Luboš Brim
Boudewijn Haverkort
Martin Leucker
Jaco van de Pol (Eds.)

LNCS 4346

Formal Methods: Applications and Technology

11th International Workshop, FMICS 2006
and 5th International Workshop, PDMC 2006
Bonn, Germany, August 2006, Revised Selected Papers

Springer

TP311-53
F723
2006

Luboš Brim   Boudewijn Haverkort
Martin Leucker   Jaco van de Pol (Eds.)

Formal Methods: Applications and Technology

11th International Workshop, FMICS 2006
and 5th International Workshop, PDMC 2006
Bonn, Germany, August 26-27, and August 31, 2006
Revised Selected Papers

Springer

Volume Editors

Luboš Brim
Masaryk University
Botanicka 68a, 602 00 Brno, Czech Republic
E-mail: brim@fi.muni.cz

Boudewijn Haverkort
University of Twente
P.O. Box 217, 7500AE Enschede, The Netherlands
E-mail: brh@cs.utwente.nl

Martin Leucker
Technische Universität München
Boltzmannstr. 3, 85748 Garching, Germany
E-mail: leucker@in.tum.de

Jaco van de Pol
Centrum voor Wiskunde en Informatica, SEN 2
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
E-mail: Jaco.van.de.Pol@cwi.nl

Library of Congress Control Number: 2007921124

CR Subject Classification (1998): D.2.4, D.2, D.3, C.3, F.3

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-540-70951-7 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-70951-0 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12021901 06/3142 5 4 3 2 1 0

Preface

These are the joint final proceedings of the 11th International Workshop on Formal Methods for Industrial Critical Systems (FMICS 2006) and the fifth International Workshop on Parallel and Distributed Methods in Verification (PDMC 2006). Both workshops were organized as satellite events of CONCUR 2006, the 17th International Conference on Concurrency Theory that was organized in Bonn, August 2006.

The FMICS workshop continued successfully the aim of the FMICS working group – to promote the use of formal methods for industrial applications, by supporting research in this area and its application in industry. The emphasis in these workshops is on the exchange of ideas between researchers and practitioners, in both industry and academia.

This year the Program Committee received a record number of submissions. The 16 accepted regular contributions and 2 accepted tool papers, selected out of a total of 47 submissions, cover formal methodologies for handling large state spaces, model-based testing, formal description and analysis techniques as well as a range of applications and case studies.

The workshop program included two invited talks, by Anna Slobodova from Intel on "Challenges for Formal Verification in an Industrial Setting" and by Edward A. Lee from the University of California at Berkeley on "Making Concurrency Mainstream." The former full paper can be found in this volume.

Following the tradition of previous workshops, the European Association of Software Science and Technology (EASST) supported a best paper award. This award was granted to Michael Weber and Moritz Hammer for their excellent paper "'To Store or Not To Store' Reloaded: Reclaiming Memory on Demand."

The primary goal of the PDMC workshop series is to present and discuss recent developments in the young area of parallel and distributed methods in verification. Several verification techniques, ranging over model checking, equivalence checking, theorem proving, constraint solving and dependability analysis are addressed by the PDMC community. Verification problems are usually very demanding tasks, especially because the systems that we build and want to verify become increasingly complex.

On the other hand, parallel and distributed computing machinery is widely available. Algorithms and tools must be developed to use this hardware optimally for our verification tasks. Traditionally, we studied algorithms for homogeneous situations, such as parallel shared-memory computers and distributed clusters of PCs. Currently, the emphasis is shifting towards heterogeneous GRIDs. But even modern desktop PCs are quite heterogeneous, consisting of multiple core processors, various memory devices and cache levels, all with their own performance characteristics.

This year's PDMC had nine submissions; six papers were selected for presentation, and four papers were accepted for publication in this volume. In addition, Luboš Brim from Masaryk University, Brno, gave an invited lecture on "Distributed Verification: Exploring the Power of Raw Computing Power." The full paper can also be found in this volume.

