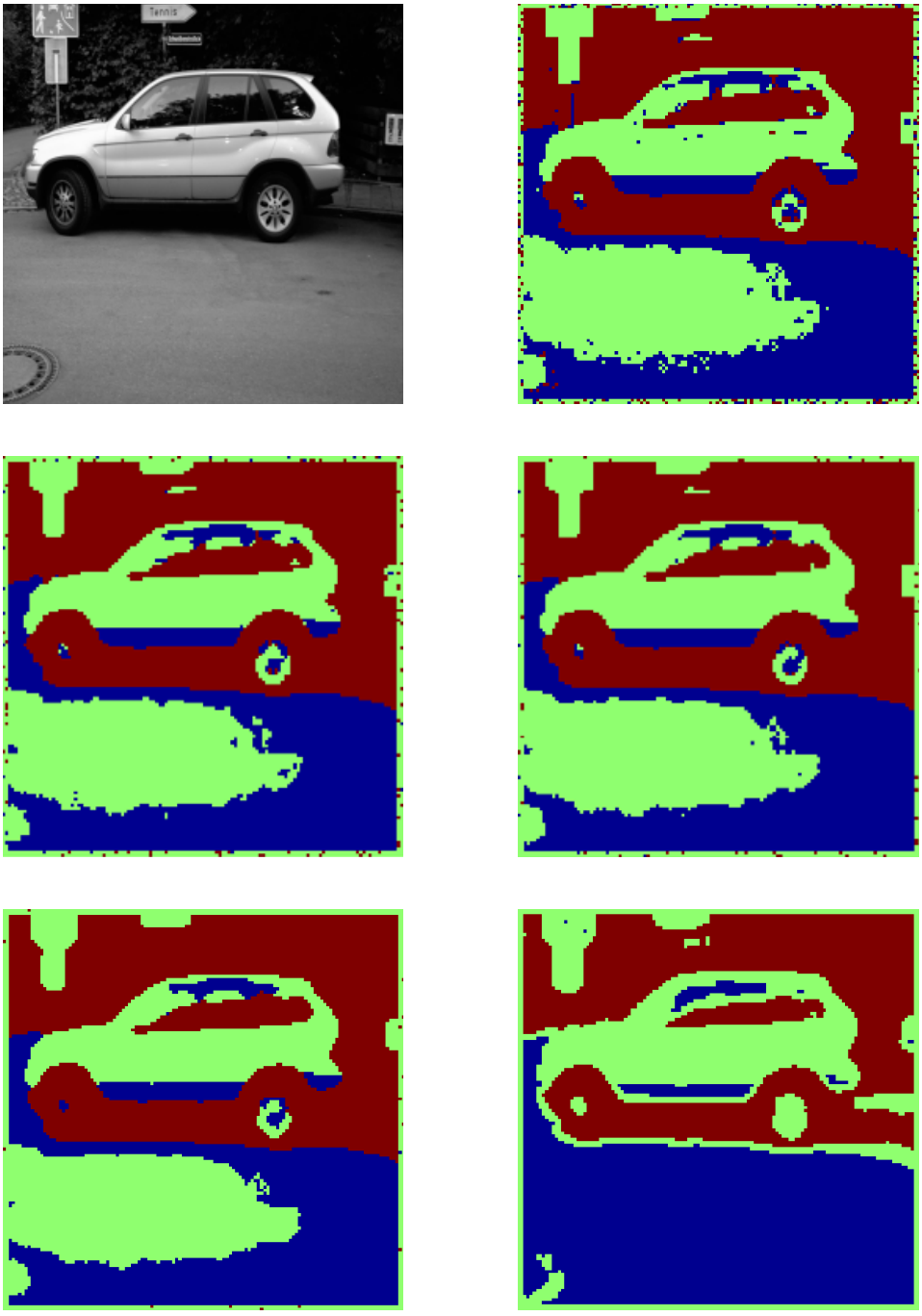


Figure A.8: Grayscale image segmentation. Segmentation results with 1st to 5th order neighborhood system (left to right and up to down; topmost left: Original image). Model/architecture settings: 3 classes, 6 equally sized and spaced bins, float-point precision and massively parallel processing dynamic. Grayscale channel: 8bit; value range [0,255].

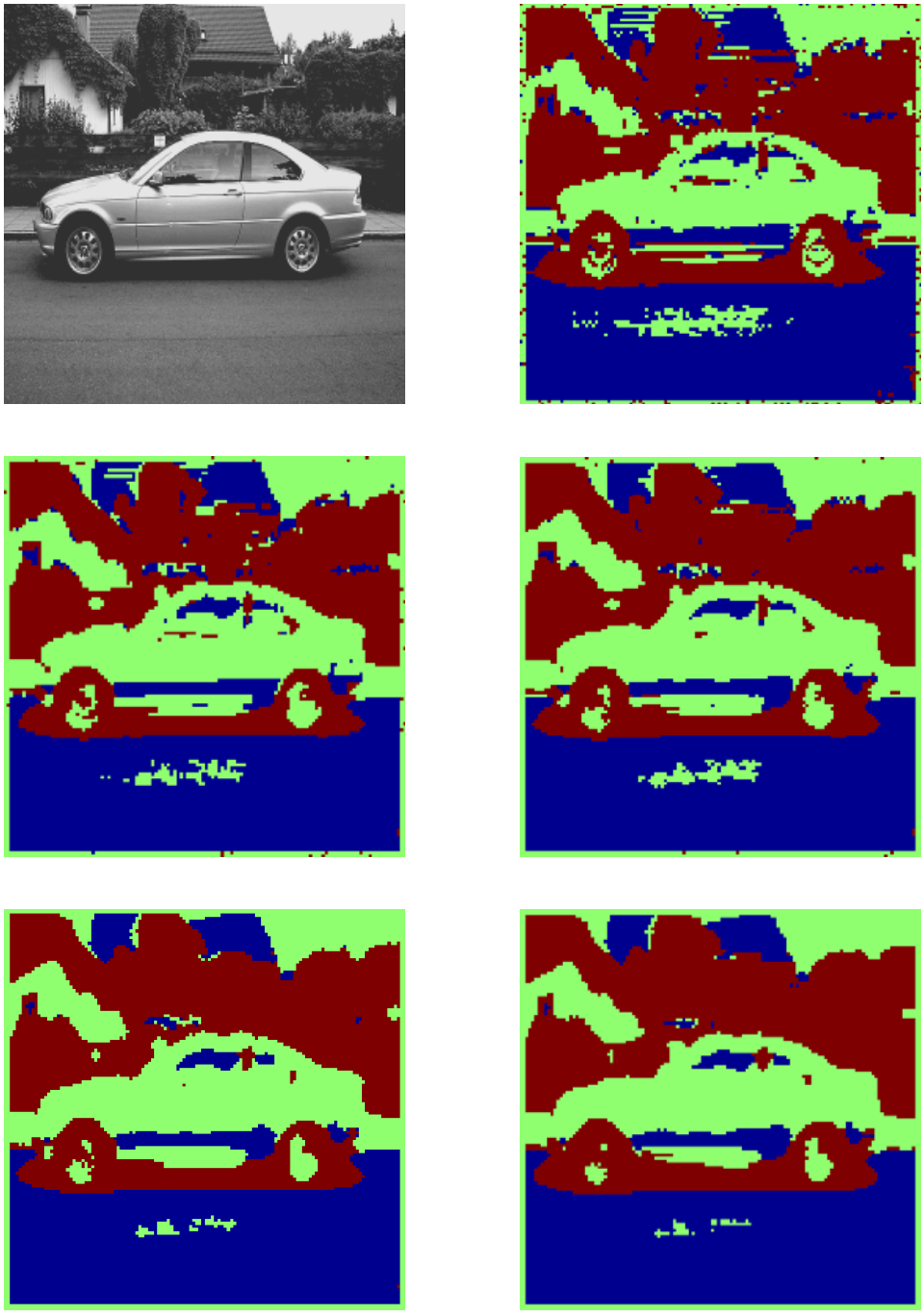


Figure A.9: Grayscale image segmentation. Segmentation results with 1st to 5th order neighborhood system (left to right and up to down; topmost left: Original image). Model/architecture settings: 3 classes, 6 equally sized and spaced bins, float-point precision and massively parallel processing dynamic. Grayscale channel: 8bit; value range [0,255].

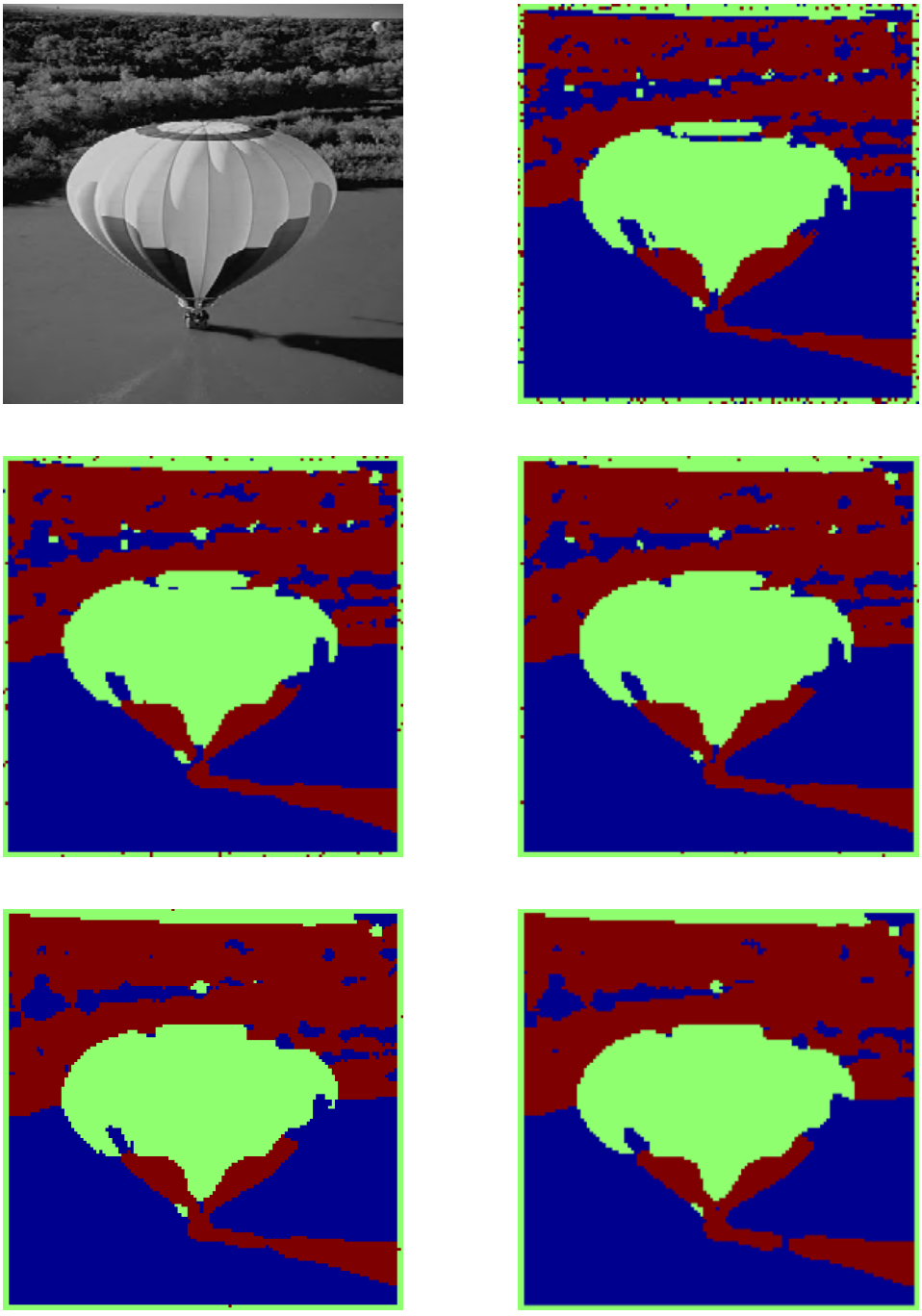


Figure A.10: Grayscale image segmentation. Segmentation results with 1st to 5th order neighborhood system (left to right and up to down; topmost left: Original image). Model/architecture settings: 3 classes, 4 equally sized and spaced bins, float-point precision and massively parallel processing dynamic. Grayscale channel: 8bit; value range [0,255].

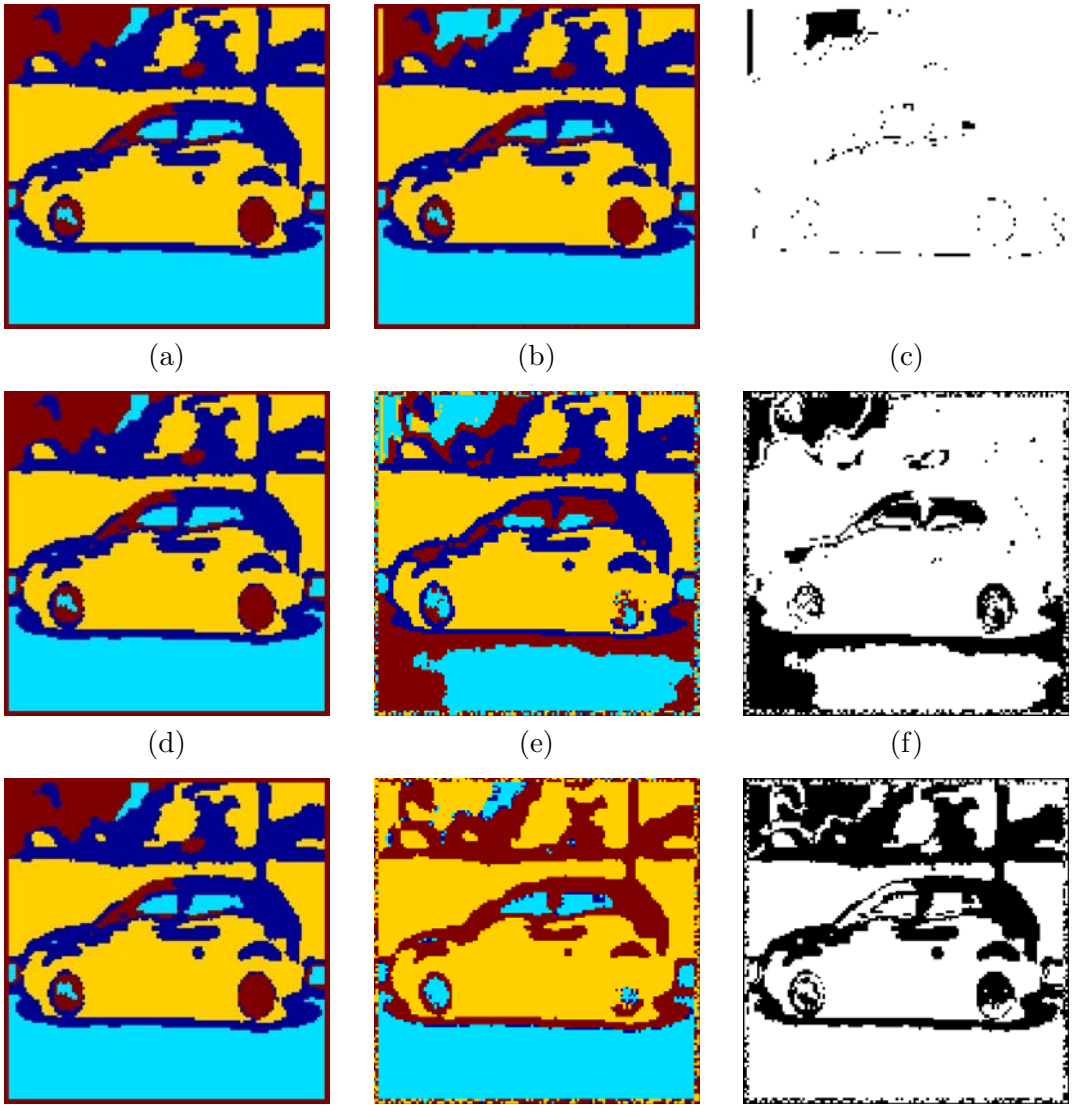


Figure A.11: Comparison of ground-truth with fixed-point model/architecture variants. (a) Ground-Truth. (b) 26bit model/architecture variant. (c) Difference (black) between (a) and (b) segmentation results. (d) Ground-Truth. (e) 20bit model/architecture variant. (f) Difference (black) between (d) and (e) segmentation results. (g) Ground-Truth. (h) 16bit model/architecture variant. (i) Difference (black) between (g) and (h) segmentation results. Ground-Truth (a),(d) and (g) are identical and only depicted for the purpose of illustration.

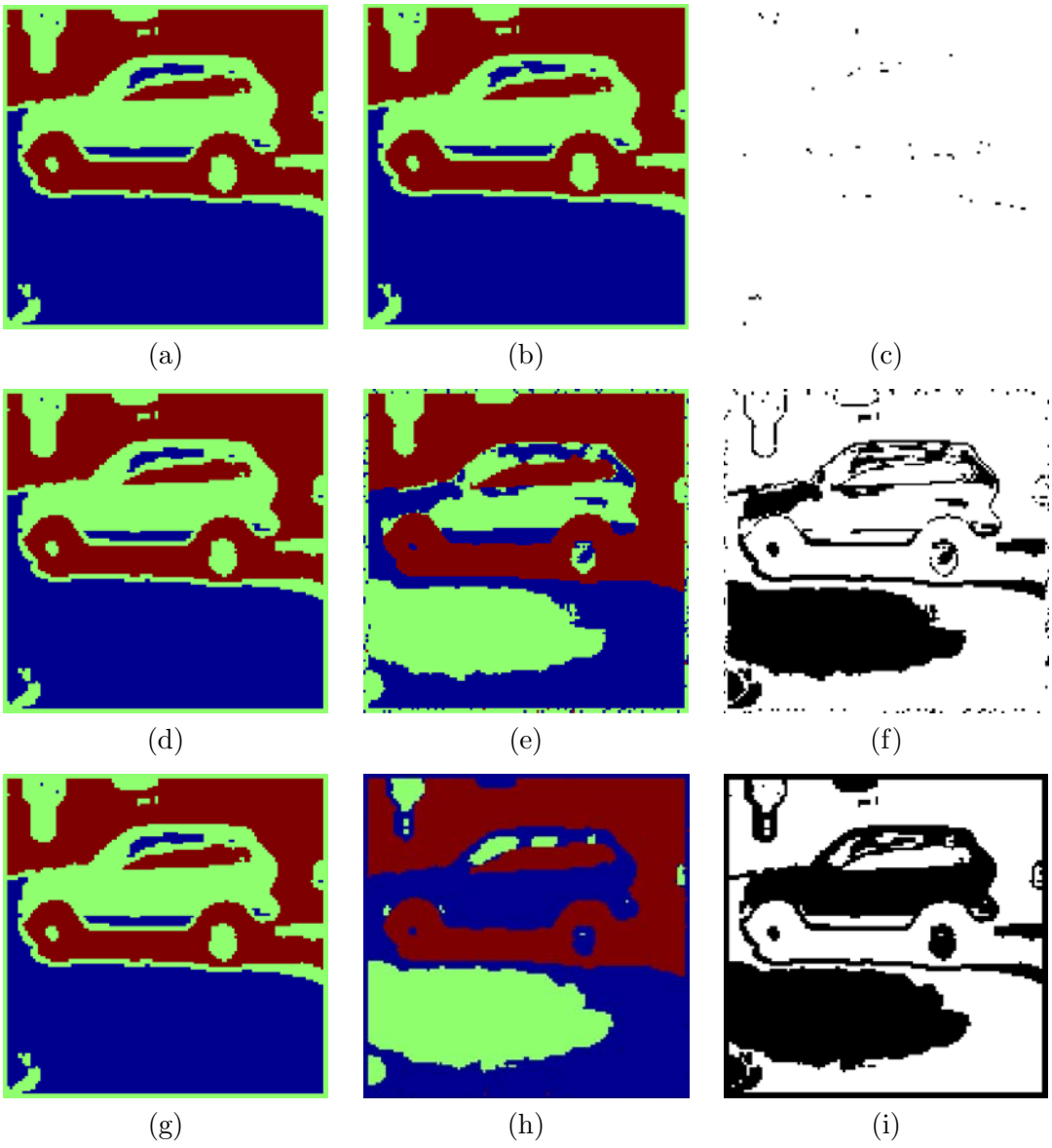


Figure A.12: Comparison of ground-truth with fixed-point model/architecture variants. (a) Ground-Truth. (b) 26bit model/architecture variant. (c) Difference (black) between (a) and (b) segmentation results. (d) Ground-Truth. (e) 20bit model/architecture variant. (f) Difference (black) between (d) and (e) segmentation results. (g) Ground-Truth. (h) 16bit model/architecture variant. (i) Difference (black) between (g) and (h) segmentation results. Ground-Truth (a),(d) and (g) are identical and only depicted for the purpose of illustration.

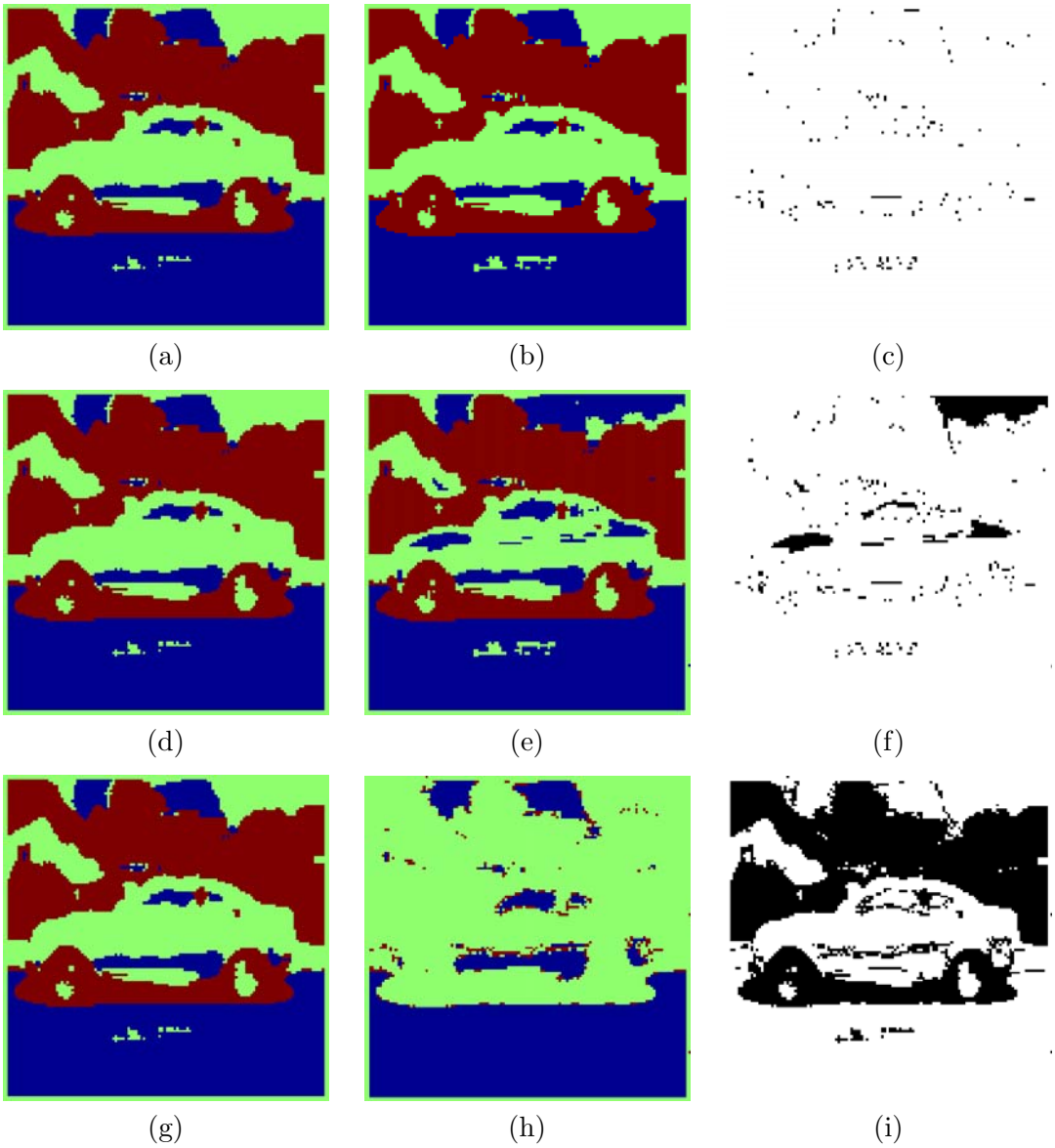


Figure A.13: Comparison of ground-truth with fixed-point model/architecture variants. (a) Ground-Truth. (b) 26bit model/architecture variant. (c) Difference (black) between (a) and (b) segmentation results. (d) Ground-Truth. (e) 20bit model/architecture variant. (f) Difference (black) between (d) and (e) segmentation results. (g) Ground-Truth. (h) 16bit model/architecture variant. (i) Difference (black) between (g) and (h) segmentation results. Ground-Truth (a),(d) and (g) are identical and only depicted for the purpose of illustration.

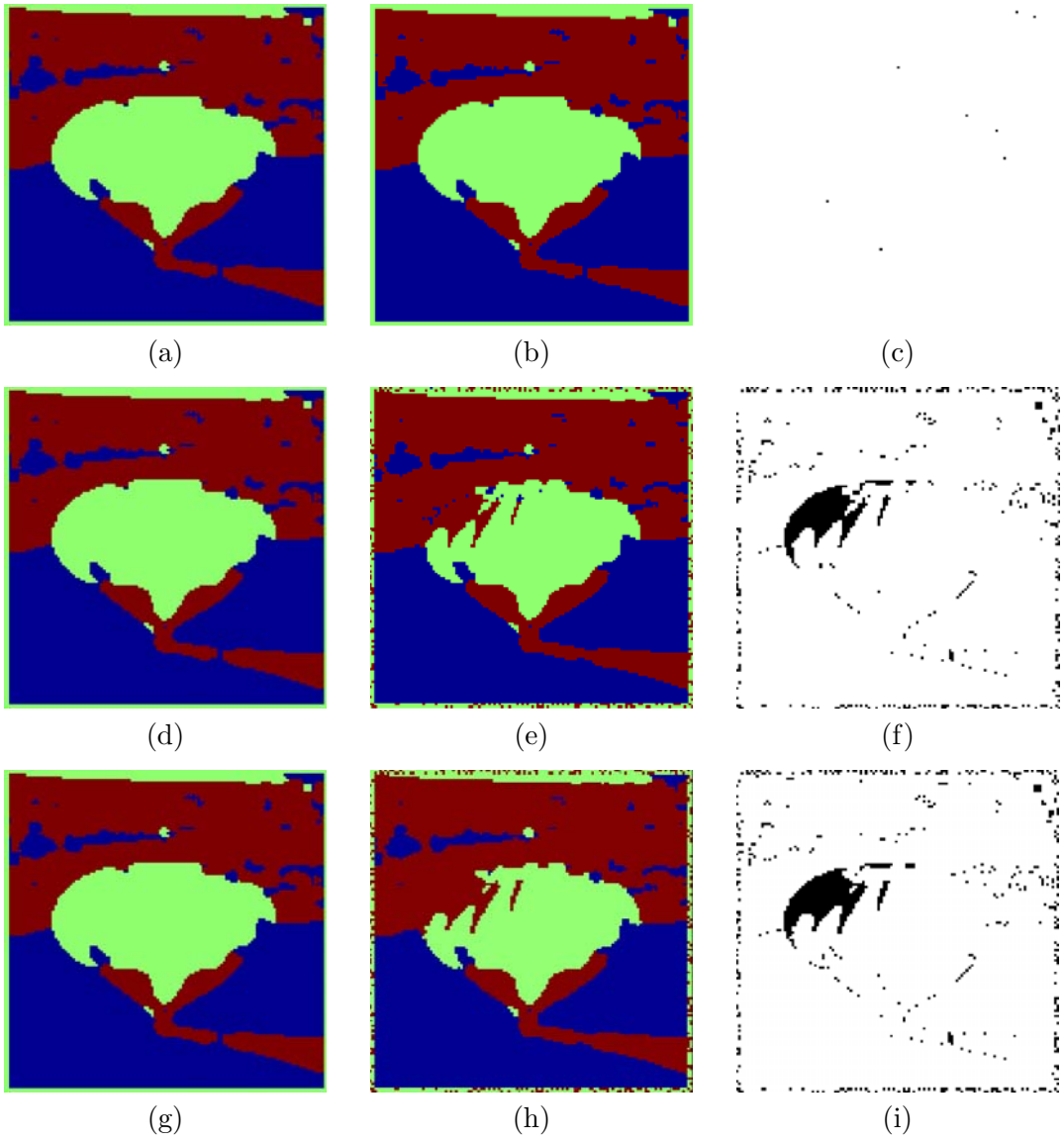


Figure A.14: Comparison of ground-truth with fixed-point model/architecture variants. (a) Ground-Truth. (b) 26bit model/architecture variant. (c) Difference (black) between (a) and (b) segmentation results. (d) Ground-Truth. (e) 20bit model/architecture variant. (f) Difference (black) between (d) and (e) segmentation results. (g) Ground-Truth. (h) 16bit model/architecture variant. (i) Difference (black) between (g) and (h) segmentation results. Ground-Truth (a),(d) and (g) are identical and only depicted for the purpose of illustration.

