Gatecomp: Equivalence Checking of Digital Circuits in an Industrial Environment

Rolf Drechsler
Institute of Computer Science
University of Bremen
28359 Bremen/Germany
drechsle@informatik.uni-bremen.de

Stefan Höreth
CL DAT DF LD V
Infineon Technologies AG
81739 München/Germany
Stefan.Hoereth@infineon.com

Abstract

This paper outlines formal verification in general and then introduces CVE's equivalence checking tool gatecomp, an equivalence checker developed in the formal verification group at Infineon, Germany. The basic verification tasks are described and the advanced features of the tool are discussed. The application of gatecomp to large industrial examples is reported. This demonstrates the power of the tool for various verification tasks, like netlist vs netlist comparison, RTL vs. netlist comparison or RTL vs. RTL comparison.

Keywords: formal hardware verification, equivalence checking, multi-engine concept.

Why use Formal Verification?

The traditional practice of functional verification by simulation reaches its limits and threatens to block exploitation of deep submicron opportunities in applications. Therefore technological alternatives to simulation have a brilliant future. Formal verification, i.e. the highly automated, mathematical analysis of logical levels of circuit design, is one of the few such alternatives. It brings to the user previously unimaginable quality, cost and time improvements for tasks constituting over 60% of the overall development efforts. Moreover, the approach allows verification to be concurrent with design, while simulation is often applied after first integration (see also [Kro99,Dre00,Pay01]).

Where to use Formal Verification?

Formal verification can be applied in nearly every stage of the design flow, as illustrated by Figure 1.

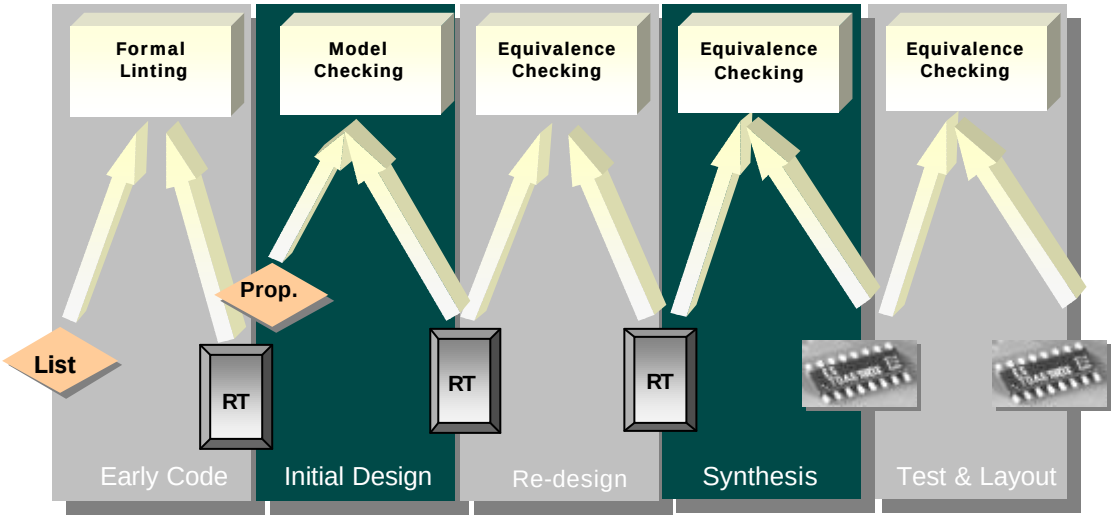

Figure 1 – Formal Verification Flow

This can start from static and dynamic checks of early RT code. Well before tests or formal properties of the code are available these checks quickly and easily improve the quality of block design and thus reduce the repair portion of subsequent verification. Then a property checker will test the RTL against system requirements (properties) providing a complete and detailed 100% analysis of blocks. Finally an equivalence checker will ensure that the design doesn't change as it is implemented and integrated. All of these tools as well as adaptations for specific usergroups (IP business, ASIC foundries) are available in the CVE toolset.

