Modell Evaluation Using Genetic Manipulation Techniques

Z. Stamenkovic[1]
University of Nis
Faculty of Electronic Engineering
Beogradska 14
18000 Nis - Yugoslavia
E-mail: szoran@elfak.ni.ac.yu
Tel: (+381)18529324
Fax: (+381)1846180

H.-Ch. Dahmen and U. Glaeser
The German National Research Center
for Information Technology (GMD)
System Design Technology Institute
Schloß Birlinghoven
D-53754 St. Augustin - Germany
E-mail: [dahmen, glaeser]@gmd.de
Tel: (+49)2241-14-2301, Fax: -2324

Abstract

Formal Verification is an important area in industry with getting more and more attention. Growing complexity of digital circuits and the use in safety critical systems are the reasons for the need of tools for checking the correctness of designs.

In this paper we present a new approach for model evaluation. With our approach we are able to increase the belief of a designer in the right functionality of a circuit without the long runtimes of classical model checking but with more reliability than testing a design via simulation with some input patterns.

To achieve this goal we use our genetic manipulation technique: a combination of classical genetic algorithms with a goal oriented mutation operator.

1. Introduction

For the first design step on the RTL-Level it is very important to ensure some properties of the design. Classical approaches based on simulation are incomplete and very sensitive to the simulation patterns, model checking is in most cases much to slow or needs to much memory.

We propose the use of our genetic manipulation technique, which is on one side based on simulation, but due to a structural analysis goal oriented to create and simulate just these patterns which are most likely to be a counter example for the property - if one exist.

Like classical genetic algorithms we use an initial population of patterns to start with, but for the mutation operator we use a structural analysis of the circuit to find the bit inside the pattern to change, which is most likely to reach the goal.

The overall flow of our method is like the following:

- Get the design on RTL- or gate-level
- Get the property on RTL- or gate-level
- Use the interface to create a gate-level representation of both
- Build one design from the original design and the property
- Use the genetic manipulation technique for finding counter examples

1) Z. Stamenkovic was during this work with GMD

2. The Interface to VHDL

For our genetic manipulation algorithm we need a representation at the gate level. This representation will be generated during a fast-path synthesis from the VHDL circuit description at the register-transfer level (RTL) to a description at the logic gate level. A flowchart of this fast-path synthesis is given in figure 1.

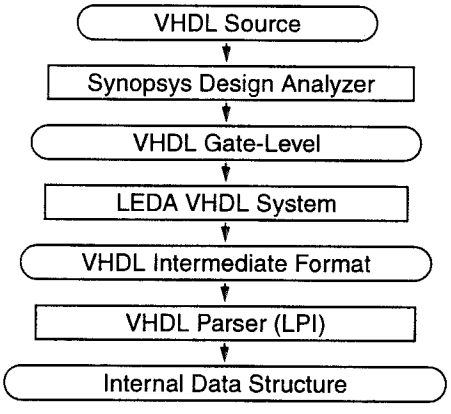

Figure 1: **VHDL-Interface flowchart**

We created a special target-library for the synthesis, consisting only of some standard gate types like AND, NAND, OR, NOR, INVERTER and REGISTER.

0-7695-0104-4/99 $10.00 © 1999 IEEE

3. Checking a Property

Once we translated the circuit into a gate-level representation we can apply the property to the circuit.

3.1. Connecting the property

This is done by connecting the relevant inputs and outputs of the circuit to the representation of the property.

For checking, if the property is correct within the circuit, we have to check, if the output of the property can be set to a certain value. This is performed by trying to generate a test-pattern for a stuck-at 1 or a stuck-at 0 test for the output of the property. In figure 2 we give an example of the connection between the circuit and the property.