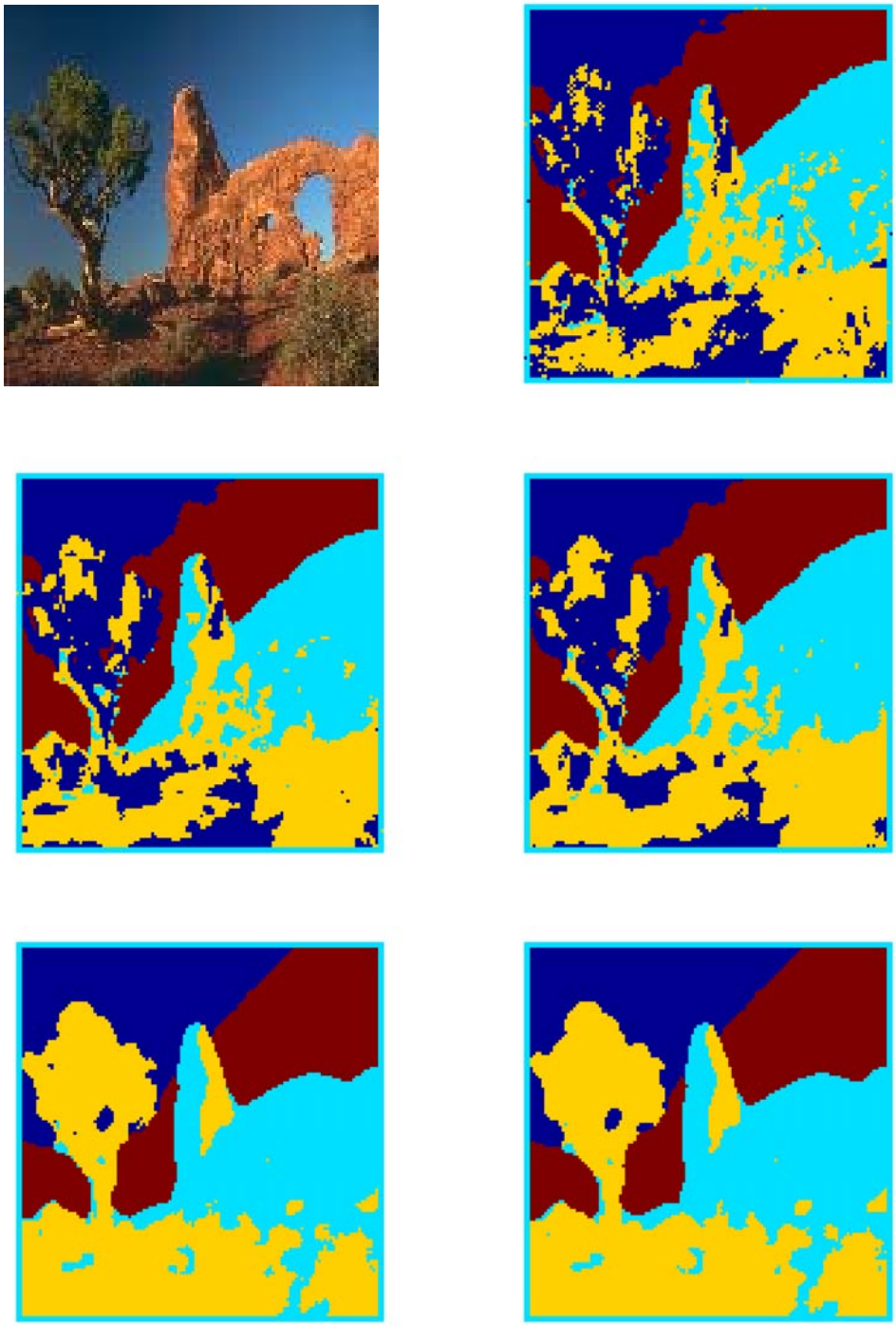


Figure A.15: Color image segmentation. Segmentation results with 1st to 5th order neighborhood system (left to right and up to down; topmost left: Original image). Model/architecture settings: 4 classes and 8 equally sized and spaced bins for each of the RGB channels. Float-point precision and massively parallel processing dynamic. RGB channels: each 8bit; value range [0,255].

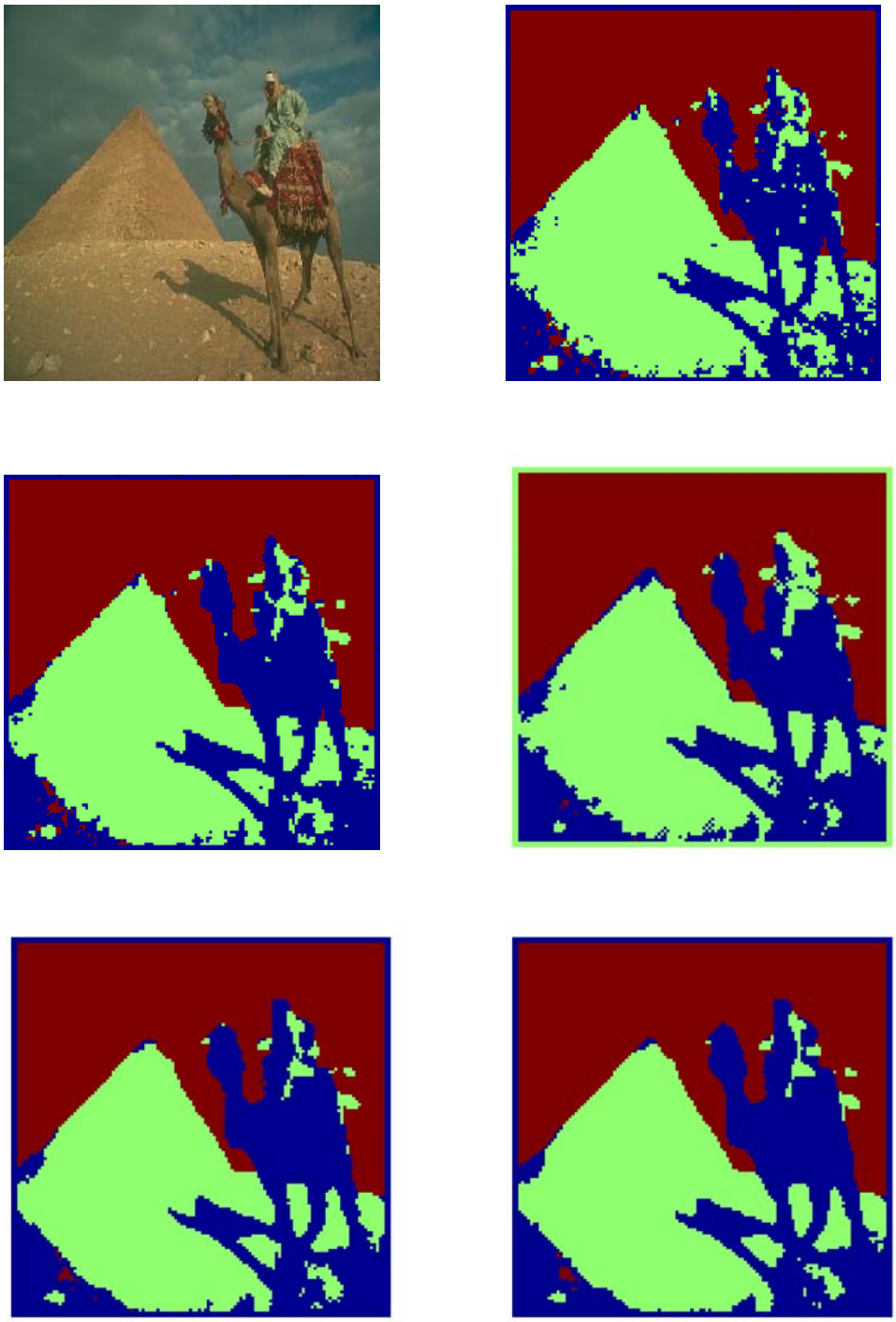


Figure A.16: Color image segmentation. Segmentation results with 1st to 5th order neighborhood system (left to right and up to down; topmost left: Original image). Model/architecture settings: 3 classes and 4 equally sized and spaced bins for each of the RGB channels. Float-point precision and massively parallel processing dynamic. RGB channels: each 8bit; value range [0,255].

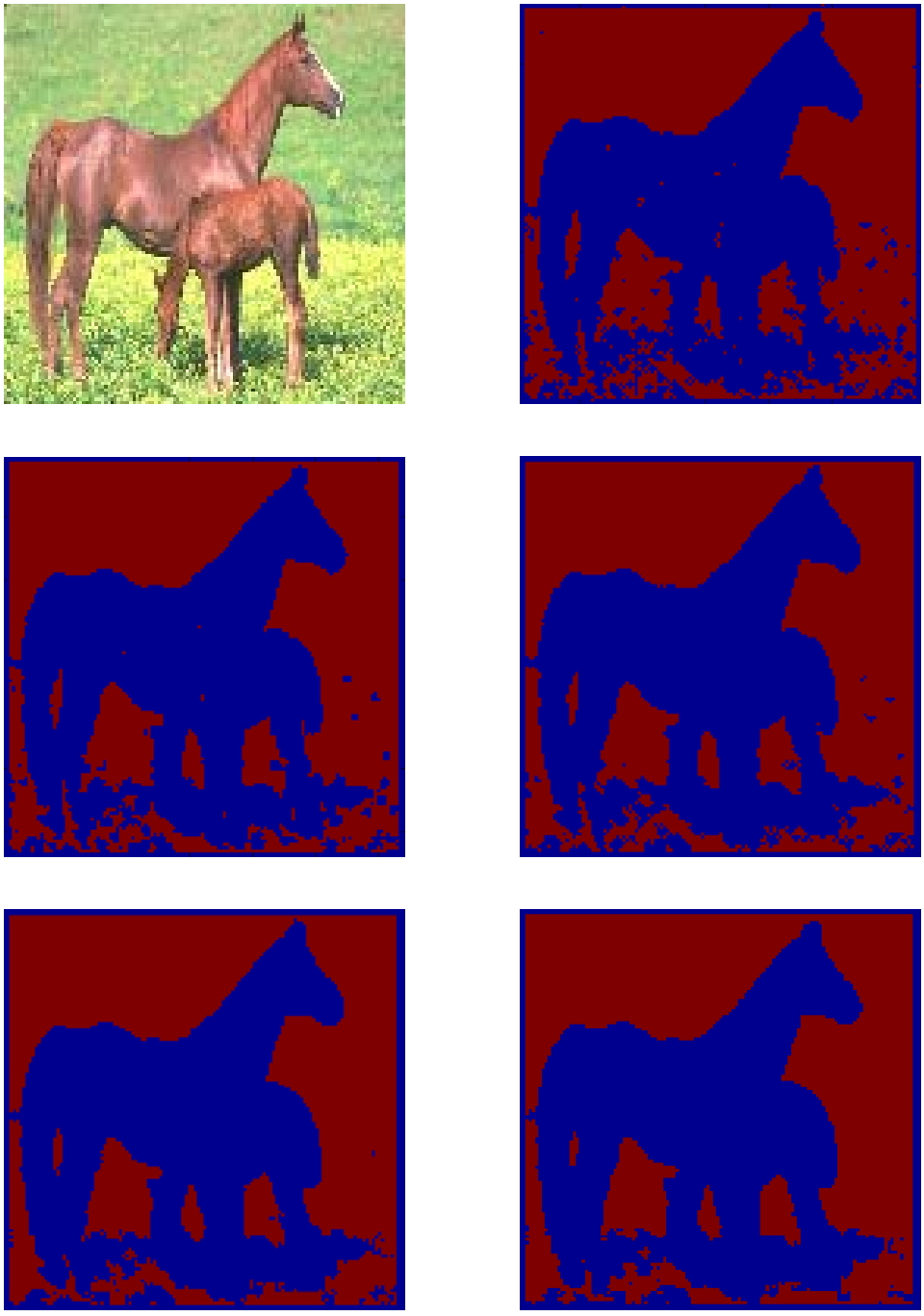


Figure A.17: Color image segmentation. Segmentation results with 1st to 5th order neighborhood system (left to right and up to down; topmost left: Original image). Model/architecture settings: 2 classes and 5 equally sized and spaced bins for each of the RGB channels. Float-point precision and massively parallel processing dynamic. RGB channels: each 8bit; value range [0,255].

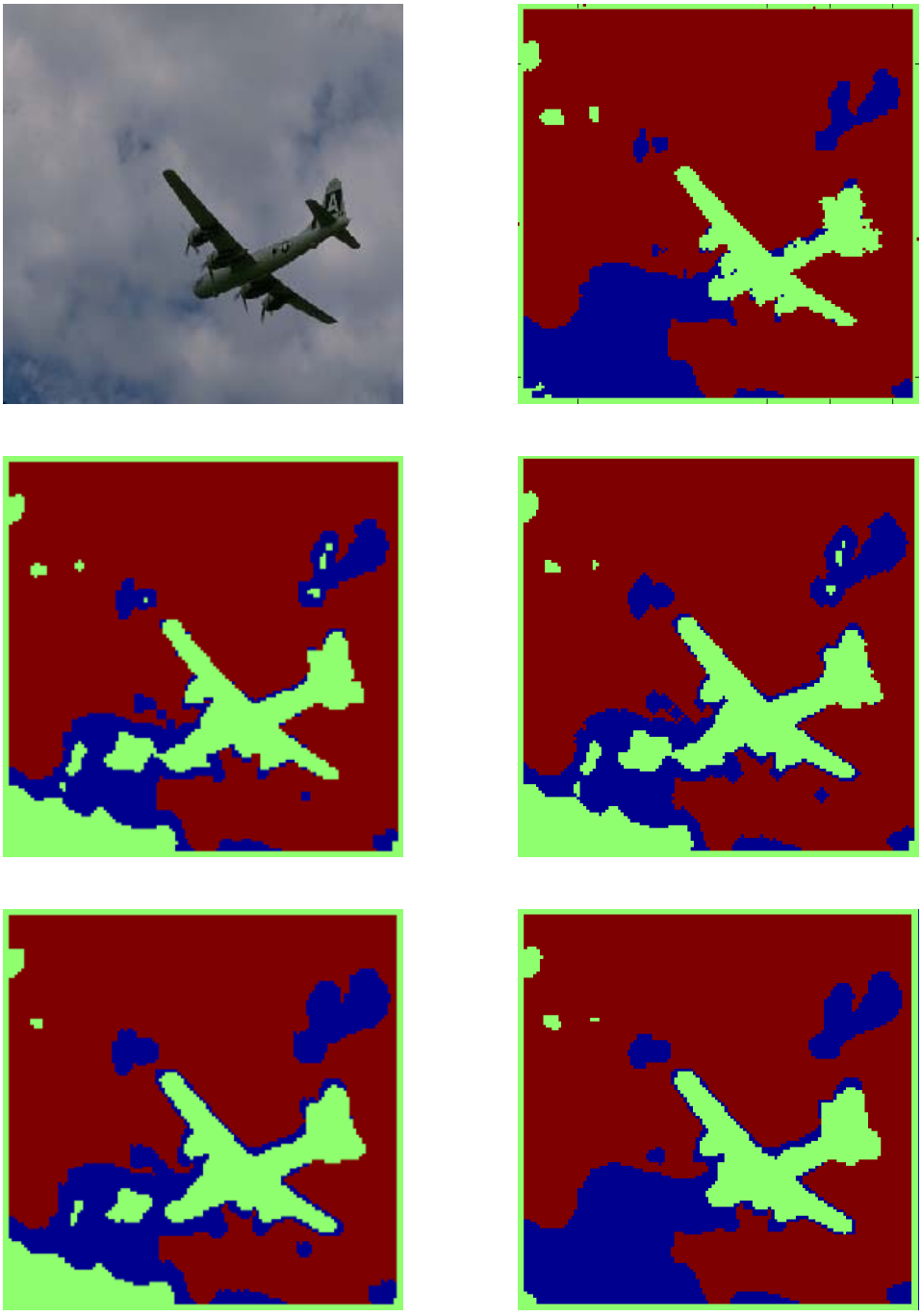
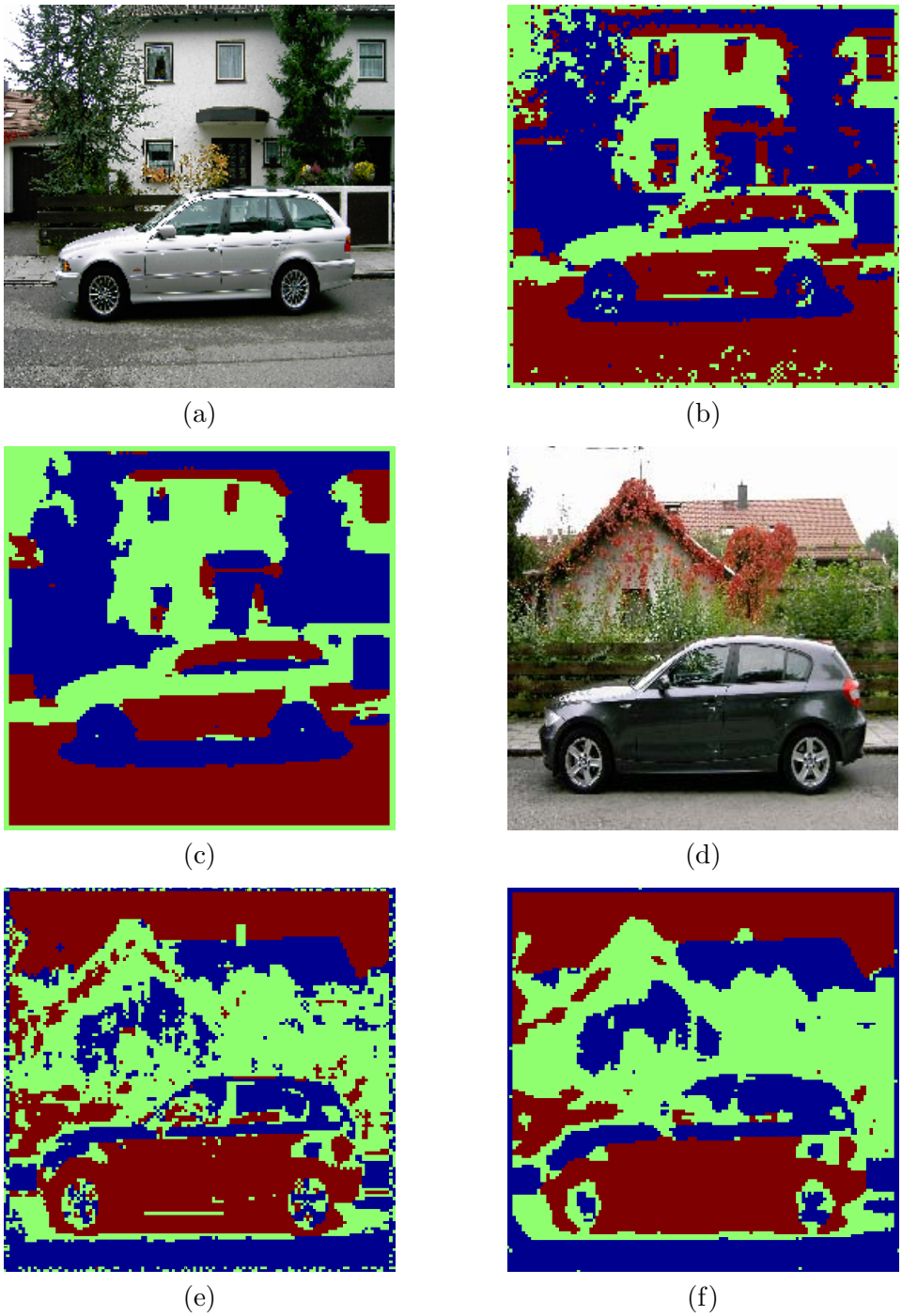


Figure A.18: Color image segmentation. Segmentation results with 1st to 5th order neighborhood system (left to right and up to down; topmost left: Original image). Model/architecture settings: 3 classes and 5 equally sized and spaced bins for each of the RGB channels. Float-point precision and massively parallel processing dynamic. RGB channels: each 8bit; value range [0,255].



(a) (b) (c) (d) (e) (f)

Figure A.19: Color image segmentation. Segmentation results with 1st and 5th order neighborhood system. (a) and (d) Original image. (b) and (e) 1st neighborhood system. (c) and (f) 5th neighborhood system. Model/architecture settings for (a)-(c): 3 classes and 5 equally sized and spaced bins for each of the RGB channels. Model/architecture settings for (d)-(f): 3 classes and 10 equally sized and spaced bins for each of the RGB channels. RGB channels: each 8bit; value range [0,255]. Float-point precision and massively parallel processing dynamic.

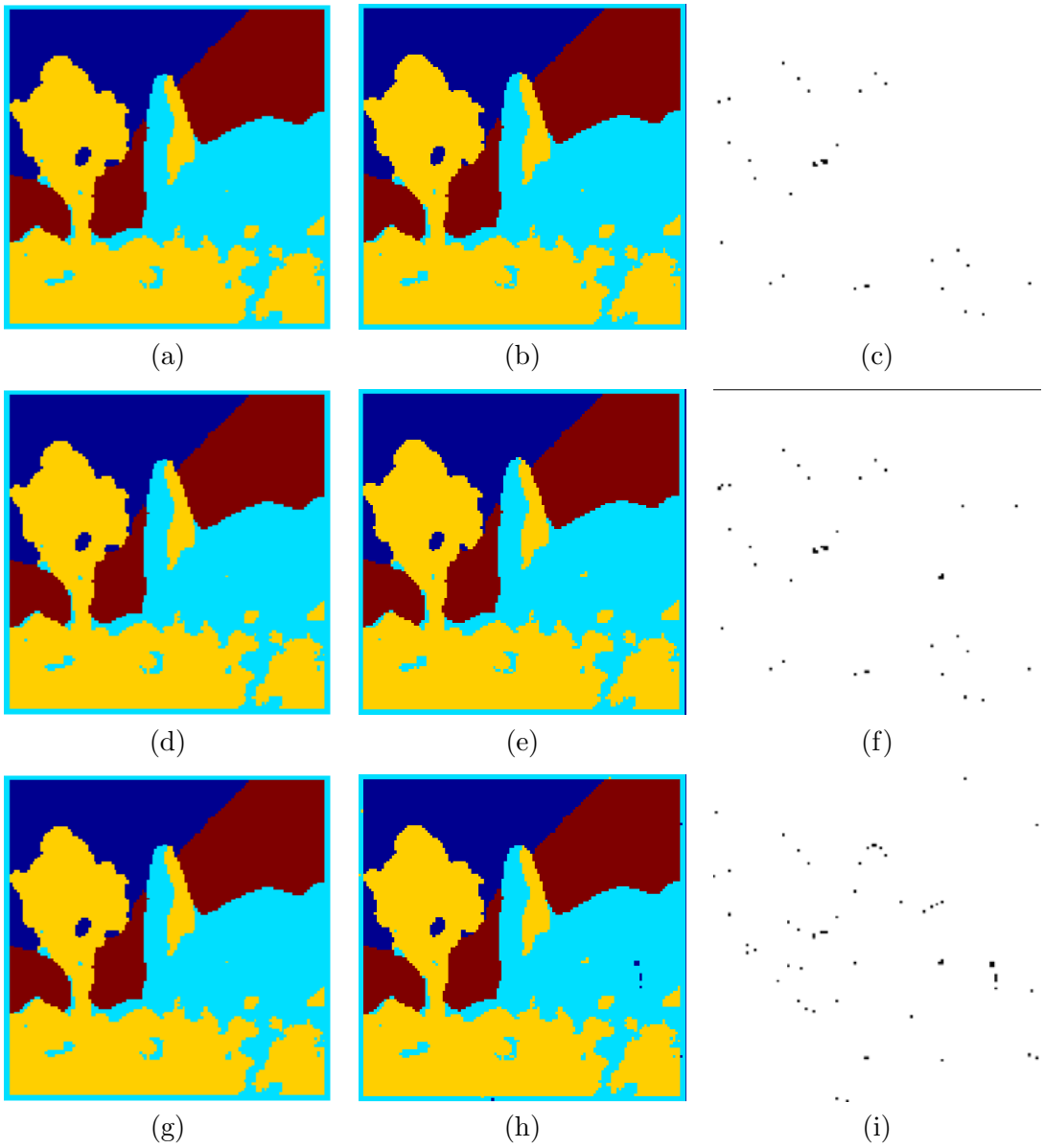


Figure A.20: Comparison of ground-truth with fixed-point model/architecture variants. (a) Ground-Truth. (b) 30bit model/architecture variant. (c) Difference (black) between (a) and (b) segmentation results. (d) Ground-Truth. (e) 26bit model/architecture variant. (f) Difference (black) between (d) and (e) segmentation results. (g) Ground-Truth. (h) 22bit model/architecture variant. (i) Difference (black) between (g) and (h) segmentation results. Ground-Truth (a),(d) and (g) are identical and only depicted for the purpose of illustration.

