Cost-efficient Formal Block Verification for ASIC Design

K. Winkelmann
Infineon Technologies AG
klaus.winkelmann@infineon.com

J. Trylus
Siemens AG
joachim.trylus@icn.siemens.de

D. Stoffel
Universität Kaiserslautern
stoffel@eit.uni-kl.de

G. Fey
Universität Bremen
fey@informatik.uni-bremen.de

Abstract

ASIC designs for future communication applications cannot be simulated exhaustively. Formal property checking is a powerful technology to overcome the limitations of current functional verification approaches. The paper reports on a large-scale experiment employing the CVE property checker for verifying the block-level functional correctness of a multi-million gates ASIC.

This new verification methodology achieves substantial quality and productivity gains. The two biggest advantages are that coding and verification can be done in parallel, and that the whole state space of a test case will be verified in a single run.

Formal property checking simplifies and shortens the functional verification of large-scale ASICs at least in the same order of magnitude as static timing analysis did for timing verification.

1 Challenges in Designing Complex Wireless Communication ASICs

In the last decade the useable chip area for semi-custom ASIC design increased rapidly. In the early 90's the maximum available gate size was about several 100k. Today it's more than 20 million in a 0.11µm technology. The attempt of using the available gate size results in the two main problems:

- how to meet the functional verification coverage
- how to meet the correct timing of the ASIC.

Using new generation layout-driven synthesis tools and static timing analysis a lot of progress has been made to solve the second problem. The first one, though, has remained a major challenge, especially for large-scale designs like the one described in this paper.

1.1 Design Characteristics

The ASIC considered in this paper is developed by Siemens Mobile Communications. It serves as a 'number-cruncher' coprocessor in the up-link of a UMTS base station. An external client controls its operations, called tasks, via the control path of the ASIC, using parameter tables. The client is also responsible for starting and stopping the tasks. This is achieved by setting execution conditions, stored in execution tables, which are monitored by the control path.

The size of the coprocessor is more than 10 million gates. Its clocking frequency is above 100MHz. The average data throughput lies in the range of several Gbit/s.

1.2 Verification Challenge

The main challenge for the verification of the ASIC is its complicated task scheduling. It is dependent on many parameters. Most of the parameters are allowed to change every 10 ms (the duration of one frame). This results in a case space of more than 10^{18} cases. Obviously, this case space could not be verified with classical approaches like simulation.

2 Infineon's CVE Circuit Verification Environment

Infineon Technologies AG offers a broad range of semiconductor products for important target markets such as mobile communications and networks, access control and network security, car electronics, and more. In order to meet its highly demanding cost and quality targets, Infineon is using an advanced design flow incorporating state-of-the-art commercial tools, as well as innovative in-house tools.

2.1 Tool environment

The Infineon design flow includes formal verification based on the in-house system CVE (Circuit Verification Environment), which has been developed by Siemens and Infineon for more than a decade, and is made available beyond in-house use at Infineon also to Siemens.

Among other components CVE comprises language front-ends including VHDL and Verilog, the equivalence checker *gatecomp* and a property checker called *gateprop*.

2.2 gateprop Property Checker

Functional verification, using *gateprop*, is based on

- compiling (automatically) the design into an internal finite state machine representation, and
- formalising (manually) its specification using a simple temporal property language, called ITL.

An ITL property is essentially a constraint on the design's signals over a finite time interval. For the property to be valid means to hold for every observation window of the appropriate length, in every run admitted by the design. The tool checks each property and, if it is found to be not universally valid, produces a counter-example.

The basic concepts of the language are:

HDL flavour: The user chooses to write in either a VHDL or Verilog syntax, using familiar language constructs to quickly get up to speed with property writing.

Time steps: A property is written over a number of time steps, from time `t` (i.e. `t+0`) to a future time (e.g. time `t+4` after 4 clock cycles). Consequently, there are a few time constructs in the language (e.g. `at t`, `during [t, t+2]`, etc.).

