Formal Methods in System Design, 19, 35–44, 2001

On the Effective Deployment of Functional Formal Verification

YAEL ABARBANEL-VINOV*, NETA AIZENBUD-RESHEF, ILAN BEER, CINDY EISNER, DANIEL GEIST, TAMIR HEYMAN, IRIS REUVENI, ERAN RIPPEL*, IRIT SHITSEVALOV, YARON WOLFSTHAL AND TALI YATZKAR-HAHAM
IBM Haifa Research Laboratory, Matam Advanced Technology Center, Haifa, 31905, Israel

Abstract. We examine IBM's exploitation of formal verification using RuleBase—a formal verification tool developed by the IBM Haifa Research Laboratory. The goal of the paper is methodological. We identify an integrated methodology for the deployment of formal verification which involves three complementary modes: architectural verification, block-level verification, and design exploration.

Keywords: formal verification, hardware verification, model checking

1. Introduction

It is becoming increasingly difficult to deliver quality hardware under the constraints of resources and time to market. As micro-architectural complexity has grown and process technologies improved, verification has become the productivity bottleneck in the design cycle. The role of formal verification is to break the bottleneck, by providing a way for systems under design to be mathematically modeled and reasoned about. However, as noted by Sangiovanni-Vincentelli [13] at the 33rd Design Automation Conference in 1996, powerful tools are not enough. A paradigm shift is required, and a verification aware discipline in the design process is the only way to ensure the correctness of hardware today. This is recognized by engineers and their management at IBM, who are increasingly turning their focus to methodological questions in design and verification. In this paper, we describe an integrated methodology for the deployment of formal verification, used in the IBM design environment. It includes three complementary modes: architectural verification, block-level verification, and design exploration.

It is worth noting at the outset that IBM takes a pragmatic view of formal verification. During more than six years of experience, formal verification at IBM has focused on finding bugs, rather than proving a design correct. We consider formal verification to be a powerful extension of the traditional process of bug finding by simulation, and as complementary to simulation in the verification process.

This paper is organized as follows. Section 2 provides some basic background on symbolic model checking with RuleBase (this section can be skipped by the reader without impact

*This work was done while the author was a member of the Formal Methods Group at IBM Haifa Research Laboratory.

on the readability of the rest of the paper, which is self-contained). Section 3 describes the three tiered design methodology. Section 4 reviews a number of formal verification projects in which RuleBase was deployed in a variety of ways according to the aforementioned methodology. Section 5 summarizes and concludes.

2. The RuleBase formal verification tool

RuleBase [2] is a formal verification tool developed by the IBM Haifa Research Laboratory. It is based on SMV [9], the symbolic model checker developed by Ken McMillan at Carnegie Mellon University. RuleBase was described in [2]; we will only briefly touch on some important points here.

The verification process with RuleBase takes three inputs. The first input is the design, which can be coded in either VHDL or Verilog. The second input is the environment, which describes the legal input sequences of the design under verification. These are coded in the language EDL, a dialect of the SMV input language. Finally, the properties to be verified are input in the language Sugar [2], which is a user-friendly superset of CTL [5]. The execution flow of RuleBase is shown below in figure 1.

Much of the RuleBase development effort has been put into various techniques which address the state explosion problem. Powerful reduction techniques, both syntactic and semantic, identify and remove logic which does not affect the properties being checked. Safety properties are checked on-the-fly [4], with the ability to apply partial search [12] and guided search [16] algorithms, as well as over-approximations. The tool orders partitions [7] and BDD variables.

Debugging aids are also an important part of the RuleBase tool. A reduction analyzer gives information regarding signals removed by reductions. Vacuous passes [3] as well as

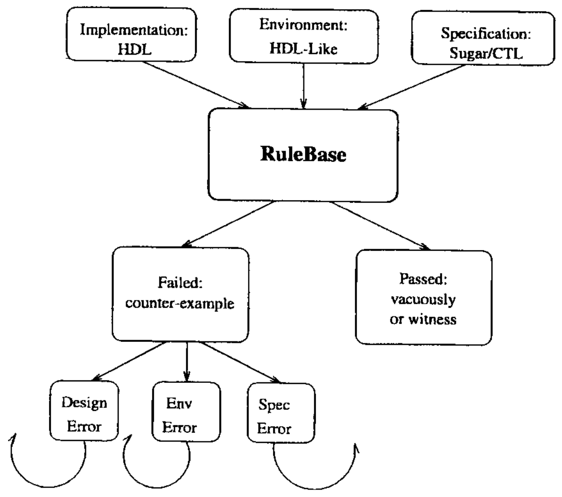

Figure 1. RuleBase execution flow.

properties which are found to be tautologies or contradictions are flagged for the user, and counter-examples and witnesses are presented as timing diagrams.

