

# Fifteen Years of Formal Property Verification in Intel

Limor Fix

Intel Research Pittsburgh  
Limor.fix@intel.com

**Abstract.** Model checking technologies have been applied to hardware verification in the last 15 years. Pioneering work has been conducted in Intel since 1990 using model checking technologies to build industrial hardware verification systems. This paper reviews the evolution and the success of these systems in Intel and in particular it summarizes the many challenges and learning that have resulted from changing how hardware validation is performed in Intel to include formal property verification. The paper ends with a discussion on how the learning from hardware verification can be used to accelerate the industrial deployment of model-checking technologies for software verification.

**Keywords:** Model checking, formal specification, formal property verification.

## 1 Hardware Verification in Intel

### 1.1 The First Generation

Since 1995 Intel engineers have been using formal verification tools to verify properties of hardware designs. The first generation of Intel formal property verification tools [1, 2, 3, 4, 5, 6], called *Prover*, included an enhanced version of SMV, the BDD-based model checker developed by Ken McMillan [17], and a specification language, called FSL, that was an hardware linear temporal language inspired by LTL, the linear temporal logic proposed by Pnueli [18]. The compiler for FSL translated the linear logic into automata using algorithms similar to [19]. FSL was used both to specify *formal properties* to be verified by the model checker and to specify *checkers* to be checked dynamically during simulation of the hardware designs.

Two lead CPU design teams used Prover for 1995 till 1999. Both teams reported successful usage of the new verification technology and in particular *high quality*

*bugs* have been discovered. These bugs were classified as bugs that either would have been found by other validation tools much later in the design cycle or bugs that might otherwise would have escaped all the validation tools and reach the silicon. Very important leanings were generated by the two design teams about: how, where, when and by whom hardware formal verification tools should be used and moreover the remaining technology challenges were identified.

*How* - *Automated abstraction* [1, 4] and *modular verification* [20] using *assume-guarantee paradigm* were used to overcome the limited capacity of the tools. Properties were developed to capture the intended behavior of the inputs and the outputs of each module. Properties on the outputs of a module were verified using the assumptions on the inputs of the module. Properties on the output signals of the module served also as assumptions on the inputs of the neighboring module.

*Where* - Only selected areas of the CPU were formally verified. These were areas of high risk in which new complex functionality was added or areas in which the properties to be proven were obvious. For example, the arithmetic units of the CPU had both characteristic.

*When* - The decision on when to use the formal tools was not easy. On the one hand, using the tools very early in the design cycle even before the entire design was coded, that is before the RTL simulation can start, was very successful. Bugs were revealed early and did not even reach the early simulation models. On the other hand, since at that time the RTL was unstable changes to the RTL required recoding of the properties and assumptions again and again. The team ended up using the tools relatively late in the design cycle and indicated that support for early verification would be very beneficial.

*By Whom* - Traditionally, limited validation was done by the RTL designers and most of the in depth verification was carried out by the validation engineers. We encouraged both groups, the designers and the validators to use Prover. In terms of number of users, we had better success with the validation groups. More resources were devoted in the validation teams to use the new formal property verification system. The small number of designers that used the tools indicated bigger success in finding more bugs with less efforts. The designers had an easier task because they were very familiar with the design and developing appropriate assumptions and properties was much easier for them.

Among the main challenges that have been identified, it became clear that it was very difficult to develop good specifications. It was hard for designers to develop high level properties that do not mimic the details of the implementation, it was hard to train the designers to use a linear temporal language, it was impossible to know if enough properties were developed to express the entire desired behavior of the design,

and it was hard to maintain the properties since the design was changing and as a result the properties had to be changed accordingly.

Other challenges were also identified. One obvious challenge was the limited capacity of the model checker. As mentioned before, modularization and abstraction techniques were developed. The design was divided into smaller components and assumptions were placed on the inputs of the blocks. The introduction of assumptions created additional challenges: it was hard to know which assumptions were needed, it was hard to identify circular reasoning, and it was hard to make sure all assumptions were verified. As for abstraction, semi-manual abstraction techniques were developed, however, it was impossible to deploy them because whenever the design changed the abstract model had to be re-built. We ended up focusing on automated abstraction only [1, 4].

The most important learning from the first deployment of formal property verification tools in two real complex design projects was the need to either identify which previously existing validation activity can be omitted and replaced by the formal verification effort. Or, alternatively, the need to smoothly integrate the formal verification activity into the rest of the design and validation efforts while minimizing any additional manual effort by either the designers or the validators.

