Falko Guderian

Developing a Design Flow for Embedded Systems

Beiträge aus der Informationstechnik

Mobile Nachrichtenübertragung
Nr. 61

Falko Guderian

Developing a Design Flow for Embedded Systems

VOGT

Dresden 2013

Bibliografische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

Bibliographic Information published by the Deutsche Bibliothek
The Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibiograpic data is available in the internet at http://dnb.ddb.de.

Zugl.: Dresden, Techn. Univ., Diss., 2013

Die vorliegende Arbeit stimmt mit dem Original der Dissertation „Developing a Design Flow for Embedded Systems“ von Falko Guderian überein.

Gesetzt vom Autor

ISBN 978-3-938860-61-8

Jörg Vogt Verlag
Niederwaldstr. 36
01277 Dresden
Germany

Phone: +49-(0)351-31403921
Telefax: +49-(0)351-31403918
e-mail: info@vogtverlag.de
Internet : www.vogtverlag.de

Technische Universität Dresden

Developing a Design Flow for Embedded Systems

Falko Guderian

der
Fakultät Elektrotechnik und Informationstechnik
der Technischen Universität Dresden

zur Erlangung des akademischen Grades

Doktoringenieur
(Dr.-Ing.)

genehmigte Dissertation

Vorsitzender: Prof. Dr.-Ing. habil. Frank Ellinger
Gutachter: Prof. Dr.-Ing. Dr. h.c. Gerhard Fettweis
Prof. Dr.-Ing. Rolf Ernst

Tag der Einreichung: 05.11.2012
Tag der Verteidigung: 24.01.2013

Danksagung (Acknowledgments)

Ich möchte die Gelegenheit nutzen, um an zahlreiche Persönlickeiten ein herzliches Dankeschön zu richten.

Der erste Dank gehört meinem Spiritus Rektor Prof. Gerhard Fettweis, welcher mir das Vertrauen schenkte, als Quereinsteiger in der mobilen Nachrichtentechnik zu promovieren. Die erstklassige Ausstattung des Lehrstuhls, hohe fachliche Kompetenz und das angenehme Miteinander waren ein Garant für den Erfolg dieser Arbeit.

Des weiteren gilt mein Dank an Stefan Krone, Rainer Schaffer, Erik Fischer, Emil Matus und die Kollegen der Hardware- und Algorithmengruppe für eine fruchtbare Zusammenarbeit in der Lehre, in Projekten und der wissenschaftlichen Arbeit. Außerdem bedanke ich mich bei Marcel Jar e Silva für seine aufopfernde Englischkorrektur. Marcel, thank you for carefully correcting the English of this thesis.

Darüber hinaus gilt der herzlichste Dank meiner Familie. Erst durch die Hingabe meiner Eltern und Fürsorge der Großeltern konnte ich genügend tiefe Wurzeln treiben, um mich der Herausforderung einer Promotion mit Neugier und Tatendrang zu stellen.

Weiterhin begleiteten mich viele gute und beste Freunde während dieser Lebensphase. Für ihre Gespräche und Ratschläge richte ich einen einzigartigen Dank an: Marco Krondorf, Martin Juhrisch, Heidi Pfütze, Stephan Preibisch, Stefan Krone und Zhijun Rong.

Abstract

This thesis covers the electronic system level (ESL) design of embedded systems motivated by the practical relevance of these systems. Thereby, the design concepts have been extended according to the state of the art addressing an early design stage. This includes a new methodology, new models, and new methods aiming at an improved time, quality, and cost for the design of embedded systems. A special focus is set to the future many-core system-on-chips (SoCs) composed of clusters, where each cluster represents a set of heterogeneous intellectual property (IP) cores. The developed methodology realizes a more systematic ESL design. Moreover, the implemented system functions are inter-dependent and integrated in the system models. Principally, a separation into computation resources, data logistic resources, and resource management is followed. In addition, the developed design methods solve complex design problems of the targeted many-core SoCs. A new programming language for the design automation is also deployed in the course of this thesis. It allows to develop, manage, and optimize design flows. Finally, the contributions of this thesis are demonstrated in a case study for the heterogeneous multicluster architecture.