We would like to thank all authors for their submissions. We would also like to thank the members of both Program Committees, and the additional referees, for their timely reviewing and lively participation in the subsequent discussion—the quality of the contributions in this volume are also due to their efforts and expertise.

The organizers wish to thank CONCUR for hosting the FMICS and PDMC 2006 workshops and taking care of many administrative aspects, and ERCIM for its financial support of FMICS. Additionally, the organizers would like to thank the EASST (European Association of Software Science and Technology), the Faculty of Informatics, Masaryk University Brno and the Technical University Munich, the CWI (Center of Mathematics and Computer Science, Amsterdam) and the University of Twente for supporting these events.

December 2006

Luboš Brim
Boudewijn R. Haverkort
Martin Leucker
Jaco van de Pol

Organization

FMICS

Program Chairs

Luboš Brim	Masaryk University Brno, Czech Republic
Martin Leucker	Technical University of Munich, Germany

Program Committee

Rance Cleaveland	University of Maryland, USA
Wan Fokkink	Vrije Universiteit Amsterdam and CWI, The Netherlands
Stefania Gnesi	ISTI-CNR, Italy
Susanne Graf	VERIMAG, France
David Harel	Weizmann Institute of Science, Israel
Klaus Havelund	Kestrel Technology, USA
Thomas A. Henzinger	EPFL, Switzerland
Leszek Holenderski	Philips Research, The Netherlands
Stefan Kowalewski	RWTH Aachen University, Germany
Marta Kwiatkowska	University of Birmingham, UK
Salvatore La Torre	Universitá degli Studi di Salerno, Italy
Tiziana Margaria	University of Göttingen, Germany
Radu Mateescu	INRIA Rhône-Alpes and ENS Lyon, France
Doron Peled	University of Warwick, UK
Ernesto Pimentel	University of Malaga, Spain
Andreas Podelski	Max-Planck-Institut für Informatik, Germany
Don Sannella	University of Edinburgh, UK
Joseph Sifakis	VERIMAG, France

PDMC

Program Chairs

Boudewijn Haverkort	University of Twente, The Netherlands
Jaco van de Pol	CWI Amsterdam, The Netherlands

Program Committee

Gerd Behrmann	Aalborg University, Denmark
Ivana Černá	Masaryk University Brno, Czech Republic
Gianfranco Ciardo	University of California at Riverside, USA
Joerg Denzinger	University of Calgary, Canada

Hubert Garavel	INRIA Rhône-Alpes, France
Orna Grumberg	Technion, Haifa, Israel
William Knottenbelt	Imperial College, London, UK
Marta Kwiatkowska	University of Birmingham, UK
Martin Leucker	Technical University of Munich, Germany

Referees (FMICS and PDMC)

C. Artho	I. Černá	M. Kuntz	D. Parker
Y. Atir	F. Ciesinski	F. Lang	G. Parlato
R. Atkey	M. Faella	P. Lopez	G. Salaün
J. Barnat	A. Fantechi	K. MacKenzie	W. Serwe
M. ter Beek	M. Felici	P. Maier	F. Sorrentino
M. van der Bijl	A. J. Fernandez	S. Maoz	J. Tenzer
B. Bollig	M. Fruth	F. Mazzanti	A. Venet
L. Bozzelli	N. Geisweiller	R. Merom	A. Wijs
A. Bucchiarone	A. Goldberg	A. Murano	T. Willemse
D. Calvanese	A. Idani	G. Norman	V. Wolf
M. V. Cengarle	C. Joubert	M. Parente	

¥562.00元

Table of Contents

Challenges for Formal Verification in Industrial Setting

Anna Slobodová

Intel
anna.slobodova@intel.com

Abstract. Commercial competition is forcing computer companies to get better products to market more rapidly, and therefore the time for validation is shrinking relative to the complexity of microprocessor designs. Improving time-to-market performance cannot be solved by just growing the size of design and validation teams. Design process automation is increasing, and the adoption of more rigorous methods, including formal verification, is unavoidable because for achieving the quality demanded by the marketplace.