## 1.2 The Second Generation

In the following years, Intel formal verification system developed very fast, moving from a single BDD-based model checking engine to multiple engines. A SAT-based model checker [21, 7] and a symbolic simulation engine [22, 8] were added. Formal specification coverage tools were introduced to be able to measure the quality and the completeness of the set of properties that have been developed [10, 11, 12, 14]. These tools indicated which parts of the implementation were not specified by the properties, which properties were vacuously true with respect to the given design and how one set of properties covers another set of properties. A database of properties and assumptions was developed to help detect circular reasoning and track the status of all properties.

With high effort, a new generation of the specification language, called ForSpec [9, 10], was developed. This language had two versions, a standalone version in which the properties are developed in a separate file detached from the RTL design and an embedded version in which properties were developed as assertions embedded inside the RTL model, that is, as part of the Verilog code. In 2003, Intel donated ForSpec to Accelera, part of the effort to make ForSpec an IEEE standard for formal verification language [23]. The resulting IEEE 1850 Standard, has adopted major parts of ForSpec standalone version. The IEEE standard for SystemVerilog has adopted major parts of ForSpec embedded version for the SystemVerilog assertions, also called SVA.

The barrier to moving from a limited deployment to wide spread deployment of formal property verification in Intel was crossed mainly due to two developments: the first was the introduction of ForSpec assertions inside the Verilog code, thus allowing the designers to easily code and maintain the properties (assertions). The second was the integration of the formal verification activity with other validation efforts. In particular, the RTL designer had two reasons to annotate his/her code with assertions. The assertions were always checked during simulation and in addition the assertions served as assumptions and properties for formal verification. In case, an assertion was too complex to be verified formally it was still very useful as a checker in simulation.

In the last two years we have extended the use of formal verification technology to other parts of the design. A very successful system has been developed for the verification of microcode [15].

## 2. Industrial Deployment of Model Checking for SW Verification

A large body of very successful research already exists in the area of software formal

verification. The goal of using these techniques widely by software developers, has not been achieved yet. As our experience in hardware verification taught us the following questions need to be answered: how, where, when and by whom. Below I present my beliefs that are based only on my hardware experience.

## **2.1 How**

Embedded assertions have been proven very successful in hardware verification, thus, I believe assertions embedded in programs is a very most promising approach. For assertions to be successful, they need to be dense enough, that is, enough assertions need to be manually inserted by the programmer or generated automatically by the compilers. A very successful method to increase the number of assertions is to develop a library of parameterized assertions that express common requirements for software correctness.

Once assertions are embedded in the program they need to be utilized for several purposes, for example, for compiler optimization, for debugging using gdb-like debuggers and for formal verification. Database of assertions need to be generated automatically from the program code for managing the status of the assertions.

## **2.2 Where**

While parallel programming and distributed algorithms were very active research fields 20 years ago but had limited deployment, these days parallel programming becomes a necessity. Due to the power wall in silicon technology all state of the art computing devices have multiple processing units and the transition to chip multiprocessors is happening very fast. New programming paradigms are being developed to combat the difficulties of parallel programming, e.g., transaction memory programming. This transition and the need for new programming paradigms and languages create opportunities for formal verification of software. The new developed parallel programming paradigm should include embedded assertions as integral part its design.

Additional area in which formal verification of software should apply to is security. The problems of viruses, spyware, and worms are growing fast and have very high costs. Extra efforts to reduce vulnerability of software are likely to be invested to prevent these costs.

### 2.3 When

As with hardware, the best person to develop the assertions is the developer (programmer) himself. Assertions development should become an integral part of writing software and assertions should be embedded in the code while the code is generated and they should be as dense as possible. The compilers should also generate embedded assertions in addition to the ones inserted by the programmer.

### 2.4 By whom

In large software companies, just as in hardware companies, large validation groups are focused on raising the quality of the code by intensive debugging. I believe most assertions need to be developed by the programmers themselves and should be always turned on. The validation teams may add more assertions later and most importantly they need to work with the assertion database to complete the verification of all assertions.

**Acknowledgments.** The success of formal verification in Intel was the result of a great collaboration between industry and academia. Intel formal verification systems have been developed with intensive collaboration with researchers around the world. In particular, my team has worked with Moshe Vardi, Amir Pnueli, Orna Grumberg, Zohar Manna, Ed Clarke, Randy Bryant, David Dill, Sharad Malik, Assaf Schuster, P.P. Chakrabarti, P. Dasgupta, Scott Hazelhurst, Enrico Giuchilia. It also required great openness and willingness to take risk from the managers of the design teams and the managers of Intel internal CAD group.