Each property consists of an “assume” and a “prove” part.

Assume part: This part allows the designer to specify the working mode of the design under inspection. Assumptions such as ’no reset occurs at time t’ are typically necessary to investigate if the design exhibits a particular behaviour. Further typical assumptions are ’at time t the input connection_request is high’ or ’there will be no write_request during time interval t+1 to t+5’.

Prove part: This part of the property specifies expected behavior. Typical assertions in the prove-part are ’the grant output is set at time t+5’ or ’the write_acknowledge output will somewhere be issued within time interval t+1 to t+3’.

There are a number of language extensions that are designed to result in *concise but intuitive* properties, including data quantifiers, a powerful macro mechanism and time variables.

3 Block Verification

The verification approach in our ASIC project combines block-level formal verification with system-level simulation. Working in a bottom-up way, each block was verified formally before the system level is even coded.

The verification team was headed by one experienced CVE verification engineer.

All verification engineers had some background in formal methods and have been trained to use CVE. The major verification load was handled by this team, separating the concerns of designing and verifying. In addition, each designer underwent the same CVE training. Over the duration of the project the designers themselves increasingly started using the formal tool, e.g. adapting properties and running regressions.

In the verification process we can discern several phases. For each of them a close interaction of designer and verifier was practised:

Preparatory formalisation: using the informal specification, and interaction with the designer team, first properties were written even before block-level architecture is available. This required at least the entity description and saves some effort for the later phases – however in most cases the actual verification also required some knowledge of implementation details, such as the names of state variables.

Initial block verification: parallel to coding of a block properties were written, and as soon as the VHDL code could be successfully compiled, formal verification was started. Alternatively, the designer often ran a first simple simulation for a few standard cases before he handed the code over for verification. The latter option filters out some trivial bugs, but makes little difference from a global perspective. In this phase both trivial and complex bugs were found (and fixed), and as a result a formal block-level specification existed which covered the block's function completely, and truthfully with respect to the implementation.

Block-level regression: when a new HDL version was checked in the existing property suite was re-run, in some cases catching errors that were introduced by the change. This was often but not always possible without adapting the property suite.

Specification adaptation: as is common in a large innovative project as this, the system specification changed during the development duration. E.g., certain additional modes and flags were proposed by system engineering, and had to be incorporated into the already coded blocks. In this case properties had to be adapted in parallel with the code change, and re-checked. The complete regression suite was very helpful in these cases as it is by no means trivial to maintain the existing function while adding new ones.

Interface verification: Once two or more communicating blocks had been completely verified at the block level, the formal specification provided an excellent means to check their mutual interfaces. The ITL properties unequivocally describe the timing, handshake protocols and dependencies for each partner of an interface.

An example of such a situation is this: block A produces a result x, together with a “valid” flag x_valid. Block B, which consumes x, assumes that if x is not valid, it will not take the special value

X'010', and acts on this assumption. None of the blocks is faulty by itself, but B's assumption is just not guaranteed by A.

Such dependencies were captured in ITL properties, and by carefully reviewing them several bugs were discovered that resulted from two designers' deviating understanding of the informal specification.

3.1 Verifying the Control Path

Verifying control logic is an ideal application of property checking. Instead of generating sophisticated stimuli to check normal operation as well as the corner cases, properties cover all cases of an expected functionality at once. Moreover the specification of the control path is done using Finite State Machines. This directly allows to derive properties from the specification and check their validity in the RTL-description.

On the first synthesisable version of the code a large number of bugs were found. This is due to the fact that no simulation run preceded the formal verification, so a lot of simple bugs occurred. The advantage was that no effort was spent setting up a test bench at module level for the task controller.

The most important contribution of the formal verification was the detection of some difficult bugs. Often these were detected by the most general properties – such as mutual exclusion of events or a condition that always has to be met. While there was no error under most circumstances, the occurrence of several rare conditions at once caused a faulty behaviour. Bugs of this kind would not have been caught by a simulation run.

