Alireza Mahzoon
Daniel Große
Rolf Drechsler

Formal Verification of Structurally Complex Multipliers

Springer

Formal Verification of Structurally Complex Multipliers

Alireza Mahzoon • Daniel Große • Rolf Drechsler

Formal Verification of Structurally Complex Multipliers

Springer

Alireza Mahzoon
University of Bremen
Bremen, Germany

Rolf Drechsler
University of Bremen/DFKI
Bremen, Germany

Daniel Große
Institute for Complex Systems
Johannes Kepler University of Linz
Linz, Austria

ISBN 978-3-031-24570-1 ISBN 978-3-031-24571-8 (eBook)
https://doi.org/10.1007/978-3-031-24571-8

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

To Tina,
Marie
and
Yuna

Preface

Back in 1970, an Intel 4004 processor had 2250 transistors. It could only support a limited number of instructions, and it was working at a very low frequency. However, digital circuits nowadays are much larger, sometimes even consisting of billions of transistors. Moreover, they are usually designed based on sophisticated algorithms, leading to fast but complex architectures. The big size and the high complexity of modern digital circuits make them extremely error-prone during different design phases. Consequently, formal verification is an important task to ensure the correctness of a digital circuit.

Formal verification of arithmetic circuits is one of the most challenging problems in the verification community. Despite the success of verification methods based on *Binary Decision Diagrams* (BDDs) and *Boolean Satisfiability* (SAT) in proving the correctness of adders, they totally fail when it comes to the verification of multipliers. In the last six years, the word-level verification methods based on *Symbolic Computer Algebra* (SCA) achieved many successes in proving the correctness of structurally simple multipliers. The proposed techniques can verify a very large multiplier in a few seconds. However, they either totally fail or support a limited set of benchmarks when it comes to verifying structurally complex multipliers.

This book addresses the challenging tasks of verifying and debugging structurally complex multipliers. In the area of verification, it first investigates the challenges of SCA-based verification when it comes to proving the correctness of multipliers. Then, it proposes three techniques, i.e., vanishing monomials removal, reverse engineering, and dynamic backward rewriting, to improve and extend SCA. As a result, a wide variety of multipliers, including highly complex and optimized industrial benchmarks, can be verified. In the area of debugging, it proposes a complete debugging flow, including bug localization and fixing, to find the location of bugs in structurally complex multipliers and make corrections.

Bremen, Germany — Alireza Mahzoon
Linz, Austria — Daniel Große
Bremen, Germany — Rolf Drechsler

Acknowledgements

We would like to particularly appreciate all those who have contributed to the results included in this book. Our special thanks go to Christoph Scholl and Alexander Konrad for their investments of time and great ideas, which were important for this book. Many thanks to Mehran Goli for his helpful feedback and inspiring discussions. We would like to express our appreciation to all of the colleagues in the Group of Computer Architecture at the University of Bremen, the Institute for Complex Systems at the Johannes Kepler University Linz, and the Cyber-Physical Systems group at the German Research Center for Artificial Intelligence for their support.

Bremen, Germany — Alireza Mahzoon
Linz, Austria — Daniel Große
Bremen, Germany — Rolf Drechsler
October 2022

Contents

Chapter 1
Introduction

With the invention of the transistor back in 1947, the cornerstone for the digital revolution was laid. As a fundamental building block, the transistor triggered the development of digital circuits. The mass production of digital circuits revolutionized the field of electronics, finally leading to computers, embedded systems, and the Internet. Hence, the impact of digital hardware on society, as well as the economy, was and is tremendous. Over the last decades, the enormous growth in the complexity of integrated circuits continues as expected. As modern electronic devices are getting more and more ubiquitous, the fundamental issue of functional correctness becomes more important than ever. This is evidenced by many publicly known examples of electronic failures with disastrous consequences. This includes, e.g., the Intel Pentium bug in 1994 [6, 28], the New York blackout in 2003 [64], and a design flaw in Intel's Sandy Bridge chipset in 2011 [18].

Such costly mistakes can only be prevented by applying rigorous verification to the circuits before they get to production [20, 21]. A lot of effort has been put into developing efficient verification techniques by both academic and industrial research. Only recently, the industry has recognized the great importance of formal verification (see, e.g., functional safety standards such as ISO 26262 [88]). Hence, in the last few years, this research area has become increasingly active. Essentially, formal verification aims to prove in a mathematical sense that an implementation is correct with respect to its specification. Formal verification is an essential task in each phase of the design flow to ensure the correctness of an implementation.