The first part of this thesis introduces the new concept of an executable design flow and describes a unified methodology for the systematic development of design flows. A first example illustrates the functional design of a digital filter. Then, the modeling of systems and design flows is unified. It is shown that domain-specific design flows can also be derived from a unified abstract design flow model. The λ-chart, proposed in this thesis, represents a flow model for the ESL design of many-core SoCs. Moreover, the developed system functions of the targeted many-core SoCs are presented to be further integrated in the simulation models. In this context, two new schemes for the fair arbitration of link bandwidth in a network-on-chip have been developed. Another important contribution concerns the introduction of several design methods solving the design problems of the many-core SoCs. This includes two estimation techniques and

several formulations of optimization problems via mixed-integer linear programming and genetic algorithms.

Further important contributions relate to new automation features for ESL design. The presented programming language allows for developing design flows in terms of a computer program, used by the introduced automation tools. In addition, the simulation models, design methods, and automation functions are accessible for a wider community via a development framework. An integrated development environment combines the development of design flows with the available solutions of the design problems.

In the final part of this thesis, a case study of the heterogeneous multicluster architecture demonstrates the usage of the ESL design methodology, simulation models, and design methods under realistic conditions. First, the focus is on the search of suitable input parameters of the design methods. Then, a systematic dimensioning of the multicluster architecture will be applied in terms of the necessary computation resources. Thereafter, an adequate interconnect topology is generated. This allows to compare the available arbitration schemes concerning their impact on the system performance. In addition, a performance evaluation of several estimators of cluster load is applied. The case study shows that the design flow of future many-core SoCs includes various complex design problems. They can be efficiently solved through an ESL design avoiding the time-consuming simulations at a lower abstraction level, such as register-transfer level.

Zusammenfassung

Die vorliegende Arbeit behandelt die Thematik des Systementwurfs von eingebetteten Systemen, welche immer häufiger im technischen Kontext Anwendung finden. Dabei wurden bestehende Entwurfskonzepte nach dem Stand der Technik erweitert. Das beinhaltet neue Methodiken, Modelle und Methoden. Die Arbeit beschränkt sich auf den Entwurf neuartiger Ein-Chip-Vielkernsysteme, bestehend aus Rechenclustern, die eine verschiedenartige Menge von Rechenkernen mit separaten Speicher- und Pheripherieelementen enthalten. Die entwickelten Verfahren berücksichtigen eine Vielzahl sich gegenseitig beeinflussender Systemfunktionen und werden zusammenhängend in einem Entwurfsfluss verwendet. Die Systemfunktionen sind dabei grundsätzlich anhand der Berechnungsressourcen, Datenlogistikressourcen und des Ressourcenmanagements unterteilt. Dabei wird auf einer höheren Abstraktionsebene modelliert und die Entwurfsraumexploration findet in einem frühen Stadium statt. Im Rahmen dieser Arbeit kommt eine neuartige Ablaufsprache für den automatischen Entwurf zum Einsatz, um bedarfsgerecht Entwurfsflüsse entwickeln, verwalten und optimieren zu können. Abschließend werden die genannten Forschungsbeiträge anhand eines Anwendungsfalles für die heterogene Mehrclusterarchitectur demonstriert.

