FormED: A Formal Environment for Debugging

Andre Suelflow Robert Wille Christian Genz Goerschwin Fey Rolf Drechsler

{suelflow,rwille,genz,fey,drechsle}@informatik.uni-bremen.de
Institute of Computer Science - University of Bremen – Germany

http://www.informatik.uni-bremen.de/agra/eng/

Abstract

FormED – a FORMal Environment for Debugging is proposed. FormED aids design debugging by computation of fault candidates, generation of high quality counterexamples and visualization of results.

1. Introduction

Whenever a complex design is created, it is likely that bugs are contained. Verification based on simulation or formal techniques is very effective in detecting the presence of bugs. But debugging, i.e. finding the source of a bug, mainly remains a time-consuming manual task. This decreases the productivity throughout the design cycle.

In the past, tool support for design analysis (e.g. [1]) and techniques that automatically determine candidate fault sites (e.g. [2]) have been proposed. Furthermore, tools to help understanding an error were introduced (e.g. [3]). But using random counterexamples for debugging cannot ensure that a fault candidate is sufficient to explain all erroneous behaviors. Moreover, the importance of having "useful" counterexamples has been observed [4].

We propose FormED - a *FORM*al *E*nvironment for *D*ebugging. FormED aids fault understanding by visualization and supports automation in debugging.
The main features are:

- Computation of candidate fault sites
- Generation of high quality counterexamples
- Visualization of results

In the following the core ideas are briefly described.

2. Computation of Candidate Fault Sites

Given an erroneous design, a set of counterexamples, and correct output responses, candidate fault sites are computed.

Cone extraction and path tracing provide basic debugging capabilities based on structural analysis [5]. All signals in the input cone of a faulty signal yield an initial set of fault candidates. Path tracing aids the formal diagnosis algorithm by considering structural properties of a circuit. Based on simulation, only fault sites responsible for the observed faulty behavior are returned.

SAT-based debugging [2,6] provides a formal method for debugging. In comparison to non-formal methods, SAT-based debugging *ensures* to determine fault candidates that can fix all counterexamples. A non-deterministic behavior of any computed fault candidate can fix the faulty design with respect to the counterexamples.

FormED highlights the results, i.e. the candidate fault sites, obtained by these approaches (see Figure 1). The cone view focuses debugging on relevant parts of the design. Path tracing (highlighted orange and green) computes an initial set of 15 fault sites. SAT-based debugging (highlighted green) reduces the fault candidates to three.

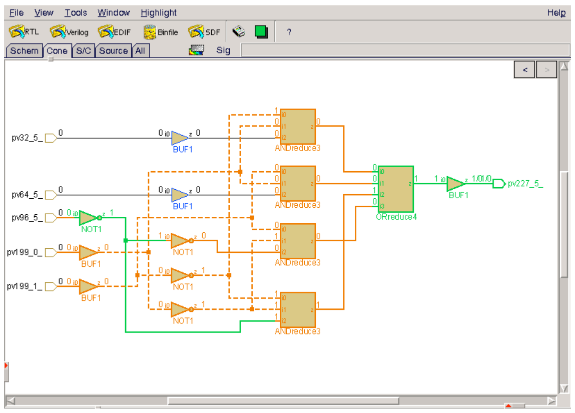

Figure 1: Computation of candidate fault sites

3. Generation of Counterexamples

A distinguishing feature of FormED is the generation of high quality counterexamples. Beside basic functionalities for equivalence and property checking, FormED integrates the approach of [7].

Equivalence and property checking for combinational and sequential models are supported. The equivalence of two designs is verified by building a miter circuit. Properties are specified in *Property Specification Language* (PSL) [8] and verified by *Bounded Model Checking* [9].

SAT-based debugging and counterexample generation are iterated to check the completeness of candidate fault sites, i.e. whether a fault candidate is sufficient to fix for any erroneous behavior [6]. If at least one of the candidate fault sites is incomplete an additional counterexample is determined, that activates still unfixable parts of the design. Each newly created counterexample strengthens the diagnosis and reduces the candidate fault sites.

In an alternative scenario, an initial set of counterexamples is checked for completeness [7]. Here, the user provides counterexamples, e.g. from simulation and a specification. Candidate fault sites are computed and validated to cover the full range of erroneous behaviors. Again, additional counterexamples are generated for uncovered scenarios.