Equivalence Checking

Most designers will first experience formal verification when using an equivalence checking tool for sign-off e.g. to check that the final netlist has the same behavior as previous netlists and even the original RTL.

As an example, the general flow for the synthesis verification, i.e. checking the equivalence of a RTL description and a netlist, is shown in Figure 2. Starting from the RTL description a netlist is generated by a synthesis tool. Then both descriptions are translated into an internal *gate* format that is used by gatecomp to prove functional equivalence. The translation is done by the CVE frontends. This independence from the synthesis tool guarantees a further improvement of quality of the overall design.

In a similar way equivalence of RTL vs. RTL and netlist vs. netlist descriptions is proven.

The steps in general are as follows:

1. The designs are translated into an internal format.
2. The correspondence between the designs is established in a matching phase.
3. Equivalence or inequivalence is proven by the equivalence checker.
4. In case of an inequivalence a counterexample is generated and the design has to be debugged.

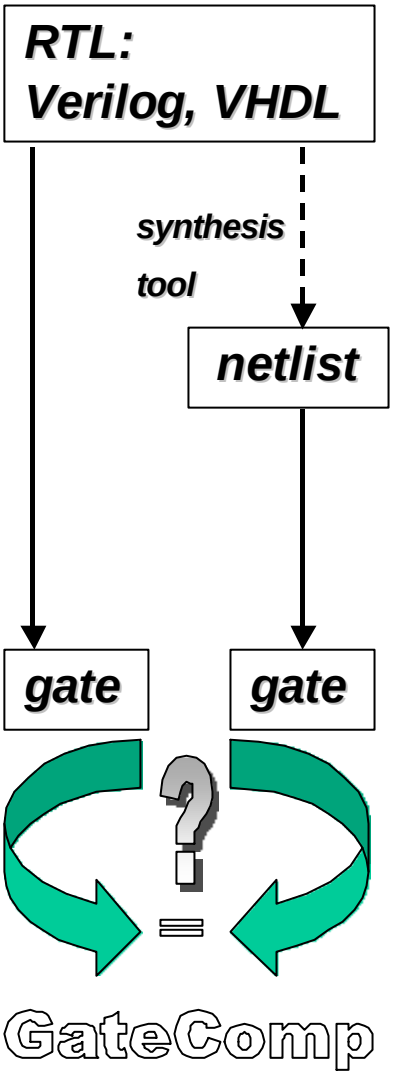

Figure 2: Synthesis verification flow

Several powerful features of gatecomp support the user during these steps.

Advanced Equivalence Checking in CVE

The CVE toolset contains the advanced equivalence checker gatecomp. The algorithms used are incredibly efficient; a typical performance figure for gatecomp is 100 k gates per minute; many multi-million gate designs have been verified so far. Before describing the examples in more detail, we first review the main features of the tool and also show the differences due to more precise modeling compared to alternative implementations.

Gatecomp is used to compare netlist vs. netlist, RTL vs. netlist or RTL vs. RTL. While other tools only focus on *bug finding*, in addition gatecomp is targeted towards simulation verification, i.e. to check that what is simulated on the RTL is also simulated on the netlist (reference design).

Gatecomp is an advanced equivalence checker due to the following differentiators:

- Speed
  - An efficient hash based data structure allows to handle complete designs by very low memory consumption (see also [KK97,vEi97]).
  - Multi-engines with multi-threading guarantee beside the very fast execution also the robustness and quality (see also [PK00,GPB00]).
  - The denotational translation schemes on word level in language frontends support the use of RTL information for the equivalence proof (see also [Joh01]).
- Capacity
  - The intelligent control of multi-engines ensures a tight integration of the different proof engines.
  - The frontends make use of compositional translation.
- Language coverage
  - The longterm experience with various description formats, like VHDL (incl. VITAL), Verilog (incl. UDPs) and EDIF, results in robust frontends and very user friendly linting tools.
- Debugging
  - A Graphical User Interface (GUI) allows for easy handling of the gatecomp results (see Figure 3).
  - A link to ModelSim and Debussy for source code/netlist debugging is provided.
  - A fast reachability analysis for eliminating spurious sequences is integrated.
- Flexibility
  - The multi-engine concept can easily be extended.
  - Gatecomp supports full Boolean constraints.
  - A 3rd party transistor extraction tool can be accessed.
  - Automatical generation of controllability and observability don't cares.
- Rich set of features
  - Multipliers of large bit-length can be handled.
  - Matching techniques: name based, simulation and prover based, structural, user defined, change name file from Synopsys.
  - Automatic removal of redundant states.
  - Support for clock-gating, tri-states, black boxing, compression of counterexamples, assertions and constraints, scan path insertion, ...