Der erste Teil der Arbeit erläutert das Konzept des ausführbaren Entwurfsflusses und beschreibt eine vereinheitlichte Methodik für die systematische Erstellung von Entwurfsflüssen. Als einführendes Beispiel dient der funktionale Entwurf eines digitalen Filters. Der anschließende Teil der Arbeit vereinheitlicht die Modellierung von Systemen und Entwurfsflüssen. Es wird gezeigt, dass konkrete Entwurfsflüsse anhand eines vereinheitlichten Entwurfsflussmodells erstellt werden können. Das im Rahmen der Arbeit entstandene λ-chart repräsentiert ein Entwurfsflussmodell für Ein-Chip-Vielkernsysteme. Im Anschluss, werden die entwickelten Systemfunktionen von Ein-Chip-Vielkernsystemen vorstellt, um als Grundlage für die angewendeten Simulationsmodelle zu dienen. In diesem Zusammenhang werden zwei neue Techniken zur fairen

Verteilung von Netzwerkbandbreite vorstellt. Die Verfahren wurden für Kommunikationsnetzwerke entwickelt, die auf einem Chip integriert sind. Eine weitere wichtige Forschungsleistung beinhaltet die Einführung mehrerer Methoden zur Lösung von Entwurfsproblemen der untersuchten Systeme. Diese beinhalten zwei Schätzverfahren und mehrere Formulierungen von Optimierungsproblemen mittels linearer Programmierung und genetischer Algorithmen.

Weitere wichtige Beiträge der Arbeit stellen neue Lösungen zur Entwurfsautomatisierung auf Systemebene dar. Die Ablaufsprache ermöglicht eine Entwicklung von Entwurfsflüssen im Sinne eines Computerprogrammes unterstützt durch Automatisierungswerkzeuge. Die entwickelten Modelle, Methoden und Automatisierungsfunktionen stehen einem breiten Nutzerkreis mittels eines Programmierungsframeworks zur Verfügung. Des Weiteren vereinigt die bereitgestellte Entwicklungsumgebung die Erstellung von Entwurfsflüssen mit der Wiederverwendung von Lösungen zu den bearbeiteten Entwurfsproblemen. Für Letztere stehen die im Framework integrierten Simulationsmodelle und die Entwurfsmethoden zur Verfügung.

Zum Ende der Arbeit werden die entwickelten Methodiken, Modelle und Methoden mittels eines Anwendungsfalles zu einer Mehrclusterarchitektur unter realistischen Bedingungen erprobt. Zu Beginn steht die Suche geeigneter Eingabeparameter für die zu verwendenden Entwurfsverfahren im Mittelpunkt. Danach erfolgt eine systematische Auslegung der Mehrclusterarchitektur hinsichtlich der notwendigen Berechnungsressourcen. Daraufhin wird eine geeignete Netzwerkstruktur generiert. Im Anschluss steht ein Vergleich der verfügbaren Arbitrierungsprotokolle hinsichtlich des Einflusses auf die Systemleistung im Vordergrund. Außerdem wird die Analyse verschiedener Lastverteilungsverfahren mit dem Ziel vorgenommen, eine gleichmäßige Auslastung der Rechencluster zu gewährleisten. Die Fallstudie zeigt, dass ein Entwurfsfluss von Ein-Chip-Vielkernsystemen vielfache Optimierungsprobleme beinhaltet, die insbesondere auf Systemebene effizient gelöst werden. Dadurch kann auf zeitintensive Simulationen auf niedrigeren Abstraktionsebenen, z.B. Registertransferebene, verzichtet werden.

Contents

List of Figures

List of Tables

List of Listings

Nomenclature

Constants

$\mathcal{N}$	Number of rows in 2D NoC or number of applications	49
M	Number of columns in 2D NoC or number of PE types	49
Q	Step width	49
K	Weighting factor	51
R	Number of routers	51
P	Number of IP cores	51
C	Total number of cluster candidates	63
$P_{\max}$	Maximum number of allowed PEs per cluster	63
Ψ_{link}	Physical link power	54
Ψ_{in}	Router input port power	54
Ψ_{out}	Router output port power	54
L	Upper limit for the total number of PEs	120
S	Smallest resolution of the parallelism value ϕ_{ij}	121

Functions

$\mathrm{dist}(i,j)$	Distance between cores i and j	54
$h(i,j)$	Number of hops between cores i and j	54
max	Maximum	51
min	Minimum	63