Intel is one of the strongest promoters of the use of formal methods across all phases of the design development. Intel's design teams use high-level modeling of protocols and algorithms, formal verification of floating-point libraries, design exploration systems based on formal methods, full proofs and property verification of RTL specifications, and equivalence checking to verify that transistor-level schematics correspond to their RTL specifications. Even with the best effort to adopt the progress in formal methods quickly, there is a large gap between an idea published at a conference and a development of a tool that can be used on industrial-sized designs. These tools and methods need to scale well, be stable during a multi-year design effort, and be able to support efficient debugging. The use of formal methods on a live design must allow for ongoing changes in the specification and the design. The methodology must be flexible enough to permit new design features, such as scan and power-down logic, soft error detection, etc. In this paper, I will share my experience with the formal verification of the floating-point unit on an Itanium(R) microprocessor design and point out how it may influence future microprocessor-design projects.

1 Introduction

Floating-point (FP) arithmetic is, with respect to functional validation, one of the critical parts of modern microprocessor designs. Even though the algorithms for FP arithmetic are well known, optimization for high performance, reliability, testability and low power, may introduce bugs into a design. The huge input data space that needs to be explored to ensure correctness of floating-point designs is beyond the limits of traditional simulation techniques (hereafter referred to as *simulation*). Fortunately, formal methods are well suited for this area and they can enhance a verification effort substantially. Formal semantics of floating-point operations can be expressed in a succinct way and the IEEE Floating-Point standard [IEEE] serves as a guide for many instruction-set architectures. In this paper, floating-point algorithms are not our concern. Instead, I focus on the correctness of their register-transfer level (RTL) implementations.

For almost a decade, papers reporting compliance proofs for circuit models with respect to the IEEE standard and particular instruction set architecture have been published. While the early research was focused on answering a principal question of the feasibility of formal proofs for computer arithmetic (e.g., [AS95, CB98, OZ+99]), recent work emanates from commercial industry. Formal tools and methods have reached the maturity necessary for their deployment in real design projects.

There are substantial differences between the methodologies used at different companies, depending on their target and available tools and resources. Intel and AMD were among first companies that applied formal methods, at first to verification of floating-point algorithms and then to RTL design. Methods reported from AMD design team in [Rus98, RF00, FK+02] are solely based on theorem proving using ACL2 system [1]. Although lot of automation has been added to building ACL2 models from RTL descriptions, and an ACL2 library of Floating-Point Arithmetic has been created to avoid repetition of implementation independent proofs, the methodology still requires high-level expertise in theorem proving and a perfect understanding of the design.

A recent paper from IBM by Jacobi *et al.* [JW+05] presents a verification method based on symbolic simulation of a RTL model and its comparison to a high-level model written in VHDL. Although highly automated, the approach is not as rigorous as the one described by AMD, and lacks the scope of the methodology developed at Intel [AJ+00][AJ+00, KA00, KN02, KK03, Sch03]. It skips the verification of the more difficult part of the design - multiplier, by removing it from the cone of influence, hence proving merely correctness of the adder and rounder. In contrast, in our approach, no abstraction or design modification took place. We return to the comparison of their work to our results in Section 7.

At Intel, an important work in the area of the verification of floating-point algorithms, and in particular, floating-point libraries for Itanium$^{(R)}$ has been done by John Harrison (see [Har05] for an overview, and [Har00a, Har00b, Har03] for details). However, in hardware verification, while many papers have been published on verification of Pentium$^{(R)}$ design, the first report on the formal verification of floating-point arithmetic for the Itanium$^{(R)}$ microprocessor family was reported on Designing Correct Circuits (DCC) Workshop [2], in March, 2004, in Barcelona [SN04]. The main result was the first successful formal verification of floating-point fused multiply-add instruction which is in repertoire of the IA-64 Instruction Set Architecture (ISA). Proofs have been constructed for a live project aimed at a next generation of Itanium$^{(R)}$ microprocessor. We continued our work presented at DCC by extending the scope and verifying correctness

