Improving Testbench Evaluation using Normalized Formal Properties

Martin Oberkönig, Martin Schickel, Hans Eveking
Technische Universität Darmstadt, Computer Systems Group
Merckstr. 25, 64283 Darmstadt, Germany
www.rs.tu-darmstadt.de
oberkoenig,schickel,eveking@rs.tu-darmstadt.de

Abstract

In this paper, we present a method to improve the testbench evaluation in a simulative verification process using formal properties. Assuming that at least for some part(s) of the design under test (DUT) a set of formal properties exists, the properties are transformed into a normalized form. The transformed properties, called microproperties, allow an objective evaluation of the testbench of the full DUT. It is possible to obtain more detailed coverage results using microproperties than by means of the original properties. This paper also describes how to obtain a unified metric from formal and non-formal verification results. Several examples including an AMBA AHB bus system are used to show the presented technique's practical applications.

Keywords: coverage, assertion, assertion-based design, properties, simulation

1. INTRODUCTION

The most time-consuming part in the design process of digital systems is the verification as an essential contribution to the quality of the final product. The functional verification of systems is done by simulation in most cases. Although formal verification methods are exhaustive, they are often limited to single modules due to complexity issues. The verification of full systems is therefore usually done using simulative methods like, e.g., constrained random simulation.

Specifications written in a formal property language like PSL [Acc04b], SVA [Acc04a] or ITL [One08] which are typically used in a formal verification process have the important advantage that they cannot be interpreted ambiguously as it is the case with informal specifications in a natural language like English. Furthermore, they offer the possibility to analyse and process properties automatically or to generate monitors or executable prototypes.

In a typical design situation, testbenches are written for simulation purposes parallel to the extraction of formal properties from informal specifications. A coverage model is created defining the set of essential requirements. Even if these tasks are closely related, currently no methods exist that directly relate formal verification properties with testbenches to employ synergies. Fig. 1 shows a scenario where formal properties already exist for module 1 which has been formally verified.

It would be nice to benefit from the formal properties of module 1 when also used in the simulation of the whole DUT. Of course, checking the formal properties of module 1 by simulation is superfluous because it was proven that the properties always hold. However, the formal properties represent essential parts of the functional behaviour of module 1 and can therefore be used to check if the testbench activates all these formal properties. The simple application of standard techniques like assertion coverage as given by [And05] might lead to very optimistic results. We will demonstrate that the transformation of the original properties into a set of normalized properties (so-called *microproperties*) significantly improves the accuracy of the testbench evaluation by means of assertion coverage.

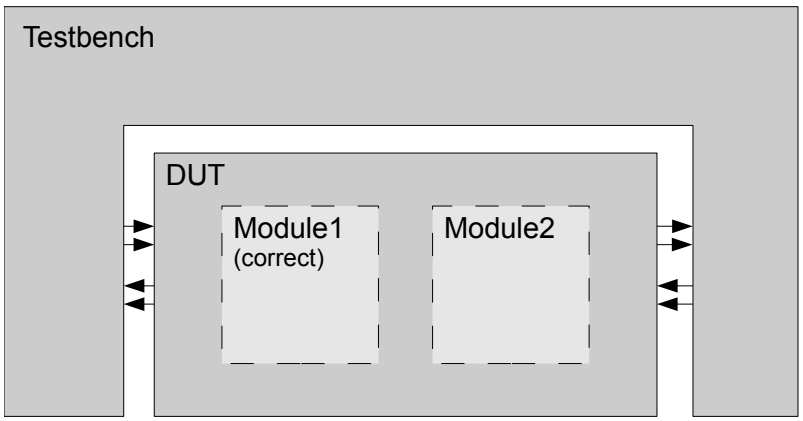

FIGURE 1: Simulation Environment for a Composed System

In the following, we present a method to integrate formal verification properties into existing simulation environments to benefit from the formal nature of the specifications. On the one hand, this method allows to quantitatively evaluate the quality of the testbench in relation to the formal specification. On the other hand, our method can discover functional gaps of the testbench more precisely. In addition, we present a technique to calculate a common coverage metric including both formal and non-formal coverage results.

The paper is structured as follows: Section 2 introduces the assertion-based design process followed by preliminaries on quantitative verification analyses (section 3). Afterwards, the normalization process towards microproperties is described in section 4. Section 5 contains the actual application of the microproperties, and section 6 presents methods to obtain a unified coverage metric. Experimental results are summarized in section 7. Sections 8 and 9 give an outline of related work and the conclusion.