An overview of the typical top-down design flow is presented in Fig. 1.1. The system specification defines the functionality and is usually the starting point for the design. The register transfer model is constructed by a designer or a high-level synthesis tool based on the system specification. At this level, the system behavior is described in terms of registers and their data flow with its operations. Logic synthesis tools transfer the model to a gate-level description, consisting of logic gates and flip-flops. Finally, the logic gates are mapped into a circuit-level description, consisting of transistors, interconnects, and other physical cells,

A. Mahzoon et al., *Formal Verification of Structurally Complex Multipliers*,
https://doi.org/10.1007/978-3-031-24571-8_1

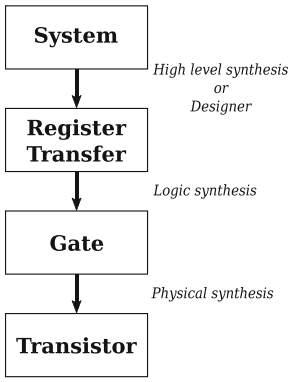

Fig. 1.1 Top-down design flow

which create the final chip implementation. There is always a risk of incorrect transformations by designers or synthesis tools when moving between different levels of abstraction. As a result, bugs might appear in each phase of the design, leading to a faulty circuit description. The fabrication and production of faulty designs cause a catastrophe, resulting in huge financial loss and endangering lives. It is thus critical to ensure the correctness of a circuit description at each level of abstraction.

Several formal verification methods have been proposed to prove the correctness of a circuit description. These methods can be categorized into three groups:

- **Equivalence checking**: The goal of formal equivalence checking is to prove that a circuit description is functionally equivalent to a specification (golden model). The specification is usually a correct description in the same level or a higher level of abstraction. For example, assuming that gate-level description A is correct, gate-level description B will be correct if A and B are functionally equivalent. As another example, the gate-level description and the high-level system specification have to be equivalent; otherwise, the gate-level description is faulty. Equivalence checking is widely employed in the automated formal verification of both combinational and sequential digital circuits.
- **Model checking**: The aim of model checking is to ensure that a property holds for a circuit description in a specific level of abstraction. This includes safety properties (nothing incorrect ever occurs) and liveness properties (something correct eventually occurs). In model checking, the circuit description is captured as a transition system, specifying its behavior in different states. Furthermore, the property is expressed in the form of a temporal logic formula, and a model checker is used to check whether the property is violated or not. Model checking is widely employed in the automated formal verification of sequential digital circuits.
- **Theorem proving**: The goal of theorem proving in verification is to prove that a circuit description satisfies its specification by mathematical reasoning. The description and the specification are expressed as formulas in a formal

logic. Then, the required relationship between them (logical equivalence or logical implication) is described as a theorem to be proven within the context of a proof calculus. A proof system, consisting of axioms and interface rules (e.g., simplification, rewriting, and induction), is used to achieve this goal.

Formal verification of arithmetic circuits is one of the most popular and challenging topics in the verification community. Arithmetic circuits are extensively used in many systems, e.g., for signal processing and cryptography, as well as for upcoming AI solutions employing machine learning and deep learning. They also constitute a big part of an *Arithmetic Logic Unit* (ALU), which is the computational heart of a *Central Processing Unit* (CPU). The top-level design flow in Fig. 1.1 is also used for the implementation of arithmetic circuits. The high-level specification is usually a mathematical expression, determining the function of an arithmetic circuit based on its primary inputs and outputs. For example, assuming A and B are two n-bit inputs and S is an $(n+1)$-bit output, the expression $S = A + B$ describes an integer adder in the highest level of abstraction. The high-level specification is transformed into the register transfer level by a designer or an arithmetic generator tool. Finally, the circuit is synthesized into gate-level and then transistor-level descriptions. Formal verification of arithmetic circuits at the register transfer level (where the hierarchical information is available) and the gate level (where no hierarchical information is at hand) is the focus of many research works.

Integer multipliers are among the most frequently used arithmetic circuits in a large variety of applications. Most of these applications require very large multipliers supporting a wide range of integer numbers. Furthermore, the multiplier architectures also vary based on the design goals of different applications. Several multiplication algorithms have been developed to satisfy the community demands for fast, area-efficient, and low-power designs or make a trade-off between several design parameters. Employing these algorithms usually results in the generation of very complex architectures. Formal verification of huge and structurally complex multipliers is on the one hand necessary to ensure the correctness of the final design. On the other hand, it is a big challenge, where most of the existing formal methods completely fail.

In the last 30 years, several formal verification methods, based on equivalence checking and theorem proving techniques, have been proposed to ensure the correctness of arithmetic circuits. Although these methods accomplished big successes in many domains, they suffer from serious limitations when it comes to verifying integer multipliers:

- Equivalence checking methods using *Binary Decision Diagrams* (BDDs) [10, 27] or *Boolean Satisfiability* (SAT) [19, 35] ensure the correctness by proving that an integer multiplier is equivalent to a correct multiplier description. However, they are not scalable and only work for very small benchmarks.
- Equivalence checking methods using *Binary Moment Diagrams* (*BMDs and K*BMDs) [23, 39] ensure the correctness by proving that an integer multiplier and its high-level specification are equivalent. These methods are scalable for