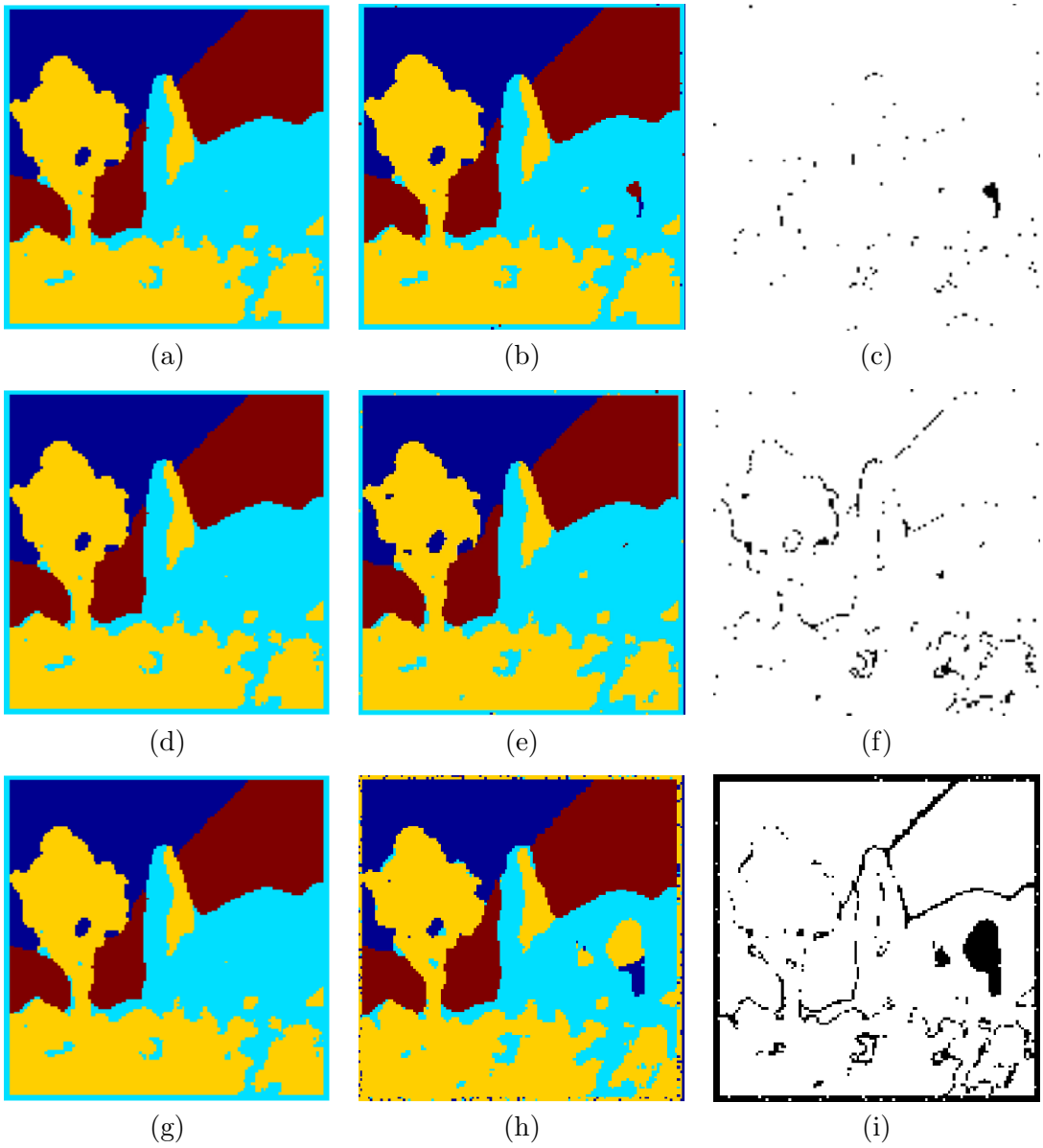


Figure A.21: Comparison of ground-truth with fixed-point model/architecture variants. (a) Ground-Truth. (b) 20bit model/architecture variant. (c) Difference (black) between (a) and (b) segmentation results. (d) Ground-Truth. (e) 18bit model/architecture variant. (f) Difference (black) between (d) and (e) segmentation results. (g) Ground-Truth. (h) 14bit model/architecture variant. (i) Difference (black) between (g) and (h) segmentation results. Ground-Truth (a),(d) and (g) are identical and only depicted for the purpose of illustration.

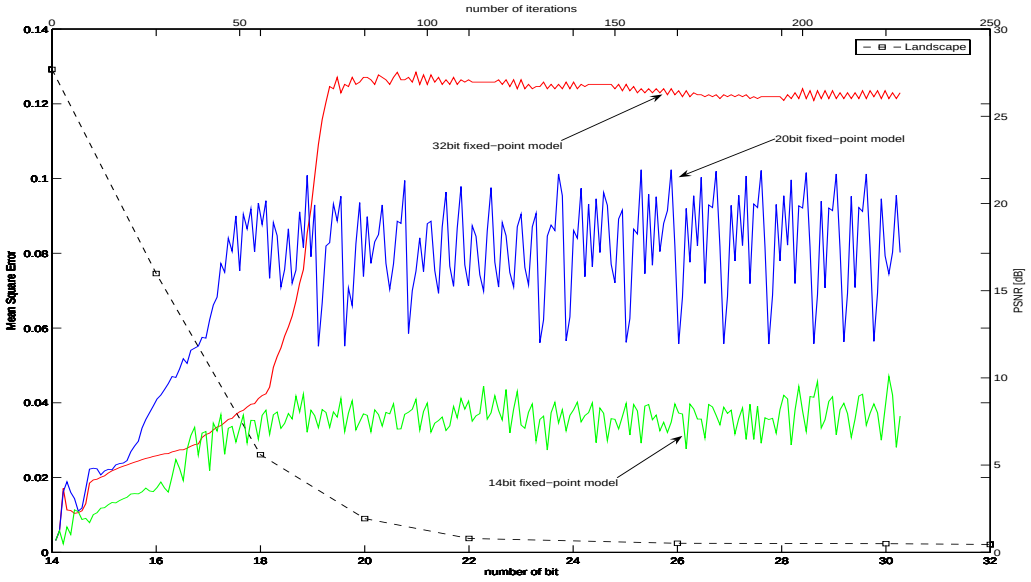


Figure A.22: Means Square Error (MSE) and Peak signal-to-noise ratio (PSNR) analysis of simulation runs (landscape image, cf. Figure A.20, A.21) of the segmentation model. MSE versus number of bit used for the model. PSNR versus number of iterations.

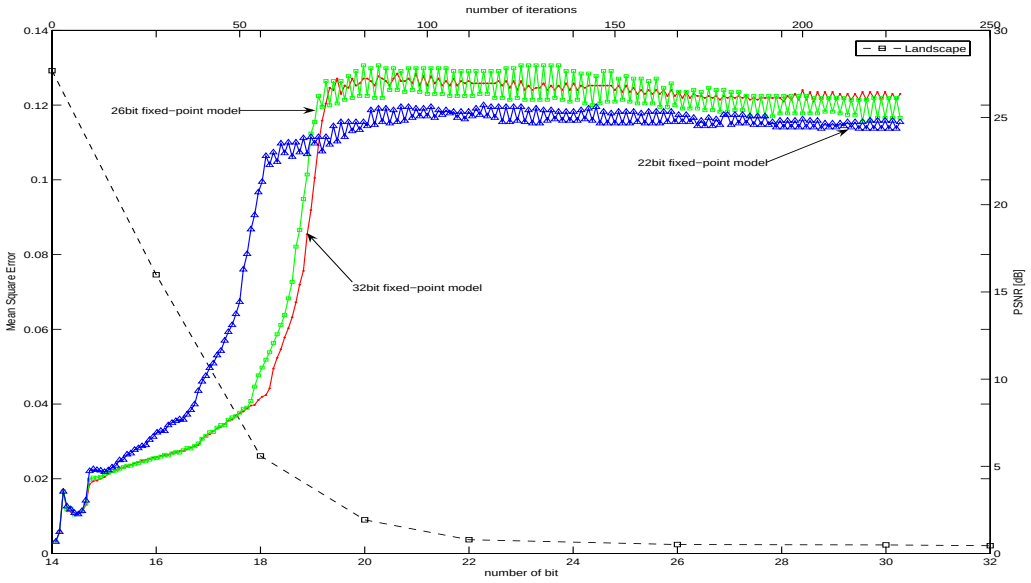


Figure A.23: Means Square Error (MSE) and Peak signal-to-noise ratio (PSNR) analysis of simulation runs (landscape image, cf. Figure A.20, A.21) of the segmentation model. MSE versus number of bit used for the model. PSNR versus number of iterations.

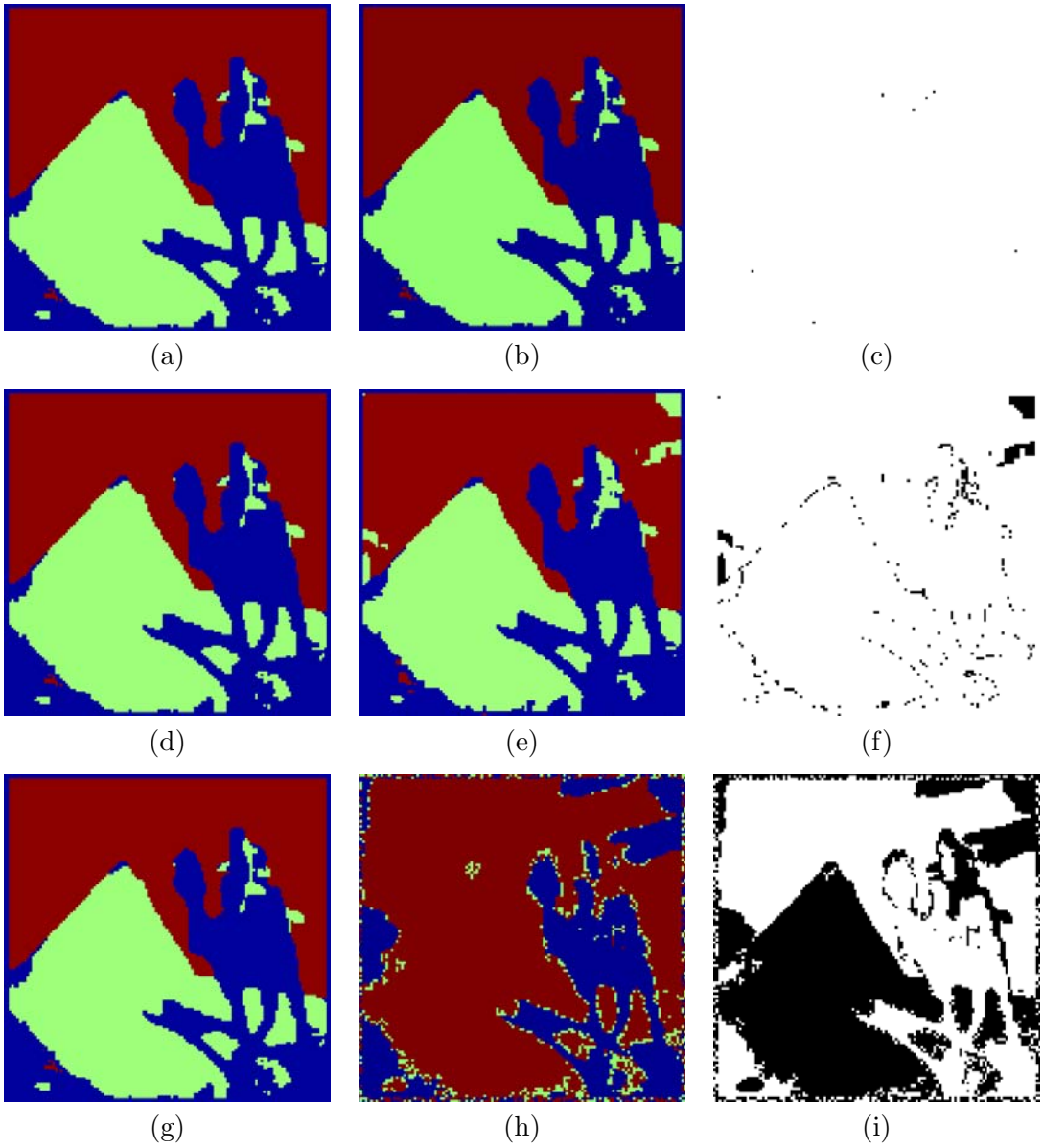


Figure A.24: Comparison of ground-truth with fixed-point model/architecture variants. (a) Ground-Truth. (b) 26bit model/architecture variant. (c) Difference (black) between (a) and (b) segmentation results. (d) Ground-Truth. (e) 20bit model/architecture variant. (f) Difference (black) between (d) and (e) segmentation results. (g) Ground-Truth. (h) 16bit model/architecture variant. (i) Difference (black) between (g) and (h) segmentation results. Ground-Truth (a),(d) and (g) are identical and only depicted for the purpose of illustration.

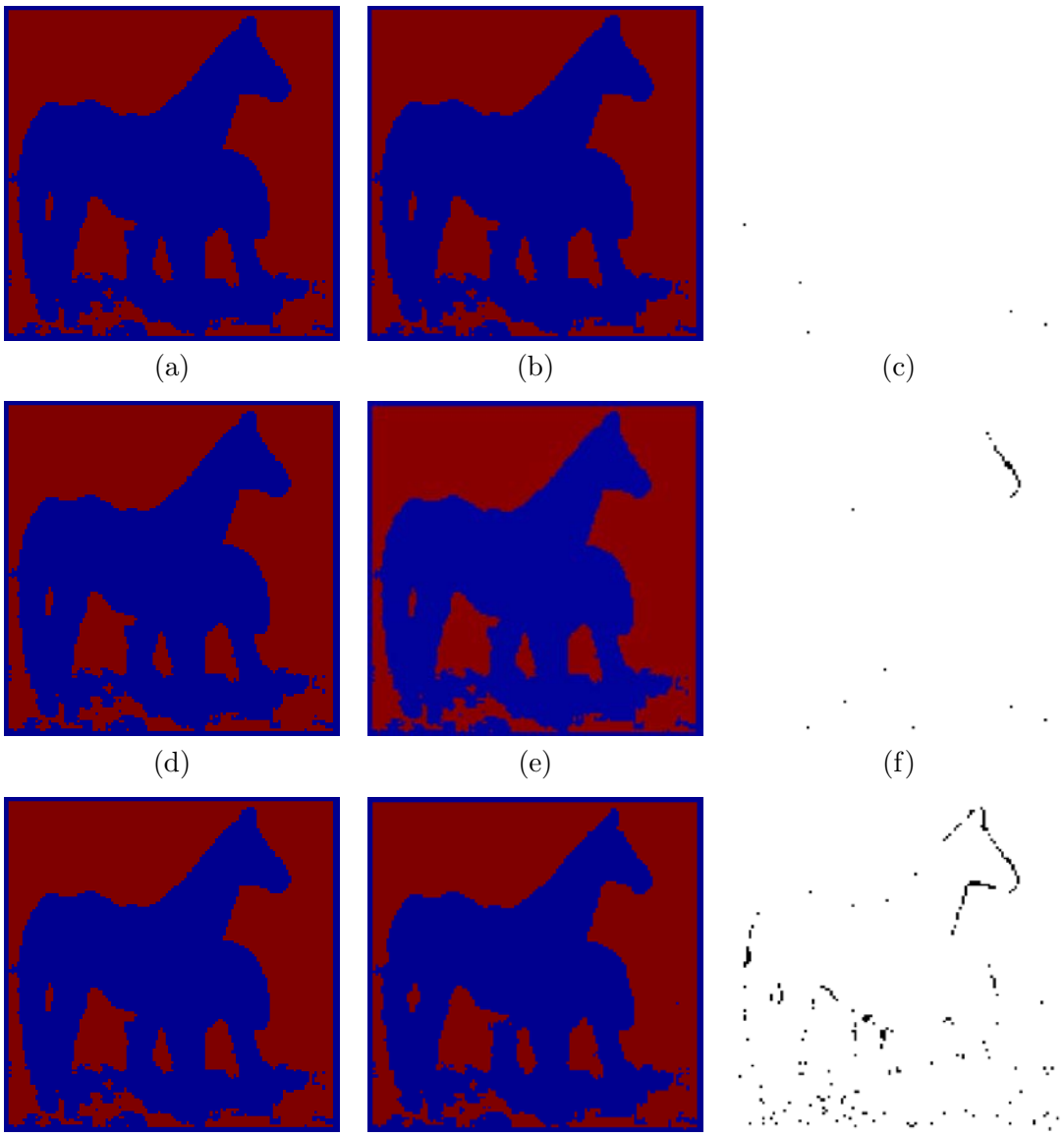


Figure A.25: Comparison of ground-truth with fixed-point model/architecture variants. (a) Ground-Truth. (b) 26bit model/architecture variant. (c) Difference (black) between (a) and (b) segmentation results. (d) Ground-Truth. (e) 20bit model/architecture variant. (f) Difference (black) between (d) and (e) segmentation results. (g) Ground-Truth. (h) 16bit model/architecture variant. (i) Difference (black) between (g) and (h) segmentation results. Ground-Truth (a),(d) and (g) are identical and only depicted for the purpose of illustration.

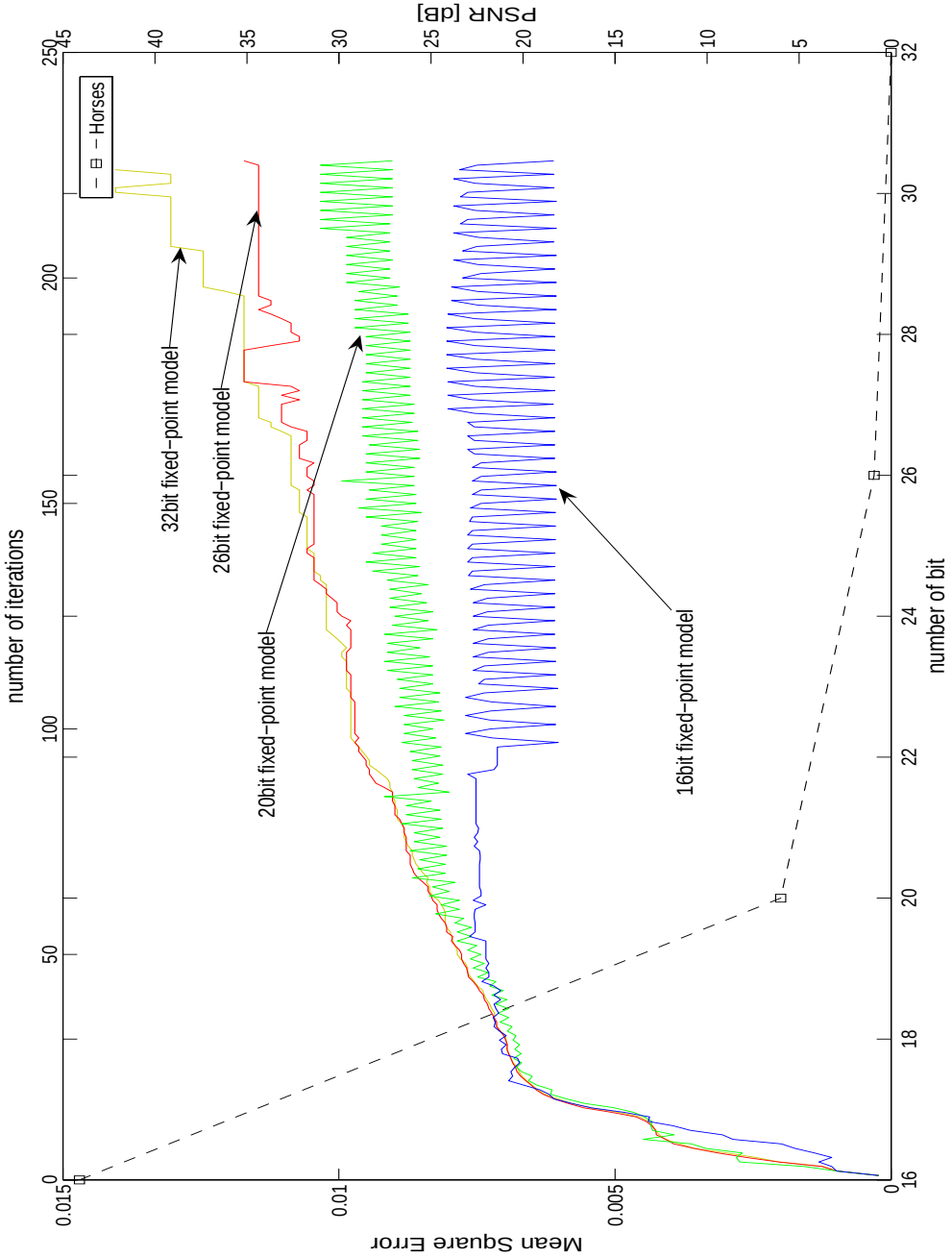


Figure A.26: Means Square Error (MSE) and Peak signal-to-noise ratio (PSNR) analysis of fixed-point simulation runs (horses, cf. Figure A.25) of the segmentation model. MSE versus number of bit used for the model. PSNR versus number of iterations.

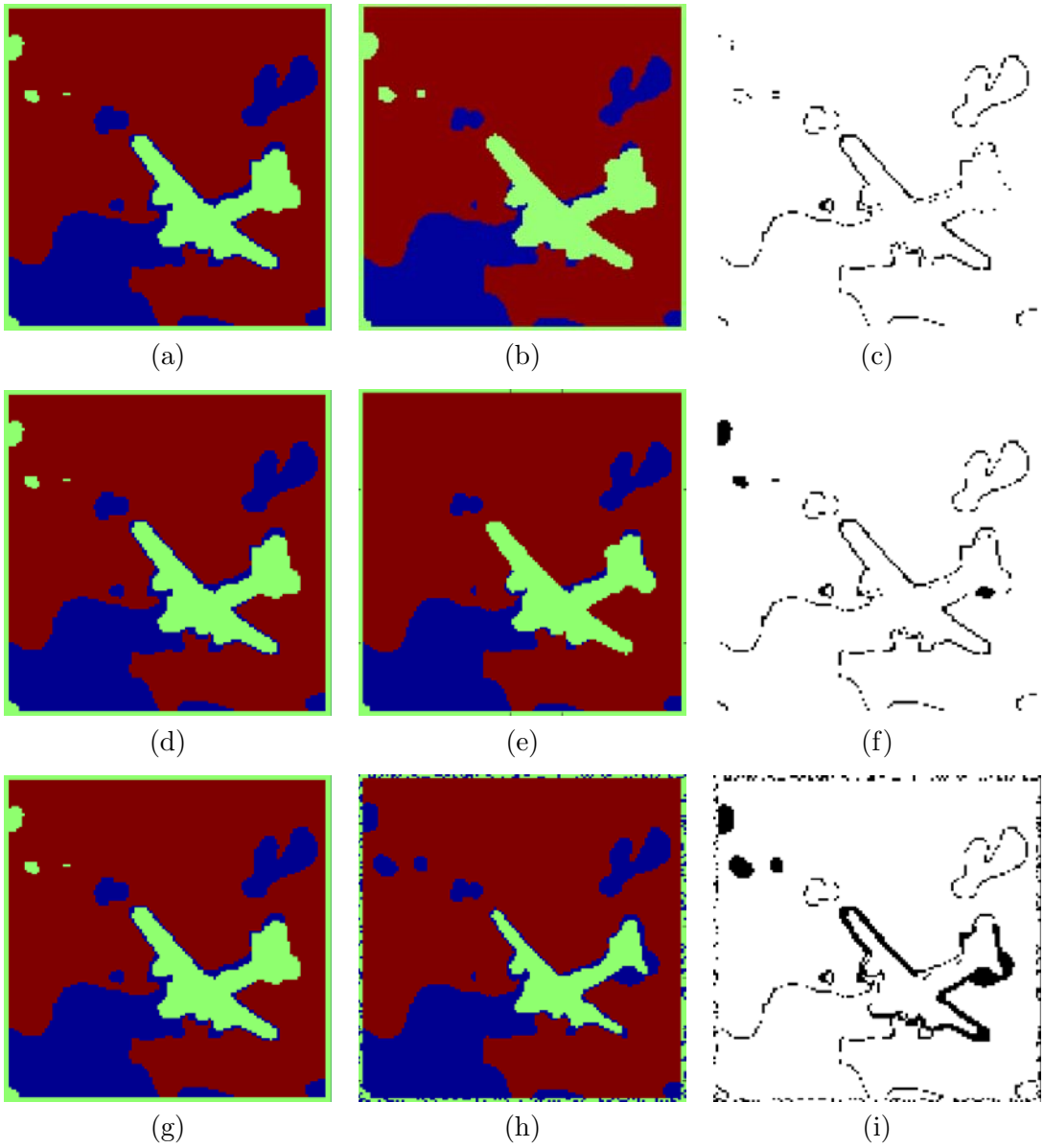


Figure A.27: Comparison of ground-truth with fixed-point model/architecture variants. (a) Ground-Truth. (b) 26bit model/architecture variant. (c) Difference (black) between (a) and (b) segmentation results. (d) Ground-Truth. (e) 20bit model/architecture variant. (f) Difference (black) between (d) and (e) segmentation results. (g) Ground-Truth. (h) 16bit model/architecture variant. (i) Difference (black) between (g) and (h) segmentation results. Ground-Truth (a),(d) and (g) are identical and only depicted for the purpose of illustration.

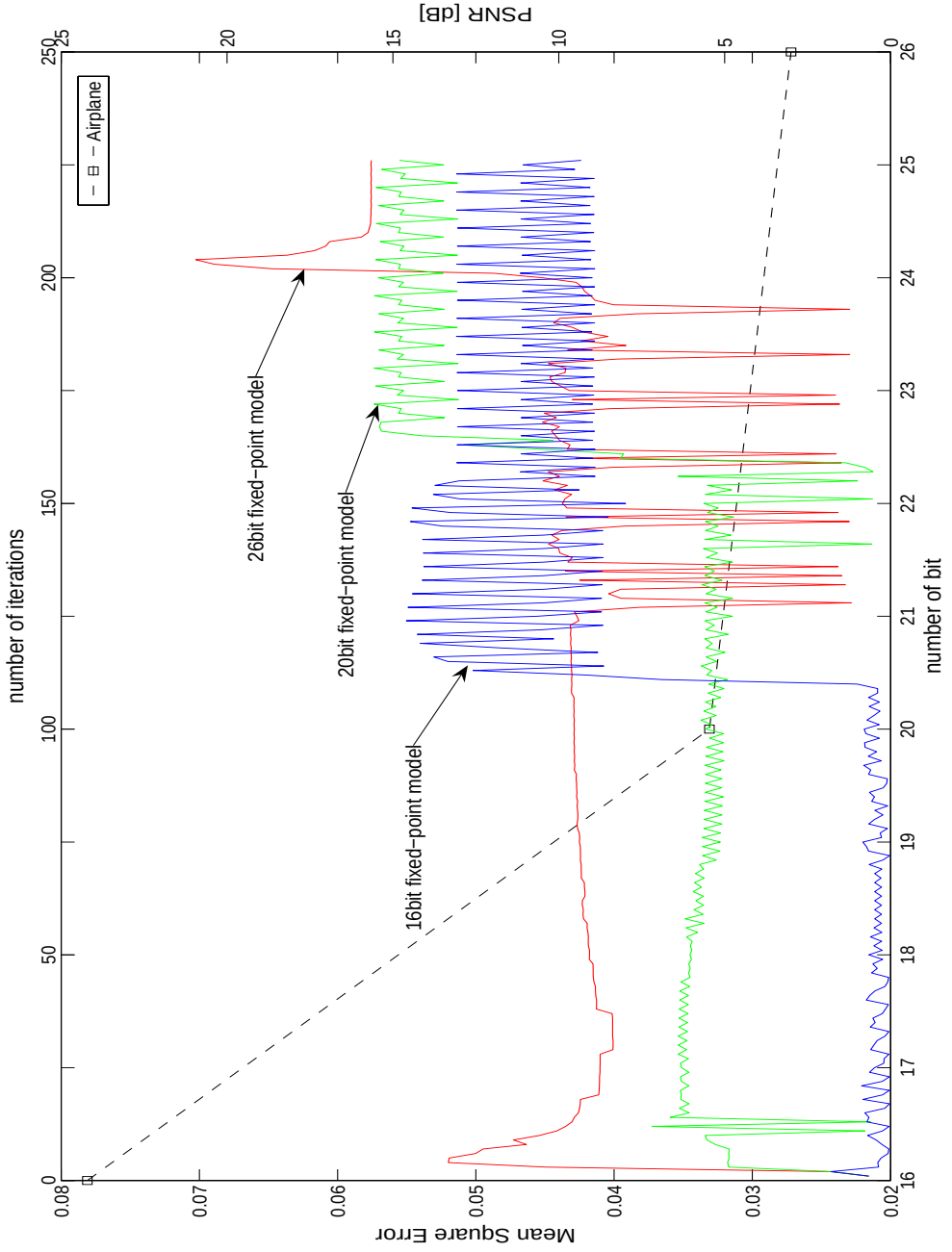


Figure A.28: Means Square Error (MSE) and Peak signal-to-noise ratio (PSNR) analysis of fixed-point simulation runs (airplane image, cf. Figure A.27) of the segmentation model. MSE versus number of bit used for the model. PSNR versus number of iterations.

