Post-Silicon and Runtime Verification for Modern Processors

Ilya Wagner • Valeria Bertacco

Post-Silicon and Runtime Verification for Modern Processors

Springer

Ilya Wagner
Platform Validation Engineering Group
Intel Corporation
Hillsboro, Oregon
USA
ilya.wagner@intel.com

Valeria Bertacco
Department of Electrical Engineering
and Computer Science
University of Michigan
Ann Arbor, Michigan
USA
valeria@umich.edu

ISBN 978-1-4419-8033-5 e-ISBN 978-1-4419-8034-2
DOI 10.1007/978-1-4419-8034-2
Springer New York Dordrecht Heidelberg London

To my niece Ellie, who showed me
the miracle of learning.

Ilya Wagner

To all my students, who make working
in the field of verification such a rewarding
experience.

Valeria Bertacco

Preface

The growing complexity of modern processor designs and their shrinking production schedules cause an increasing number of errors to escape into released products. Many of these escaped bugs can have dramatic effects on the security and stability of consumer systems, undermine the image of the manufacturing company and cause substantial financial grief. Moreover, recent trends towards multi-core processor chips, with complex memory subsystems and sometimes non-deterministic communication delays, further exacerbate the problem with more subtle, yet more devastating, escaped bugs. This worsening situation calls for high-efficiency and high-coverage verification methodologies for systems under development, a goal that is unachievable with today's pre-silicon simulation and formal validation solutions. In light of this, functional post-silicon validation and runtime verification are becoming vitally important components of a modern microprocessor development process. Post-silicon validation leverages orders of magnitude performance improvements over pre-silicon simulation while providing very high coverage. Runtime verification solutions augment the hardware with on-chip monitors and checking modules that can detect erroneous executions in systems deployed in the field and recover from them dynamically.

The purpose of this book is to present and discuss the state of the art in post-silicon and runtime verification techniques: two very recent and fast growing trends in the world of microprocessor design and verification. The first part of this book begins with a high-level overview of the various verification activities that a processor is subjected to as it moves through its life-cycle, from architectural conception to silicon deployment. When a chip is being designed, and before early hardware prototypes are manufactured, the verification landscape is dominated by two main groups of techniques: simulation-based validation and formal verification. Simulation solutions leverage a model of the design's structure, often written in specialized hardware programming languages, and validate a design by providing input stimuli to the model and evaluating its responses to those stimuli. Formal techniques, on the other hand, treat a design as a mathematical description of its functionality and focus on proving a wide range of properties of its functional behavior. Unfortunately, these two categories of validation methods are becoming increasingly inadequate

in coping with the complexity of modern multi-core systems. This is exactly where post-silicon and runtime validation techniques, the primary scope of this book, can lend a much needed hand.

Throughout the book we present a range of recent solutions in these two domains, designed specifically to identify functional bugs located in different components of a modern processor, from individual computational cores to the memory subsystem and on-chip fabrics for inter-core communication. We transition into the second part of the book by presenting mainstream post-silicon validation and test activities that are currently being deployed in industrial development environments and outline important performance bottlenecks of these techniques. We then present *Reversi*, our proposed methodology to alleviate these bottlenecks in processor cores. Basic principles of inter-core communication through shared memory are overviewed in the following chapter, which also details new approaches to validation of communication invariants in silicon prototypes. We conclude the discussion of functional post-silicon validation with a novel echnique, targeted specifically to modern multi-cores, called *Dacota*.

The recently proposed approaches to validation that we collected in part two of this book have an enormous potential to improve verification performance and coverage; however, there still is a chance that complex and subtle errors evade them and escape into end-user silicon systems. Runtime solutions, the focus of the third part of this work, are designed to address these situations and to guarantee that a processor performs correctly even in presence of escaped design bugs without degrading user experience. To better analyze these techniques we investigate the taxonomy of escaped bugs reported for some of the processor designs available today, and we also classify runtime approaches into two major groups: *checker-* and *patching-based*. In the remainder of part three we detail several runtime verification methods within both categories, first relating to individual cores and then to multi-core systems. We conclude the book with a glance towards the future, discussing modern trends in processor architecture and silicon technology, and their potential impacts on the verification of upcoming designs.