Matrices

Φ	Parallelism value matrix	43
$\boldsymbol{A_C}$	Communication affinity matrix	46
$\boldsymbol{A_M}$	Management affinity matrix	46
$\boldsymbol{A}$	Affinity matrix	47

$\boldsymbol{L}$	Locality matrix	48
$\boldsymbol{G}$	2D/3D matrix chromosome	52

Sets

$\mathcal{C}$	Set containing all IP cores	24
$\mathcal{M}$	Set of distance metrics	122
$\mathcal{C}_{\mathcal{M}}$	Subset of IP cores related to resource management	49
$\mathcal{C}_{\mathcal{D}}$	Subset of IP cores related to data logistic resources	24
$\mathcal{C}_{\mathcal{C}}$	Subset of IP cores related to computation resources	63

Variables

t	Time variable	42
p	Parallelism variable	42
Δt	Time uncertainty	42
ϕ_{ij}	Parallelism value of PE type j and application i	43
t_t	Transfer latency	46
t_e	Execution latency	46
d	Data value	46
d'	Weighted data value	46
$\alpha, \beta, \gamma, \omega$	Weights	46
i, j	IP core/ application/ PE type variable	51
b	Binary variable	51
r	Router variable	51
a_{ij}	Affinity value for cores i and j	51
c	Cost variable	54
s_{total}	Total chip area	53
$U_{\max}$	Maximum number of PEs among the clusters	63
$U_{\min}$	Minimum number of PEs among the clusters	63
k	Cluster candidate variable	63
P_k	Number of candidates in cluster k	63
w	Word length	66
e_{abs}	Absolute error	66
l	Locality value in locality matrix $\boldsymbol{L}$	122

τ	Pairwise difference in the locality matrix $\boldsymbol{L}$	122

Vectors

$\boldsymbol{g}$	Array chromosome	56

Abbreviations

2D	Two dimensional
3D	Three dimensional
ALAP	As latest as possible
AM	Architecture model
APM	Application model
ASAP	As soon as possible
ASIC	Application-specific integrated circuits
BL	Bottom-left
CAD	Computer aided design
CP	Control processor
cRate	Crossover rate
DFL	Design flow language
DFM	Design flow model
DoDF	Design of design flow
DSE	Design space exploration
DSP	Digital signal processor
EDA	Electronic design automation
ES	Exhaustive search
ESL	Electronic system level
FIR	Finite impulse response
GA	Genetic algorithm
Gbps	Gigabit per second
GPP	General purpose processor
GUI	Graphical user interface
HPC	High performance cluster

IC	Integrated circuit
IDE	Integrated development environment
IP	Intellectual property
IF	Interface
LP	Linear programming
Mbps	Megabit per second
MEM	Memory
MILP	Mixed integer linear programming
MPEG4	MPEG4 decoder
MPSoC	Multi-processor system-on-chip
mRate	Mutation rate
MWD	Multi-window displayer
nGen	Number of generations
NoC	Network-on-chip
PCB	Printed circuit board
PE	Processing element
pSize	Population size
PP	Packaging problem
RR	Round robin
rRate	Replacement rate
RTL	Register-transfer level
SDF	Synchronous data flow
SDL	Specification and description language
SLDL	System-level design language
SoC	System-on-chip
Tcl	Tool command language
TLM	Transaction-level model
UML	Unified modeling language
VCG	Visualization of compiler graphs
YML	Y-chart modeling language