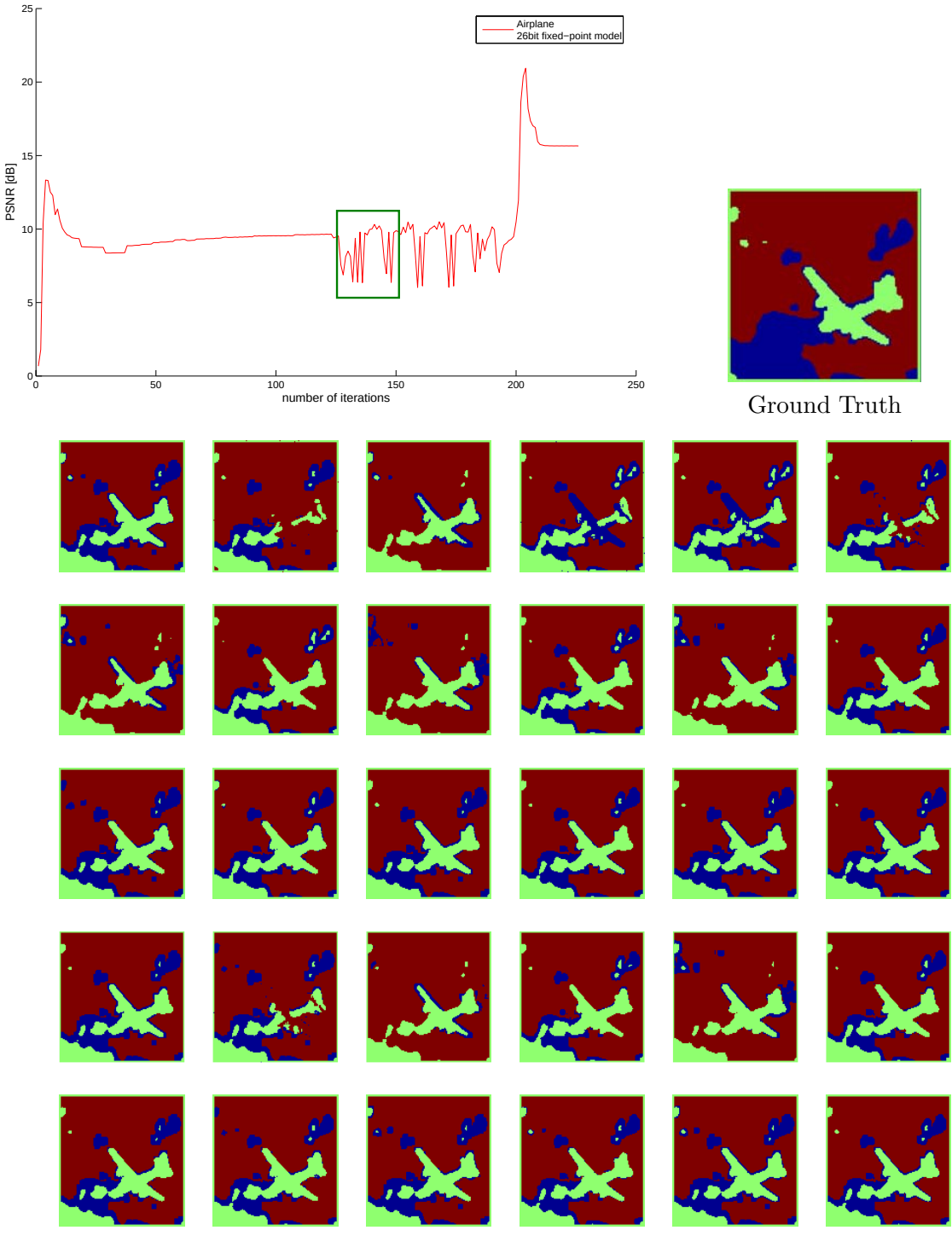


Figure A.29: Peak signal-to-noise ratio (PSNR versus number of iterations) analysis of 26bit fixed-point model. PSNR for iteration steps 125-154 (green box) and the corresponding segmentation results shown beneath. Iteration steps 125-154: Image sequence from left to right and top to bottom. PSNR with respect to binary difference data.

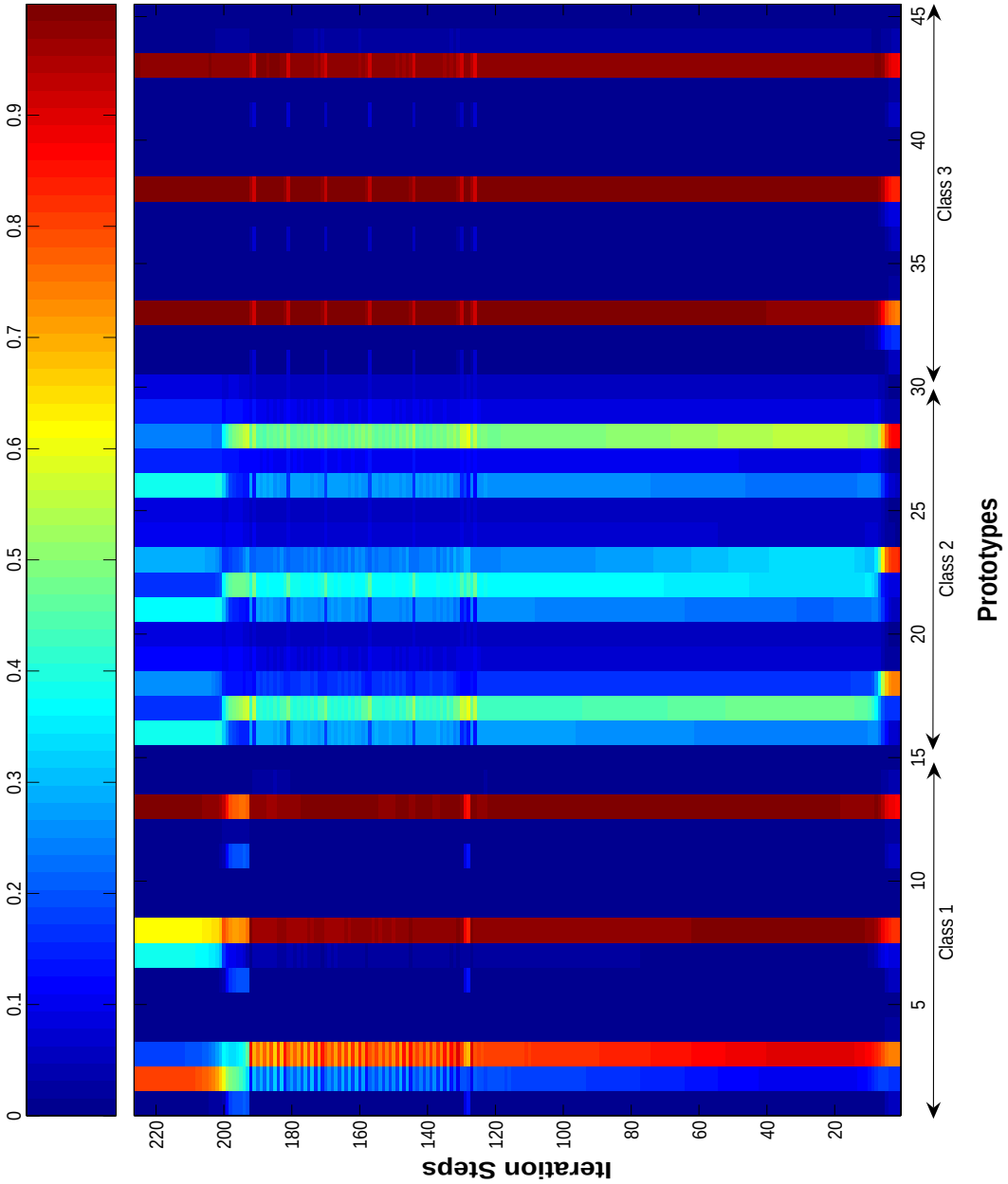


Figure A.30: Estimated prototypes q_ν. Prototypes of the architecture/model variant with 26bit fixed point precision and with respect to the airplane image (rotate to view). Each class prototype is composed of probability distributions for the RGB channels. The probability values are encoded corresponding to the colorbar.

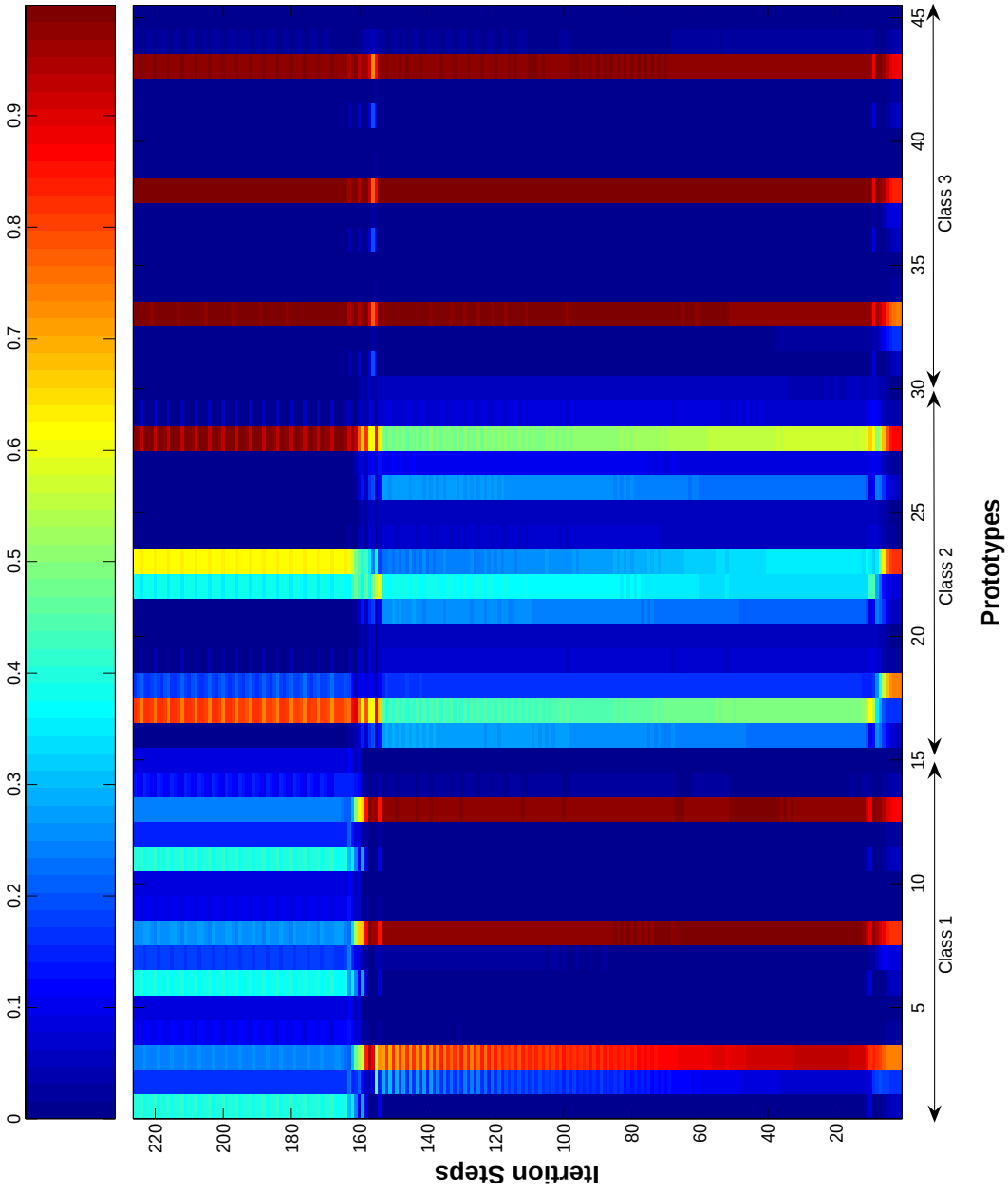


Figure A.31: Estimated prototypes q_ν. Prototypes of the architecture/model variant with 20bit fixed point precision and with respect to the airplane image (rotate to view). Each class prototype is composed of probability distributions for the RGB channels. The probability values are encoded corresponding to the colorbar.

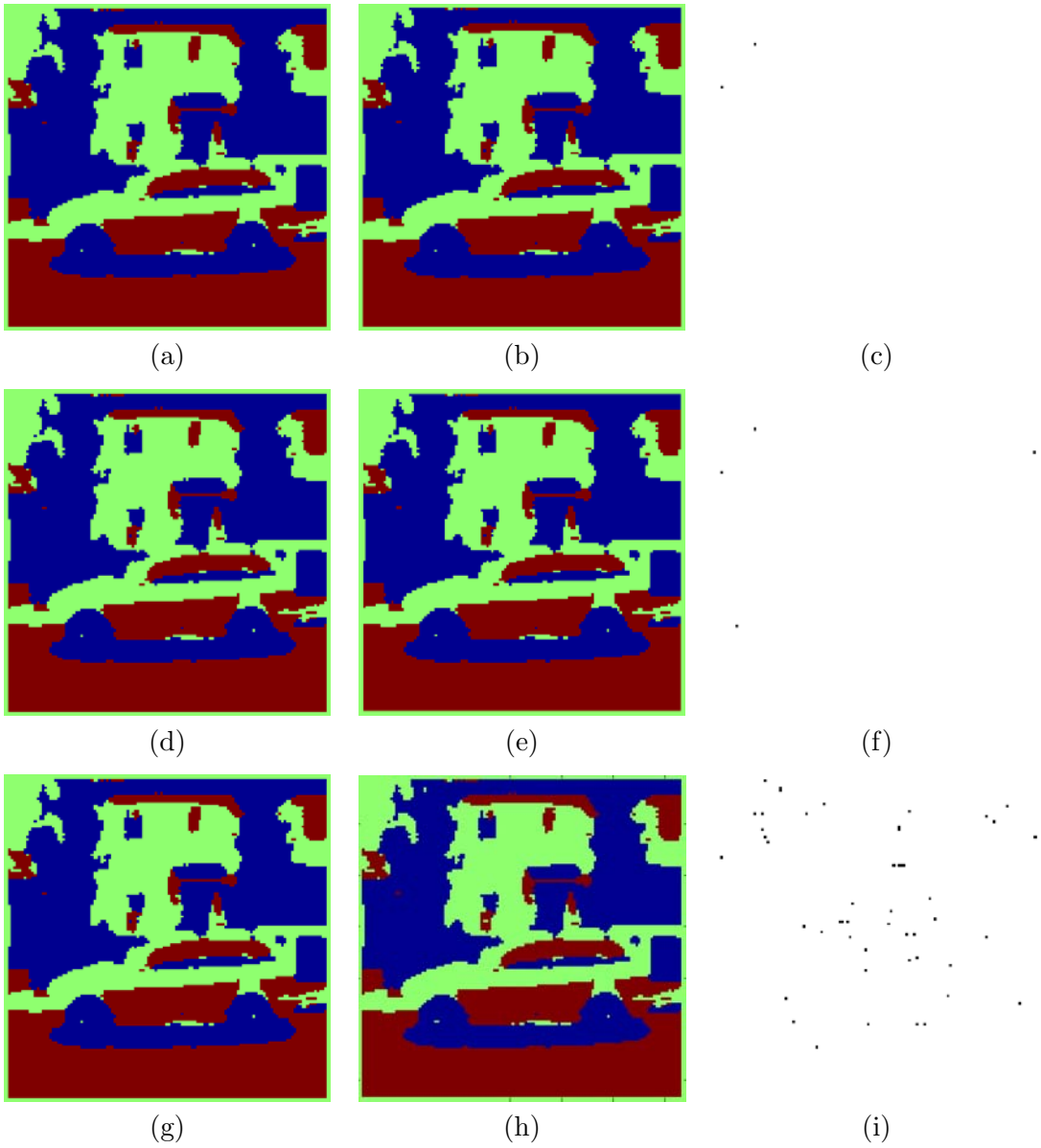


Figure A.32: Comparison of ground-truth with fixed-point model/architecture variants. (a) Ground-Truth. (b) 26bit model/architecture variant. (c) Difference (black) between (a) and (b) segmentation results. (d) Ground-Truth. (e) 20bit model/architecture variant. (f) Difference (black) between (d) and (e) segmentation results. (g) Ground-Truth. (h) 16bit model/architecture variant. (i) Difference (black) between (g) and (h) segmentation results. Ground-Truth (a),(d) and (g) are identical and only depicted for the purpose of illustration.

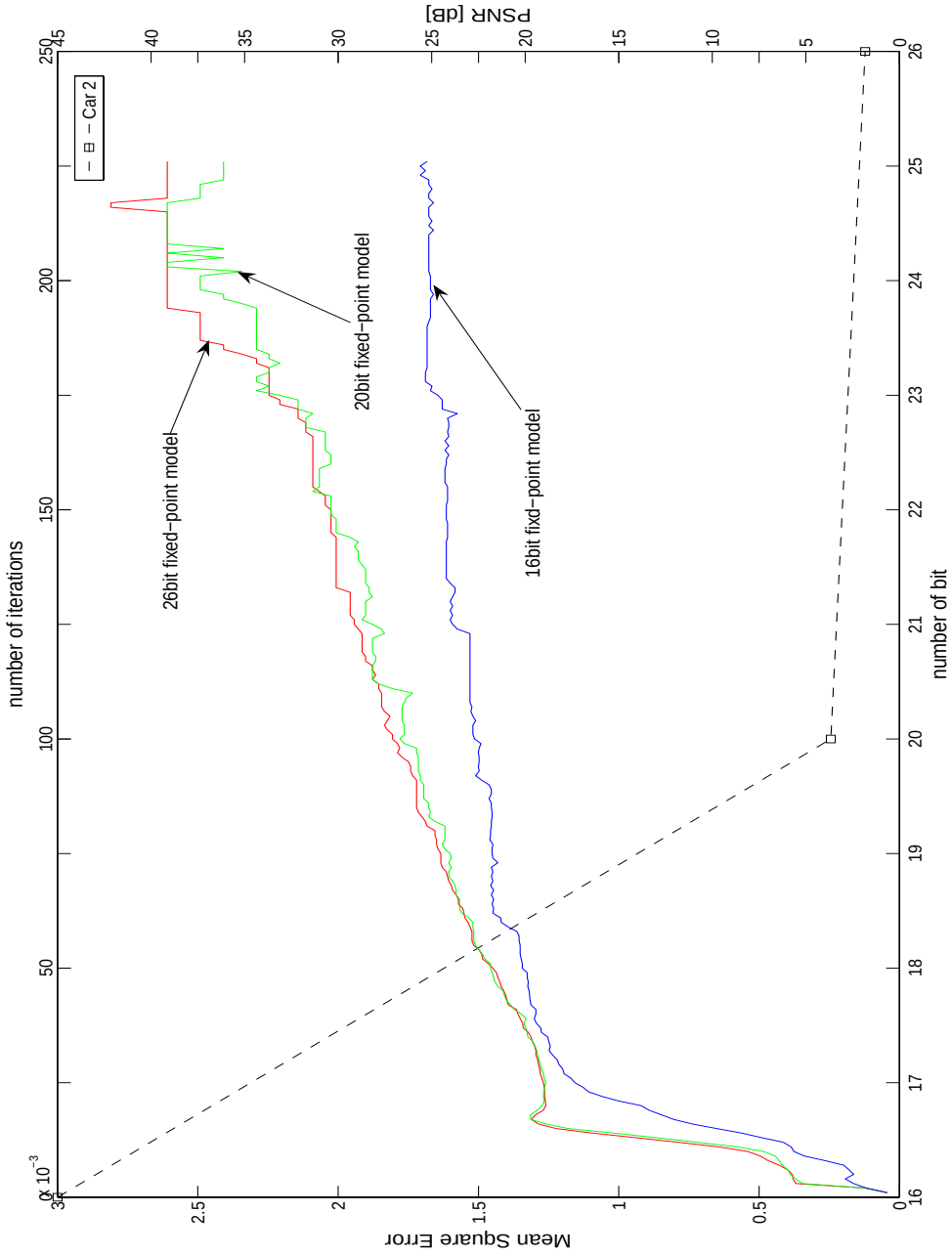


Figure A.33: Means Square Error (MSE) and Peak signal-to-noise ratio (PSNR) analysis of fixed-point simulation runs (car 2 image, cf. Figure A.32) of the segmentation model. MSE versus number of bit used for the model. PSNR versus number of iterations.

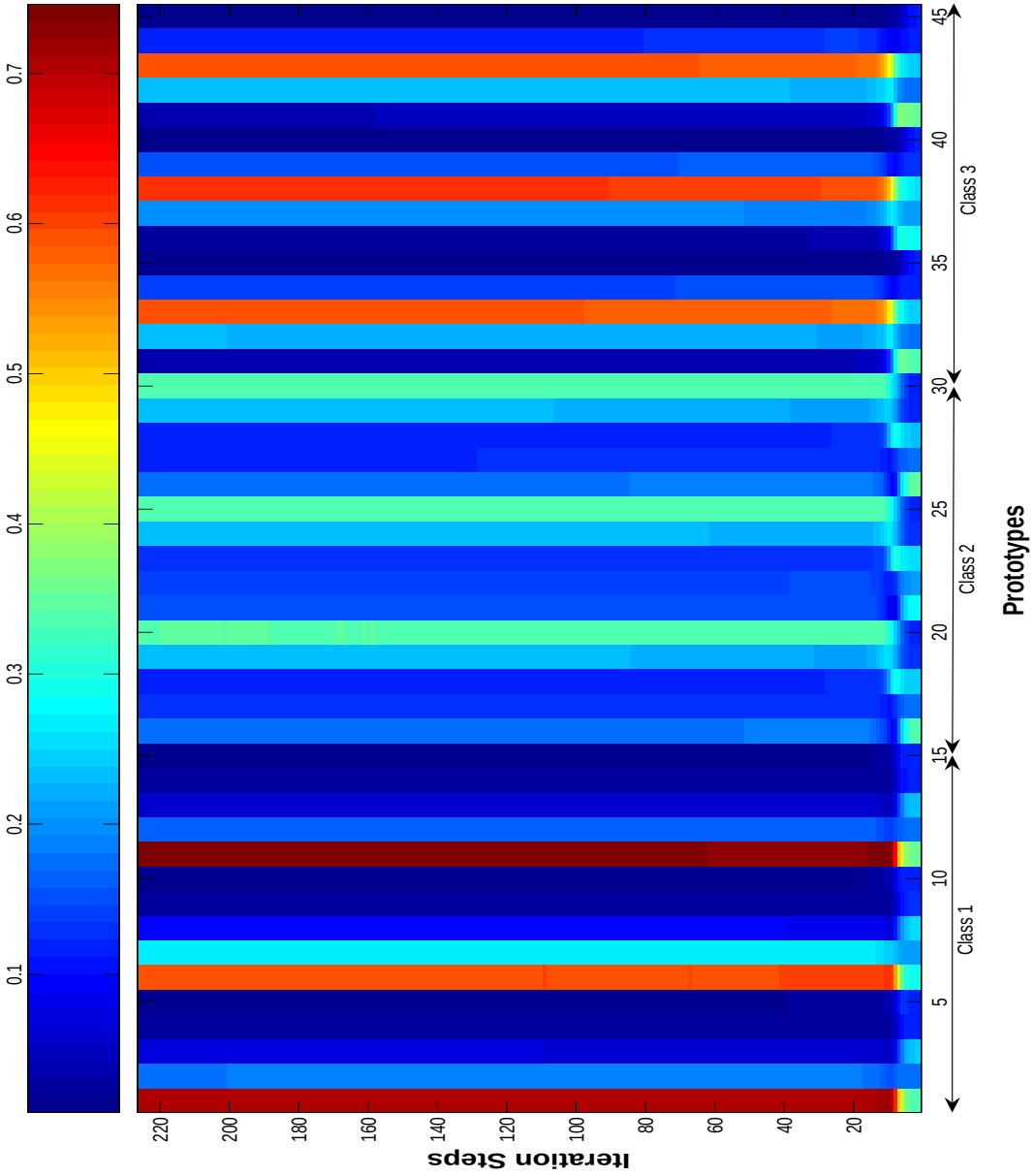


Figure A.34: Estimated prototypes q_ν. Prototypes of the architecture/model variant with 20bit fixed point precision and with respect to the utility car image (rotate to view). Each class prototype is composed of probability distributions for the RGB channels. The probability values are encoded corresponding to the colorbar.

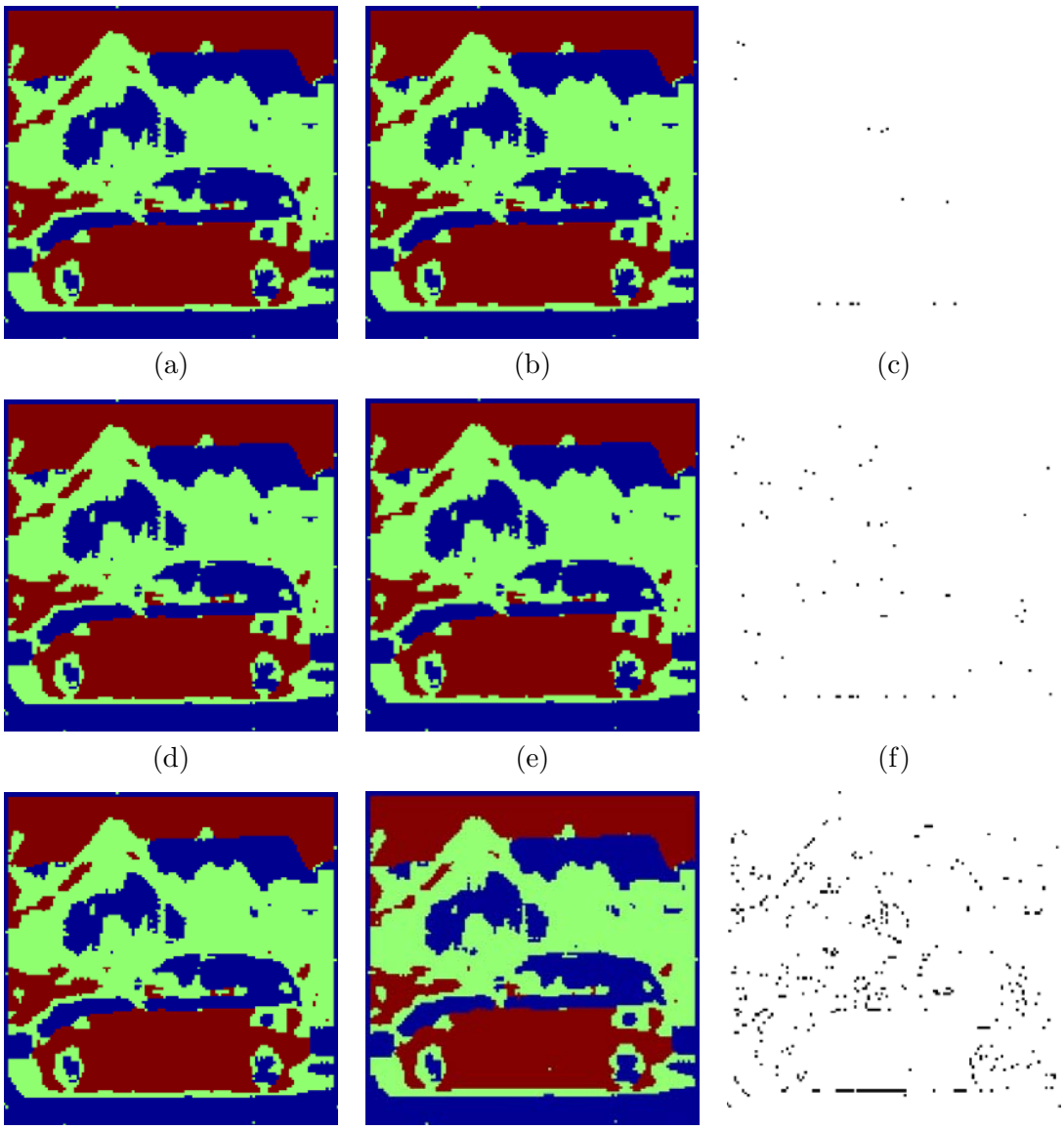


Figure A.35: Comparison of ground-truth with fixed-point model/architecture variants. (a) Ground-Truth. (b) 26bit model/architecture variant. (c) Difference (black) between (a) and (b) segmentation results. (d) Ground-Truth. (e) 20bit model/architecture variant. (f) Difference (black) between (d) and (e) segmentation results. (g) Ground-Truth. (h) 16bit model/architecture variant. (i) Difference (black) between (g) and (h) segmentation results. Ground-Truth (a),(d) and (g) are identical and only depicted for the purpose of illustration.