[1] *http://www.cs.utexas.edu/moore/acl2/*

[2] *http://www.math.chalmers.se/~ms/DCC04/*

of the rest of floating-point instructions (about 40) issued to execution pipe All instructions have been verified with respect to eight precisions and four IEEE rounding modes, including dynamic rounding specified in floating-point status register. The verification was based on symbolic trajectory evaluation and arithmetic libraries previously proven within the same system [KN02]. The proof includes correctness of the result, update of floating-point status register, and correctness of more than a dozen interrupt signals. Behavior of the floating-point circuitry for invalid instructions and/or instructions with false qualifying predicates have been considered as well. All proofs have been regularly rerun as a regression suite to ensure the consistency of any changes in the design. Formal sequential equivalence checking was used to finish the validation of low-level design proving its correctness with respect to the RTL. However this last phase of verification is out of scope of this paper.

In the process of constructing our proofs we found many bugs and issues that required RTL changes. Our work also helped to clarify incomplete and ambiguous parts of our micro-architectural specification, and it contributed to some hardware optimizations. The proofs are automated and portable to other Itanium$^{(R)}$ micro-processor designs.

The goal of this paper is to describe the scope and results of our work, and to provide some insight into challenges of using formal verification in an industrial environment, where a fine balance between rigorous verification methods and traditional simulation-based methods is crucial for success of the validation. Although the approach we choose is a combination of known techniques already documented in context of the verification of floating-point adders and multipliers, we believe that it has many aspects that might be interesting to researchers in academia as well as validation engineers.

The paper is organized in following way: Next section describes tools and methodology developed for formal verification of floating-point arithmetic at Intel Corporation and specifics of our approach. The core of our work is described in Section 3, where we dive into details of the verification of the most interesting operation - fused multiply-add, and report what has been covered by our proofs. Since debugging of failing proofs is one of the concerns in the use of formal methods, we touch this question in Section 4. Section 5 focuses on benefits of our effort for the design project. We describe our experience with proof management in Section 6. Concluding section contains summary of our work and detailed comparison to related published work.

2 Our Approach to Formal Verification of FP Arithmetic

Intel's approach to the validation of floating-point arithmetic includes a huge database of corner test-cases and pseudo-random generators for simulation, as well as Intel's FORTE formal verification tool that combines theorem-proving with model-checking capabilities [3]. The methodology described below does not

[3] A publicly available version of the tool that can be used for non-commercial purposes can be downloaded from *http://www.intel.com/software/products/opensource/*

rely on FORTE specifics and can be reproduced using any tool with capability of symbolic trajectory evaluation (STE) and some means of composing results obtained by STE. We believe that formal proofs coupled with traditional pseudo-random and focused simulation is a good way to achieve thorough functional validation. In our project, the formal and simulation based validation teams mutually benefited from their collaboration. However, this is out of the scope of this paper and we will focus on formal verification only.

2.1 FORTE System and STE

The history of formal verification of floating-point arithmetic at Intel has been motivated by two controversial trends: promising results in academia that were followed by proof of concept at Intel Research Lab [OZ+99]; and bugs that escaped to the micro-processor products [Coe96, Fis97]. Today, formal proofs developed for Pentium$^{(R)}$ designs [AJ+00, KA00, KN02, KK03, Sch03] are re-used and even put into hands of validation engineers that are not experts on formal methods. These proofs have been done using FORTE – a system built on top of VOSS. In this section, we give a rather informal description of the technology inside the FORTE system, just enough to understand the paper; details can be found in the referred publications. FORTE includes a light-weight theorem prover and a symbolic trajectory evaluation (STE) engine [STE]. The theorem prover is based on a higher-order logic. The interface language for FORTE is FL - a strongly-typed functional language in the ML family [Pau96]. One good property of FL as a specification language is its executability. While creating specifications, we often ran sanity checks. For instance, the translation from the memory format to register-file format was written as specified by the Software Developers Manual [ISA], and then checked whether consequent inverse translations yield consistent values. FL includes Binary Decision Diagrams (BDDs)[Bry86] as first-class objects and STE as a built-in function. For more information we refer the interested reader to the online documentation for the FORTE system and [KA00]. Here we describe the basic mechanisms of STE and the framework in which we work.