CHAPTER 1

Introduction

1.1. Electronic System Level (ESL) Design

Since the invention of the integrated circuit (IC) in the late fifties, semiconductor technology scaling has followed Moore's law doubling the number of transistors in an IC about every two years [7]. IC-based products facilitate our daily life and IC design is one of the main forces driving technology innovation and economic growth. The number of transistors in modern IC designs have already crossed the one billion mark [8] and is approaching the ten billion mark. These systems, which combine both hardware (HW) and software (SW), are more and more embedded in a technical domain [9]. Traditionally, embedded applications, such as in multimedia, wireless communications, automotive industry, and networking, have been integrated on a printed circuit board (PCB). A PCB connects a set of discrete ICs, such as processors, memories and peripherals. The ongoing technology scaling is driving an integration of the discrete ICs, from board-level, towards system-on-chip (SoC) level. Future embedded systems are expected to evolve even further, towards many-core SoCs with hundreds or even thousands of processors [10]. Wireless communications, characterized by an ever-increasing need for bandwidth, is a possible application domain for these large-scale embedded systems. The wireless roadmap anticipates 100 gigabit per second (Gbps) short-range links, 10 Gbps mid-range links, and 1 Gbps long-range (cellular) links by the end of 2015 [11]. A recently developed solution for 60 GHz short-range communications at 10 Gbps shows the feasibility to meet the expected demand for bandwidth [12][13]. For example, shrinking this technology to a credit card size promises for new applications, such as in the medical sector [14, 15].

The increasing complexity in the design of embedded systems follows the continuously rising application requirements. On the one hand, there is an ever-increasing number of integrated system functions. On the other hand, the interaction between the numerous system components results in a further complexity increase. As a result, the design time is stretched by such an almost exponential growth of the system complexity. Moreover, the design process gets even more challenging by considering the tight constraints and market pressures in terms of short time-to-market, costs, reliability, etc. To address this issue, novel electronic design automation (EDA) approaches are required in order to improve both the design time, quality, and costs in future embedded systems [16].

It is commonly accepted by all major semiconductor roadmaps that, only by raising the design process to higher levels of abstraction, will designers be able to cope with the existing design challenges. This leads to an *electronic system level* (ESL) design methodology. The term *system level* refers to the use of abstract system functions in order to improve comprehension about a given system. ESL design aims at a seamless transformation of a system specification into an HW/SW implementation [17]. Hence, EDA requires a system specification that can be executed in a computer simulation. An *executable specification* is a simulation model of the intended system functions, also called a virtual prototype [9][18].

Today's ESL *design flows*, referred to as *flows*, are typically based on a specify-explore-refine methodology [1]. Such flows include a sequence of design steps, referred to as *steps*, successively refining a system model. Each step solves a design problem, such as an application mapping. Moreover, a *specification model* defines the starting point representing the targeted application characteristics and requirements. According to [17], "Specification model is used by application designers to prove that their algorithms work on a given system platform". Following, each exploration step creates a design decision that continuously increases the accuracy of the system model. Thereafter, the refined model is passed to the next exploration step. Recently developed ESL design environments, as proposed in the MULTICUBE project [19] and NASA framework [20], back away from an ad-hoc software infrastructure. Their generic EDA environments provide modularization and well-defined interfaces. Despite these advancements, the problem of a large number of possible flow sequences has not been addressed yet.

Since future many-core SoCs imply an increasing design complexity, the number of steps in the design process (flow) also increases. For example, optimization of the resource management requires additional design tasks [2]. Furthermore, the huge design space will draw more attention to the ESL design at an early design stage. Consequently, suitable system models, simulation models, and design methods, referred to as *methods*, need to be developed. In addition, a systematic methodology to develop, manage, and optimize flows promises a significantly improved design process. In this thesis, this approach is denoted as the *design of design flow* (DoDF). Similar methodologies have been developed in other scientific fields, such as physics [21], mechanical engineering [22], and software engineering [23]. The challenge is to integrate the models, methods, and DoDF approach into an EDA environment.

1.2. A System-on-Chip (SoC) Design Flow

In the following, the author discusses the question of how to develop an ESL flow for a SoC's example. The example assumes a single-core SoC comprising a processor, a global memory, and a bus connecting processor and memory. The starting point is a specification model in terms of a functional description of the target application, as seen in Figure 1.1. The final result is a transaction-level model (TLM), where suitable IP cores, the network structure, communication, and management behaviour have been explored and refined. In order to design the system, each component has to be optimized and the performance needs to be validated against given constraints, such as latency, power consumption, area, and costs. It is important to note that optimizing each component separately and connecting the explored components does not necessarily mean that the system itself is optimal. Hence, all interacting components have to be covered ensuring a system-level optimization.

