
Advanced Automation in Formal Verification of Processors

Ulrich Kühne
Arbeitsgruppe Rechnerarchitektur
Universität Bremen

Dissertation
zur Erlangung des Grades eines Doktors
der Ingenieurwissenschaften
– Dr.-Ing. –

Vorgelegt im Juli 2009 an der Universität Bremen
Kolloquiumstermin: 25. September 2009

Erster Gutachter: Prof. Dr. Rolf Drechsler,
Universität Bremen

Zweiter Gutachter: Prof. Dr. Wolfram Büttner,
Abstract RT Solutions GmbH, München

Prüfungsausschuss: Prof. Dr. Hans-Jörg Kreowski,
Prof. Dr. Jan Peleska,
Dr. Daniel Große,
Markus Groß

Berichte aus der Informatik

Ulrich Kühne

**Advanced Automation in
Formal Verification of Processors**

D 46 (Diss. Universität Bremen)

Shaker Verlag
Aachen 2009

Bibliographic information published by the Deutsche Nationalbibliothek

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at <http://dnb.d-nb.de>.

Zugl.: Bremen, Univ., Diss., 2009

Copyright Shaker Verlag 2009

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the publishers.

Printed in Germany.

ISBN 978-3-8322-8619-4

ISSN 0945-0807

Shaker Verlag GmbH • P.O. BOX 101818 • D-52018 Aachen

Phone: 0049/2407/9596-0 • Telefax: 0049/2407/9596-9

Internet: www.shaker.de • e-mail: info@shaker.de

For Cécile

Abstract

This thesis addresses automation and enhanced user support in functional verification of digital systems with a special focus on processors. Verification is an important issue in the design process of digital systems. In contrast to traditional simulation based methods, formal verification can provide a mathematical correctness proof. But, up to today, these methods are difficult to use in practice, without a strong background in formal techniques. This thesis aims at improving the usability and increasing the productivity of formal hardware verification.

The functional verification of digital systems can be attacked bottom-up or top-down. For an ad hoc verification of single components or simple hardware systems, the specification is formalized step by step in a bottom-up manner. Thereby, the verification engineer is faced with the task to identify the implementation principles of the design and to capture the behavior in temporal logic properties. In this thesis, techniques are presented to speed up this process. This includes an easy to use coverage check that gives feedback about the quality of the written properties. The process of writing the properties can be supported by strengthening properties automatically. Another approach automatically generated properties for a given expected behavior of the design.

When facing the verification of a restricted class of designs, further automation can be achieved in a top-down flow. In particular, for processors, the task is to find a mapping between the instruction set architecture – the abstract programmer’s view – and the implementation. In this thesis, it is examined how parts of this process can be automated. The presented approach provides a guided verification flow, starting with a formal architecture description that is then related to the implementation. The use of architectural models in a structured verification flow can increase the productivity significantly. Further applications of such high-level models include the automatic generation of instruction set simulators and the automatic synthesis of embedded software. These issues are also addressed in this thesis.

Zusammenfassung

Diese Arbeit beschäftigt sich mit Automation und erweiterter Benutzerunterstützung in der funktionalen Verifikation von digitalen Systemen, mit einem Schwerpunkt auf Prozessoren. Die Verifikation ist ein entscheidender Faktor im Entwurfsprozess von Hardware-Systemen. Im Gegensatz zu traditionellen simulativen Methoden kann mittels formaler Verifikation ein mathematischer Korrektheitsbeweis erbracht werden. Allerdings ist der praktische Einsatz dieser Techniken bis heute mit Schwierigkeiten verbunden, da sie teilweise erhebliches Vorwissen in formalen Methoden erfordern. Diese Arbeit zielt darauf ab, die Anwendbarkeit formaler Hardware-Verifikation zu verbessern und ihre Produktivität zu erhöhen.

Die funktionale Verifikation kann auf verschiedene Weisen angegangen werden, bottom-up von der Implementierung zur Spezifikation oder top-down von der Spezifikation ausgehend. In der ad hoc Verifikation einzelner Module und einfacher Systeme wird die Spezifikation schrittweise in einem bottom-up Ansatz formalisiert. Dabei ist es die Aufgabe des Verifikations-Ingenieurs, die Funktionsweise der Implementierung zu verstehen und das Verhalten des Systems in temporaler Logik zu beschreiben. In dieser Arbeit werden Techniken vorgestellt, um diesen Prozess zu vereinfachen und zu beschleunigen. Mit einer einfach anzuwendenden Coverage-Technik kann die Qualität eines Eigenschaftssatzes überprüft werden. Das Schreiben der Eigenschaften kann durch automatische Stärkung einzelner Eigenschaften unterstützt werden. Ein weiterer Ansatz beschäftigt sich mit der automatischen Generierung von Eigenschaften, die ein erwartetes Verhalten der Implementierung beschreiben.

Ein höherer Automationsgrad kann bei der funktionalen Verifikation von eingeschränkten Klassen von Systemen erreicht werden. Für Prozessoren besteht die Verifikationsaufgabe darin, eine Abbildung zwischen der Instruktionssatzarchitektur – der abstrakten Sicht des Programmierers – und der Implementierung zu finden. In dieser Arbeit wird untersucht, wie dieser Prozess teilweise automatisiert werden kann. Der entwickelte Ansatz stellt einen werkzeugunterstützten Verifikationsablauf zu Verfügung, ausgehend von einer formalen Architekturbeschreibung. Durch die Nutzung von Architekturmodellen in einem klar strukturierten Ablauf lässt sich die Produktivität der Verifikation erhöhen. Weitere Anwendungen solcher abstrakter Modelle sind die automatische Generierung von Instruktionssatzsimulatoren und die automatische Synthese eingebetteter Software-Programme, welche ebenfalls Gegenstand dieser Arbeit sind.