2. ASSERTION-BASED DESIGN PROCESS

In software development, the use of assertions has been a common technique for many years. An assertion consists of a boolean expression which will evaluate to true or false. Assertions provide computer engineers with the means to explicitly formulate a design's expected behaviour at specific points in the source code. The assertions are evaluated during the run-time of the program. If an assertion fails a message is thrown.

Hardware description languages like VHDL also support assertions in the source code. These assertions are being evaluated during the simulation of the (VHDL-) model. Assertions have no impact on the synthesis later on. A hardware assertion describes the valid states of the system and holds if the boolean expression evaluates true. If the assertion fails, the simulator throws a message. In VHDL, messages on different levels can be generated. Thus, for example, only a warning is given or because of a critical error the simulation is stopped:

```
assert(valid behaviour) report "unexpected behaviour!" severity ERROR;
```

While assertions in programming languages are state predicates, i.e., refer to only one timepoint, temporal logics provide a rich repertoire of temporal operators to reference, for instance, distinct timepoints. Assertion languages like PSL, SVA or ITL include such temporal operators. Many simulation tools allow to bind PSL properties to a certain module and evaluate these assertions during simulation.

3. COVERAGE METRICS

A large variety of metrics to quantitatively analyse testbenches has been developed in the recent years that determine the verification progress in different ways. Structural coverage metrics are based on analyses that record which parts of the source code have been simulated:

Code/Statement Coverage measures the part of executed, i.e., simulated statements of the hardware description [CK93, CKV04].

Transition Coverage measures the percentage of transitions that were taken in a control automaton [HYHD95, HMA95].

Toggle Coverage is based on the subset of signals that have changed their value at least once during simulation [CKV04].

Branch Coverage states which branches have been taken during simulation if conditional branches exist in the control flow [CKV04].

Condition Coverage is more detailed than branch coverage. Condition coverage determines which subterms of an expression led to the evaluation to true of the whole expression [CKV04].

Mutation Coverage measures the number of mutations that have been inserted for test purposes and discovered during simulation. As a mutation a signal can, e.g., be inverted or statically pulled to '0' or '1'.

In contrast to the structural analysis procedures, the functional coverage metrics like assertion coverage quantify the compliance of certain criteria or properties independently of the implementation's source code. For example, the amount of satisfied PSL assertions can be measured. [And05]

The list above presents the most common approaches to evaluate the simulation progress.

In addition to these simulation metrics, there are also metrics for the quantitative analysis of a property-set's completeness in the field of formal verification. The metrics from [HKHZ99, CKV03] are defined relative to a given implementation. [OSE07] provides a quantitative measurement that considers only the property-set.

4. GENERATION OF MICROPROPERTIES

As a formal basis in this paper, we use safety properties which can be expressed as $G(P)$ where G is the Globally operator of LTL [Pnu77, Eme90] meaning that P holds on all executable paths. These properties are often written in an assume-guarantee form

$$P : Assumption \implies Commitment \tag{1}$$

or can be directly transferred into that form. The interval language ITL of the verification tool OneSpin 360 MV developed by OneSpin Solutions [One08] directly supports this form of property. The assumption A and the commitment C may contain input signals (I), internal state signals (S), and output signals (O) at different timepoints of a finite time window $tmin \ldots tmax$ where $tmin$ labels the earliest and $tmax$ the latest timepoint referenced in the property.

These formal properties can be transformed into a normal form using the technique described in [SNBE06]: As a first step all ITL specific syntactical constructs are eliminated: Macros and loops are unrolled, expressions are simplified, and variables are evaluated. Afterwards we temporally normalize the properties such that the latest timepoint in every property is $t+1$. This is the intuitive form of a causal specification: The past and the present imply the future. The properties now have the following form:

$$P : A(I_{tmin}, \ldots I_{tmax-1}, S_{tmin}, \ldots S_{tmax-1}, O_{tmin}, \ldots O_{tmax-1}) \implies C(I_{tmax}, S_{tmax}, O_{tmax}) \tag{2}$$

These bit-level properties pass through another normalization process as described in [OSE07] developed for the completeness analysis of property-sets. The properties are transformed such that the commitment of a property only contains a single state or output signal by the following steps:

In a first step, we transform the commitment into a conjunctive normal form and for each clause we generate one new property. For example,

$$A \implies clause_1 \wedge clause_2$$

will be transformed into

$$A \implies clause_1 \qquad \text{and} \qquad A \implies clause_2$$

At this point of the normalization process, each commitment consists of a single clause. This clause is always a disjunction of single literals. In a further step, we split these disjunctions according to the following example:

$$A \implies S_1 \vee S_2$$

will be transformed into

$$A \wedge \neg S_2 \implies S_1 \qquad \text{and} \qquad A \wedge \neg S_1 \implies S_2$$

This transformation step might generate properties which have an input signal as a commitment. Such properties will be deleted because it is not possible to force the value of an input signal. Now, all commitments of the properties reference only one single state or alternatively one output signal at the latest timepoint $tmax$.

$$P : A(I_{tmin}, ...I_{tmax}, S_{tmin}, ...S_{tmax}, O_{tmin}, ...O_{tmax}) \implies C(S_{tmax}/O_{tmax}) \tag{3}$$

We split up the properties so that the assumptions only consist of a single product term. The assumption will be transformed into a disjunctive normal form and will be separated into single product terms. For example,

$$producttерm_1 \vee producttерm_2 \implies C$$

results in

$$producttерm_1 \implies C \qquad \text{and} \qquad producttерm_2 \implies C$$

A property in the form:

$$\mu P : producttерm \implies signal \tag{4}$$

is called a *microproperty*. In a microproperty, there are no disjunctions in the assumption and the commitment consists of a single signal. Thus, this property describes a state of the system in which a single state/output signal must have one defined value. Microproperties can be seen as a kind of atomic functionality. Using this normalization procedure, an arbitrary formal specification can be split up into all single aspects of its functionality.

As microproperties are properties at the bit-level, we can easily generate boolean expressions that can be integrated as a condition into the assertions. To do so, the implication of the microproperty for a VHDL assertion will be equivalently transformed into

$$\mu P : \neg assumption \vee commitment \tag{5}$$

The resulting VHDL assertion is of the form:

assert(μP) report "Microproperty violated!" severity ERROR;

In case of a PSL assertion, the implication can be preserved syntactically:

assert always{$assumption$ -> $commitment$};

The generation process for PSL/VHDL-assertions from ITL-properties is shown in fig. 2. Arbitrary ITL-properties can be transformed into microproperties and PSL- or VHDL-assertions can be obtained.

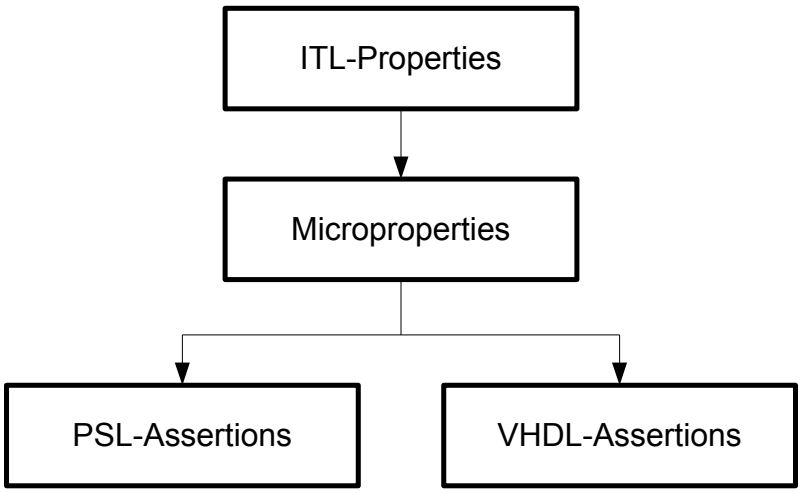

FIGURE 2: Workflow of the Assertion Generation

In contrast to PSL, VHDL does not naturally support the possibility to access past signal values in order to implement the possibly complex temporal relations between the signals within the boolean expressions. If signal values were accessed in ITL or PSL, e.g., with $prev(signal)$, these values must be made accessible by means of shift registers.

As this method can only provide signal values from the past, all (relative) timepoints of the referenced signals must be shifted back so that the latest referenced timepoint is t, denoting the present time.

The VHDL assertions generated according to the described method can be integrated in a concrete VHDL implementation and may, e.g., be simulated using a testbench. The PSL assertions can be bound to an implementation and also be simulated within a testbench.