A problem that arises when considering the entire system design as a single problem is its complexity. For example, optimization of a multiprocessor scheduling, being part of an application *mapping*, is known to be an NP optimization problem [24]. In this thesis, *scheduling* denotes a temporal planning of operations accessing the resources. Due to the problem's complexity, the design is divided into subproblems. Consequently, the sub-flows are connected in order to realize a complete flow. In Figure 1.1, the SoC flow follows a typical specify-explore-refine methodology as introduced in [1]. The flow consists of exploration and refinement tasks applying methods, such as estimation,

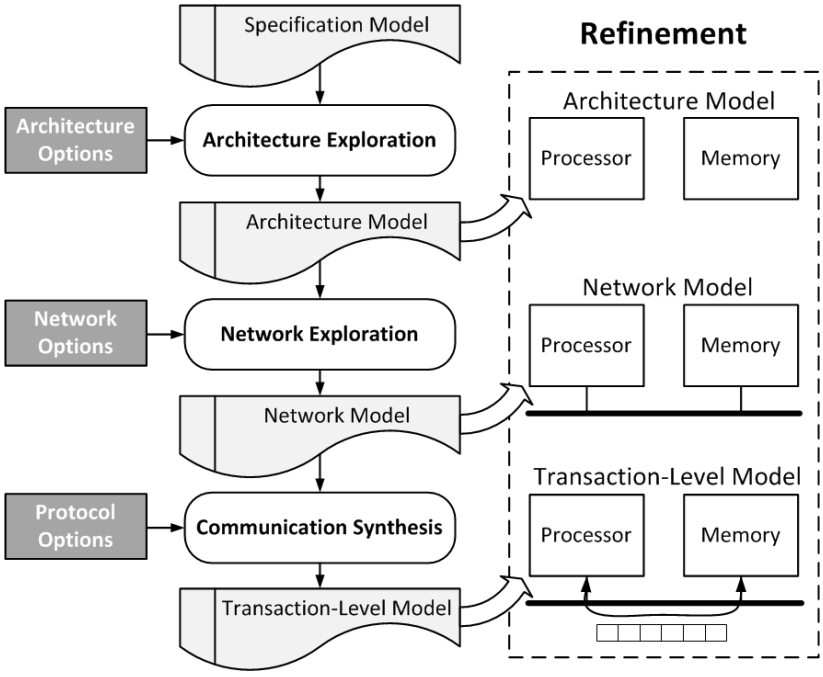

Figure 1.1.: An SoC design flow following the specify-explore-refine methodology [1]

optimization, simulation, etc. First of all, an architecture model is built from a specification model and architecture options, such as processor type, memory type, etc. The second step is to explore a network model refining the architecture model with network elements, such as buffers, links, etc. Finally, communication synthesis is applied, deriving an accurate TLM by including communication protocols, such as routing, switching, and arbitration mechanisms.

The SoC flow, shown in Figure 1.1, has been presented in an abstracted manner. In practice, the exploration and synthesis steps are rather sub-flows built from more fine-grained steps. In general, developing a flow requires one to define methods, to select and arrange steps, to define inputs/outputs for the steps, etc. Moving from the SoC to a many-core SoC, the design space increases with the number of components and their interactions. The resultant large complexity can only efficiently be handled by introducing more methods and steps. Another challenge is introduced by alternative flow configurations. For example, one can start with a network exploration assuming a given network traffic. Furthermore, an exploration strategy, which was left out in the example, can also be carried. This would require the choice of a search method, design objective, selection criteria, etc. From the SoC flow example, seen in Figure 1.1, it can be concluded that the DoDF is a challenging task which can be relaxed by providing