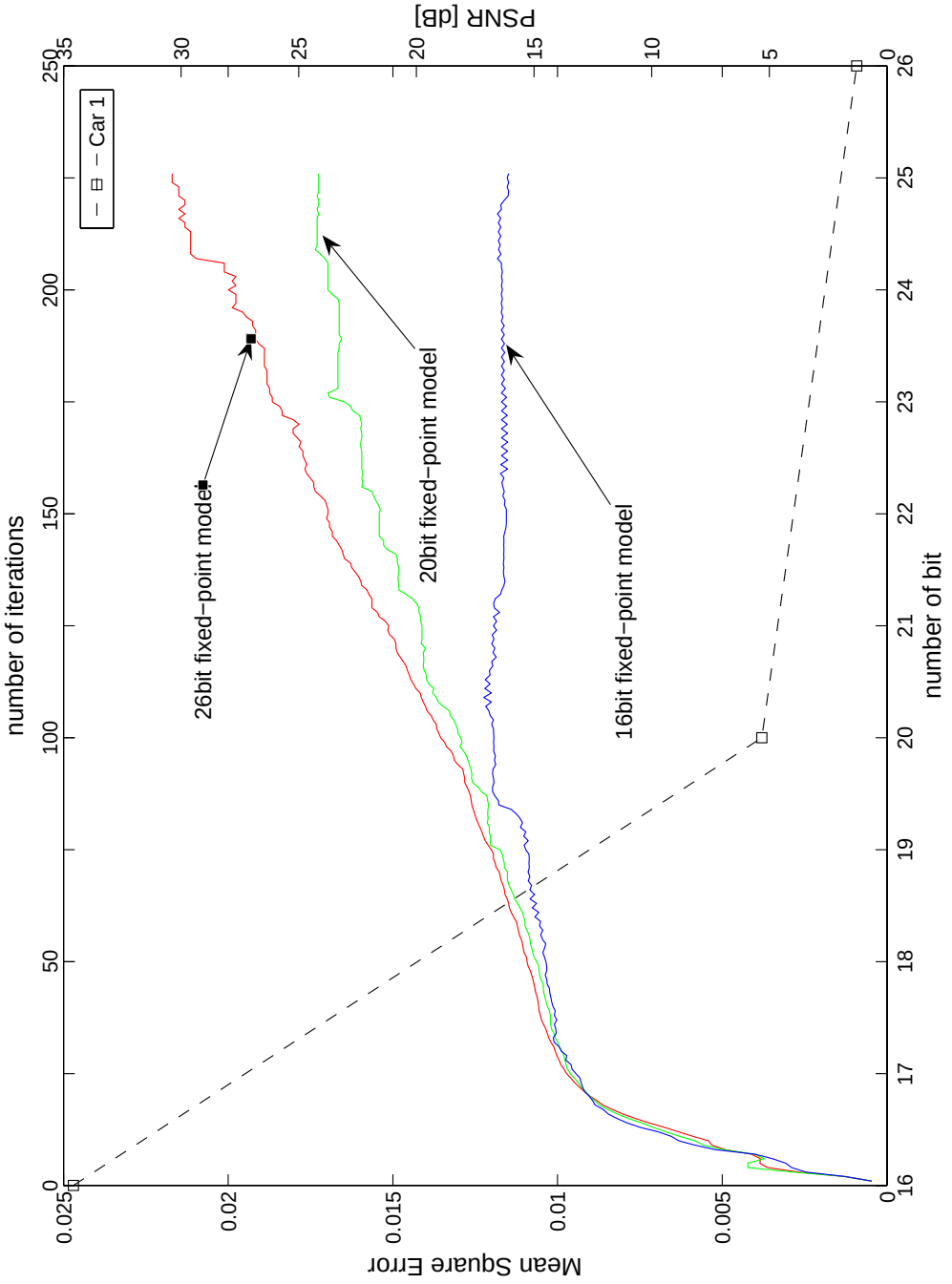


Figure A.36: Means Square Error (MSE) and Peak signal-to-noise ratio (PSNR) analysis of fixed-point simulation runs (car 1 image, cf. Figure A.35) of the segmentation model. MSE versus number of bit used for the model. PSNR versus number of iterations.

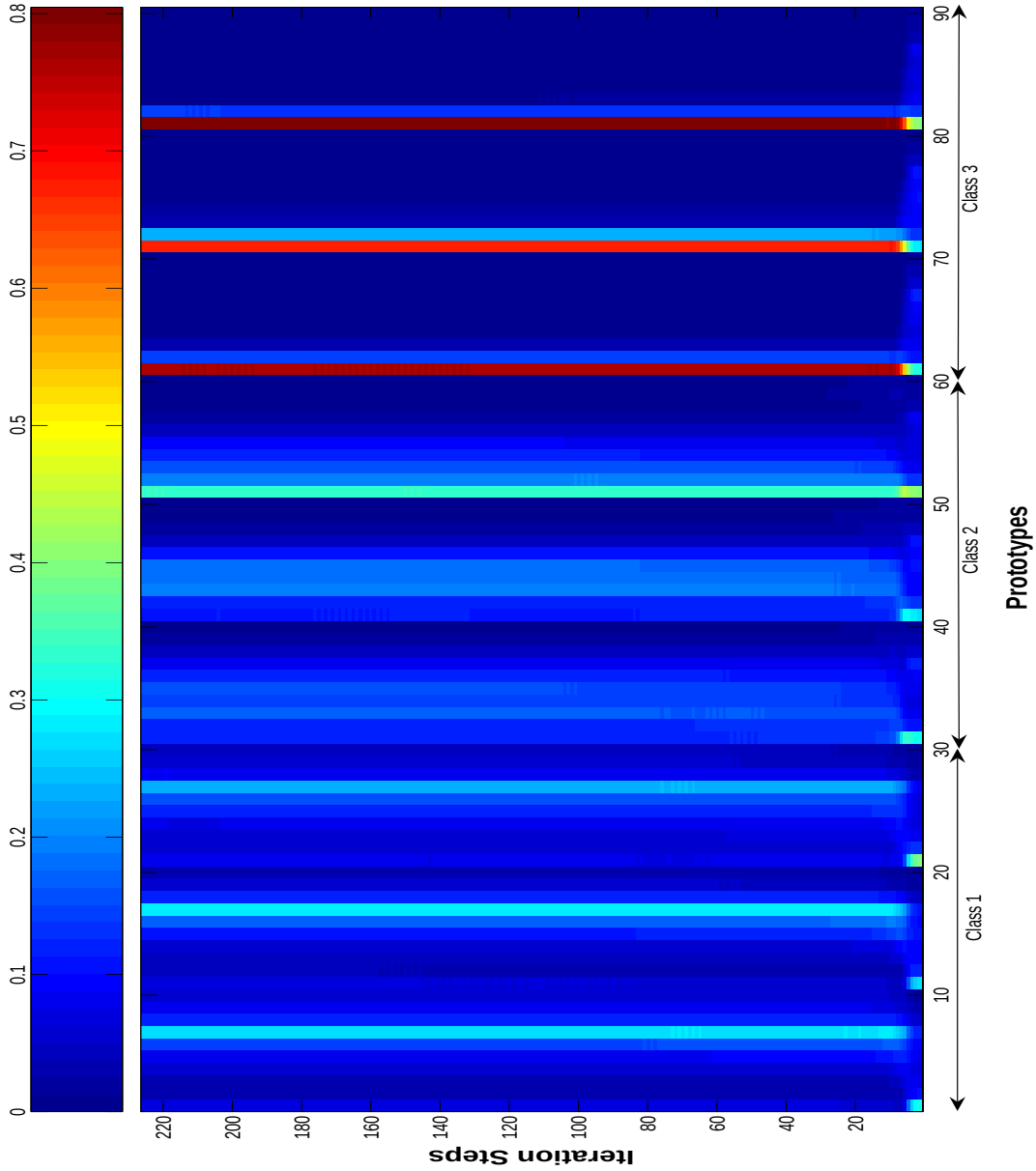


Figure A.37: Estimated prototypes q_ν. Prototypes of the architecture/model variant with 16bit fixed point precision and with respect to the sports car image (rotate to view). Each class prototype is composed of probability distributions for the RGB channels. The probability values are encoded corresponding to the colorbar.

Appendix B

Design Framework Results

This appendix presents various prototypical VLSI results and abstract graph representations, which have been relocated from the main text and the result Section 5.5 to this appendix to enhance the readability of this thesis. Several VLSI results and graph representations have been included to illustrate the capabilities of the proposed design framework and hence to underpin its practical relevance. All VLSI implementations have been done with Xilinx FPGA technologies of different gate-capacities and family-types. The particular graphs are the original representations, which have been taken from the canonical design representation of the proposed design framework.

Results - Design Framework

Figure	**Description**
Figure B.1	Illustration of cell-cluster wrapper by means of gate-level schematic.
Figure B.2	Floorplan guided by hierarchy- and planarity feature encoded within the HDL of 64×64 large cell grid.
Figure B.3	Detailed floorplan and resulting place & route of 4×4 site cluster.
Figure B.4	Utilization of hierarchy- and planarity feature for different groups of cell-cluster within one MRF grid.
Figure B.5	Original graph of global memory hierarchy. Extracted from canonical design representation.
Figure B.6	Floorplan and place & route of exemplary memory hierarchy. Illustration of overall structure and wiring characteristic.
Figure B.7	Constraint-driven floorplan of combined cell-cluster and memory hierarchy structures.

Results - Design Framework

Figure	**Description**
Figure B.8	Graph representation of port memory block with a 2-step send and receive procedure. Port memory block for first neighborhood system. (Schematic cf. Figure B.9)
Figure B.9	RTL schematic of port memory block with a 2-step send and receive procedure. Port memory block for first neighborhood system. (Graph representation cf. Figure B.8)
Figure B.10	Close-up of RTL schematic. Schematic of port memory block for 5th order neighborhood system.
Figure B.11	Schematic cut-out of histogram forming design. Illustration of design.
Figure B.12	Place & Route and extracted FSM of histogram forming design. Illustration of design and FSM details.
Figure B.13	Complete graph representation of noise removing and edge preserving energy functional.
Figure B.14	Close-up of Figure B.13 - graph representation of noise removing and edge preserving energy functional. Illustration of graph details.
Figure B.15	Close-up of Figure B.13 - graph representation of noise removing and edge preserving energy functional. Illustration of graph details.
Figure B.16	RTL schematics of a specific portion (cup-function) of the noise removing and edge preserving energy functional. Illustration of combined control- and data-path.
Figure B.17	Close-up of RTL schematic of a specific portion (cup-function) of the noise removing and edge preserving energy functional. Illustration of operator-wrapping.
Figure B.18	Extracted FSM of a specific portion (cup-function) of the noise removing and edge preserving energy functional. Illustration of FSM state transitions.

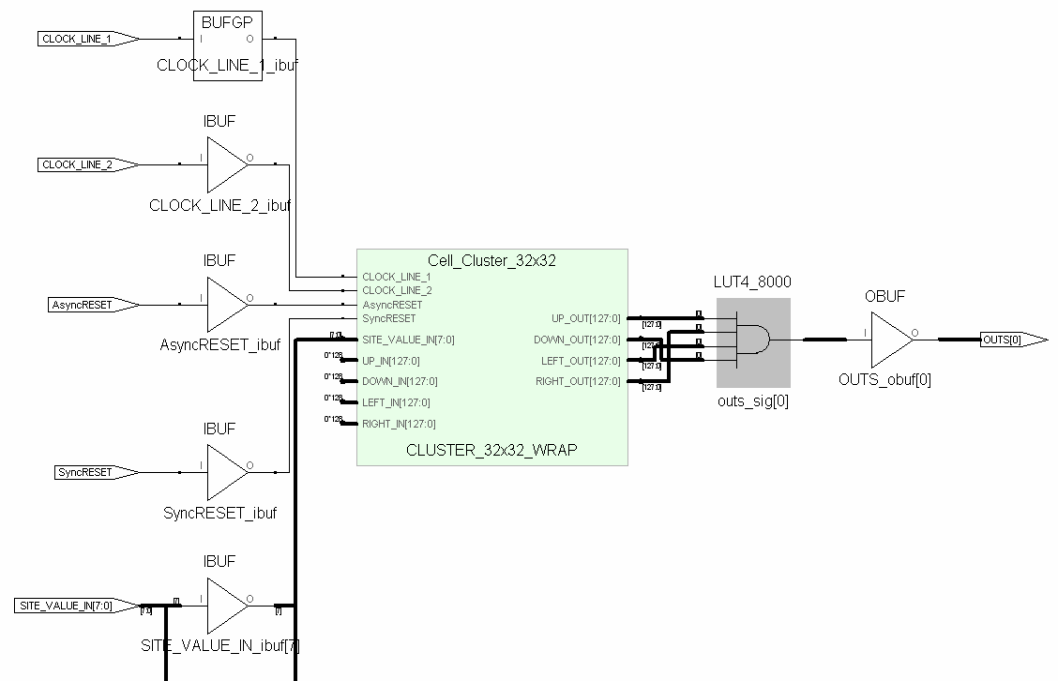


Figure B.1: Illustration of cell-cluster wrapper by means of gate-level schematic. Controlled handling of incomplete neighborhoods at the boarder of the MRF grid. 32×32 large cell grid.

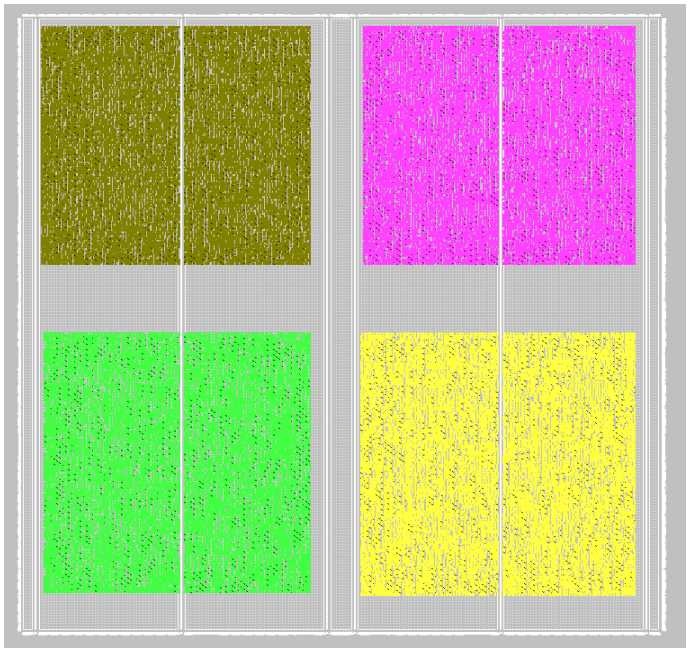
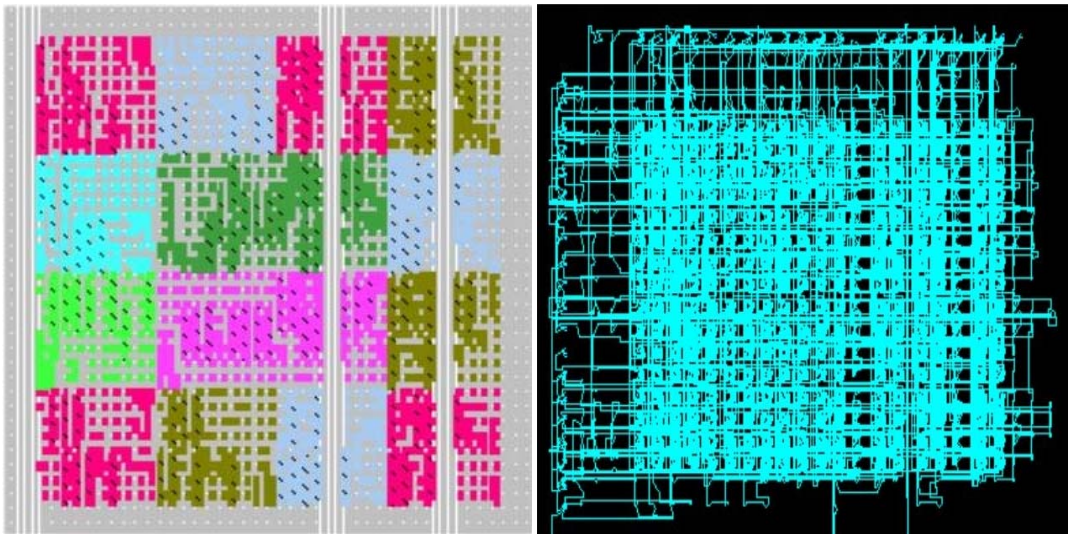
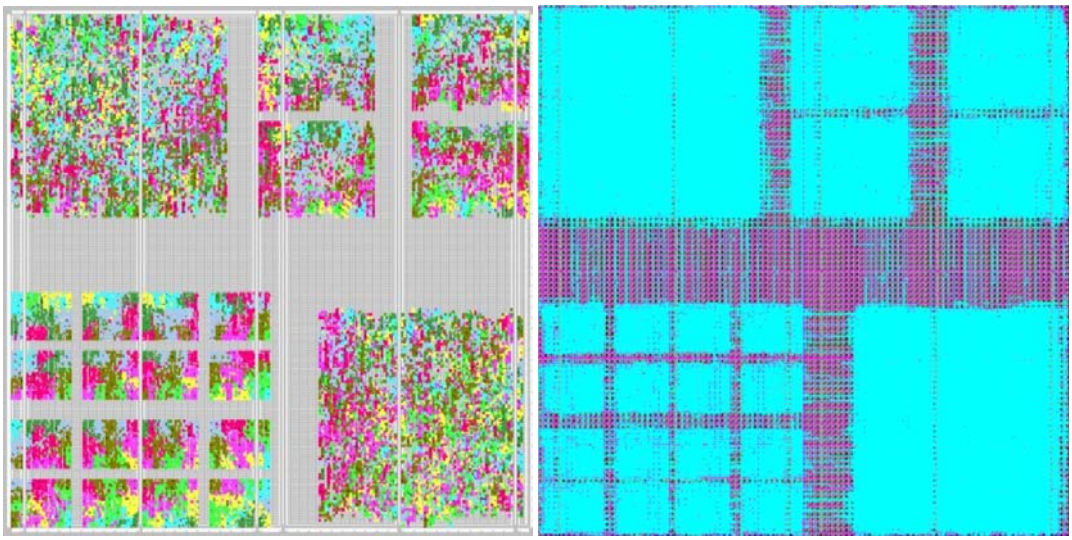


Figure B.2: Floorplan of 64×64 large cell grid. Clustering and quadrant organized floorplanning of four 32×32 sub-cluster. Floorplanning guided by hierarchy- and planarity feature encoded within the HDL code and preserved from representing graph.



(a) (b)

Figure B.3: Detailed floorplan and resulting place & route of 4×4 site cluster. (a) Floorplan of the 16 particular sites generated by using the embedded hierarchy- and planarity feature. (b) Place and Route of floorplan.



(a) (b)

Figure B.4: Utilization of embedded hierarchy- and planarity feature for different groups of site cluster. Illustrated by means of 32 cell grid. (a) Topmost left and bottom right cluster: Floorplan derived from placer without using the hierarchy- and planarity feature. Topmost right cluster: Split of cluster and floorplan derived from placer. The floorplan becomes largely unstructured even for small cell-cluster. Bottom left: (b) Place and Route of floorplan. Illustration of neighborhood structure between the cell and hence the cell-cluster.

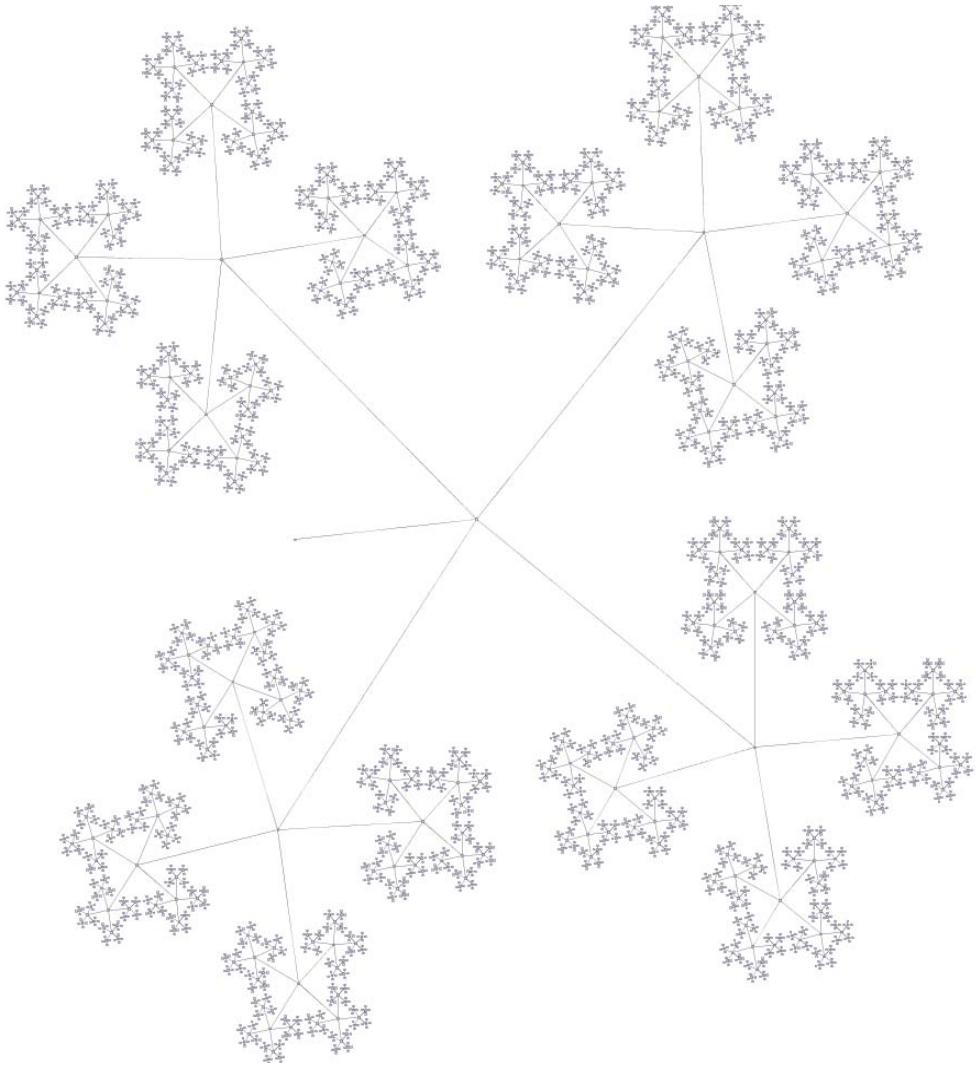
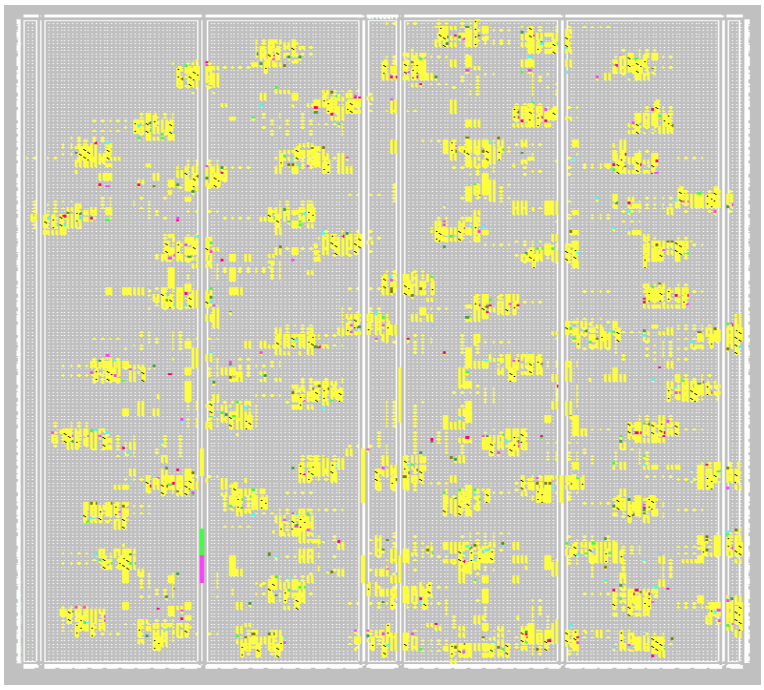
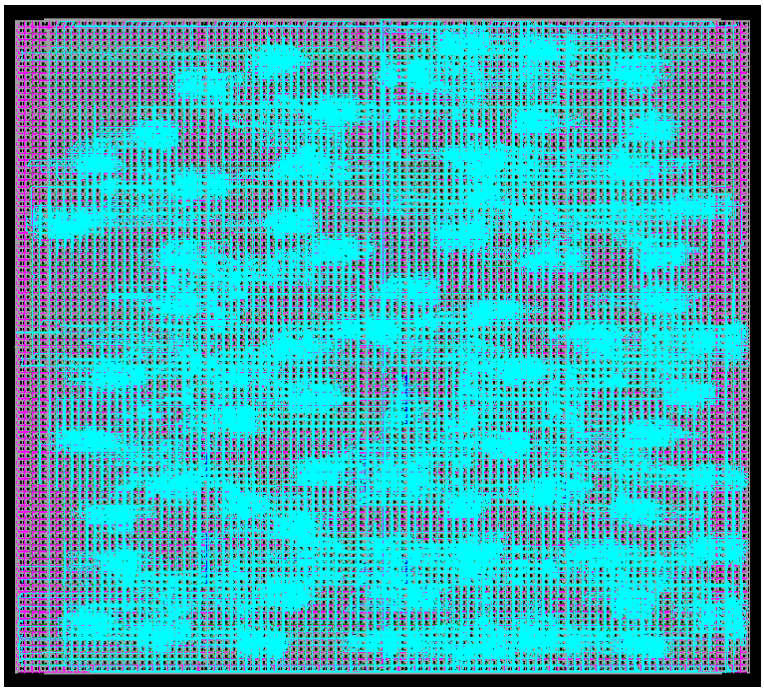


Figure B.5: Overview memory hierarchy graph. Original graph extracted from canonical design representation. Suitable for a 64×64 MRF grid. Illustration of principle hierarchy-structure and arrangement.



(a)



(b)

Figure B.6: Floorplan and place & route of memory hierarchy. (a) Floorplan of the particular memory elements of the hierarchy. Usage of different memory blocks and their corresponding placing. (b) Place & Route of the floorplan. Illustration of the memory hierarchy's wiring characteristic.