Acknowledgements

We would like to acknowledge several people that made the writing of this book possible. First, and foremost, we express our gratitude to our colleagues, who worked with us on the research presented in this book. In particular, we would like to thank Professor Todd Austin, who was a vital member of our runtime verification research and provided critical advice in many other projects. Andrew DeOrio has contributed immensely to our post-silicon validation research and has helped us greatly in the experimental evaluations of several other techniques. Both Todd and Andrew have also worked tirelessly on the original development of these works and provided valuable insights on the presentation of the material in this manuscript.

We would also like to thank many students working in the Advanced Computer Architecture Lab, and, especially, all those who have devoted their research to hardware verification: Kai-hui Chang, Stephen Plaza, Andrea Pellegrini, Debapriya Chatterjee, Rawan Abdel-Khalek have worked particularly closely to us among many others. Every day these individuals are relentlessly advancing the theory and the practice of this exciting and challenging field. We also acknowledge all of the faculty and staff at the Computer Science and Engineering Department of The University of Michigan, as well as all engineers and researchers in academia and industry who provided us with valuable feedback on our work at conferences and workshops, reviewed and critiqued our papers and published their findings, upon which much of our research was built. These are truly the giants, on whose shoulders we stand.

We also thank our families, who faithfully supported us throughout the years of research that led to the publication of this book. Each and every one of them was constantly ready to offer technical advice or a heartwarming consolation in difficult times, and celebrate the moments of our successes. Indeed, without their trust and encouragement this writing would be absolutely impossible.

Finally, we would like to acknowledge our editors, Mr. Alex Greene, Ms. Ciara Vincent and Ms. Katelyn Chin from Springer, who worked closely with us on this publication with truly angelic patience. Time and again, they encouraged us to continue writing and gave us valuable advice on many aspects of this book.

Contents

Acronyms

ALU	Arithmetic-logic unit. A hardware block that performs integer arithmetic and logic functions, such as addition, subtraction, logic AND, *etc.*
API	Application programming interface. A set of functions and routines which describe the interface of a software application. API description is typically limited to the application interface, and does not specify the way the functionality of the program is actually implemented.
ATPG	Automatic test pattern generation. A technique for post-silicon validation of a manufactured circuit, which attempts to expose errors in fabrication of individual gates and interconnect in the design. ATPG tools use a software representation of the netlist to derive input stimuli that can expose a variety of manufacturing defects and then apply these tests to the prototype.
BDD	Binary decision diagrams. A data structure for compact representation and fast operations on Boolean logic functions. BDDs are commonly used as an underlying engine of formal verification approaches, such as symbolic simulation, reachability analysis and model checking.
BIOS	Basic input-output system. A firmware residing on the motherboard, that tests and configures various modules of a computer system upon startup.
BMC	Bounded model checking. A pre-silicon verification technique which establishes adherence of design's behavior, within a finite number of clock cycles, to formal specifications, often written as temporal logic formulas.
CPI	Cycles per instruction. A metric of processor performance, which measures the average number of clock cycles that the device needs to perform one operation.
CPU	Central processing unit. Electronic circuit capable of executing software programs, synonymous to the word "microprocessor".
DFT	Design for testability. A class of techniques, which enable testing and debugging of digital circuits, *e.g.,* scan-chains, boundary-scan, on-chip logic analyzers, *etc.*
ECC	Error-correcting code. A special redundant encoding of the data that allows to recover the information even if some of the bits of it are corrupted. Typically used to protect computer storage, such as memory and caches.

EDA Electronic design automation. A generic name for computer-aided tools for electronics design, as well as for the companies producing and deploying such tools.

FPGA Field-programmable gate array. A digital circuit, which can be programmed to implement arbitrary logic functions. FPGAs are commonly used to emulate behavior of complex devices, such as microprocessors, before the first prototype is manufactured.