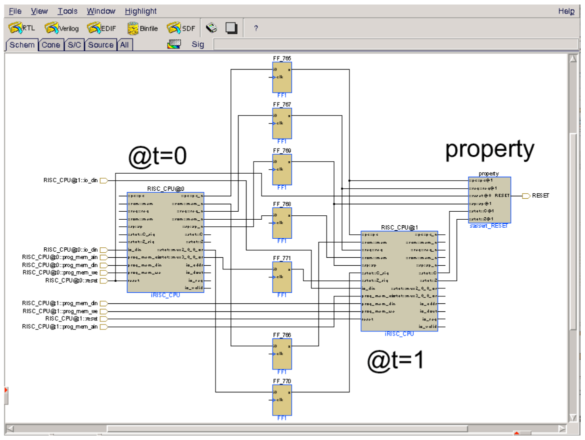
File View Tools Window Highlight Help
RTL Verilog EDIF Binfile SDF
Schem Cone S/C Source All Sig
@t=0
property
@t=1
RISC_CPU@0
RISC_CPU@1
RESET
Figure 2: Property checking for a RISC-CPU

Figure 2 illustrates the model for property checking in a schematic view. A property specifies expected behavior of a RISC-CPU for two time frames. For each of the time frames one copy of the RISC CPU is shown (big blocks on the left and the lower right). The property is shown by a separate block (upper right) that connects into the CPU.

4. Visualization of Results

The visualization engine of [10] is the front-end for debugging (see Figure 3). Some of the basic features are:

- navigation in a hierarchical schematic view
- cone extraction
- source code browsing
- cross-probing between source code and schematic view

The wide range of features provides a powerful environment for debugging. A designer can e.g. navigate in a schematic view and extract the input cone of erroneous outputs.

FormED interfaces with the visualization engine and extends the basic debugging capabilities by formal and non-formal methods. In an interactive session candidate fault sites and counterexamples are computed and annotated in the design.

5. Summary

In this work a debugging environment powered by formal methods has been proposed. The framework provides an environment to visualize erroneous behavior, to apply formal verification and gives hints for correction. Candidate fault sites are automatically determined and the time-consuming manual debugging process is partially automated.

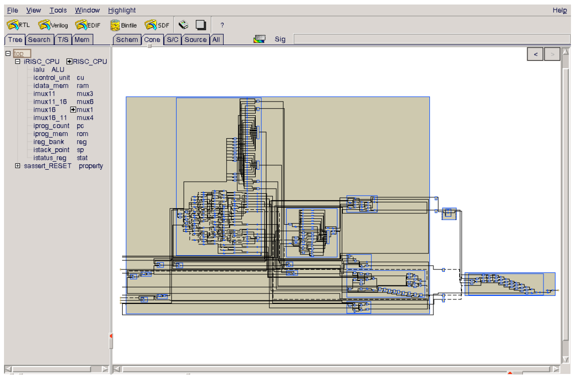

Figure 3: Cone extraction for a RISC-CPU

6. References

[1] SpringSoft Inc., "*Verdi Automated Debug*", http://www.springsoft.com, 2009.
[2] A. Smith, A. Veneris, M. Fahim Ali, and A. Viglas, "Fault diagnosis and logic debugging using Boolean satisfiability", *IEEE Trans. on CAD*, Vol. 24, No 10, pp. 1606 - 1621, 2005.
[3] A. Groce, D. Kroening, and F. Lerda, "Understanding counterexamples with explain", in *Computer Aided Verification* (CAV), Ser. LNCS, Vol. 3114, pp. 453 – 456, 2004.
[4] K. Ravi and F. Somenzi, "Minimal assignments for bounded model checking", in *Tools and Algorithms for the Construction and Analysis of Systems*, Ser. LNCS, Vol. 2988, pp. 31 – 45, 2004.
[5] M. Abramovici, P.R. Menon, and D.T. Miller. „Critical path tracing - an alternative to fault simulation", in *Design Automation Conf. (DAC)*, pages 214–220, 1983.
[6] G. Fey, S. Staber, R. Bloem, and R. Drechsler, "Automatic fault localization for property checking", in *IEEE Trans. on CAD*, Vol. 27, No. 6, pp. 1138 – 1149, 2008.
[7] A. Suelflow, G. Fey, C. Braunstein, U. Kuehne, and R. Drechsler, "Increasing the accuracy of SAT-based debugging", in *Design, Automation and Test in Europe (DATE)*, 2009.
[8] Accellera. "Property Specification Language version 1.1", http://www.accellera.org, 2004.
[9] M.D. Nguyen, M. Thalmaier, M. Wedler, J. Bormann, D. Stoffel, and W. Kunz. "Unbounded protocol compliance verification using interval property checking with invariants", in *IEEE Trans. on CAD*, Vol. 27, No. 11, pp. 2068 – 2082, 2008.
[10] Concept Engineering GmbH, "*RTLvision PRO*", http://www.concept.de, 2009.

Acknowledgement

We would like to thank Gerhard Angst and Lothar Linhard from Concept Engineering GmbH, Freiburg, Germany for their support.