If the original properties or the microproperties have already been formally verified on the implementation, no assertion will fail during simulation.

5. GUARDED ASSERTIONS

The integration of assertions into a simulation environment is well-known, and helps to discover errors in an implementation. If all assertions of a module hold because, for instance, all discovered errors have been corrected or because the assertions were formally verified, further simulation of the assertions is irrelevant for the correctness of the module.

In order to employ a module's assertions to assess a system's testbench, the assertions may be viewed as representations of the essential functional behaviour of the module. Using that approach, the verification engineer can determine which part of the module's functionality as defined by the assertions is activated by the testbench.

If a microproperty in the form of eq. 5 holds, then either the assumption is not satisfied or the commitment must be satisfied. If a testbench does not drive the system into one of the

```
architecture rtl_modified of implementation is
...
begin
        process(clk)
        begin
                if(A1) then assert(C1) end if;          --uP1
                if(A2) then assert(C2) end if;          --uP2
                ...
        end process;

        implementation: process()
        ...
end;
```

FIGURE 3: Guarded Assertions in VHDL

states denoted by a property's assumption, the assumption will never trigger; consequently, the microproperty is always satisfied. The testbench has never driven the system into an activating state. Hence, the actual commitment was never used to check the behaviour specified for the circuit by the microproperty. We say that a testbench *activates* a microproperty if the assumption part of the microproperty is true for at least one clock cycle.

Using assertion coverage, it is not possible to determine whether a microproperty is satisfied due to an assumption that never holds or whether the commitment was actually checked. For this purpose, we could use condition coverage (sect. 3). In general, this analysis allows to identify all the subterms that satisfy a boolean expression. If the assumptions and the commitments of the microproperties are rather complex, the identification of these simulation gaps is not trivial.

In order to avoid the effort of studying and interpreting all the condition coverage results, we propose to decompose a microproperty into the assumption part, called *guard* of the microproperty, and the commitment. A microproperty is activated by the testbench if its guard holds for at least one clock cycle.

In general, if there is a condition (a guard) preceding a VHDL assertion, the assertion will only be checked in the situation described by the guard. This gives us the possibility to preserve the character of the implication of microproperties even in VHDL. The assumption of the microproperty becomes the guard and the VHDL assertion contains the commitment. Fig. 3 shows the necessary VHDL statements and the embedding in the VHDL implementation in principal.

This technique enables the engineer to see, by just looking at a simple statement coverage report, which VHDL assertions were activated at all. For all untested assertions, the assumption of the microproperty, namely the preceding guard, has never been satisfied. This precisely identifies all the functional gaps in the testbench relative to a given set of microproperties.

In order to obtain similar results when using PSL, we propose to include `cover`-statements into the simulation in addition to the PSL assertions. The `cover`-statement in PSL provides a possibility to check if a specific situation has occurred during a simulation run. If the assumptions of the microproperties are used in the `cover`-statements, the functional coverage analysis of the `cover`-statements shows which of the microproperties were activated:

$$\texttt{cov\_uP: cover}\{A\};$$

6. QUANTITATIVE ASSERTION ANALYSIS

If VHDL assertions are used, we can obtain a quantitative measure from the statement coverage report which shows all simulated `assert`-statements. If PSL-`cover`-statements are used, we get the results from the functional coverage analysis, respectively.

The *degree of assertion coverage* can generally be defined as the quotient between activated assertions and assertions at all:

$$\text{degree of assertion coverage} = \frac{\text{\#activated assertions}}{\text{\#all assertions}} \tag{6}$$

The amount of activated assertions can be obtained either from the satisfied PSL-`cover`-statements or from the guarded and executed `assert`-statements in VHDL. The total number of assertions corresponds to the number of embedded microproperties.

In order to establish a relationship with the completeness of a formal property-set, we weight the degree of assertion coverage with the microproperties' degree of determination according to [OSE07]. The degree of determination quantitatively measures a property-set's completeness. This relationship allows to scale the 100 %-mark of the degree of assertion coverage to the determination (completeness) of the used formal properties:

Definition: The *formal coverage degree* as the weighted degree of assertion coverage is given by:

$$\text{formal coverage degree} = \frac{\text{\#activated assertions * degree of determination}}{\text{\#all assertions}} \tag{7}$$

This degree represents the amount of formal microproperties which have in fact been checked within the testbench. As it is natural to a simulation, checking of assertions is only the testing of specific situations - in contrast to a formal proof using property checking.