STE is a weak form of model-checking where a formal (gate-level) model is subjected to a symbolic simulation. The idea of a symbolic simulator is similar to that of standard simulator but it differs in that symbolic values (besides explicit binary values) are assigned to each signal and these values propagated through the design model. Results of such simulations are formulas for specified signals at specified times.

STE is an enhancement of symbolic simulation where Boolean logic has been extended to a lattice [STE] with X as a bottom (no information) and T as a top element (overconstrained). X is automatically assigned (by the STE simulator) to signals to which no value has been specified. X can be thought of as an unknown value. Its semantics and use are discussed later. Symbolic values are bound to signals at specified times to form signal trajectories. Trajectories that prune possible computations by restricting the values of some signals at specific times are called *antecedents*; they can be interpreted as assumptions. Trajectories that specify expected responses of the circuit are called *consequents.* A specification is written in a form of Boolean expressions that constrain symbolic values in antecedents and consequents. Trajectory evaluation correctness statement $\models_{ckt} [ant \Longrightarrow\!\!> cons]$ means: all circuit computations that satisfy antecedent *ant* also satisfy consequent *cons.* If any of consequent is violated, a STE run (proof) fails and a counterexample can be extracted from this failure. In fact, the failed proof provides all possible counterexamples and the user may select one for debugging purposes. If all consequents hold at every point of the simulation, success is reported by the tool.

2.2 Pre- and Post-condition Framework

Because of capacity limitations inherit in the STE engine, we may be forced to break our model into smaller pieces. In this case, we make sure that those pieces perfectly fit together. Informally, this means that the border signals of the decomposition match exactly and that nothing is left out of the design. Also the times at which we extract the values of the signals must be consistent. In terms of STE, consequents that include border signals serve as antecedents in the following step of the proof. In this way, we can use facts proved in one part as assumptions for later proofs.

The idea of proof (de)composition described above comes from the *pre-and-post-condition theory* used for verification of sequential programs. It was first applied to STE by Kaivola and Aagaard [KA00]. It allows one to prove the statements of the form $\{P\}S\{Q\}$, where P and Q are logical properties and S is a program. In our case, the program is replaced by a circuit and trajectories that bind values inputs and outputs of the circuit at specific times. $\{P(x)\}(pretr_x, ckt, posttr_y)\{Q(x,y)\}$ represents the statement: if $pretr_x$ binds the Boolean vector x to signals (usually inputs) of the circuit *ckt* and $posttr_y$ binds the Boolean vector y to signals of the circuit (usually outputs), then the property $P(x)$ guarantees property $Q(x,y)$.

$\{P(x)\}(pretr_x, ckt, posttr_y)\{Q(x,y)\}$ is a shorthand for the following formula:

$$\begin{aligned}\forall x(P(x) \Rightarrow (\exists y(\models_{ckt} [pretr_x \Longrightarrow\!\!> posttr_y])) \wedge \\ (\forall y((\models_{ckt} [pretr_x \Longrightarrow\!\!> posttr_y]) \Rightarrow Q(x,y)))) \qquad (1)\end{aligned}$$

In our methodology, P is a conjunction of an *initial condition* that describes the restriction of inputs to the circuit, and *an auxiliary pre-condition* that is used to further restrict the simulation. For consistency, we use the same initial conditions throughout all proofs for every instruction analyzed, except when we weaken an initial condition to *true.* An example of an initial input condition is a statement that the specified input signals have value of a specific opcode. Auxiliary pre-conditions are usually used to simplify a particular STE run by restricting symbolic values (meaning that the inputs or internal nodes are restricted). An example of an auxiliary pre-condition is a restriction specifying