### References

1. G. Kamhi, O. Weissberg, L. Fix, Z. Binyamini, Z. Shtadler: "Automatic data-path extraction for efficient usage of HDD", 9th International Conference, CAV'97, pp. 95-106 LNCS 1254, Springer.
2. G. Kamhi, L. Fix: "Adaptive variable reordering for symbolic model checking", IEEE/ACM International Conference on Computer Aided Design (ICCAD) 1998.
3. G. Kamhi, L. Fix, Z. Binyamini: "Symbolic Model Checking visualization", Second International Conference on Formal Methods in Computer-Aided Design (FMCAD) 199

4. S. Mador-Haim, L. Fix: "Inputs elimination and data abstraction in model checking", Second International Conference on Formal Methods in Computer-Aided Design (FMCAD) 1998.
5. Ranan Fraer, Gila Kamhi, Limor Fix, Moshe Vardi: "Evaluating Semi-Exhaustive Verification Techniques for Bug Hunting" SMC, 1999 (CAV'99 workshop).
6. Ranan Fraer, Gila Kamhi, Barukh Ziv, Moshe Y. Vardi, Limor Fix: "Prioritized Traversal: Efficient Reachability Analysis for Verification and Falsification", CAV 2000.
7. Fady Copt, Limor Fix, Ranan Fraer, Enrico Giunchilia, Gila Kamhi, Armando Tacchella, Moshe Vardi: "Benefits of Bounded Model Checking at an Industrial Settings", CAV2001, LNCS 2102.
8. Scott Hazelhurst, Osnat Wiessberg, Gila Kamhi, Limor Fix: "A hybrid verification approach: getting deep into the design" DAC 2002.
9. R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M. Vardi, Y. Zbar: "The ForSpec temporal Logic: A new Temporal Property Specification Language". TACAS 2002, 296-311.
10. R. Armoni, L. Fix, A. Flaisher, O. Grumberg, N. Piterman, A. Tiemeyer, M. Vardi: "Enhanced Vacuity Detection in Linear Temporal Logic", CAV 2003, 368-380
11. P. Basu, S. Das, P. Dasgupta, P.p. Chakrabarti, C.R. Mohan, L. Fix, "Formal Verification Coverage: are the RTL-properties covering the design's architectural intent. DATE 2004, 668-669.
12. P. Basu, S. Das, P. Dasgupta, P.p. Chakrabarti, C.R. Mohan, L. Fix: "Formal Verification Coverage: Computing the coverage gap between temporal specifications", ICCAD'2004
13. R. Armoni, L. Fix, R. Fraer, S. Huddleston, N. Piterman, M. Vardi: "SAT-based induction for temporal safety properties", BMC workshop at CAV'04
14. P. Basu, S. Das, P. Dasgupta, P.p. Chakrabarti, C.R. Mohan, L. Fix, "Formal methods for analyzing the completeness of assertions suite against a high level fault model". To be published VLSI Design'2005 conference at Kokata.
15. Tamarah Arons, Elad Elster, Limor Fix, Sela Mador-Haim, Michael Mishaeli, Jonathan Shalev, Eli Singerman, Andreas Tiemeyer, Moshe Vardi, Lenore Zuck, "Formal Verification of Backward compatibility of Microcode", 17<sup>th</sup> International Conference on Computer Aided Verification, July 2005, Edinburgh.
16. Limor Fix, Orna Grumberg, Tamir Heyman, Assaf Schuster, "Verifying very large industrial circuits using 100 processes and beyond", Third International Symposium on Automated Technology for Verification and Analysis, Oct 2005. Best paper award.
17. Ken .L. McMillan. "Symbolic Model Checking: an approach to the state explosion problem", PhD Thesis. CMU CS-929131, 1992.
18. Amir Pnueli. "The temporal logic of programs", In Proc. 18<sup>th</sup> IEEE Symposium on Foundation of Computer Science, 1977.
19. E. Clarke, O. Grumberg, H. Hamaguchi: "Another Look at LTL Model Checking", Formal Methods in System Design, Volume 10, Number 1, February 1997. Also in CAV'94.

20. Amir Pnueli. "In Transition from Global to Modular Temporal Reasoning about Programs", in K.R. Apt, editor, *Logics and Models of Concurrent Systems*, sub-series F: Computer and System Science, pages 123-144. Springer-Verlag, 1985.
21. A. Biere, A. Comatti, E. Clarke, and Y. Zhu. "Symbolic model checking without BDDs". In *Proc. of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, Lecture Notes in Computer Science. Springer-Verlag, 1999.
22. R.E. Bryant and C-J. Seger. "Formal verification of digital circuits using symbolic ternary system models." In E.M. Clarke and R.P. Kurshan, eds. *Workshop on Computer Aided Verification. 2<sup>nd</sup> International Conference, CAV'90*, Lecture Notes in Computer Science 531, Springer-Verlag 1990.
23. Moshe Vardi. "From Church and Prior to PSL: Standing on The Shoulders of Giants". This volume.