tudapc — Bounded Model Checker Using Property Based Automated Abstractions

Ingo Schäfer

Dept. of Electrical and Computer Engineering
Darmstadt University of Technologie
schaefer@rs.tu-darmstadt.de

Abstract. Bounded Model Checking offers the possibility to proof the correctness of an implementation of a digital hardware system with respect to a formal specification. However, due to computational complexity, this technique is typically limited to the verification of different system blocks instead of whole digital hardware systems. To enhance the capabilities, abstraction techniques were developed, eg. [1]. This contribution introduces an experimental Bounded Model Checker called tudapc which applies automatic abstraction techniques.

Overview

Model Checking using tudapc is done in the following steps:

1. Generation of a Model to examine
2. Formulation of the property to check
3. Creation of the abstract model & proof
4. Examination of the result

The creation of the abstract model and the proof itself are fully automated, whereas the user is responsible for the creation of design and property. If the property is disproven, a diagnosis is presented which helps the user in debugging the model and / or the property. Figure 1 shows the connections between the different parts.

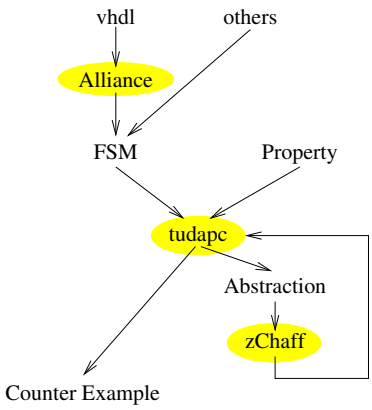

Figure 1: Overview

Models and Frontends

tudapc uses final state machines (FSMs) for representation of the model in question. These FSMs can be derived by synthesis of hardware description languages such as verilog or vhdl. Currently we are using [2] for this step, which enables the use of many common VHDL circuits.

Properties

In contrast to most common Bounded Model Checkers, tudapc does not check the reachability of an error state during an increasing number of time steps starting from the initial state. tudapc does check for a specific boolean relation between signals belonging to a limited number of consecutive steps. This check does not start with the initial state, in fact the initial state is no longer part of the model. An example of the property language used is diplayed in figure 2. The language is very intuitive and semantically similar to timing diagrams widely used in informal specifications.

```
if   state_i@t = idle and
     req_i@t = '1'
then grant_o@t+1='1';
```

Figure 2: Example Property

Abstracting Model Checker

The abstracting model checker tudapc itself is working without much user interaction. The input consists of a design and a property which are used to build an abstraction that is sufficient for the (dis-)proof of the property but still as small as possible. As an underlying proof technique, the satisfiability solver *zChaff* [3] is used. If computational resources are not exceeded, the model checker either reports the proof or the disproof of the property.

Output

If a property fails, a signal trace is produced that describes a situation in which the model violates the property. This trace can be viewed with any vcd-viewer and helps understanding the faulty situation and correcting the design and / or the property.

Experimental Results

tudapc can be used to evaluate different abstraction techniques. During tests, major improvements in memory consumption compared to a simple cone of influence reduction were observed using more advanced techniques. The cone of influence of the properties used during the experiments was between 20k and 1000k Gates.

References

[1] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu and H. Veith. Counterexample-guided Abstraction Refinement. In *Computer Aided Verification*, pages 154-169, 2000.

[2] Alliance VLSI CAD system, Université Pierre et Marie Curie, Paris.

[3] Z. Fu, Y. Mahajan, S. Malik. New Features of the SAT'04 Versions of zChaff. In *Contest Booklet to SAT'04 Competition*, Vancouver, 2004.