Our defined formal coverage degree enables the verification engineer to establish an objective and quantitative relationship between a testbench and a set of formal properties.

7. EXPERIMENTAL RESULTS

We extended the tool Candogen [SNBE07] so that it accepts formal properties in ITL and transforms the properties into VHDL or PSL assertions.

To test our approach, we used a FIFO memory interface taken from [Cla07]. The interface is shown in fig. 4. In addition to data input and output, control signals exist for reading from and writing into the memory. num represents the number of elements saved in the FIFO, and err signals an error.

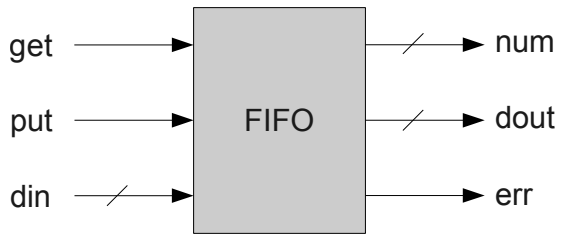

FIGURE 4: Interface of the FIFO

We manually implemented the behaviour specified by the formal properties using VHDL. The properties can be formally verified on the VHDL model. The property-set is complete according to the analysis method described in [OSE07] if the actual values saved in the memory are specified as a degree of freedom with an additional property. Thus, we restrict the specification to the control behaviour.

The properties were normalized and the microproperties were embedded as VHDL assertions into the VHDL implementation. Afterwards, we simulated this modified implementation together

TABLE 1: Experimental Results for FIFO (Overview)

Task	Time
Generation of the PSL/VHDL microproperties	0.6 s
Simulation without assertions	0.2 s
Simulation with VHDL microproperties	0.37 s
Simulation with PSL microproperties	1.4 s
Simulation with original properties in PSL	0.47 s

with a testbench using ModelSim from Mentor Graphics. Fig. 5 shows the simulation trace using a specific testbench. It tries to write more values into the FIFO than possible, and then it tries to read out more values from the memory than available. Both errors are indicated by err.

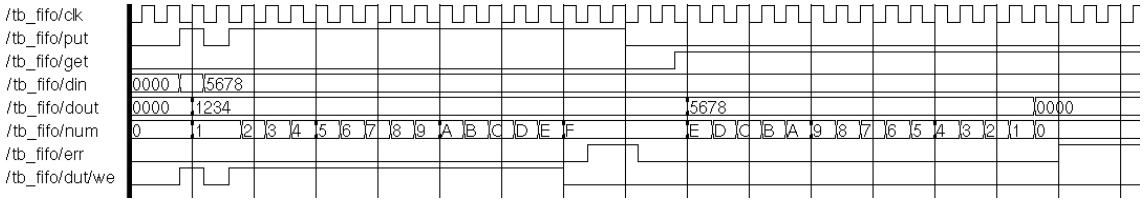

FIGURE 5: Simulation Trace of the FIFO

438 normalized microproperties were generated from 11 original formal ITL properties. But only 290 assertions were in fact checked during simulation, and they were of course satisfied. The assumptions of 148 microproperties were not triggered by this testbench, and so the assertions were not activated. According to eq. 7, this yields to a formal coverage degree of 66.3 %, weighted with a degree of determination of 100 %.

If the non-normalized original properties were used as assertions and were simulated instead of the normalized microproperties, all assumptions were fulfilled at least once. Thus, the formal coverage degree would be 100 %. This shows that microproperties allow a more detailed analysis of the testbench stimuli.

In order to check every microproperty during simulation for this implementation, the testbench must be adjusted manually so that all assumptions are satisfied at least once. In our case, the unsatisfied assumptions mainly belonged to microproperties that specify the data transfer from `din` to `dout`. More varying input data to the FIFO is needed to satisfy all the assumptions and thus to increase the formal coverage degree.

For example, the following guard has never been satisfied which shows that a dataword has never been written with bit 15 equal to 1 into an empty FIFO:

```
(num="0000" AND din(15)='1' AND put='1')
```

Tab. 1 gives the timing results for the generation of the assertions and the different simulation runs.

As a second example, we examined an arbiter with an appropriate testbench. The arbiter controls the access of three devices to a common resource with fixed priority. The generation of the 104 microproperties took 7.5 s. They had a degree of determination according to [OSE07] of 60.4 %, and the testbench triggered 19 microproperties. This results in a formal coverage degree of 11.03 %.