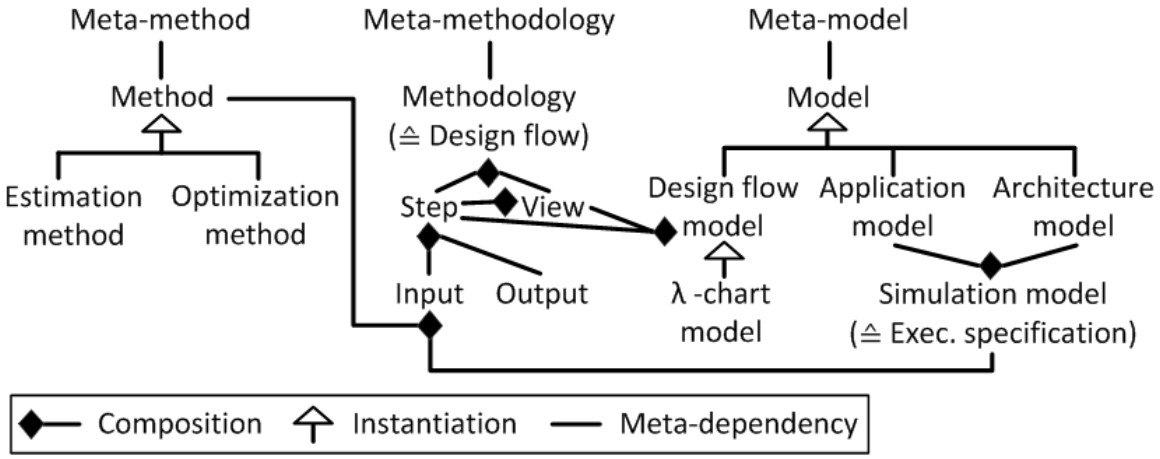

Figure 1.2.: Classification and relationships of methodology, model, and method

a systematic approach. Therefore, a unified methodology to create, modify, and reuse flows is highly desirable.

1.3. Terminology

This section provides a terminology for the classification as well as the relationships of the terms used for the development of ESL flows. This is illustrated in the Figure 1.2. The classification relates to the *methodology*, *model*, and *method* and will be explained in the following. Composition refers to an element which is part of another element. Instantiation means that an element is derived from another element. Moreover, the term *meta* is used in order to describe an abstraction of a subject. An example is the meta-data which means data about data. Referring to Figure 1.2, a meta-dependency characterizes such an abstraction.

In the subject of the methodology, the following terms are defined:

- ▷ meta-methodology
- ▷ methodology (flow)
- ▷ view
- ▷ step
- ▷ input / output

The *meta-methodology* defines a methodology realizing another methodology. In Section 2.3, a meta-methodology for the development of flows, also considered as a DODF, is introduced. Hence, a flow represents a *methodology* composed of steps in order to

build the intended design. A *view* allows for a partitioning of a flow, resulting in a subset of the steps in the flow. Furthermore, a *step* solves a design problem via a method or simulation model. Each step takes inputs and produces outputs. An *input* can be an executable file, configuration, parameter, or constraint. A method or simulation model is compiled into the executable file. Moreover, an *output* will be a configuration which is produced when the step is finished.

In the subject of the model, the following terms are introduced:

- ▷ meta-model
- ▷ model
- ▷ flow model
- ▷ application model
- ▷ architecture model
- ▷ simulation model (executable specification)

The meta-modeling describes the modeling of the modeling languages. This includes an abstract syntax and the semantics. For example, a *meta-model* enables heterogeneous models of computations in the ESL design, as presented in [25]. In this thesis, a meta-model of a flow is introduced in Section 3.1.4. The intension is to avoid a discussion about the best definition of the term *model*. The considered example is a suitable definition, found in Wikipedia [26]: "A model is a pattern, plan, representation (especially in miniature), or description designed to show the main object or workings of an object, system, or concept.". A flow model is derived from the meta-model. It defines a set of steps and views in order to build a flow. The λ-chart [2], described in Section 3.2, represents a *flow model* following the meta-model. Meta-models can also be defined for the application and architecture models being implemented in a simulation model and executable specification, respectively. The *application model* represents the functions and the data exchange between the functions of a target application. Moreover, an *architecture model* describes the structure and functions of the intended system, such as the computation architecture, interconnecting topology, management infrastructure, communication protocols, etc. Referring to Figure 1.2, an application and architecture model for future many-core SoCs is introduced in the Section 3.1.