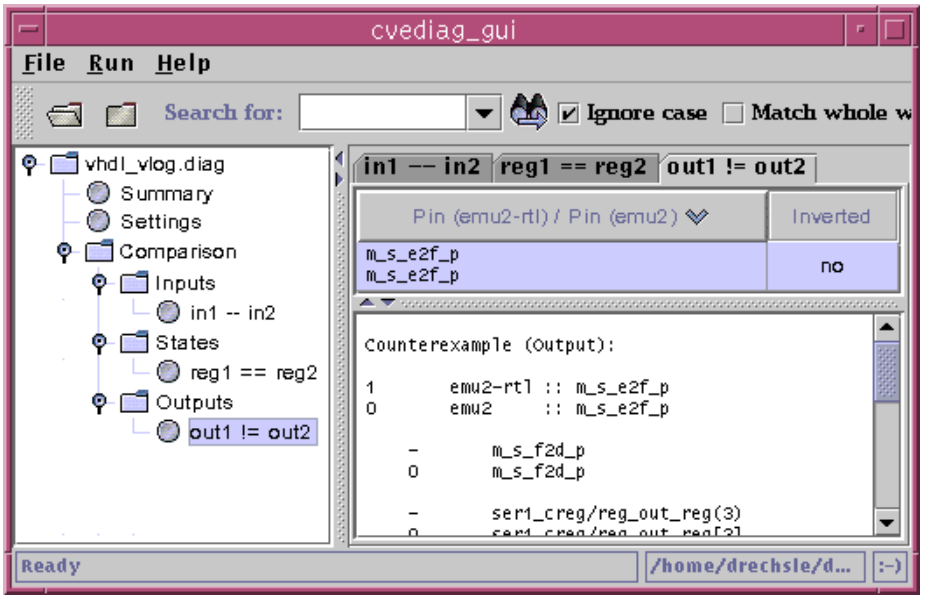

Figure 3: Graphical user interface

Precision - what you simulate on RTL is also simulated on the netlist

Gatecomp uses a different approach for the proof algorithms. Instead of a two- a four-valued logic is used, that allows to use synthesis and simulation semantics. While other tools reduce the simulation to two values only, gatecomp can model the precise language semantics, e.g. 9-valued in VHDL. The main features are:

- A formal library qualification tool in CVE guarantees conformance of synthesis and simulation view and issues warnings otherwise. The libraries are compiled into simple functional replacements. This - in addition to being very robust and reliable - results in very fast runtimes during the equivalence checking phase.
- The simulation/synthesis mismatches are proven/highlighted.
- Gatecomp is independent of the internal workings of the synthesis tools.
- The tool allows a formal handling of internal don't cares for RTL/RTL and RTL/netlist comparison.

Examples

To demonstrate the usage of the tool in equivalence checking of ASICs we report on three verification scenarios: a netlist vs. netlist comparison, a RTL vs. netlist comparison, and a RTL vs. RTL comparison.

Netlist vs. netlist comparison

First, we describe the verification of a synthesized netlist against its description after test logic insertion. The verified designs contained approximately 2.6 million gates. Details are given in Table 1.

The verification times were less than 20 CPU minutes on a four processor machine. Less than 0.5 GByte of main memory were used.

characteristics	synthesized netlist	final netlist
inputs	2843	2843
outputs	4178	4178
states	150218	150215
gates (million)	2.635	2.634
lines of code	222610	3861939

Table 1: Information on ASIC complexity

RTL vs. netlist comparison

This is the typical scenario for synthesis verification as shown in Figure 2. In our example the RTL description had more than 50.000 lines of code and the resulting netlist consists of over 2 million gates. Gatecomp took less than 23 CPU minutes and less than 420 MByte of main memory to prove functional correctness.