3. A three-tiered methodology

The use of RuleBase in IBM is based on a multi-level deployment, comprised of high-, mid-, and low-level applications of the RuleBase tool. Each of these is described in detail below.

3.1. High-level verification

Complicated hardware control typically implements some concurrent algorithm, in which a set of communicating processes cooperate in order to achieve the desired functionality. If the micro-architecture describes the algorithm in sufficient detail, it can be verified against the desired functionality in a process known as high-level, or architectural verification. In this mode, the set of properties to be verified is very simple, and directly expresses the desired functionality of the algorithm. For instance, the desired functionality of the memory/cache controller in a cache coherent multi-processor system is that it return correct data. Thus, the central safety property (simplified here slightly) can be expressed as "If a data value D is written to address A, then the next time address A is read from, the read will return data value D." In addition, one liveness property ("every transaction is eventually completed") is checked. Other safety properties (for instance, that cache coherence is maintained, that buffers do not overflow, etc.) may be checked because they are easier to verify and find bugs more quickly than the central safety property. However, because the central safety property directly expresses the desired functionality, the question of whether or not enough properies have been written is not relevant.

High-level verification is applied at the very early stage of high level design, when the system architecture is still being developed. In this mode, application is by formal verification specialists, who work together with the architects and designers in a process which can be described as formal design [6]. In architectural verification, an abstract model of the system is built and model checked. This process allows fast focus on algorithmic problems and thus finds errors in early design approaches. At this level of application, the goal is to completely verify the system architecture, with the intent that the design be free of architectural bugs late in the design cycle, when fixing them is much more difficult and costly.

Formal verification specialists are needed in this mode because of the aggressive goals of high-level verification. A formal verification specialist is experienced in modeling and abstracting systems for formal verification, and is familiar also with the internals of the model checking process. The specialist uses this knowledge to avoid and/or deal with size problems inherent in a system-level verification effort.

3.2. Mid-level verification

Mid-level verification is the process of validating a functional unit or block against a comprehensive set of properties, derived directly from a detailed specification. For instance, a bus interface unit may be verified in this manner against a set of properties describing the bus protocol. The goal is to attain a high degree of confidence in the correctness of the design under verification.

To this end, trained formal verification engineers devise and follow a thorough verification plan. Critical logic circuits of up to 5000 latches (pre-reduction) are verified in this way. Because of the large pre-reduction sizes, it is not possible to completely verify designs at this level. Rather, the verification plan addresses various reduced environments which will result in a number of reduced models, each of which attempts to cover a portion of the design. Thus, mid-level verification is not only mid-way between high- and low-level verification in the size of the designs to which it is applied, it is also mid-way between the complete, exhaustive expectations of high-level verification and the exploratory nature of low-level verification.

Dedicated formal verification engineers are needed in this mode because of the specialized knowledge required to deal with large pre-reduction sizes of the unit under verification. A formal verification engineer is experienced in environment modeling strategies which give good verification of large designs, and is an expert user of the formal verification tool. The formal verification engineer uses this knowledge to develop a verification plan for a large piece of logic, as well as to perform the verification.

3.3. *Low-level verification*

Low-level verification, or design exploration, is the activity of exploring a partly-specified piece of logic in order to obtain insight into the design behavior. In this mode, the application is usually by logic designers to their own design, and is limited to small pieces of no more than 100 latches (pre-reduction). This work is not driven by the expectation of completely verifying the design, but rather of exploring the logic in order to obtain better insight into the state space. Typical properties are therefore aimed at checking whether an Finite State Machine (FSM) can move from state s1 to state s2, or whether a specific bus transaction can occur, rather than whether the FSM always makes the correct transition, or whether a bus transaction always completes correctly. This type of application works in the early design stages, when the design is still unstable, and much of it not yet coded. It results in a cleaner design being released for higher verification stages, and frequently results in a simpler design due to a better comprehension by the designer of the design behavior.