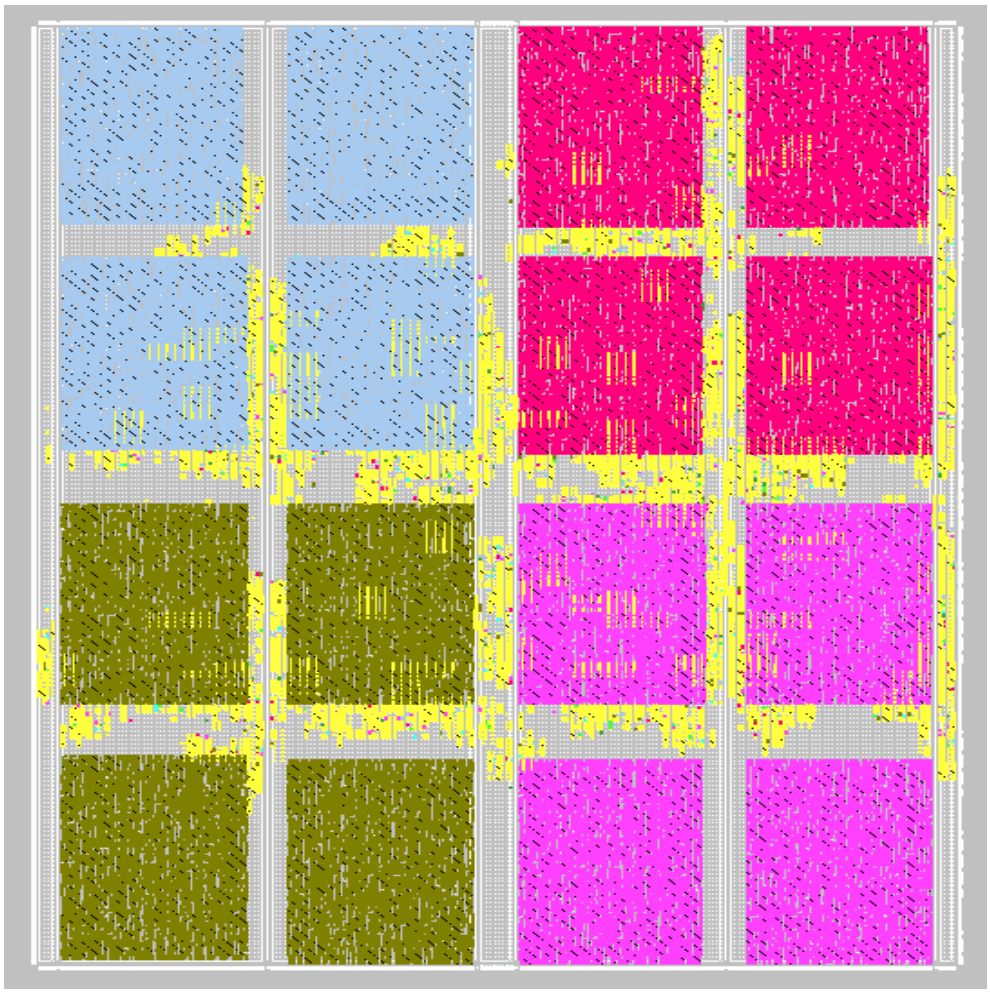


Figure B.7: Constraint-driven floorplan of combined cell-grid and memory hierarchy. Clearly separated structures. Top row of cell-cluster equipped with constraints to partially relocate the elements of the memory hierarchy in the floorplan. Bottom row of cell-cluster equipped with constraints to completely relocate the elements of the memory hierarchy in the floorplan.

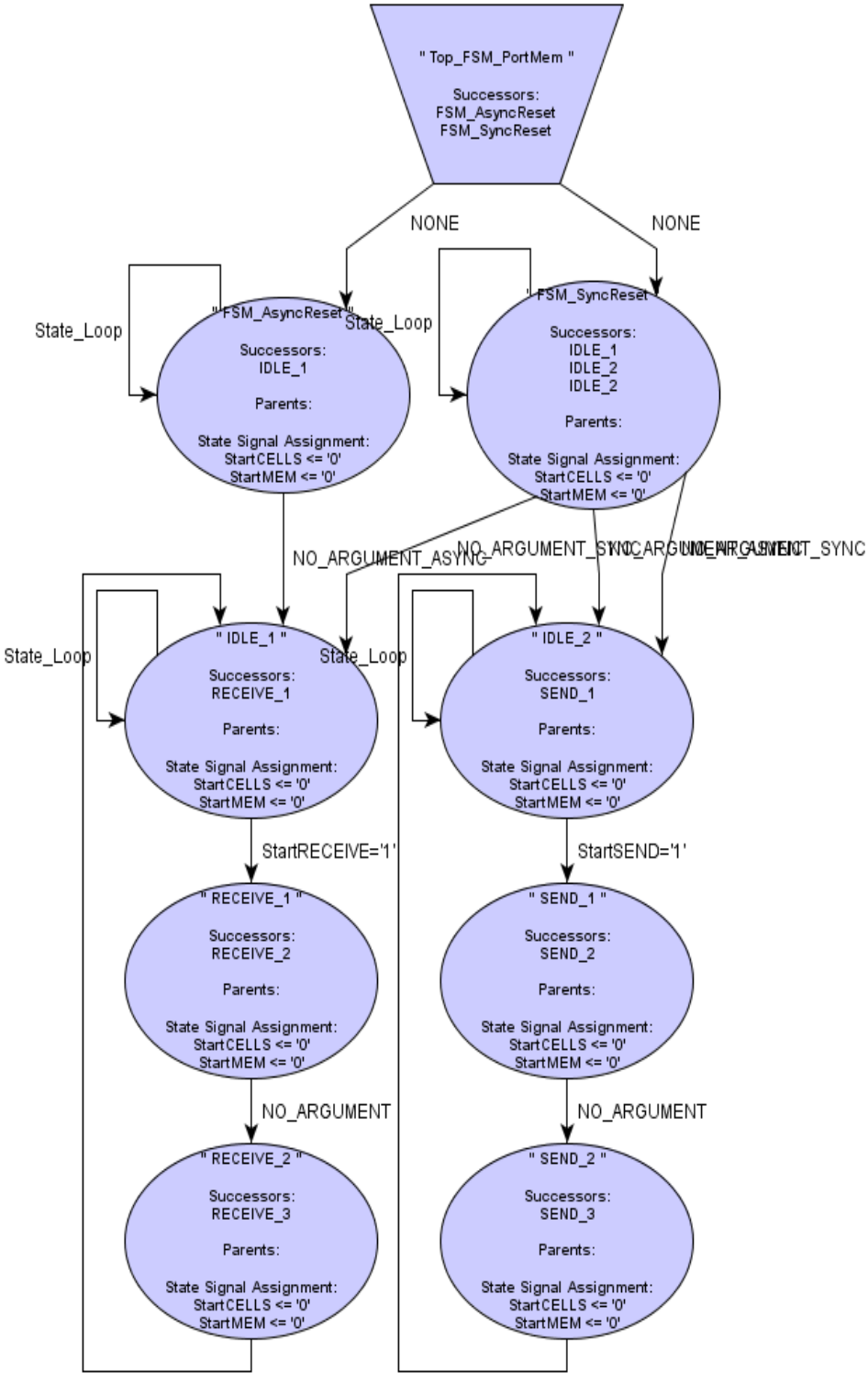


Figure B.8: Graph representation of port memory block with a 2-step send and receive procedure. Original graph representation extracted from canonical design representation (not all node information is shown).

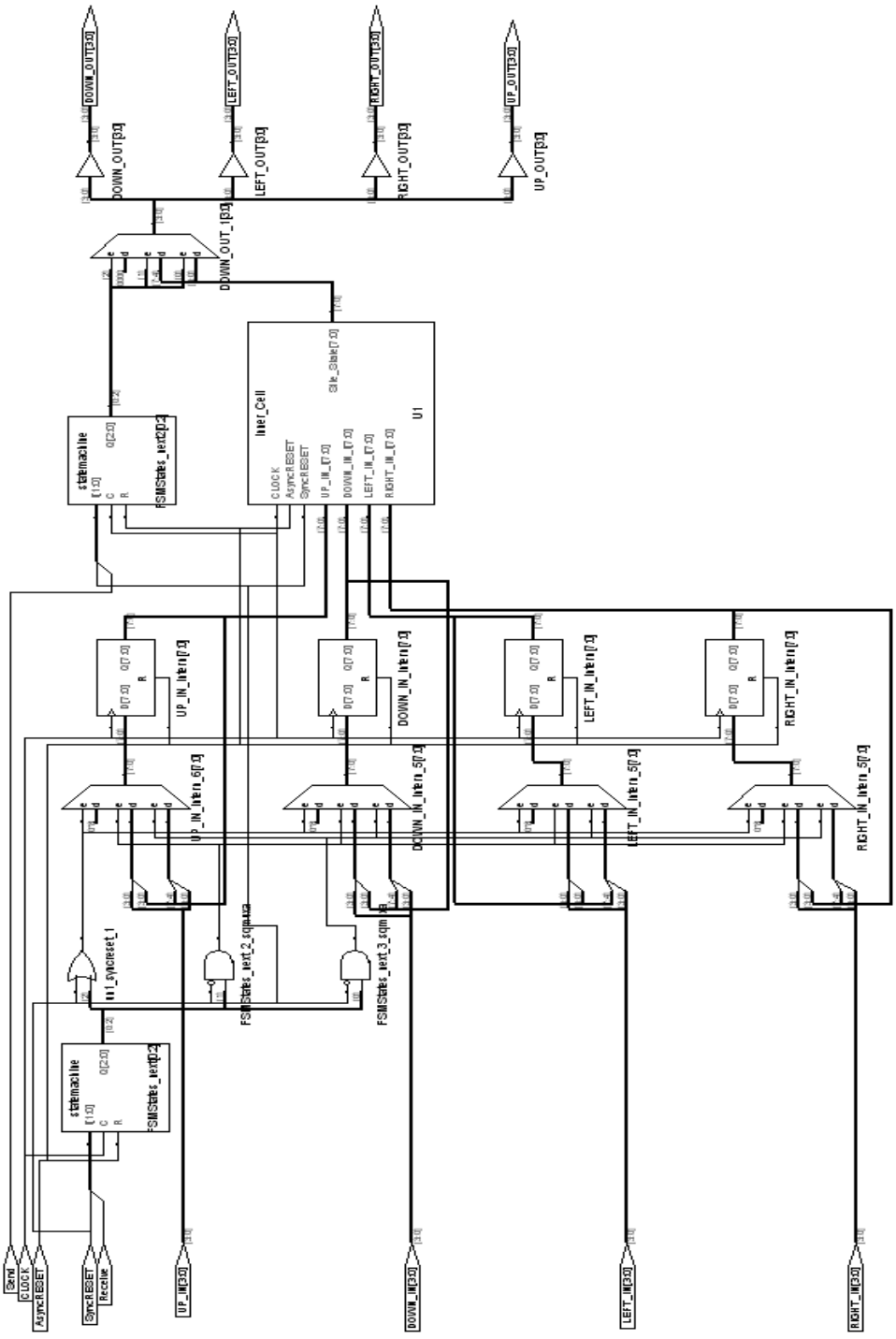


Figure B.9: RTL schematic of port memory block with a 2-step send and receive procedure. RTL schematic of port memory block synthesized by 3rd party tool from generated HDL code. Port memory for first order neighborhood system.

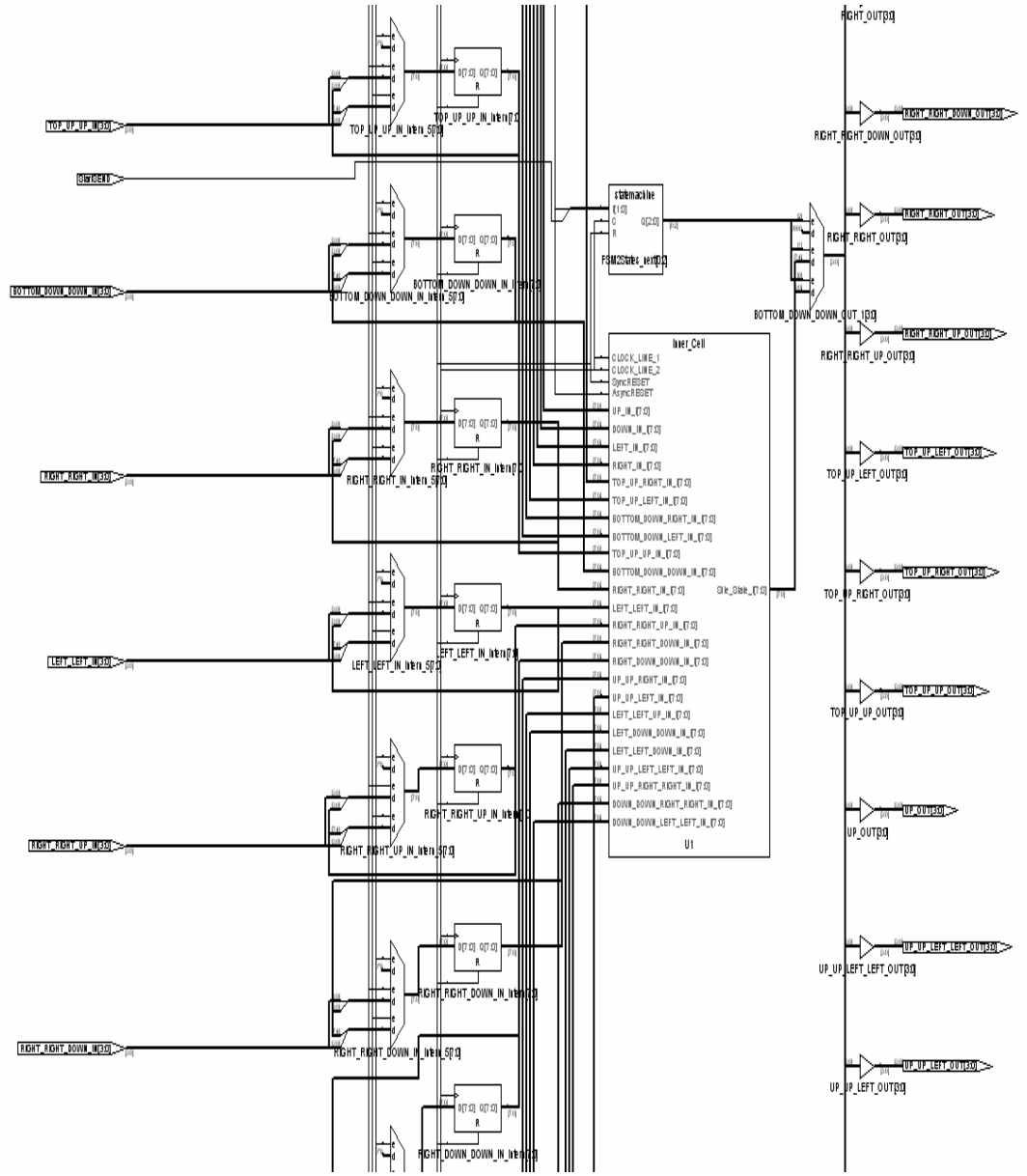


Figure B.10: Close-up of RTL schematic. Schematic of particular port memory block with a 2-step send and receive procedure. Support of 5th order neighborhood system. Data-flow from left to right. Left most side: Inputs (not complete) from 5th order neighbors and corresponding register-banks. Middle part: FSM block and "Inner_Cell" block encapsulating the rest of the cell circuits. All register-banks (only partially shown) of the port memory block are connected to the "Inner_Cell" block to provide the neighborhood data for the cell internal processing. Right most side: Distribution of cell-state datum to the 5th order output ports. Multiplexer unit (topmost right) realizes the 2-step send process; controlled by FSM.

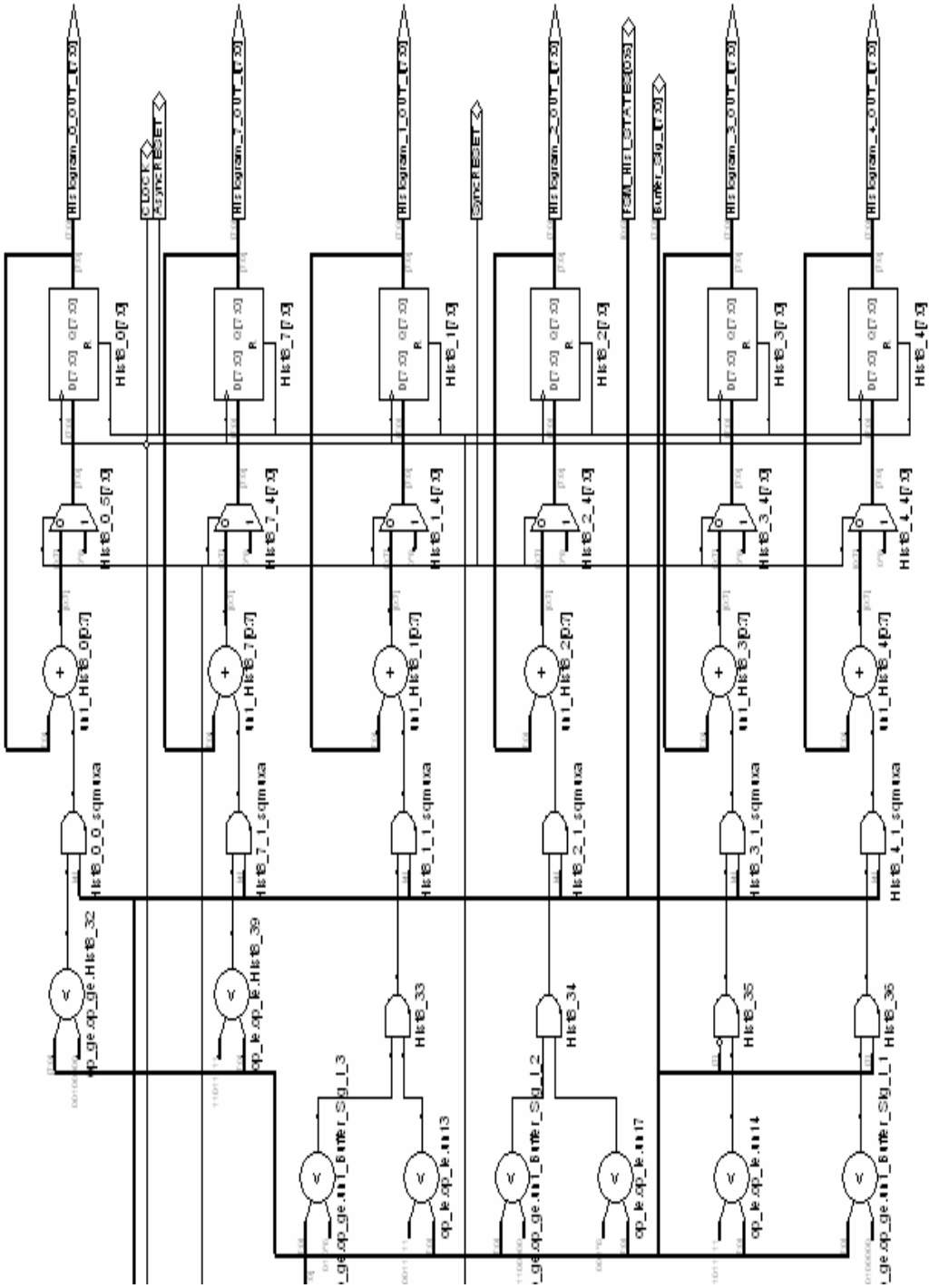
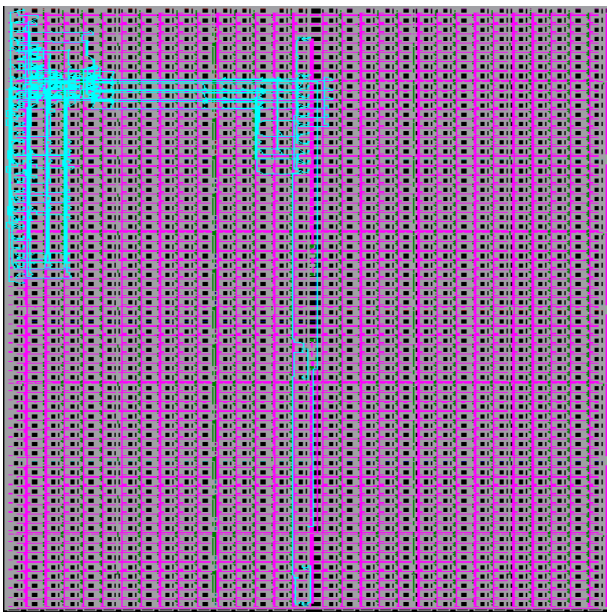
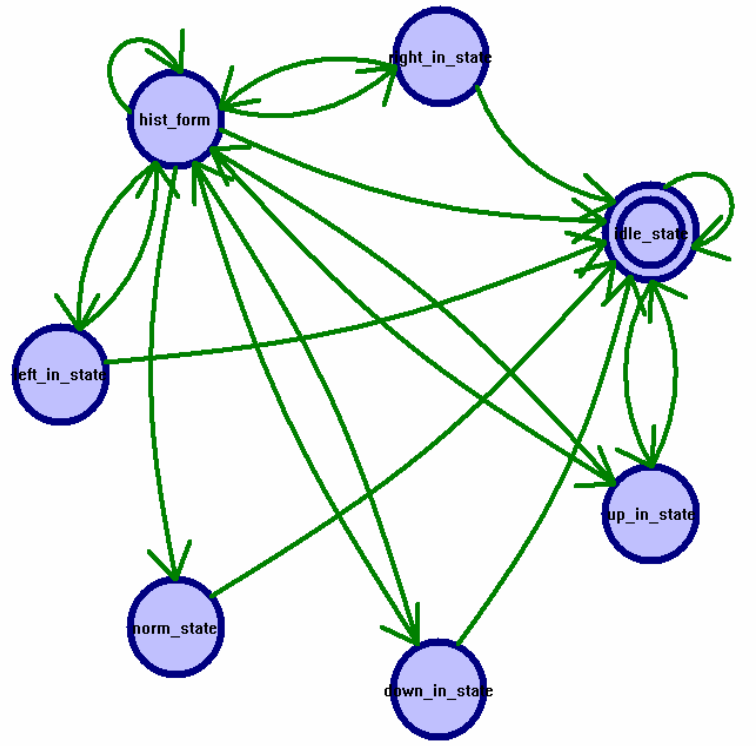


Figure B.11: Design of histogram forming process (1st order neighborhood system, 8 equally spaced bins, channel value-range [0,255], support for 1 channel). Cut-out of RTL schematic. Illustration of comparator operators to determine the bin of a value, and register-banks with feedback to plus operator to store and increase the bin values.



(a)



(b)

Figure B.12: Design of histogram forming process (1st order neighborhood system, 8 equally spaced bins, channel value-range [0,255], support for 1 channel). (a) Compact Place & Route result of histogram design. No place and route difficulties identified for this design. (b) Extracted Finite State Machine of histogram design.

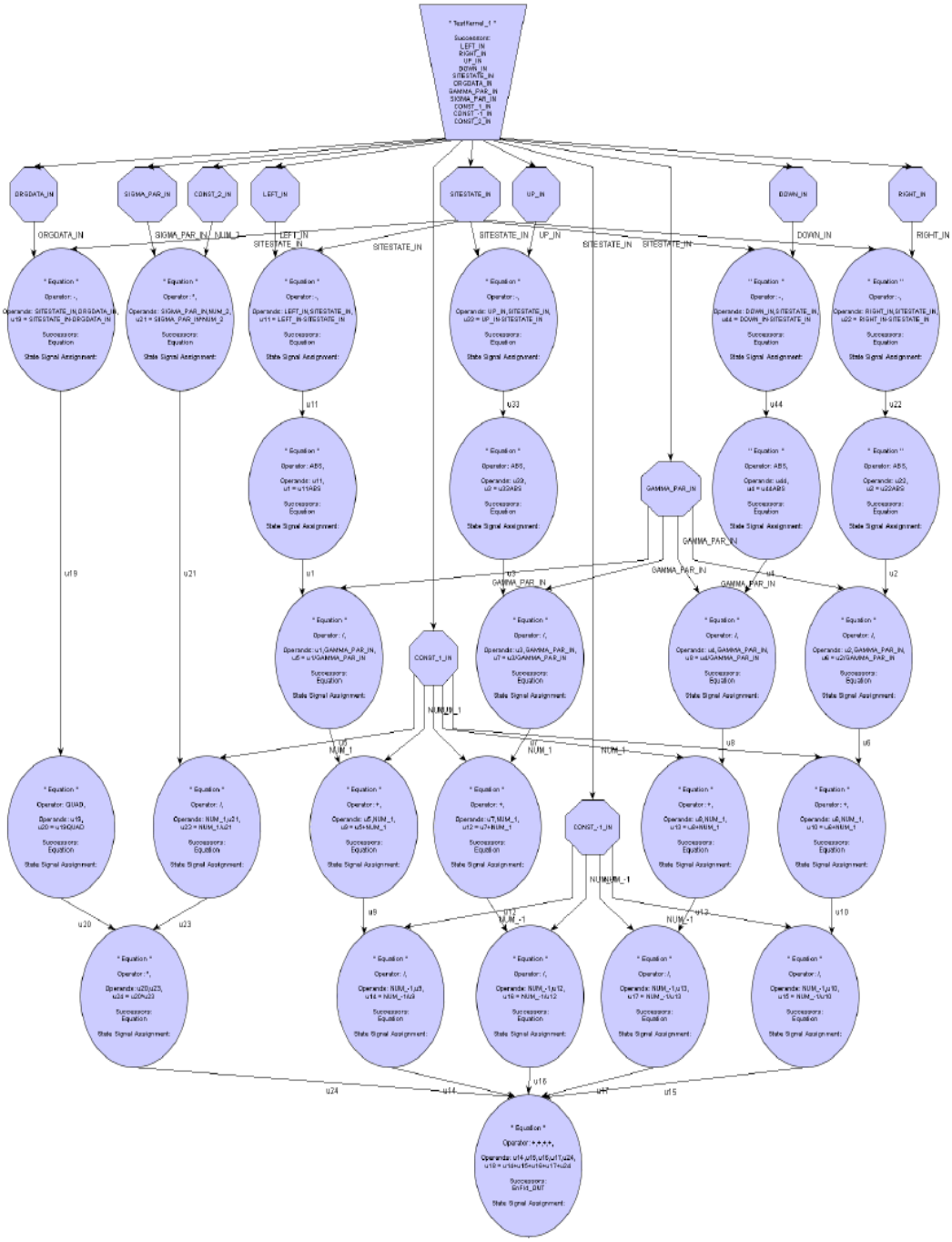


Figure B.13: Overview - original graph representation of noise removing and edge preserving energy functional. Graph extracted from canonical design representation. Trapezoid shaped node: Top node of the graph. Octagon shaped node: Data-Input node. Ellipse shaped node: Operator node.