3.2 Data path results

Our ASIC has many blocks with arithmetic functions. These contain more of the standard arithmetic such as complex addition and multiplication. Although arithmetic is typically problematic for formal verification tools, most blocks could be verified by CVE. In some cases we did encounter complexity problems, which, however, could always be alleviated by manual interaction, for example, by reducing the bit widths of the arithmetic operators. Typical design errors found in these blocks include wrong operand signs, wrong comparison operators (‘<’ instead of ‘=’) and typical entity interface problems.

Note that property checking cannot completely replace simulation. For example, as stated before, designers usually maintain a test bench during the design process to be able to quickly simulate the basic functionality concurrently with coding and to immediately remove the most obvious design errors. Interestingly, after running these simulations, the designers very often felt sure that their designs were free of error, especially because their test benches operated the design at maximum load. A number of errors stayed undetected just because the designed system was simulated at extreme operating conditions, thereby simply disregarding the “ordinary” stimuli. These errors were, however quickly discovered by formal verification.

4 Overall Result

Formal verification has very rarely been applied to a large design project with the outcome that the expected degree of coverage was met. In this project, block-level verification was completely covered by formal property checking. Comparing this to a traditional simulation-based approach, we have to consider several factors, as follows.

4.1 Verification Cost

It would be naive to believe that the benefits of formal verification come for free. The total *human effort* for writing properties in our project was in the order of one person year.

This has to be compared to the total human effort for HDL coding on the one hand, and writing test benches on the other. The total coding effort was close to 2 person years. As test benches at the module level were not used in this project, we can only relate this to previous projects and conclude that the formal approach requires less total effort than thorough block-level verification, but not drastically so.

The *computation time* for a complete regression run has to be compared to the total simulator run time. On a basis of 30 blocks verified, we find that the sum of all verification run times is in the order of 50 CPU hours. However, fewer than 2 % of all theorems (10 out of a total of about 700) account for more than 90% of this computation time. In other words 98% functional coverage is possible within only 5 hours. Given that simulator time is today one of the severely limiting factors in the ASIC quality assurance process, these figures show that formal property checking provides a very valuable progress here.

4.2 Quality Improvements

The *quality achieved* by block-level verification is probably the most important factor. It can be measured by the number of bugs which make it to the later stages of system simulation and emulation, or even to silicon. Although the project is on-going at the time of this report, we have already reasonable grounds to estimate that this number is cut down substantially. This claim is based on analysing and classifying more than 100 bugs discovered by property checking. This analysis showed that more than a third of these would have been missed by block-level as well as system-level simulation.

4.3 Outlook

Time to market and first-time-right silicon are the most important targets in today’s ASIC development.

The promising results in this challenging ASIC project shows that formal property checking has the ability to give ASIC verification a substantial boost.

At Siemens Mobile Networks’ UMTS-ASIC development formal property checking is now one of the supporting pillars of future design flows.

References

[1] J. Bormann, C. Spalinger, Formale Verifikation für Nicht-Formalisten, IT+TI 2/2001.

[2] M. Bartley, D. Galpin and T. Blackmore: “A Comparison of Three Verification Techniques: Directed Testing, Pseudo-Random Testing and Property Checking”, Proc. Intl. Design Automation Conference (DAC-02), pp. 819-823.

[3] A. Biere, A. Cimatti, E. Clarke and Y. Zhu, “Symbolic Model Checking without BDDs”, Lecture Notes in Computer Science, vol. 1579, pp. 193-207.

[4] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang and S. Malik: “Chaff: Engineering an Efficient SAT Solver”, Proc. Intl. Design Automation Conference (DAC-01), pp. 530-535, Las Vegas, June 2001.

[5] 3GPP TS 25.213 V3.6.0 (2001-06) Spreading and Modulation (FDD).

[6] H. Holma, A. Toskala, “WCDMA For UMTS”, John Wiley & Sons, 2001.