Application in this mode is necessarily by the logic designer because low-level verification is applied to small, usually undocumented pieces of logic. Since the goal is not a full verification, but rather design exploration, the logic designer is not required to be an expert user of the verification tool. Logic designers can begin using the tool in this mode after a short (1–2) day introduction to RuleBase.

Together, these three modes of using formal verification provide a framework for the analysis of the hardware throughout its life cycle, starting from concept development, and continuing throughout design implementation and verification. In the next section, several formal verification projects, at varying modes of application, are described.

4. Review and analysis of some formal verification efforts

This section focuses on several product design efforts across IBM during the years 1996–1999. These include an AS/400 compression cache, a Netfinity SMP Server, a SP/2 System Node Controller, a SP/2 Communication Switch, an AS/400 Storage Control Unit, and an AGP 2X Core. We describe each project, and examine in detail one or more bugs found by formal verification. The type of bugs described varies; some are complex deadlocks or livelocks, while others are simpler bugs found in the earliest design stages. Together, this variety of bugs represents the different roles which formal verification takes at different points in the hardware design flow, according to the methodology described in Section 3.

4.1. High-level verification

AS/400 Compression Cache (1998). An 8-way AS/400 MP system containing a compression cache was modeled and formally verified by a formal verification specialist in Rochester, Minnesota. AS/400 is IBM's mid-range server designed for business use.

Rules were coded to check coherence between the compression cache and other caches in the system, and to verify that correct data was delivered to the processors. Four confirmed architectural flaws in the specification of the compression cache were found, having to do with the interaction of compression requests and other requests on the bus. For instance, the bus protocol defines a time lag between receiving a request and responding to it. The original architecture implied that a bus response should be calculated as early as possible after receiving a request. Architectural verification showed that, in rare cases, too early a calculation could result in loss of coherence, and confirmed that the solution was to delay the calculation until the response was actually needed.

The most important contribution of architectural verification to this project was confirmation that the particular use of the bus by the compression cache, which was a new and untried concept, was indeed workable. No additional architectural errors in the compression cache were found by simulation.

Netfinity SMP Server (1999). A Netfinity SMP server was modeled and verified by a team of formal verification specialists from Rochester, Minnesota, and Haifa, Israel, working closely with the Netfinity chipset system architects. IBM's Netfinity line consists of Intel processor based industry standard servers for business class applications.

Rules were run to check coherence of processor caches, and that correct data was delivered to the processors. Results are shown in figure 2 below. Bug count is for bugs found during formal verification of the chipset high level design only. Simulation (implementation) bugs are not shown.

Fifty chipset specification bugs were found during high level design. A few were simple misses to tables. Many were corner cases involving unusual interactions of two or more transactions. Three were deadlocks, and three of the bugs were violations of data consistency, despite the fact that cache coherence was maintained.

The chipset architects commented that the problems found by architectural verification speeded the design team's understanding of the Intel coherence architecture, and helped

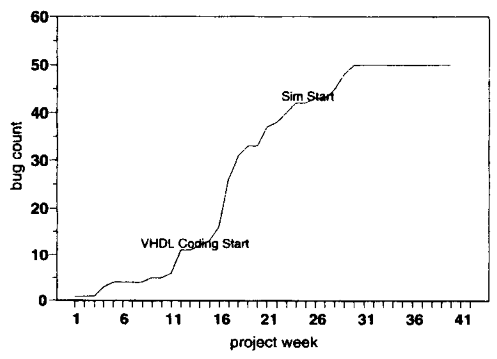

Figure 2. Coherence unit specification bugs.

to find problems in early chipset design approaches. It drove rigor, completeness and correctness of the chipset high level design, and resulted in VHDL which was derived directly and rapidly from a complete, algorithm-oriented specification. Millions of simulation cycles were run on this design, and tens of implementation bugs were found. However, no coherence-related architectural bugs were found by simulation. A more in-depth discussion of the design process used in this project can be found in [6].