RTL vs. RTL comparison

A Verilog design was automatically translated into a VHDL description. After translation each module was checked by equivalence checking for functional correctness. For almost all blocks the verification was done in no time and fully automatic. In only a few cases – where the Verilog-VHDL-translation was erroneous – the tool took a few CPU minutes.

In case of a block with more than 600 outputs and over 1000 state variables the verification took 7 CPU minutes and less than 80 MByte were used. Based on the counterexample generated by gatecomp the design bug, that was due to a wrong assignment of don't care values, could easily be fixed.

In all three cases, this high performance is to be seen as a result of tight interaction between different tool components, i.e. the frontends, the proof engine and the debugging environment. The multi-engine concept used in gatecomp and its intelligent control guarantees high flexibility and robustness also on large designs with several million gates.

Comparison

Finally, we report about a comparison of various equivalence checking tools carried out by one of our customers. All experiments were carried out on a SUN Sparc 2 with 256 Mbyte running SunOS 5.7. An initial netlist is compared to a post layout netlist including routing that has been obtained by application of Magma *Blast Fusion™*, one of the leading physical design systems that also applies logic synthesis techniques. By this, the comparison often becomes more difficult. The initial netlist consists of 370 k gate equivalents. The netlist has more than 4500 outputs and more than 21000 states bits

The runtimes of gatecomp in comparison to three other commercially available tools[1] are given in Table 2. All tools are started with their default settings, i.e. no tuning of the parameters is done. As can be seen, significant reductions in runtime can be obtained.

[1] Names not given to guarantee anonymity.

characteristics	tool 1	tool 2	tool 3	gatecomp
runtime	>1 week[2]	~21h	~18h	~3.5h

Table 2: Information runtime

Conclusion

CVE is being developed at the design automation department of Infineon. The tools have been in productive use for years in the design centers of Siemens telecom and industrial automation divisions and in Infineon's semicustom design flow.

The examples described above show the application to real-world examples. Using the powerful tool gatecomp, equivalence checking of multi-million gate designs can be performed within minutes, and by this is superior to classical simulation - not only with respect to quality - but also regarding runtime. This has a direct impact on the costs of the verification process that can be reduced significantly based on formal techniques.
A comparison to other equivalence checking tools on a difficult example has shown a significant speed-up demonstrating the power of the tool.

References

[Pay01] *Payer, M.:* Industrial Experience with Formal Verification, it+ti, Oldenbourg Wissenschaftsverlag, Volume 43, Issue 1, pages 16-21, 2001

[KK97] *Kuehlmann, A. and Krohm, F.*: Equivalence checking using cuts and heaps, ACM/IEEE Design Automation Conference, pages 263-268, 1997

[vEi97] *van Eijk, C.*: Formal Methods for the verification of digital circuits, PhD thesis, Eindhoven University of Technology, 1997

[Kro99] *Kropf, T.*: Introduction to Formal Hardware Verification, Springer, 1999

[Dre00] *Drechsler, R.*: Formal Verification of Circuits, Kluwer Academic Publisher, 2000

[GPB00] *Goldberg, E., Prasad, M. and Brayton, R.*: Using SAT for combinational equivalence checking, International Workshop on Logic Synthesis, pages 185-191, 2000

[PK00] *Paruthi, V. and Kuehlmann, A.*: Equivalence checking combining a structural SAT-solver, BDDs and Simulation, International Conference on Computer Design, pages 459-464, 2000

[Joh01] P. Johannsen: BooStER: Speeding up RTL property checking of digital designs by word-level abstraction, CAV'01, 2001

[2] After a finetuning of the parameters for this tool a reduction of runtime to 4 hours has been achieved. But similar results can be expected for the other tools by variation of the parameters, e.g. gatecomp can do the comparison in less than 1 hour, if specialized parameters are chosen.