Danksagung

Mit dieser Arbeit beende ich meine Promotion im Fach Informatik. Dass es so weit kommen konnte, dafür bin einigen Leuten zum Dank verpflichtet.

An erster Stelle danke ich Rolf Drechsler, dem Leiter der Arbeitsgruppe Rechnerarchitektur und Betreuer meiner Promotion. Er hat mich bereits während meines Studiums an der Universität Bremen für die Forschung begeistert und mich seitdem stets in meinen Zielen unterstützt, hat zahlreiche Veröffentlichungen gelesen und korrigiert und mich immer wieder motiviert. Auch den Mitarbeitern der Arbeitsgruppe danke ich für die tolle Zusammenarbeit. Mit Daniel Große hatte ich viele fruchtbare Diskussionen, aus denen auch viele gemeinsame Arbeiten entstanden sind. Genauso danke ich allen anderen dort, die zur Verlängerung meiner Publikationsliste beigetragen haben: André Süllow, Görschwin Fey, Robert Wille und Stefan Frehse.

Während meiner Promotion hatte ich die Möglichkeit, einige Zeit in der Stadt München zu verbringen. In Kooperation mit der Firma OneSpin Solutions sind dort große Teile dieser Arbeit entstanden. An dieser Stelle danke ich Wolfram Büttner, der mich dort sehr freundlich aufgenommen hat und der auch einer der Gutachter dieser Arbeit ist. Jörg Bormann danke ich für die Betreuung und seine Verbesserungsvorschläge für meine Dissertation. Auf Grundlage seiner Arbeiten ist die Eigenschaftsgenerierung für Prozessoren entstanden. Ein herzliches Dankeschön geht an Sven Beyer, mit dem ich in München zusammengearbeitet habe und der in Fragen der Verifikation stets Rat wusste und ebenfalls meine Dissertation gelesen hat. Außerdem danke ich Sebastian Skalberg und Christian Pichler, die mich zu Beginn der Kooperation in München in ihre Untersuchungen zur Simulatorgenerierung einbezogen haben.

Vielen Dank schließlich an die Mitglieder des Prüfungsausschusses, die meinem Promotionskolloquium in der Universität Bremen beiwohnten, an Hans-Jörg Kreowski, Jan Peleska, Daniel Große und Markus Groß.

Contents

1	Introduction	1
2	Background	11
2.1	Binary Decision Diagrams	11
2.2	Decision Procedures	13
2.2.1	Boolean Satisfiability	14
2.2.2	Satisfiability Modulo Theories	15
2.2.3	Quantified Boolean Formulae	16
2.3	Bounded Model Checking	16
2.3.1	Semantics of Bounded PSL	17
2.3.2	Encoding as a SAT Problem	21
2.3.3	k -Induction	23
2.3.4	0-Induction	24
2.3.5	Interval Property Checking in the OneSpin Flow	25
2.4	The WoLFram Framework	28
3	Bottom-Up – Formalizing the Specification	31
3.1	Bounded Model Checking	33
3.1.1	RISC CPU	33
3.1.2	Block Level	34
3.1.3	Instruction Set Architecture Level	38
3.1.4	Software Level	41
3.1.5	Summary	44
3.2	Coverage Analysis	45
3.2.1	Related Work	45
3.2.2	Idea	47
3.2.3	Generating the Coverage Property	47
3.2.4	Applicability and Limitations	51
3.2.5	Results	52
3.2.6	Summary	57
3.3	Property Analysis	58

3.3.1	Related Work	59
3.3.2	Idea	59
3.3.3	Algorithm	60
3.3.4	Discussion	62
3.3.5	Results	63
3.3.6	Summary	64
3.4	Inverse Property Checking	65
3.4.1	Related Work	65
3.4.2	Algorithm	65
3.4.3	Integration with Coverage Analysis	68
3.4.4	Results	70
3.4.5	Summary	71
3.5	Discussion	73
4	Top-Down – Processor Verification Methodology	75
4.1	Processor Design and Verification	77
4.1.1	Automated Pipeline Design	77
4.1.2	Incremental Design and Verification	78
4.1.3	Processor Verification	78
4.2	Verification of Processors Based on Architectural Models	81
4.2.1	Contribution and Related Work	82
4.2.2	Overview	83
4.2.3	Architecture Description	86
4.2.4	General Processor Model	88
4.2.5	Generating the Property Suite	100
4.2.6	Completeness	105
4.2.7	Applications	106
4.2.8	Applicability and Limitations	114
4.2.9	Summary	115
4.3	Generating an Instruction Set Simulator	117
4.3.1	Background and Related Work	119
4.3.2	Generation of the ISS	120
4.3.3	Results	126
4.3.4	Summary	128
4.4	Automated Software Synthesis with Architectural Models	129
4.4.1	Architecture and Software Specification	130
4.4.2	QBF Model and Algorithm	133
4.4.3	Applications	135
4.4.4	Summary	137
5	Conclusions and Perspectives	139

A Bounded PSL Syntax	143
B ISA description Syntax	149
Glossary	157
Bibliography	159