4.2. Mid-level verification

SP/2 System Node Controller (11/96–6/97). The SP/2 system node controller was formally verified by a team of verification specialists in Austin, Texas and in Haifa, Israel, using the mid-level verification mode. The SP/2 parallel computing architecture is part of the Department of Energy's Accelerated Strategic Computing Initiative (ASCI), which is building the world's fastest supercomputer [15]. The node controller (NC) is an interface between four processors and the SP/2 system. The NC is comprised of four ports, each of which is partitioned into five blocks. Three more blocks interact with the ports. The system is shown in figure 3. Formal verification was done on all the NC blocks.

An elusive deadlock was found by this work. It involved an address multiplexer used to route high and low priority transactions coming from the ports: Processors P1 and P2 constantly have low priority requests pending. After P2 is served, but before P1 is served, another processor, P3, issues a high priority request. This high priority request is immediately served, and by the time it is completed, the arbitration cycle returns to P2. This results in the starvation of processor P1.

This bug was not found during several months of simulation over millions of simulation cycles. One month after formal verification began, this bug was discovered. In addition, 47 other bugs were found by the formal verification effort described here. The NC formal verification experience has been described in more detail in [1].

SP/2 Communication Switch (11/96–6/98). The SP/2 communication switch is also part of the SP/2 parallel computing architecture 4.2. It was formally verified by a team of formal

, completeness and
ıich was derived di-
illions of simulation
found. However, no
in-depth discussion

controller was for-
ınd in Haifa, Israel,
chitecture is part of
'e (ASCI), which is
NC) is an interface
f four ports, each of
ıe ports. The system
s.
ss multiplexer used
ocessors P1 and P2
before P1 is served,
request is immedi-
ıs to P2. This results

llions of simulation
ɔvered. In addition,
ere. The NC formal

ı switch is also part
by a team of formal

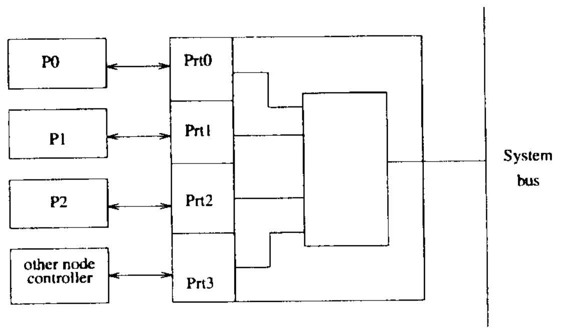

Figure 3. SP/2 System node controller.

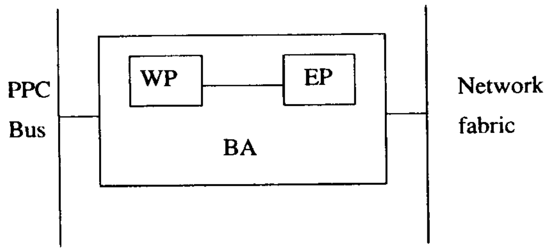

Figure 4. SP/2 Bus adapter.

verification engineers in Poughkeepsie, New York, and Haifa, Israel. The communication switch is comprised of three ASICs, of which the bus adapter (BA) was formally verified. The BA is shown in figure 4 below. It is the interface between the computer's native memory interface (PowerPC bus) and the rest of the adapter. The BA consists of two parts: the East Part (EP), which contains the adapter logic, and the West Part (WP), which is the PowerPC bus interface. Both were formally verified.

A subtle violation of the PowerPC bus protocol was found by this work. The PowerPC bus can serve the WP, or delay service by responding with a rerun response.[1] After a WP transaction receives a rerun response, it can be reissued only after receiving a rerun request on the PowerPC bus. The following scenario was found: a WP address transaction was delayed with the rerun response. After receiving two rerun requests, the WP correctly reissued the transaction. However, this transaction was also delayed with a rerun response. This time, the WP did not wait for a rerun request before reissuing the transaction, thus violating the PowerPC bus protocol.