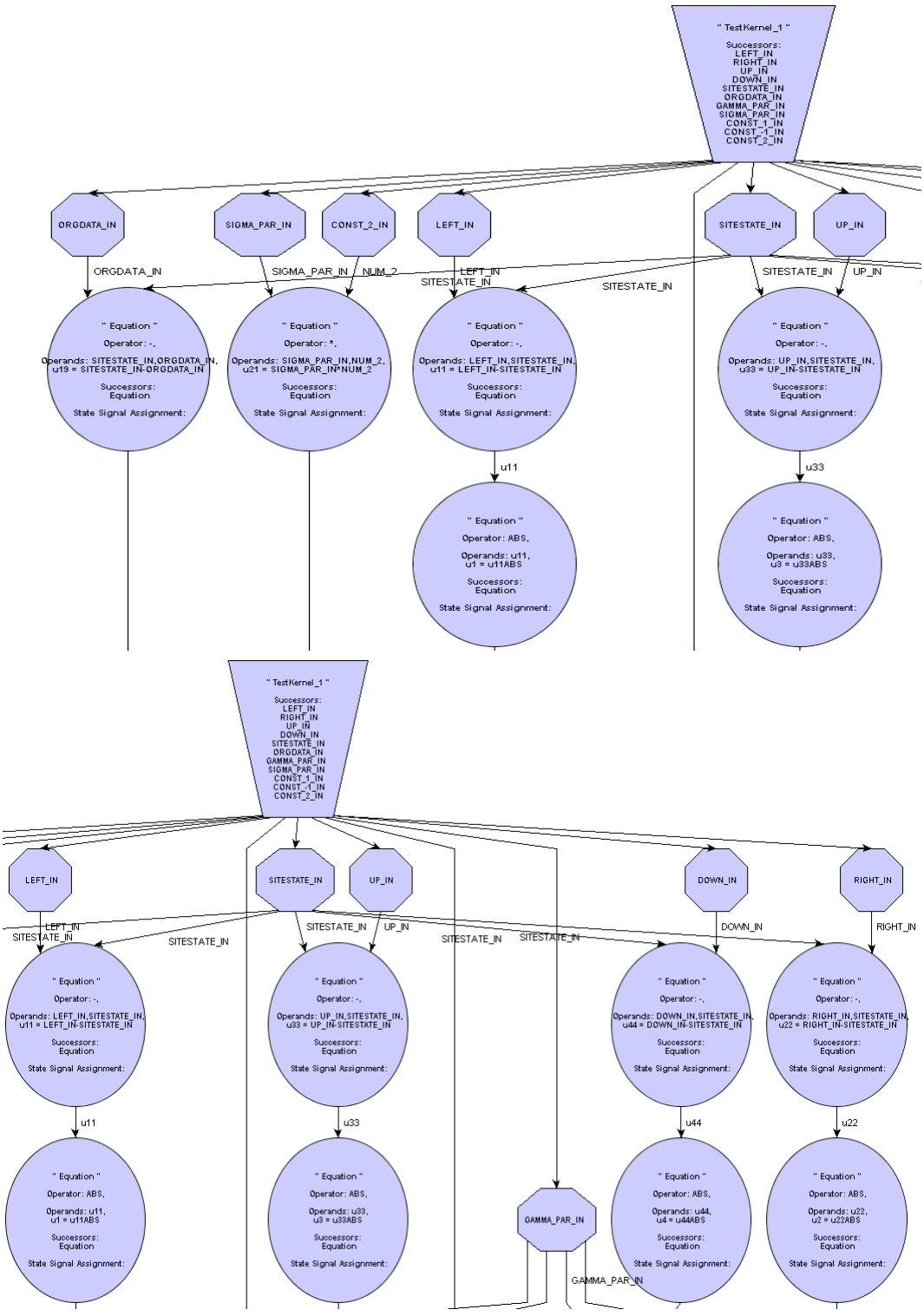


Figure B.14: Close up - Top left and top right part of Figure B.13. Graph extracted from canonical design representation. Trapezoid shaped node: Top node of the graph. Octagon shaped node: Data-Input node. Ellipse shaped node: Operator node.

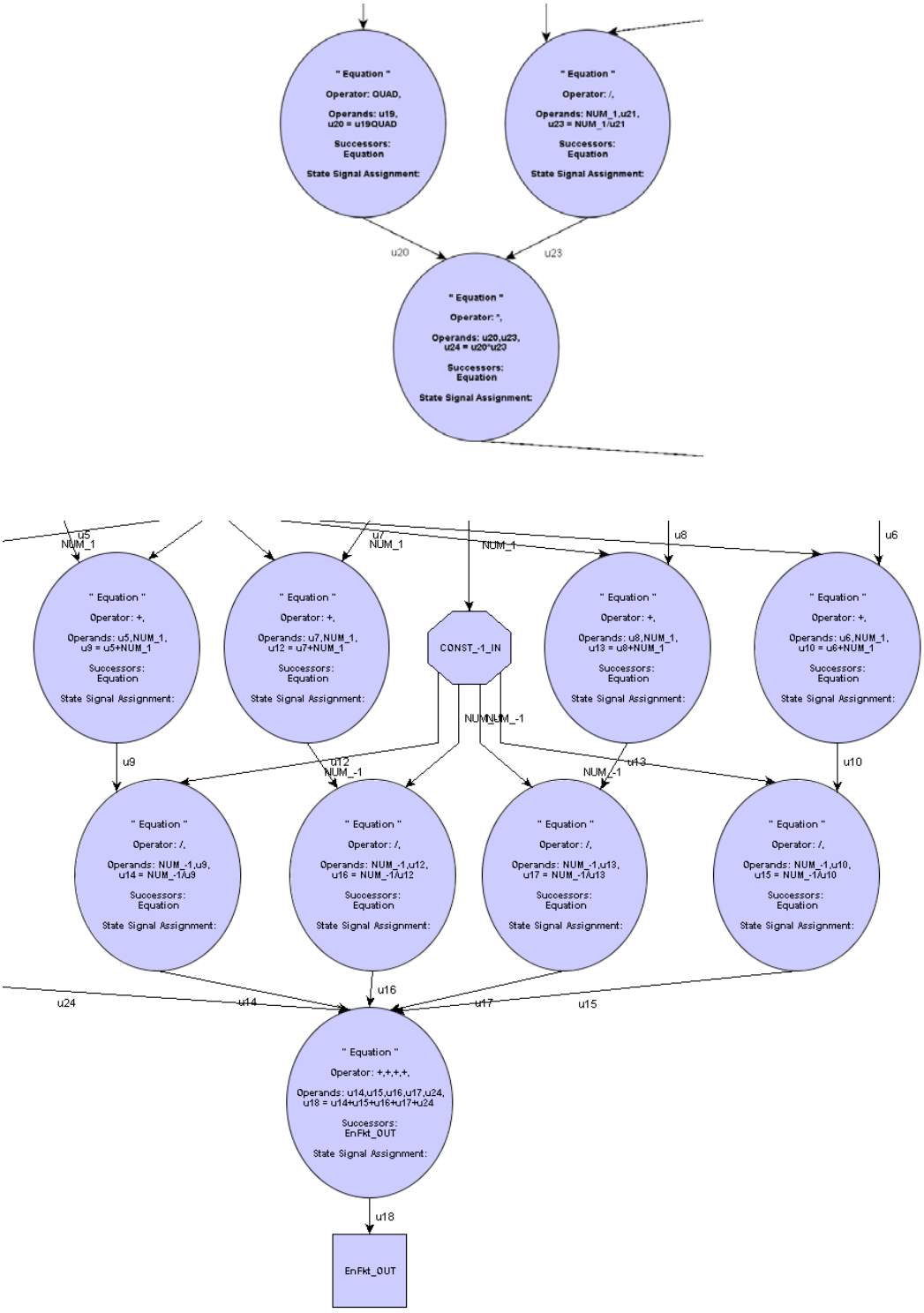
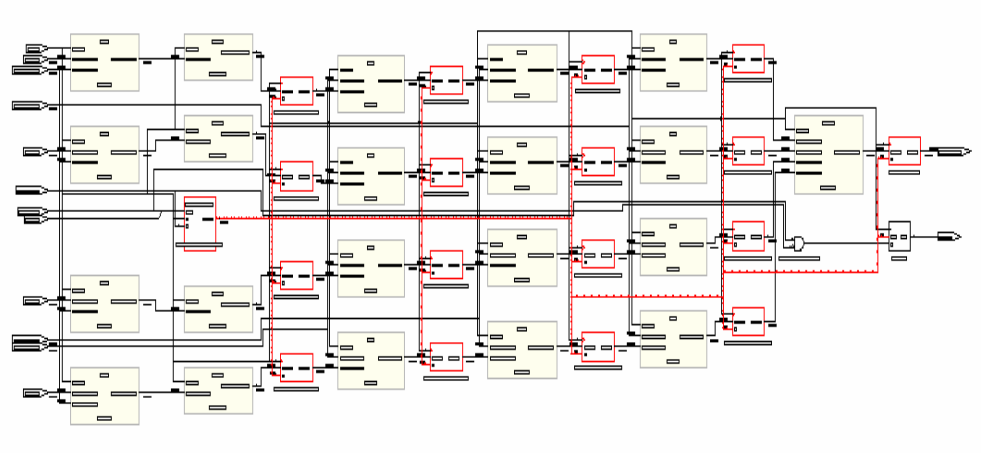
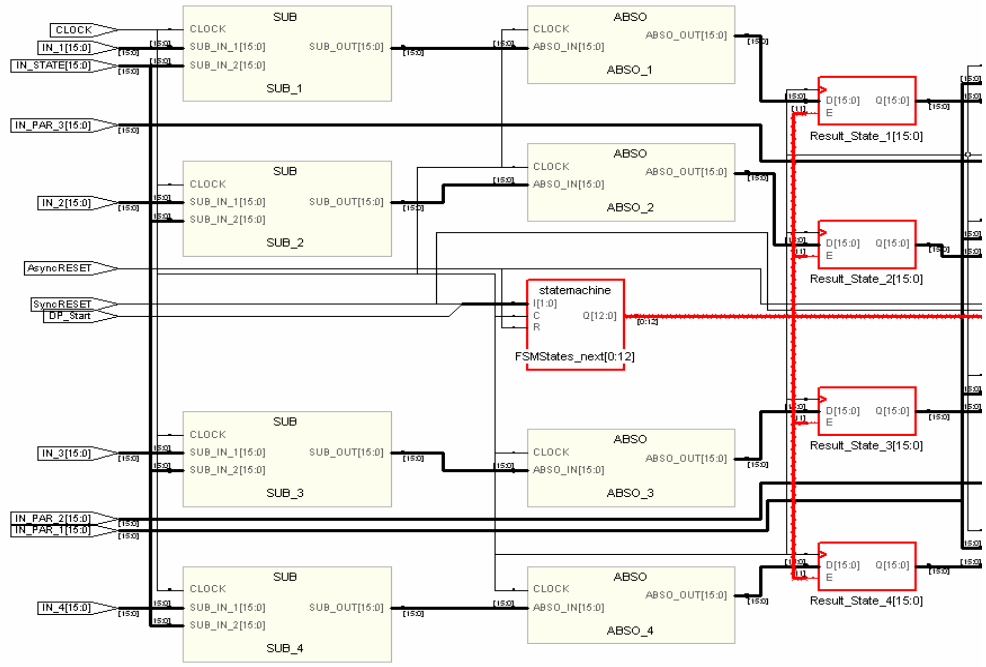


Figure B.15: Close up - Bottom right part of Figure B.13. Graph extracted from canonical design representation. Octagon shaped node: Data-Input node. Ellipse shaped node: Operator node. Rectangular shaped node: Data-Output node.



(a)



(b)

Figure B.16: RTL schematics of a specific portion (cup-function) of the noise removing and edge preserving energy functional. (a) Overview schematic of cup function. Highlighted red: FSM (left most block), register banks between operators and control signals from the FSM to the register banks. (b) Close-up of schematic. Highlighted red: FSM block, connections and first register bank.

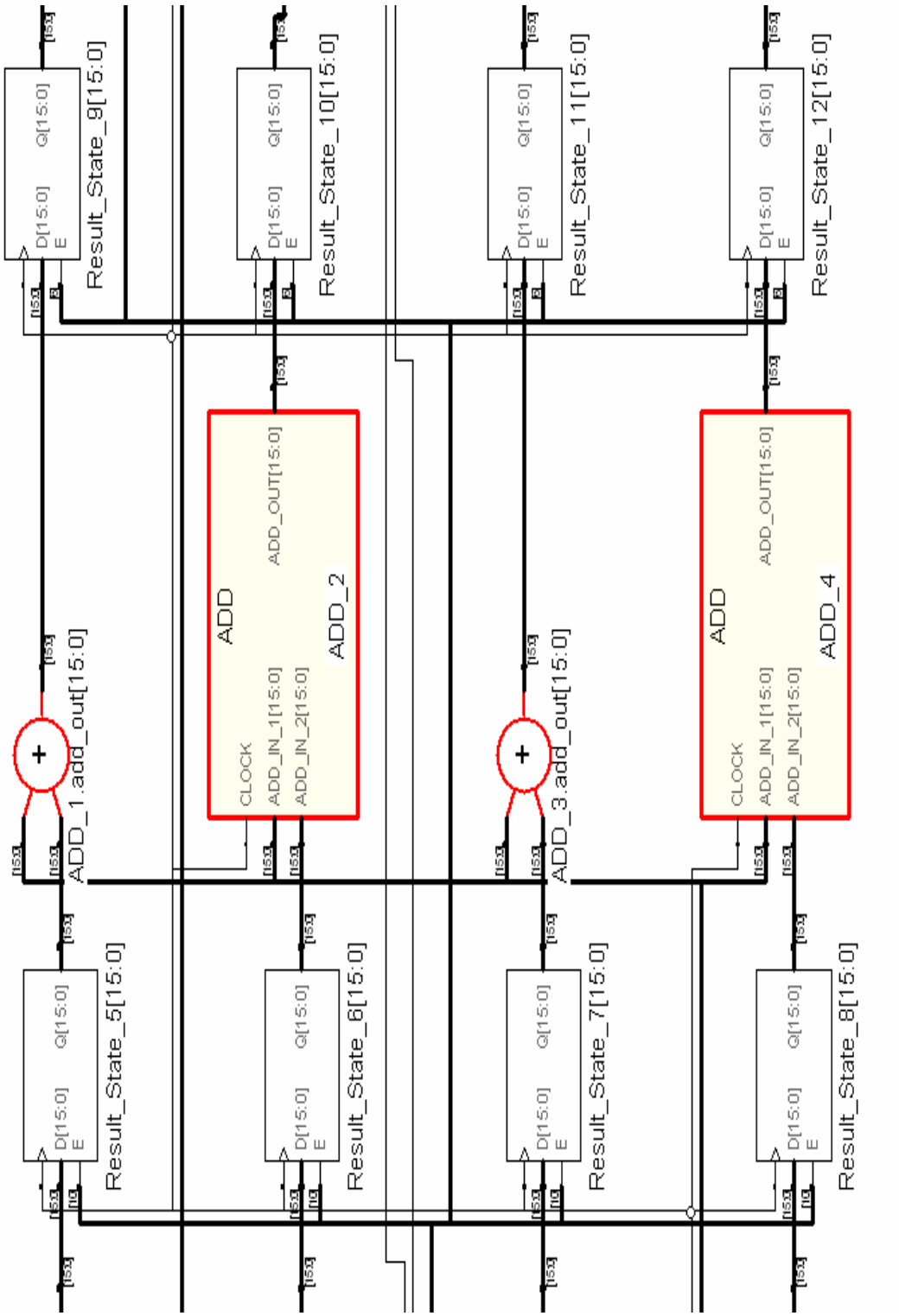


Figure B.17: Operator wrapping. Close-up RTL schematic (cup function). Highlighted red: Partially resolved operator wrapping for the plus operator.

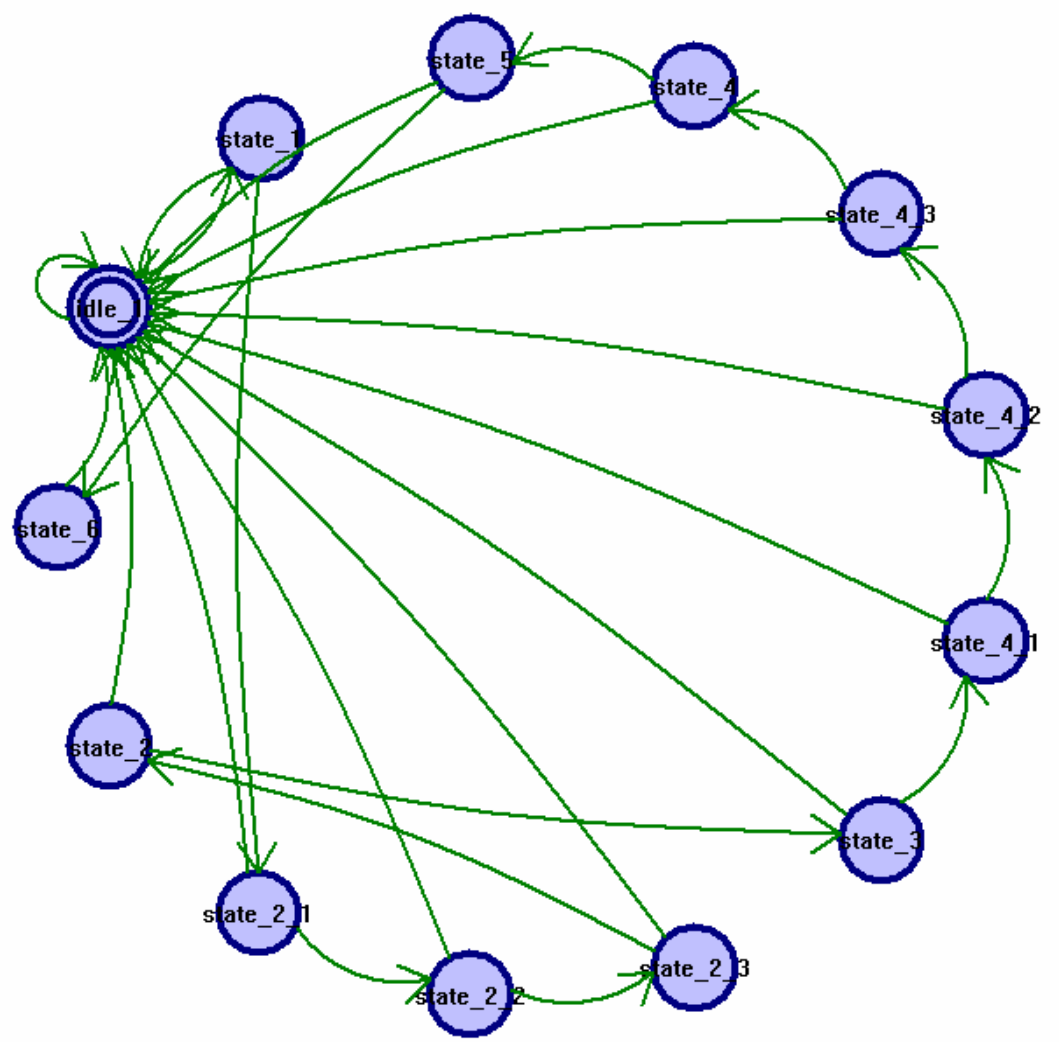


Figure B.18: FSM state transition graph. State transition graph of FSM (cup function). State 2 and 4 are assigned to the division operators and possess three sub-states 2_1, 2_2, 2_3 respectively 4_1, 4_2, 4_3. These consecutive states model the divisions as multi-cycle (4-cycle) operations.

Bibliography

[1] A.V. Aho, R. Sethi, and J.D. Ullman. *Compiler: Principles, Techniques and Tools*. Addison-Wesley, 1986.

[2] T.D. Albright, T.M. Jessell, E.R. Kandel, and M.I. Posner. A century of progress and the mysteries that remain. *Cell and Neuron*, 1(25):1–55, 2000.

[3] J.R. Armstrong. *Chip Level Modeling with VHDL*. Prentice-Hall, 1989.

[4] R. Azencott. Parallel Simulated Annealing: An Overview of Basic Techniques. In R. Azencott, editor, *Simulated Annealing, Parallelization Techniques*, chapter 4, pages 37–46. John Wiley & Sons, Inc., 1992.

[5] R. Azencott, editor. *Simulated Annealing, Parallelization Techniques*. John Wiley & Sons, 1992.

[6] R. Azencott and C. Graffigne. Parallel Annealing by Periodically Interacting Multiple Searches: Acceleration Rates. In R. Azencott, editor, *Simulated Annealing, Parallelization Techniques*, chapter 6, pages 81–90. John Wiley & Sons, Inc., 1992.

[7] M.A.E. Beaumont and D. Jackson. Visualising complex control flow. In *IEEE Symposium on Visual Languages*, pages 244 – 251. Proceedings, September 1998.

[8] K. Beck and D. Andres. *Extreme Programming Explained*. Addison-Wesley Professional, October 1999.

[9] N. Bergmann. A Case Study of the F.I.R.S.T. Silicon Compiler. *Third Caltech Conference on VLSI*, 1983.

[10] J. Besag. Nearest-neighbour systems and the auto-logistic model for binary data. *Journal of the Royal Statistical Society*, **34** (1):75–83, 1972.

[11] J. Besag. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society*, series B, **36** (2):192–236, 1974. (with discussion).

[12] J. Besag. Statistical analysis of non-lattice data. *The Statistician*, 24:179–195, 1975.

[13] J. Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society*, series B, **48** (3):259–302, 1986. (with discussion).

[14] J. Besag. Towards Bayesian image analysis. *Journal of Applied Statistics*, Vol. 16(3):395–407, 1989.

[15] D. Binkley and M. Harman. Results from a large-scale study of performance optimization techniques for source code analyses based on graph reachability algorithms. In *Third IEEE International Workshop on Source Code Analysis and Manipulation*, pages 203 – 212. Proceedings, September 2003.

[16] A. Blake and A. Zissermann. *Visual Reconstruction*. Series in Artifical Intelligence. The MIT Press, 1987.

[17] M.T. Bohr. Nanotechnology goals and challenges for electronic applications. *IEEE Trans. Nanotechnol.*, 1:56–62, March 2002.

[18] V. Bonaiuto, A. Maffucci, G. Miano, M. Salerno, F. Sargeni, and C. Visone. Design of a cellular nonlinear network for analogue simulation of Reaction-Diffusion PDE's. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 431–434, 2000.

[19] C.F. Borges. On the Estimation of Markov Random Field Parameters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(3):216–224, March 1999.

[20] G. Borriello. Combining Event and Data Flow Graphs in Behavioral Synthesis. In *Proceedings of the International Conference on Computer-Aided Design*, pages 56–59, 1988.

[21] R. Boute. Declarative Languages - Still a Long Way to Go. In *Proceedings of the 10th International Symposium on Computer Hardware Description Languages and their Applications*. Invited Paper, 1991.

[22] R.K. Brayton, R. Camposano, G. De Micheli, R.H.J.M. Otten, and J.T.J. van Einjndhoven. *The Yorktown Silicon Compiler System*, chapter in D.D. Gajski, Editor, Silicon Compilation. Addison-Wesley, 1988.

[23] Pierre Brémaud. *Markov Chains, Gibbs Fields, Monte Carlo Simulation, and Queues*. Number 31 in Texts in Applied Mathematics. Springer-Verlag, 1999.

[24] D. Brook. On the distinction between conditional probability and joint probability approaches in the specification of nearest-neighbourhoodsystems. *Biometrika*, 51:481–483, 1964.

[25] A.S. Brown. Flat, Cheap, and under control. *IEEE Spectrum*, pages 34–39, January 2005.

[26] Cadence. Inc. http://www.cadence.com.

[27] R. Camposano. Path-based scheduling for synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(1):85 – 93, January 1991.

[28] R. Camposano and W. Rosenstiel. Synthesizing Circuits from Behavioral Descriptions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8:171–180, 1989.

[29] R. Camposano, L.F. Saunders, and R.M. Tabet. VHDL as input for high-level synthesis. *IEEE Design & Test of Computers*, 8:43–49, 1991.

[30] R. A. Carmona, G. Linan, R. Dominguez-Castro, S. Espejo, and A. Rodriguez-Vázquez. SIRENA: A CAD Environment for Behavioural Modelling and Simulation of VLSI Cellular Neural Network Chips. *Int. J. Circuit Theory and Applications - Special Issue: Theory, Design and Applications of Cellular Neural Networks: Part II: Design and Applications*, 27:43–76, 1999.

[31] V. Cerny. Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm. *J. Opt. Theory Appl.*, 45(1):41–51, January 1985.

[32] N. Chabini and W. Wolf. Unification of Scheduling, Binding, and Retiming to Reduce Power Consumption Under Timing and Resources Constraints. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(10):1113–1126, October 2005.

[33] Bernard Chalmond. *Modeling and Inverse Problems in Image Analysis*. Number 155 in Applied Mathematical Sciences. Springer-Verlag, 2003.

[34] L.O. Chua and I. Yang. Cellular neural networks: Applications. *IEEE Transactions on Circuits and Systems*, 35:1273–1290, October 1988.

[35] L.O. Chua and I. Yang. Cellular neural networks: Theory. *IEEE Transactions on Circuits and Systems*, 35:1257–1272, October 1988.

[36] N. Collins, R. Eglese, and B. Golden. Simulated Annealing - an annotated bibliography. *American Journal of Mathematical and Managment Science*, 8:209–308, 1988.

[37] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms -VI Graph Algorithms*. MIT Press, 1994.

[38] F. Curatelli, L. Mangeruca, and M. Chirico. S-CFG: a representation model for system synthesis. In *International Symposium on Signals, Systems, and Electronics*, pages 326 – 331. ISSSE 98, September-October 1998.

[39] A.P. Dempster, N.M. Laired, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society B*, 39:1–38, 1977.

[40] Reinhard Diestel. *Graphentheorie*. Springer-Verlag, second edition, July 2000.

[41] R. Dominíguez-Castro, S. Espejo, A. Rodríguez-Vázquez, I. García-Vargas, J. F. Ramos, and R. A. Carmona. SIRENA: A Simulation Environment for CNNs. In *Proceedings of IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, pages 417–422. Proceedings, 1994.

[42] P. Donaldson. Collision Avoidance Systems Mature. *Defence Helicopters*, pages 41–45, April/May 2005.

[43] J. E. Dowling. *The Retina - An Approachable Part of the Brain*. Harvard University Press, 1987.

[44] T. Roska et al. A digital multiprocessor hardware accelerator board for cellular neural networks. *Int. J. Circuit Theory*, 20:589–599, 1992.

[45] European Conf. on Circuit Theory and Design. *Special Session on Cellular Neural Networks*, September 1991.

[46] M. Fowler. *Refactoring - Improving the Design of Existing Code*. Addison-Wesley, 1999.

[47] D.J. Frank, R.H. Dennard, E. Nowak, P.M. Solomon, Y. Taur, and H.-S. P. Wong. Device scaling limits of Si MOSFET's and their application dependencies. *Proc. IEEE*, 89:259–288, March 2001.

[48] D.G. Fritz and R.G. Sargent. An overview of hierarchical control flow graph models. In *Simulation Conference Proceedings, 1995. Winter*, pages 1347 – 1355, December 1995.

[49] D. Gamerman. *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*. Chapman & Hall/CRC, 1st edition, 1997.

[50] E. Gamma, R. Helm, and R.E. Johnson. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, July 1997.

[51] D.D. Gasjski, N.D. Dutt, A.C-H. Wu, and S.Y-L. Lin, editors. *High-Level Synthesis: Introduction to Chip and System Design*. Kluwer Academic, 1992.

[52] M. Gasteier and M. Glesner. Bus-based communication synthesis on system-level. In *Proc. 9th Int. Symp. Syst. Synthesis*, pages 65–70, November 1996.

[53] Zhiguo Ge, Jirong Liao, and Weng-Fai Wong. Compiling to FPGAs via an EPIC compiler's intermediate representation. In *IEEE International Conference on Field-Programmable Technology (FPT)*, pages 431 – 434. Proceedings, December 2003.

[54] D. Geman. Stochastic model for boundary detection. *Image and Vision Computing*, 5:61–65, 1987.

[55] D. Geman, S. Geman, and C. Graffigne. Locating texture and object boundaries. In P. A. Devijver and J. Kittler, editors, *Pattern Recognition Theory and Applications*. Springer Verlag, Heidelberg, 1987.

[56] D. Geman, S. Geman, C. Graffigne, and P. Dong. Boundary detection by constrained optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-12(7):609–628, July 1990.

[57] D. Geman and B. Gidas. Image analysis and computer vision. In *Spatial Statistics and Digital Image Analysis*, chapter 2. National Academy Press, Washington, D.C., 1991.

[58] D. Geman and G. Reynolds. Constrained restoration and the recovery of discontinuities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-14(No. 3):367–383, 1992.

[59] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-6(1):721–741, 1984.

[60] Sabih H. Gerez. *Algorithms for VLSI Design Automation*. Wiley and Sons, 2002.

[61] G.R. Grimmet. A theorem about random fields. *Bull. Lond. Math. Soc.*, 5:81–84, 1973.

[62] M. M. Gupta and G. K. Knopf, editors. *Neuro-Vision Systems*. IEEE Press, 1994.

[63] Sumit Gupta, Nick Saoiu, Nikil Dutt, Rajesh Gupta, and Alex Nicolau. Using Global Code Motions to Improve the Quality of Results for High-Level Synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(2), February 2004.

