Visualization of Diagnosis Results for Design Debugging

Görschwin Fey Rolf Drechsler

Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
{fey,drechsle}@informatik.uni-bremen.de

Abstract. Multi-million gate designs of circuits and systems can only be handled by means of computer aided design tools. Testbenches or formal verification are used to ensure correctness of these large designs. But most tools only return a single simulation trace to expose a failure. Then, usually simulation has to be used for manual debugging.
Here, a tool that automatically diagnoses an error and intuitively visualizes diagnosis results for equivalence checking is presented. The diagnosis engine facilitates a fast algorithm to calculate candidate error sites from counter-examples. The visualization engine was developed under the objective to handle large industrial designs.

1 Introduction

Design and verification of todays integrated circuits with several million gates heavily depend on *computer aided design* (CAD). Tools are employed for synthesis of the initial HDL-description, optimization of the first gate-level description and during verification. While synthesis and logic optimization benefit from very efficient fully automatic tools, ensuring correctness has become the major bottleneck in the design process. Already up to 80% of the design costs are caused by verification.

During verification either testbenches or formal methods [1] are used to check the behavior of a design. When using testbenches debugging is only aided by simulators. Also in the context of property checking recent techniques for debugging essentially rely on simulation to navigate through the erroneous design [2, 3]. The methods based on property checking additionally offer some hints, which stimuli are essential to produce an erroneous output. But the designer can only use the simulated values to find the error. No automatic identification of candidate error sites is done, i.e. the tools do not show areas in the design that are capable of error correction.

First attempts to automate debugging in equivalence checking are investigated in the area of *Design Error Detection and Correction* (DEDC) [4–7]. All of these methods calculate a set of candidate error sites. But no efficient tool that integrates diagnosis and visualization of the result was shown.

Here, we present a tool that tightly integrates diagnosis and visualization of the diagnosis result. Our tool is applied in the context of combinational equivalence checking. The diagnosis engine uses counter-examples to the equivalence of two designs for calculation of candidate error sites. The diagnosis result is then visualized. Displayed are the schematic, information regarding the hierarchy and the source code. In all these views candidate error sites are shown. Also cross-links between the different views are established. This enables an easy and intuitive navigation through the design and through the candidate error sites. By this a large speed-up of the debugging process is to be expected.

2 Diagnosis Engine

The in-equivalence of two designs is exposed by counter-examples. Multiple counter-examples are used by the diagnosis engine to calculate candidate error sites. Details are given in [8]. Assumed is a single design error of the types defined by the error model in [9]. A procedure called *path tracing* is facilitated to mark a set of candidate error sites for a single counter-example [10]. The intersection of the sets for all given counter-examples are the final candidate error sites.

This approach for diagnosis has several advantages over structure based or formal approaches. Only counter-examples are needed for diagnosis. By this, the specification may be given in a completely different format (e.g. HDL), may be incomplete and no structural similarities are necessary to retrieve candidate errors. Moreover the method is simulation based. This allows for diagnosis of large designs.

3 Visualization Engine

The visualization engine is capable of handling large designs. Different views of the design are offered and cross-links between these views are established. Moreover extra functionality can easily be integrated. These features and their relevance for debugging are described in more detail in this section.

Figure 1 shows a screenshot including schematic, source code and diagnosis result for the circuit 'alu4' from the LGSynth93 set. Candidate error sites are highlighted in all different views.

Cone View The cone view (middle window) shows the diagnosis result, i.e. all candidate error sites. This allows to get a fast overview of the candidates and to find areas that are erroneous with high probability. Moreover cone extraction can be used to explore the neighborhood of a gate: upon a mouse-click on a pin, neighboring gates are shown.

Schematic View The broader context of the candidates is determined by dragging one of the gates into the schematic (top window). Upon this the dragged gate is marked. To maintain an easy overview a large schematic is divided into pages. Navigation between the pages is done by clicking on connectors at nets crossing a page border.