In the subject of the method, the following terms are defined:

- ▷ meta-method
- ▷ meta-optimization
- ▷ method

A *meta-method* is a method to analyze another method. As an example, *meta-optimization* is an optimization method to tune another optimization method. In [27], a genetic programming technique has been used for the meta-optimization in order to fine-tune compiler heuristics. In Section 6.2, the author applies meta-optimization via an exhaustive search in order to find suitable input parameters of a genetic algorithm (GA). Referring to Figure 1.2, the *method* denotes a technique for solving an ESL design problem. The method and simulation model is compiled into an executable file used as step input. In Chapter 4, new optimization and estimation methods, developed in this thesis, are presented.

1.4. Thesis Outline and Contribution

This thesis extends available methodologies, models, and methods in ESL design motivated by the practical relevance of the embedded systems. It goes far beyond the state of the art in ESL design which is typically realized with a dedicated flow. The increasing complexity of future many-core SoCs in terms of the large number of inputs and steps in the flow necessitates the development of a systematic design for the flow approach. The huge design space of these systems will draw more attention to ESL design at an early design stage, avoiding time consuming TLM and register-transfer level (RTL) simulations. At the high level of abstraction, the developed simulation models make use of system-level representations of the architecture and application. Special focus is given to many-core SoCs composed of clusters, each representing a set of heterogeneous intellectual property (IP) cores, such as computation resources, data logistic resources, and resource management units. Numerous system functions for the heterogeneous multicluster architecture, resource management, interconnect, and communication protocols have been implemented enabling a validation of the system performance. Moreover, the developed methods include estimation and optimization techniques for the architecture design in terms of a computation and interconnect topology. Furthermore, existing design automation languages do not support a systematic development of flows. Therefore, a programming language and the corresponding automation tools are introduced in order to develop, manage, and optimize the flows. A case study for the heterogeneous multicluster architecture demonstrates how the systematic development of flows helps to cope with the design complexity of future many-core SoCs.

The outline of this thesis is as follows. In Chapter 2, the author introduces the principle of an executable flow. The chapter also focuses on an explanation of the DoDF approach. Then, the introduced concepts are first exemplified via a functional exploration of a finite impulse response (FIR) filter. In later chapters, more complex examples are given. In Chapter 3, the modeling principles of future embedded systems and flows are explained. Then, the idea of abstracting the flows and the corresponding derivation of a domain-specific flow are introduced. The chapter also presents simulation models mainly targeted to clustered many-core SoCs. Also, the introduction of a timing model allows for defining performance metrics. In Chapter 4, several methods for solving ESL design optimization problems in future embedded systems are introduced. The presented methods comprise estimation techniques and formulations of optimization problems via genetic algorithms (GAs) and mixed-integer linear programming (MILP). Search and exploration techniques, applicable to a step and flow, are further covered. In Chapter 5, a design flow language (DFL) is defined allowing to develop, manage, and optimize a flow. Furthermore, a development framework is presented which opens the models, methods, and automation tools to a wide community. An integrated development environment (IDE) combines the programming of DFL flows with the use of the framework. Finally, Chapter 6 applies the previously developed models, methods, and automation tools for an ESL design of an heterogeneous multicluster architecture [3]. The case study focuses on ESL design at an early stage with several flows arranged in a sequence of flows. Each flow outputs experimental results representing suitable solutions for the individual design problems.