[64] X. Guyon. *Random Fields on a Network, Modeling, Statistics and Applications*. Probabilities and its Application. Springer-Verlag, 1991.

[65] B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13(2):311–329, 1988.

[66] M. Hanggi, S. Moser, E. Pfaffhauser, and G. S. Moschytz. Simulation and visualization of CNN dynamics. *Int. Journal of Bifurcation and Chaos*, 9:1237–1261, 1999.

[67] W.K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrica*, 57:97–109, 1970.

[68] L. Hermes. *Entropy-Based image segmentation and its application to remote sensing*, volume 727 of *Fortschritt Berichte VDI Reihe 10*. VDI Verlag, 2003.

[69] L. Hermes, T. Zöller, and J.M. Buhmann. Parametric Distributional Clustering for Image Segmentation. In *Computer Vision - ECCV 2002*, pages 577–591. LNCS 2352, Springer, 2002.

[70] T. Hofmann. *Data Clustering and Beyond - A Deterministic Annealing Framework for Exploratory Data Analysis*, volume D98. Shaker Verlag, computer science edition, 1997.

[71] T. Hofmann, J. Puzicha, and J.B. Buhmann. Unsupervised texture segmentation in a determinisitc annealing framework. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):803–818, August 1998.

[72] A. Holmes-Siedle and L. Adams. *Handbook of Radiation Effects*. Oxford University Press, Oxford, 1993.

[73] C. Huang, S. Ravi, A. Raghunathan, and N.K. Jha. Generation of Distributed Logic-Memory Architectures Through High-Level Synthesis. *IEEE Transactions on Computer-Aided Desing of Integrated Circuits and Systems*, 24(11):1694–1711, November 2005.

[74] IEEE. *Proceedings CNNA 1990,1992, IEEE International Workshop on Cellular Neural Networks and their Applications*, 1990,1992.

[75] IEEE. *Proceedings CNNA 1994, IEEE International Workshop on Cellular Neural Networks and their Applications*, 1994.

[76] IEEE. *Proceedings CNNA 1996, IEEE International Workshop on Cellular Neural Networks and their Applications*, 1996.

[77] IEEE. *Proceedings CNNA 1998, IEEE International Workshop on Cellular Neural Networks and their Applications*, 1998.

[78] IEEE. *Proceedings CNNA 2000, 2002, 2004, IEEE International Workshop on Cellular Neural Networks and their Applications*, 2000, 2002, 2004.

[79] IEEE. *Proceedings CNNA 2005, IEEE International Workshop on Cellular Neural Networks and their Applications*, 2005.

[80] Trends in Neurosciences. TINS, 1997-2004.

[81] The Open SystemC Initiative. SystemC Community. http://www.systemc.org, 2004.

[82] R.D. Isaac. The future of CMOS technology. *IBM J. Res: Develop.*, 44:369–378, May 2000.

[83] *ITRS 1999. International Technology Roadmap for Semiconductors. Available: http://public.itrs.net/files/1999_SIA_Roadmap/.*

[84] *ITRS 2003. International Technology Roadmap for Semiconductors. Available: http://public.itrs.net/Files/2003ITRS/Home2003.htm.*

[85] A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[86] S. Jankowski, R. Buczynski, A. Wielgus, W. Pleskacz, T. Szoplik, I. Veretennicoff, and H. Thienpont. Digital cnn with optical and electronic processing. In *Proceedings of 14 European Conference on Circuit Theory and Design*, pages 1183–1186. Proceedings, 1999.

[87] S. Jankowski, A. Wielgus, W. Pleskacz, B. Buczynski, and M. Wisniewski. Ic design of 8x8 digital cnn with optoelectronic interface. In *Proceedings of IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, pages 431–436, 2000.

[88] E. Jaynes. Information theory and mechanics II. *Physical Review*, 108(2):171–190, 1957.

[89] E. Jaynes. Information theory and statistical mechanics. *Physical Review*, 106(4):620–630, 1957.

[90] M.I. Jordan, editor. *Learning in Graphical Models*. MIT Press, 1999.

[91] Eric R. Kandel, James H. Schwarz, and Thomas M. Jessell, editors. *Principles of Neural Science*. McGraw-Hill, 4th edition, 2000.

[92] P. Keresztes, Á. Zarándy, T. Roska, P. Szolgay, T. Bezák, T. Hídvégi, P. Jónás, and A. Katona. An emulated digital cnn implementation. *Journal of VLSI Signal Processing Special Issue: Spatiotemporal Signal Processing with Analogic CNN Visual Microprocessors*, 23:291–304, 1999.

[93] R.W. Keyes. Fundamental limits of silicon technology. *Proc. IEEE*, 89:227–239, March 2001.

[94] A. Khachaturyan, S. Semenovskaya, and B. Vainshtein. The Thermodynamical Approach to the Structure Analysis of Crystals. *Acta Cryst.*, A(37):742–754, 1981.

[95] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by Simulated Annealing. Technical report, IBM Research Report RC 9355, 1982.

[96] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[97] C. Koch and H. Li (editors). *Vision Chips: Implementing Vision Algorithms with Analog VLSI Circuits*. IEEE Computer Society Press, 1995.

[98] J.-T. Kong. CAD for Nanometer Silicon Design Challanges and Success. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(11):1132–1147, November 2004.

[99] Bernhard Korte and Jens Vygen. *Combinatorial Optimization - Theory and Algorithms*. Springer-Verlag, 2nd edition, October 2001.

[100] K. R. Krieg and L. O. Chua. ASIM, an Efficient Simulation Environment for Cellular Neural Networks. In *Proceedings of IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, pages 149–15. Proceedings, 1990.

[101] S. Kullback and R.A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86, 1951.

[102] S. Kullback and R.A. Leibler. *Information Theory and Statistics*. John Wiley, New York, 1959.

[103] K. Kuratowski. Sur le problème des courbes gauches en topologie. *Fund. Math.*, 15:271–283, 1930.

[104] C. C. Lee and J. Pineda de Gyvez. Single-Layer CNN Simulation. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 217–220, 1994.

[105] S. Lee and M.M. Crawford. Unsupervised multistage image classification using hierarchical clustering with a bayesian similarity measure. *IEEE Transactions on Image Processing*, **14** (3):312–320, March 2005.

[106] P. Lévy. Chaînes doubles de Markov et fonctions aléatoires de deux variables. *Académie des Sciences*, 226:53–55, 1948.

[107] S.Z. Li. *Markov Random Field Modeling in Image Analysis*. Number XIX in Computer Science Workbench. Springer Verlag, 2001.

[108] A. Loncar, R. Kunz, and R. Tetzlaff. SCNN 2000 - Part I: Basic Structure and Features of the Simulation System for Cellular Neural Networks. In *Proceedings of IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, pages 123–128. Proceedings, 2000.

[109] A. Loncar, R. Kunz, and R. Tetzlaff. SCNN 2000 - Part II: The Simulation Control System. In *Proceedings of IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, pages 129–134. Proceedings, 2000.

[110] T. P. Ma and P. V. Dressendorfer. *Ionizing Radiation Effects in MOS Devices and Circuits*. Wiley-Interscience, New York, 1989.

[111] D. MacMillen, R. Camposano, D. Hill, and T.W. Williams. An Industrial View of Electronic Design Automation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19:1428 – 1448, 2000.

[112] M.A. Mahowald and C.A. Mead. A silicon model of early visual processing. *Neural Networks*, 1:91–97, 1988.

[113] M.A. Mahowald and C.A. Mead. Silicon retina. In *Analog VLSI and Neural Systems*, pages 257–278. Addison-Wesley, 1989.

[114] Yury Makarychev. A Short Proof of Kuratowski's Graph Planarity Criterion. *Journal of Graph Theory*, 25:129–131, 1997.

[115] J. Malik, S. Belongie, T. Leung, and J. Shi. Contour and texture analysis for image segmentation. *International Journal of Computer Vision*, 43(1):7–27, 1982.

[116] David Marr. *Vision*. W.H. Freeman and Company, New York, 1982.

[117] R. Matei and L. Goras. On the Discrete Simulation of 1D CNN's. In *Proceedings of IEEE International Symposium on Signals Circuits and Systems*, pages 113–115, 1997.

[118] M.C. McFarland, A.C. Parker, and R. Camposano. High-Level Synthesis of Digital Systems. *Proceedings of the IEEE*, 78(2):301–318, February 1990.

[119] S.O. Memik, R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh. A Scheduling Algorithm for Optimization and Early Planning in High-Level Synthesis. *ACM Transactions on Desing Automation of Electronic Systems*, 10(1):33–57, January 2005.

[120] G. C. Messenger and M. S. Ash. *The Effects of Radiation on Electronic Systems*. Van Nostrand Reinhold, New York, 1992.

[121] G. C. Messenger and M. S. Ash. *Single Event Phenomena*. Kluwer Academic Publishers, New York, 1997.

[122] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and M. Teller. Equation for state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[123] N. Metropolis and S. UIam. The Monte Carlo method. *J. Amer. Statist. Assoc.*, 44:335–341, 1949.

[124] Alireza Moini. Vision chips or seeing silicon. Technical report, The University of Adelaide, March 1997.

[125] G. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), 1965.

[126] D. Mumford and J. Shah. Optimal approximation by piecewise smooth functions and associated variational problems. *Comm. Pure Appl. Math.*, 42:577–685, 1989.

[127] OMNet++. *http://www.omnetpp.org*.

[128] OPNET Technologies, Inc. *http://www.opnet.com/*.

[129] Manfred Opper and David Saad, editors. *Advanced Mean Field Methods - Theory and Practice*. Neural Information Processing Series. The MIT Press, 2001.

[130] A. Orailoglu and D.D. Gajski. Flow graph representation. In *Design Automation Conference*, 1982.

[131] A. Paasio, J. Paakkulainen, and J. Isoaho. A compact digital cnn array for video segmentation system. In *Proceedings of IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, pages 229–234, 2000.

[132] A. Paasio, J. Paakkulainen, and J. Isoaho. A compact digital cnn array for video segmentation system. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 710–713, 2000.

[133] F.C.N. Pereira, N. Tishby, and L. Lee. Distributional clustering of english words. *Metting of the Association for Computational Linguistics*, pages 183–190, 1993.

[134] M. Perko, I. Fajfar, T. Tuma, and J. Puhan. Fast fourier transform computation using a digital cnn simulator. In *Proceedings of IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, pages 230–236, 1998.

[135] M. Pincus. A Monte Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems. *Oper. Res.*, 18:1225–1228, 1970.

[136] A. Prince and P. Smolensky. Optimality: From neural networks to universal grammar. *Science*, 275:1604–1610, 1997.

[137] J. Puzicha. *Multiscale Annealing for Grouping, Segmentation and Image Quantization*, volume 601 of *Fortschritt Berichte VDI Reihe 10*. VDI Verlag, 1999.

[138] J. Puzicha, T. Hofmann, and J. Buhmann. Historgram clustering for unsupervised segmentation and image retrieval. *Pattern Recognition Letters*, 20:899–909, 1999.

[139] Qt. http://www.trolltech.com, 2005.

[140] M. Rahmouni and A.A. Jerraya. Formulation and evaluation of scheduling techniques for control flow graphs. In *European Design Automation Conference with EURO-VHDL*, pages 386 – 391. Proceedings EURO-DAC '95, September 1995.

[141] P.N. Robillard and M. Simoneau. A new control flow representation. In *Proceedings of the Fifteenth Annual International Computer Software and Applications Conference*, pages 225 – 230. COMPSAC '91, September 1991.

[142] K.K. Ryu and V.J. Mooney III. Automated bus generation for multiprocessor soc design. *IEEE Transactions on Computer -Aided Desing of integrated Circuits and Systems*, 23(11):1531–1549, November 2004.

[143] M. Zhang S. Vassiliadis and J.G. Delgado-Frias. Elementary Function Generators for Neural-Network Emulators. *IEEE Transactions on Neural Networks*, 11:1438–1449, November 2000.

[144] P. Salamon, P. Sibani, and R. Frost. *Facts, Conjectures, and Imporvements for Simulated Annealing*. Monographs on Mathematical Modeling and Computation. SIAM (Society for Industrial and Applied Mathematic), October 2002.

[145] E. Schikuta. Data Parallel Software Simulation of Cellular Neural Networks. In *Proceedings of IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, pages 267–272, 1996.

[146] Robert Sedgewick. *Algorithms in C++, Part 5 Graph Algorithms*. Addison-Wesley, 2002.

[147] C. Shannon. A mathematical theory of communication. *Bell System Tech. Journal*, 27:379–423,623–659, 1948.

[148] P. Smolensky. *Information processing in dynamical systems: Foundations of harmony theory*, volume 1: Foundations of *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pages 194–281. MIT Press / Bradford Books, 1986.

[149] P. Smolensky. *The Harmonic Mind*. MIT Press, 2005.

[150] P. Smolensky and M.S.Riley. Harmony theory: Problem solving, parallel cognitive models, and thermal physics. Technical report, Institute for Cognitive Science, University of California at San Diego, April 1984.

[151] S.C. Stilkerich. *Behavioral generic hardware-description of statistical models for low-level image processing problems (in german)*, volume 16. GMD Research Series, ISBN 3-88457-365-9, 1999.

[152] S.C. Stilkerich. Generic Simulation- and Development-Environment for Massively Parallel Markov Random Field VLSI-Implementations. In *Proceedings of the 2003 IEEE Third International Workshop on Spectral Methods and Multirate Signal Processing. SMMSP '03*, 13-14 September 2003.

[153] S.C. Stilkerich. MRF-Simulation- and MRF-SoC-Development System for Massive Parallel Digital Processing Architectures. In *Proceedings of the 2003 International Conference on Signal Processing and Embedded Systems. ISPC/GSPx '03*, 31 March - 3 April 2003.

[154] S.C. Stilkerich. Massively Parallel System-On-Chip Grid-Architecture of Statistical Image-Processing Models. In *Proceedings of the 2004 International Embedded Systems Conference. GSPx '04*, 27-30 September 2004.

[155] S.C. Stilkerich. Graph Theoretical Modelling and VLSI Design Framework for Entropy based Signal Processing Models. In *Proceedings of the 2005 International Embedded System Conference. GSPx '05*, 24-27 October 2005.

[156] S.C. Stilkerich. Simulation-Framework for Purely Digital CNN/MRF-Architectures. In *Proceedings of the 2005 IEEE International Workshop on Cellular Neural Networks and their Application. CNNA '05*, pages 94–97, 28-30 May 2005.

[157] S.C. Stilkerich. Graph Theoretical Representation of Grid-based ANN Architectures for VLSI Implementations. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computing. CEC '06. Held at the 2006 IEEE World Congress on Computational Intelligence. WCCI '06*, 16-21 July 2006.

[158] S.C. Stilkerich. On the Hardware-Relevant Simulation of Regular Two-Dimensional CNN Processing Grids. In *Proceedings of the 2006 IEEE International Joint Conference on Neural Networks. IJCNN '06. Held at the 2006 IEEE World Congress on Computational Intelligence. WCCI '06*, 16-21 July 2006.

[159] S.C. Stilkerich and J.K. Anlauf. High-Level Design Environment for Massive Parallel VLSI-Implementations of Statistical Signal- and Image Processing Models. In *Proceedings of the 2004 IEEE International Symposium on Circuits and Systems. ISCAS '04*, pages 37–40 Vol. 3, 23-26 May 2004.

[160] S.C. Stilkerich and J.M. Buhmann. Massively Parallel Architecture for an Unsupervised Segmentation Model. In *Proceedings of the 2004 IEEE International Conference on Signal and Electronic Systems. ICSES '04*, 13-15 September 2004.

[161] S.C. Stilkerich and R. Reiger. On the Simulation and Development of Massively Parallel Digital Architectures for Markov Random Fields. In *Proceedings of the 2004 IEEE International Conference on Acoustic, Speech & Signal Processing. ICASSP '04*, 17-21 May 2004.

[162] C. Thomassen. Kuratowski's theorem. *Journal of Graph Theory*, 5:225–241, 1981.

[163] M.K. Unaltuna, M.E. Dalkilic, and V. Pitchumani. Solving the scheduling problem in high level synthesis using a normalized mean field neural network. *IEEE International Conference on Neural Networks*, Proceedings:275 – 280, March-April 1993.

[164] Gabriel Valiente. *Algorithms on Trees and Graphs*. Springer-Verlag, July 2002.

[165] P. J. M. van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. Mathematics and Its Applications. D. Reidel Publishing Company, 1987.

[166] G. Winkler. *Image Analysis, Random Fields and Dynamic Monte Carlo Methods*. Number 27 in Applications of Mathematics. Springer-Verlag, 1995.

[167] G. Winkler. *Image Analysis, Random Fields and Dynamic Monte Carlo Methods*. Number 27 in Applications of Mathematics. Springer-Verlag, 2003.

[168] Xerces. *http://xml.apache.org/xerces-c/*.

[169] T. Zöller, L. Hermes, and J.M. Buhmann. Combined Color and Texture Segmentation by Parametric Distributional Clustering. In *Proceedings of the IEEE International Conference on Pattern Recognition, ICPR 2002*, pages 627–630. IEEE Computer Society, 2002.

Index

Acyclic, 147, 151
Annealing, 15
Architectural Building Blocks
  Energy Functional, 60
  Energy Functional Control Units, 69–70
  Global Memory Hierarchy, 57–59
  Memory Hierarchy Control Unit, 67–68
  Optimization, 61–62
  Optimization Control Units, 70–71
  Parameter Estimation, 62–64
    Schemes, 63
  Port Memory, 55–57
    Arrangement, 56
  Port Memory Control Units, 68–69
  Site Hull, 51–52
    Grid Coordinates, 52
    VLSI Shape, 52
  Site Wiring, 53–55
    Preferred Direction, 53, 54
  System Control Unit, 65–67

Behavioral Synthesis, 136
Building Blocks
  Control, 64–71
  Processing, 59–64
  Topology & Structure, 50–59

Canonical Design Representation, 137, 141–144
Control Flow Graph, 158
Control Machine
  Energy Functional, 175
  Memory Hierarchy, 173
  Optimization, 175
  Parameter Estimation, 175
  Port Memory, 174
  System, 172
Cycle Template, 75–78
  Frame Representation, 76–77
  Graph Representation, 77–78

Data Container, 144–145
Data Flow Graph, 155
  Energy Functional, 155
  Optimization, 156
  Parameter Estimation, 156
Design Flow, 161–176
Design Framework
  Arrangement, 137
  Back-End Code Generator, 141
  Canonical Design Representation, 141
  Control-Path Generator, 140
  Data Container, 144–145
  Data- & Control Path Extractor, 139
  Data-Path Generator, 140
  FSM Embedding, 141
  Memory Hierarchy Generator, 140
  Parser Front End, 139
  Scheduler, 140
  Structure & Topology Extractor, 139
  Structure & Topology Generator, 140
Design Gap, 136
Design Graph, 160
Design Methodology
  Behavioral Synthesis, 136
  High-Level Synthesis, 136
  RTL Synthesis, 136

Exemplary Architectures
  Edge Preserving & Noise Removing, 84–87
    $\mathcal{BB}$ Resources, 85
  Unsupervised Histogram Segmentation, 87–90
    $\mathcal{BB}$ Resources, 88
Extractor

Data- & Control Path, 139
Structure & Topology, 139

Finite State Machine
Energy Functional, 175
Memory Hierarchy, 173
Optimization, 175
Parameter Estimation, 175
Port Memory, 174
System, 172

Generator
Control-Path, 140
Data-Path, 140, 167
Memory Hierarchy, 140, 164
Structure & Topology, 140, 162, 164
Generic $\mathcal{H}$, 15
Graph Compilation
Control, 183–185
Processing, 181–183
Synthesis Algorithm, 177, 178, 181–184
Topology & Structure, 176
Topology&Structure, 180
Graph Feature
Acyclic, 147, 151
Combinational-Logic Neutral, 149, 153
Planarity, 148, 149, 152
Topologically Sortable, 148
Graph Representation
Control Functionality, 157
Processing Functionality, 154
Topology & Structure, 145

High-Level Synthesis, 136
Design Flow, 161–176

Image Processing Model
Restoration, 24–28
Application Scenarios, 40–41
Cost Function, 27
Cup Function, 25
Update Formula, 27
Unsupervised Segmentation, 24, 28–33
Application Scenarios, 41–42
Cost Function, 31
Update Formula, 32, 33
Implementation Technology
Design Methodologies, 36–37
HDL, 36
Representation Capabilities, 37
Technology Trend, 35–36
Variants, 33–38
VHDL, Verilog, 36

Markov Random Field
Cliques, 11–12
Equivalence Theorem, 14
Free Energy, 13
Generic $\mathcal{H}$, 15
Gibbs Distribution, 13
Neighborhood System, 11–12
Potential, 13
Maximum Entropy, 15

Optimization Scheme
Deterministic Annealing, 19
Energy Landscape, 15
Gibbs Sampler, 18
Heat-Bath Criterion, 18
Iterated Conditional Modes, 19
Metropolis Criterion, 17
Simulated Annealing, 16
Site Visitation, 16
Oversized Images, 79–84
System Variants, 79–80

Parallel Processing
Convergence, 20–22
Degree of Parallelism, 22
Independent Sets, 21
Local Characteristic, 20
Site Partitioning, 21
Site Sweep, 21
Planarity, 149, 152
Prototype Module
Cell, 102
Cell-Cluster, 104
Control, 114
Energy Functional, 110
Optimization Method, 111
Parameter Estimation, 111
Wiring, 106

Scheduling

ALAP, 170
ASAP, 169
Sequence, 168
Step, 168
Simulation Framework
Data Storage System, 99
Display & Analysis Monitor, 99
Overview, 95
Simulation-Model Generator, 98
Supporting Modules, 100
SystemC, 95
Simulation Modules
Cell & Cell-Cluster, 97, 102, 104
Control, 114
Control Functionality, 113–116
Energy Functional, 98, 110
Optimization Method, 98, 111
Parameter Estimation, 111
Processing Functionality, 108–113
Topology & Structure, 100–106
Wiring, 97, 106
Simulation Results, 211
Supporting Modules
Control GUI, 99
Frame-Cell Modules, 97
Framer Modules, 98
Interface Modules, 98
System-Architecture Template
$\mathcal{BB}$ Resources, 85, 88
Architectural Building Blocks, 49–71
Cycle Scheme, 75–78
Exemplary Architectures, 84–90
Oversized Images, 79–84
Universal Constituents, 47–49

Thermal Equilibrium, 15

Universal Constituents
Central Equation, 48
Constituents, 48

VLSI Results, 251

PUBLICATIONS

Stephan C. Stilkerich. Graph Theoretical Representation of Grid-based ANN Architectures for VLSI Implementations. *Special Issue on Hardware Architectures for Neural Networks. Elsevier Journal on Neurocomputing.* (invited journal contribution of extended CEC'06 conference paper, accepted)

Stephan C. Stilkerich. Graph Theoretical Representation of Grid-based ANN Architectures for VLSI Implementations. *Special Session on Hardware Architectures for Genetic, Neural and Fuzzy Systems. In Proceedings of the 2006 IEEE Congress on Evolutionary Computing.* CEC'06, Vancouver, British Columbia, Canada, 16-21 July 2006.

Stephan C. Stilkerich. On the Hardware-Relevant Simulation of Regular Two-Dimensional CNN Processing Grids. *Special Session on Cellular Sensory Wave Computers. In Proceedings of the 2006 IEEE International Joint Conference on Neural Networks.* IJCNN'06, Vancouver, British Columbia, Canada, 16-21 July 2006.

Stephan C. Stilkerich, P. Dunn, N. Harold, C. Petri and D. Stark. High Performance FPGA Computing Platform for a Closed-Loop Flight Control Turbulence Detection System. *In Proceedings of the 2005 International Embedded System Conference.* GSPx'05, Santa Clara, CA, USA, 24-27 October, 2005.

Stephan C. Stilkerich. Graph Theoretical Modelling and VLSI Design Framework for Entropy based Signal Processing Models. *In Proceedings of the 2005 International Embedded System Conference.* GSPx'05, Santa Clara, CA, USA, 24-27 October, 2005.

Stephan C. Stilkerich. Simulation Framework for Purely Digital CNN/MRF Architectures. *In Proceedings of the 2005 IEEE International Conference on Cellular Neural Networks and their Applications.* CNNA'05, pages 94-97, Hsinchu, Taiwan, May 28-30, 2005. (invited contribution)

Stephan C. Stilkerich and J.M. Buhmann. Massively Parallel Architecture for an Unsupervised Segmentation Model. *In Proceedings of the 2004 IEEE International Conference on Signals and Electronic Systems.* ICSES'04, Poznan, Poland, 13-15 September 2004.

Stephan C. Stilkerich. Massively Parallel System-On-Chip Grid-Architecture of Statistical Image-Processing Models. *In Proceedings of the 2004 International Embedded Systems Conference.* GSPx'04, Santa Clara, CA, USA, 27-30 September 2004.

Stephan C. Stilkerich and J.K. Anlauf. High-Level Design Environment for Massive Parallel VLSI-Implementations of Statistical Signal- and Image Processing Models. *In Proceedings of the 2004 IEEE International Symposium on Circuits and Systems.* ISCAS'04, Vancouver, British Columbia, Canada, 23-26 May 2004.

Stephan C. Stilkerich and R. Reiger. On the Simulation and Development of Massively Parallel Digital Architectures for Markov Random Fields. *In Proceedings of the 2004 IEEE International Conference on Acoustic, Speech & Signal Processing.* ICASSP'04, Montreal, Quebec, Canada, 17-21 May 2004.

Stephan C. Stilkerich. Generic Simulation- and Development-Environment for Massive Parallel Markov Random Field VLSI-Implementations. *In Proceedings of the 2003 IEEE Third International Workshop on Spectral Methods and Multirate Signal Processing.* SMMSP'03, Barcelona, Spain, 13-14 September 2003.

PUBLICATIONS

Stephan C. Stilkerich. MRF-Simulation- and MRF-SoC-Development System for Massive Parallel Digital Processing Architectures. *In Proceedings of the 2003 International Signal Processing Conference.* ISPC'03/GSPx'03, Dallas, Texas, USA, 31 March - 3 April 2003.

Stephan C. Stilkerich. Behavioral hardware-description of statistical models for low-level image processing problems. (in german), Vol. 16 in GMD Research Series, GMD, 1999.

ADDITIONAL PUBLICATIONS

Stephan C. Stilkerich. Reconfigurable Computing - An Alternative Architecture Paradigm for Space Applications. *In Proceedings of the 2006 International Embedded System Conference.* GSPx'06, Santa Clara, CA, USA, October 30- November 2, 2006. (**in press**)

N. Schmitt, P. Zeller, W. Rehm, S. C. Stilkerich, K. Schertler, H. Zinner and H. Diehl. The AWIATOR Airbrone LIDAR Turbulence Sensor. *In Proceedings of DGLR Deutscher Luft- und Raumfahrtkongress.* Friedrichshafen, Bavaria, Germany, September 26-29, 2005.

N. Schmitt, S. C. Stilkerich, K. Schertler, H. Zinner, P. Zeller and H. Diehl. The AWIATOR Airbrone LIDAR Turbulence Sensor. *In Proceedings of the 2005 IEEE Conference on Laser and Electro-Optics / European Quantum Electronics.* CLEO/EQEC 2005, Munich, Bavaria, Germany, June 12-17, 2005.

H. Zinner, N. Schmitt, S. C. Stilkerich and P. Zeller. Flight test of a short pulse UV Doppler LIDAR for Turbulence Detection. *Proceedings of the International Optronics Symposium.* OPTRO 2005, Paris, France, May 9-12, 2005.

H. Zinner, H. Diehl. T. Halldorsson, T. Pistner, W. Rehm, K. Schertler, N. Schmitt, S. C. Stilkerich, P. Zeller. Short pulse UV Doppler LIDAR for Turbulence and Wake Vortex measurement. *In Proceedings of the 2nd France-Singapure-Workshop on Optoelectronics and RF Photonics Technology.* October 27-28, 2005.