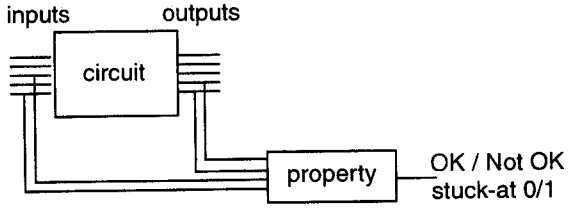

Figure 2: **The Evaluation Model**

3.2. Checking the property

For checking the property we use our genetic manipulation technique. In this section we give an overview of the algorithm, the details are described in [DaGl98].

We start with an initial population of input patterns for the circuit. Using a structural analysis we measure a distance between a pattern sequence and the goal of the algorithm, a 1 or a 0 at the output of the property. This function gives us the fitness of each pattern.

After selecting a number of the fittest input patterns we use an efficient multiple backtracing method [FuSh83] together with a probabilistic analysis in order to find a bit in the pattern sequence to be switched (mutated).

We use the genetic mutation operator for switching the selected bit in the pattern sequence or enlarge the pattern sequence if an initial state bit would have to be switched with highest probability.

The idea of switching a pattern sequence bit is illustrated in figure 3:

In this example it is quite simple to see which pattern bit should be modified in order to reach the goal (indicated by the arrow in the figure). In general, switching the right pattern bit is much higher in complexity because for every pattern bit there are usually several reasons for keeping the current value and there are usually several reasons for switching the current value.

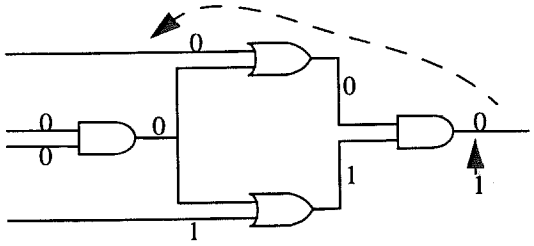

Figure 3: **Switching a bit in a test pattern**

4. Experimental Results

In our experiments we compared our software with the approach proposed in [HuCh96] gaining the following results:

		AQUILLA		
		detected	CPU AQUILLA/ gen_man	not detected
gen_man	detected	16	15000 / 87 sec.	2
	not detected	6	-----	2

As example circuits we used some of the ISCAS '89 benchmark circuits. Within the circuit we replaced randomly one gate in its functionality (for example And with Exor). In most of the examples both AQUILLA and gen_man were able to detect the different functionality, but the overall CPU-time of AQUILLA was roughly 150 times worse than gen_man. In 6 examples AQUILLA could identify the difference while gen_man was not able to detect. In 2 examples gen_man detected a difference while AQUILLA ran out of time (1 hour was the maximum limit for both software packages), and finally in two of the examples both software packages failed.

5. References

[CoRe96] F. Corno, P. Prinetto, M. Rebaudengo, M. Sonza Reorda, R. Mosca: *Advanced Techniques for GA-based sequential ATPGs*, Proc. IEEE European Design & Test Conf., 1996, pp. 75-79

[DaGl98] H.-Ch. Dahmen, U. Glaeser: *ATPG based on Genetic Manipulation Techniques*, Proc. IEEE North Atlantic Test Workshop '98, pp. 48-53

[FuSh83] H. Fujiwara, T. Shimono: *On the Acceleration of Test Generation Algorithms*, IEEE Trans. on Computers (C-32), pp. 1137-1144, 1983

[HuCh96] S.Y. Huang, K.T. Cheng, U. Gläser,: *An ATPG-based Framework for Verifying Sequential Equivalence.* Proc. of IEEE ITC 1996, Pittsburgh, pp. 145-157

[RuPa95] E.M. Rudnick, J.H. Patel: *Combining Deterministic and Genetic Approaches for Sequential Circuit Test Generation*, Proc. Design Automation Conf., 1995, pp. 183-188

[SaSa94] D.G. Saab, Y.G. Saab, J. Abraham: *Iterative [Simulation-Based + Deterministic Techniques] = Complete ATPG*, Proc. Int. Conf. on Computer Aided Design, 1994, pp. 40-43