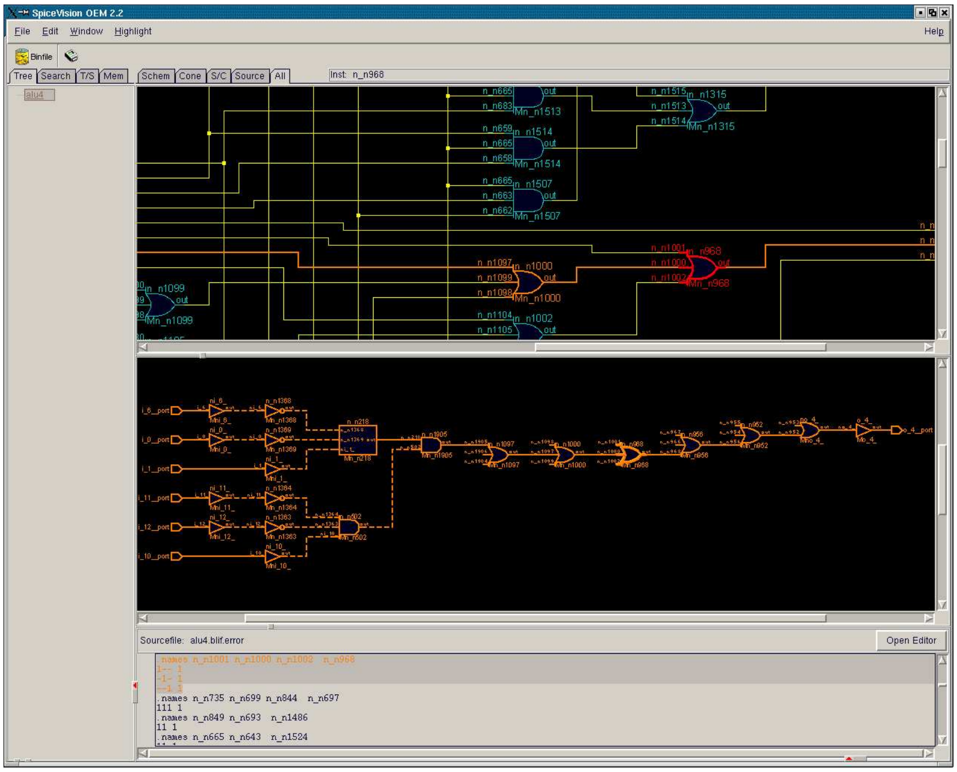

Fig. 1. Diagnosis of alu4: schematic, cone and source code view

Source Code View Also available is the cross-link to the source code by dragging a gate into the source code window (bottom window). This enables the designer to directly jump to the source for a candidate error site and to analyze the functionality in detail. The opposite - jumping from the source code into the schematic or cone - is possible in the same way.

Hierarchy Tree View Because 'alu4' is given as a flat design, there is only the topmodule 'alu4' visible in the hierarchy tree (left window). Basic gates are not shown. For more complex designs the hierarchy information is most important. Navigation through the hierarchy is possible using the hierarchy tree or directly within schematic and cone view. Expansion and reduction of submodules are triggered by intuitive mouse-interactions.

Integrating Extra Functionality Integration of extra functionality is easily possible by using an API. External functions can be specified using Tcl/Tk. Moreover the different visualization modules are available as widgets for Tcl/Tk, Microsoft Foundation Classes, Qt or Perl/Tk. By this, a custom tool can easily be developed.

The Tcl/Tk-API was used to integrate diagnosis and visualization. The internal database is loaded using commands of this API. Properties like names, colors or values can be annotated to each of the components in the circuit. Also the hierarchy information and links to the source code are established this way.

Experiments were carried out to prove the efficiency of the approach. Errors were randomly injected into circuits of the LGSynth93 benchmark set. In most cases only a few candidate error sites were marked. For the larger circuits of the benchmark set the number of candidates was always below 3%.

4 Conclusions

A push button tool for debugging in hardware equivalence checking has been described. The use of an efficient visualization engine ensures robust handling even of large circuits. Different design views and hierarchy information together with cross-links between the views provide an easy navigation. The tight integration with a fast diagnosis engine based on multiple counter-examples closes the gap between diagnosis and visualization. The intuitive way of identifying candidate error sites and locating them within the design can significantly speed up the debugging process.

It is focus of current work to provide a similarly powerful environment for sequential debugging as well.

References

1. Kropf, T.: Introduction to Formal Hardware Verification. Springer (1999)
2. Copty, F., Irron, A., Weissberg, O., Kropp, N., Kamhi, G.: Efficient debugging in a formal verification environment. Software Tools for Technology Transfer **4** (2003) 335–348
3. Hsu, Y.C., Tabbara, B., Chen, Y.A., Tsai, F.: Advanced techniques for RTL debugging. In: Design Automation Conf. (2003) 362–367
4. Kuehlmann, A., Cheng, D.I., Srinivasan, A., LaPotin, D.P.: Error diagnosis for transistor-level verification. In: Design Automation Conf. (1994) 218–224
5. Lin, C.C., Chen, K.C., Chang, S.C., Marek-Sadowska, M., Cheng, K.T.: Logic synthesis for engineering change. In: Design Automation Conf. (1995) 647–651
6. Veneris, A., Hajj, I.N.: Design error diagnosis and correction via test vector simulation. IEEE Trans. on CAD **18** (1999) 1803–1816
7. Hoffmann, D.W., Kropf, T.: Efficient design error correction of digital circuits. In: Int'l Conf. on Comp. Design. (2000) 465–472
8. Fey, G., Drechsler, R.: Finding good counter-examples to aid design verification. In: MEMOCODE. (2003) 51–52
9. Abadir, M.S., Ferguson, J., Kirkland, T.: Logic verification via test generation. IEEE Trans. on CAD **7** (1988) 172–177
10. Venkataraman, S., Fuchs, W.K.: A deductive technique for diagnosis of bridging faults. In: Int'l Conf. on CAD. (1997) 562–567