Formal verification accounted for 58 bugs found in the SP/2 Switch design pass I (January 1997–December 1997). During pass II (January 1998–June 1998), an addition ten EP and one WP bugs were found. The development team in Poughkeepsie has subsequently initiated a technology transfer process whereby a local team owns the responsibility and knowledge required to deploy RuleBase in Poughkeepsie.

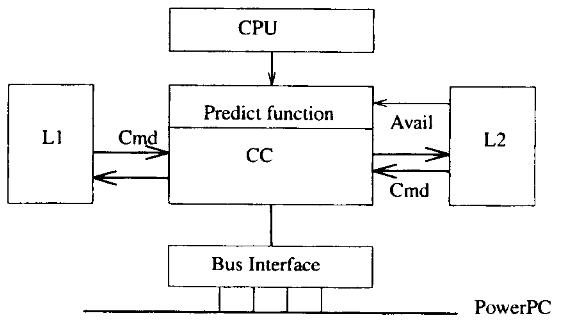

Figure 5. CC Block and environment.

4.3. Low-level verification

AS/400 Storage Control Unit (3/96–2/97). The storage control unit (SCU) of the IBM AS/400 system was formally verified jointly by the SCU design team in Rochester, Minnesota, and verification engineers in Haifa, Israel, using a mix of mid- and low-level verification. The SCU is comprised of five blocks, two of which were formally verified. The Cache Control (CC) block was one of the two, and is responsible for controlling requests to and from the L1 and L2 caches and main memory. The CC block and its environment are shown in figure 5. The environments are non-deterministically coded in EDL.

A subtle logic bug found in the AS/400 SCU using formal verification involved a deadlock in the CC. The L2 "free" signal indicates to the CC that the L2 can accept a command. The lookahead function indicates to the CC that it is likely that an L1 command will be driven the next cycle. When the CC sees that an L1 command is likely to be driven, it does not process an L2 command, so that it can remain free to process the expected L1 command. In certain situations the lookahead function gets "precisely out of sync" with the L2 free signal, so that the L2 asserts the free signal only during those cycles in which the lookahead function prevents the L2 commands from being processed. The result is a deadlock.

A simpler instance of this bug was found by both simulation and formal verification, and fixed. When the simulation test passed, the issue was closed. Simulation continued for three weeks without finding a problem in this portion of the CC logic. Only when a formal verification regression was finally run was the significantly more complex instance of this bug found and fixed.

Formal verification accounted for 68 of the bugs found in the two AS/400 blocks formally verified, and for most of the complex deadlocks found by the verification effort. The formal verification work on the SCU began quite a while before the simulation environment was available, and the first bugs were found by RuleBase nearly six weeks before simulation start.

A notable benefit of using formal verification on the SCU has been in the analysis of logic that was not easily controllable by the simulation environment. For example, formal verification found an error in the buffer allocation algorithm: in a rare case, the algorithm allocated the same buffer to two distinct resources. Creating a simulation scenario capable of finding such a situation was extremely difficult, even after the bug was known.

The AS/400 SCU design team closely examined its experiences with formal verification [8]. Some key observations were that formal verification is a powerful means for verifying designs with many different timings, nearly unreachable states, and possibilities for deadlocks. It should be used from the early stages of the design, while simulation is still not available, later to verify areas that do not have adequate simulation support, and as a regression tool. The detailed and exact specification required to build environments had a positive affect on the definition of the interfaces, by making them simpler than they would have been otherwise. They concluded that formal verification should be used not merely as a verification method, but as a design tool from the earliest stages of the design.

AGP 2X Core (1/97–6/97). The AGP 2X core [14], designed by the IBM Micro-electronics Division, is a dedicated high speed data path from the 3D graphics subsystem to the processor system memory. It is a logical superset of the PCI standard [10], with some modifications and enhancements targeted to reducing the entry code of 3D hardware accelerators. Formal verification was performed by a formal verification engineer in Haifa, Israel, in design exploration mode and in close collaboration with the team developing the core logic specification and micro-architecture.

In the AGP 2X core development effort, RuleBase was used for a few weeks in design exploration mode before the simulation environment was stable, and was the only means of verification in the early design stages. The ability to observe and validate design behavior in its early stages helped the design team pinpoint error-prone logic while it was still easy to fix. This resulted in quick turnaround time between the verification engineer and the design team.