FPU Floating point unit. A hardware block that performs floating point computation inside of the processor.

FRCL Field-repairable control logic. A hardware patching solution, which augments a processor with a programmable matcher to detect and recover from erroneous control logic states.

FSM Finite state machine. A graph description of hardware block operation. Consists of graph vertices, describing states of the machine, and edges, identifying legal transitions between the states.

GPU Graphics processing unit. An electronic circuit dedicated to processing graphical information, before it is sent to a computer display. GPU chips are commonly placed on the motherboard or implemented as a separate graphics card.

HDL Hardware description language. A class of programming languages that are used to describe functionality and organization of digital hardware, so the behavior of the design can be simulated.

ILP Instruction level parallelism. A potential overlap in the execution of independent instructions. ILP measures how many operations in a program can be performed in parallel.

ISA Instruction set architecture. A complete list of all instructions and operations that a processor can execute. ISA also often includes a complete specification of the processor interface, in terms of communication protocols, interrupt handling, *etc.*

JTAG Joint test action group. Initially, an industry group that designed boundary-scan technique for circuit board testing. Later, term JTAG became synonymous with the boundary scan architecture.

NMR N-modular redundancy. A resiliency technique, where N modules (identical or heterogeneous) compute the same function in parallel. Errors in one unit can then be detected and corrected through majority voting scheme.

OCLA On-chip logic analyzer. A circuit, residing on a processor die, which can be programmed to monitor for specific activity of the processor and record the internal state of the chip upon occurrence of the trigger.

OS Operating system. A software supervisor that manages the hardware and user-level programs of a computer system. An operating system provides applications with access to the hardware and coordinates the resource sharing in a computer system.

PCI Peripheral component interconnect. A type of computer bus that connects peripheral devices of a system, *e.g.,* graphics and network cards, to the motherboard and the processor.

PSMI Periodic state management interrupt. A technology developed at Intel, which allows to periodically stop program execution and collect the internal state of a silicon prototype for post-silicon debugging.

QoS Quality of service. A network control mechanism that distributes resources to different classes of traffic to achieve deterministic or statistical guarantees of system performance.

ROB Re-order buffer. A hardware block for instruction re-ordering used in complex out-of-order processor architectures. To hide the effects of long-latency operations a processor may issue instructions in an order different than that of the original program and use an ROB to ensure that the instruction stream is committed in strict program order.

ROM Read-only memory. A class of storage of computer devices, where stored data cannot be modified. In microprocessor domain read-only memory is typically implemented as a hard-wired lookup table.

RTG Random test generator. A software that creates randomized sequences of inputs to the design for testing purposes. Typically the test stimuli are not entirely random, but are constrained to a subset of all valid inputs to the design.

RTL Register transfer level. An register transfer level description of a logic device consists of memory elements (registers) and functions that transfer the data between them. An RTL description is typically implemented in a hardware description language.

SAT Boolean satisfiability. A class of problems that establishes if there exist an assignment to variables of a Boolean formula that evaluates it to true. SAT-solvers and their derivatives are often used by formal verification approaches, such as equivalence checking.

SEU Single event upset. A change or corruption of the state of a latch (or a flip-flop), due to an energetic particle strike.

SPICE Simulation program with integrated circuit emphasis. A class of simulation software that can evaluate the behavior of an integrated circuit from the electrical standpoint. SPICE technique solve complex differential equations, which describe how voltage and current at in various points in the design change over time, thus, operating at lower abstraction level than logic simulation solutions.

STG State transition graph. A graph description of states and transitions between the states that hardware can assume at runtime. Synonymous to finite state machine.

SoC System-on-a-chip. A hardware device that integrates multiple components of a computer system, *e.g.,* processing cores, non-volitile memory, peripheral controllers, *etc.* on a single silicon die.

TLB Translation look-aside buffer. A hardware module that acts as a fast cache to a larger and slower lookup table, thus increasing processor performance for accesses that hit in the TLB.