Formal verification accounted for 18 of the bugs found in the AGP 2X core design. The primary target was the arbitration logic. With a small set of rules, RuleBase discovered quite a few bugs and logic holes in the arbiter, which helped the designers stabilize their arbitration scheme quickly and efficiently.

The design team commented that they implemented the design with much simpler logic than would have been used if formal verification had not been performed. With intensive and quick feedback by formal verification for every definition change, the designers managed to successfully build a simple and robust interface and control logic for the AGP 2X core.

5. Conclusion

We have described a multi-level application methodology for formal verification. Model checking is applied at various levels of abstraction (system vs. RTL), by persons variously trained and specialized (formal verification experts vs. hardware designers), with various intents (exhaustive verification through design exploration). We take a pragmatic view of the possibilities of formal verification. Thus, while this methodology does not provide an absolute guarantee of success, we have found it to provide important added value to the traditional simulation-based verification process.

Acknowledgments

In addition to the authors, the following members of the Formal Methods Group at the IBM Haifa Research Laboratory have greatly contributed to the development of the model checking technology upon which the work described in this paper is based: Shoham Ben-David, Dana Fisman, Leonid Gluhovsky, Anna Gringauze, Sharon Keidar, Mark Matusevich, Yoav Rodeh. The verification work described in this paper is the work of the following people in addition to the authors: Jason Baumgartner, Chris Daniel, Don Freerksen, Gary Lippert, Kyle Nelson.

Note

1. The concepts of rerun response, rerun request, and reissue are described in detail in the PowerPC Architectural Definition [11].

References

1. J. Baumgartner and T. Heyman, "An overview and application of model reduction techniques in formal verification," in *Proc. IEEE IPCCC*, Phoenix, Arizona, pp. 165–171, 1998.
2. I. Beer, S. Ben-David, C. Eisner, and A. Landver, "RuleBase: An industry-oriented formal verification tool," in *Proc. 33rd Design Automation Conference*, 1996, Las Vegas, Nevada, pp. 655–660.
3. I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh, "Efficient detection of vacuity in ACTL formulas," in *Proc. CAV '97*, Springer-Verlag, Haifa, Israel, pp. 279–290.
4. I. Beer, S. Ben-David, and A. Landver, "On-the-fly model checking of RCTL formulas," in *Proc. CAV '98*, Springer-Verlag, Vancouver, BC, Canada, pp. 184–194.
5. E.M. Clarke and E.A. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," in *Proc. Workshop on Logics of Programs*, Lecture Notes in Computer Science, Vol. 131, Springer-Verlag, Berlin, 1981, pp. 52–71.
6. C. Eisner, R. Hoover, W. Nation, K. Nelson, I. Shitsevalov, and K. Valk, "A methodology for formal design of hardware control with application to cache coherence protocols," in *Proc. DAC 2000*, Los Angeles, California, pp. 724–729.
7. D. Geist and I. Beer, "Efficient model checking by automated ordering of transition relation partitions," in *CAV 1994*, LNCS 818, Stanford, California, pp. 299–310.
8. G. Lippert and D. Freerksen, "Northstar SCU formal verification," Internal Memo, 1997.
9. K.L. McMillan, Symbolic Model Checking, Kluwer Academic Publishers, Boston/Dordrecht/London, 1993.
10. PCI specifications, www.pcisig.com/tech/index.html
11. PowerPC User Manuals, www.chips.ibm.com/techlib/products/powerpc/manuals
12. K. Ravi and F. Somenzi, "High-density reachability analysis," in *Proc. Intl. Conference on Computer-Aided Design (ICCAD '95)*, San Jose, California, pp. 154–158.
13. A.L. Sangiovanni-Vincentelli, P.C. McGeer, and A. Saldanha, "Verification of electronic systems," in *Proc. DAC '96*, Las Vegas, Nevada, pp. 106–111.
14. www.chips.ibm.com/products/asics/cores/briefs/agp_2x.html
15. www.llnl.gov/sccd/lc/asci
16. C.H. Yang and D.L. Dill, "Validation with guided search of the state space," in *Proc. DAC '98*, San Francisco, California, pp. 599–604.