The last example is an AMBA AHB system from OpenCores [Agl07]. The system consists of one master, an arbiter (with integrated decoder), and two slave devices. Fig. 6 shows the interconnection of the modules. The slaves have different address spaces, and in the testbench in use only transfers to the second slave occur. The first slave is not addressed by the master.

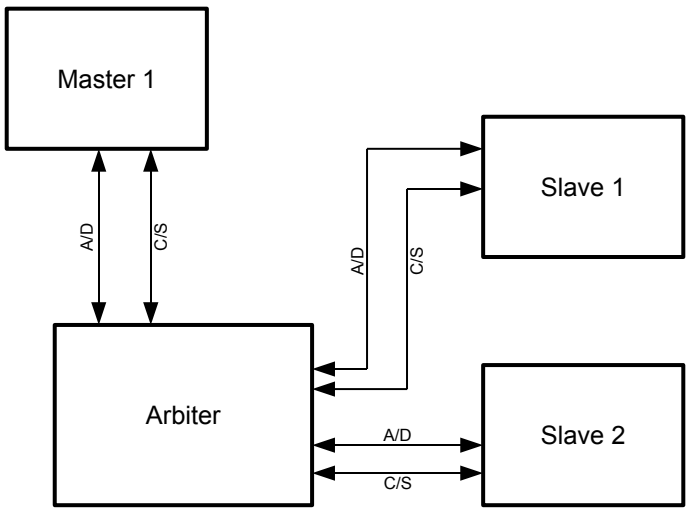

FIGURE 6: Block Diagram of the AMBA Test System

For the evaluation of the proposed methods in such a composed environment, we have written formal ITL-properties for the master component. After the normalization, the microproperties in PSL were bound to the master. During the simulation of the complete system with a testbench, several coverage metrics were calculated.

Fig. 7 shows the relation between traditional code coverage of the different modules and the assertion coverage with microproperties of the master component over (simulation) time. Even if all of the code coverage values are high, the assertion coverage with microproperties only increases to about 60 %. This shows that the usage of microproperties allows a more critical and detailed evaluation of a testbench.

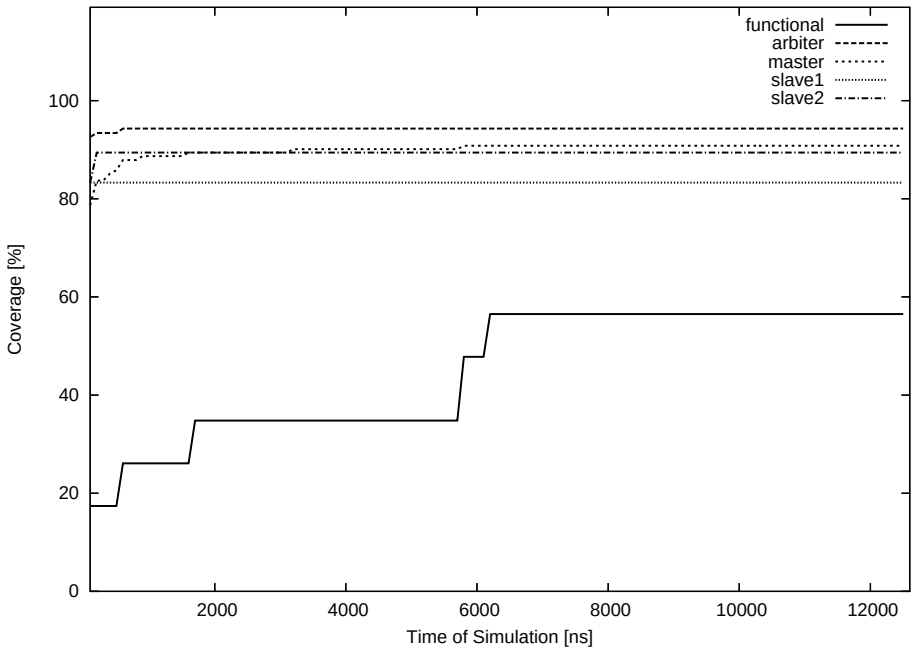

FIGURE 7: Functional Coverage using Microproperties vs. Code Coverage in an AMBA Test System

Fig. 8 relates different metrics like code coverage, branch coverage, condition coverage, and expression coverage with the assertion coverage using microproperties.

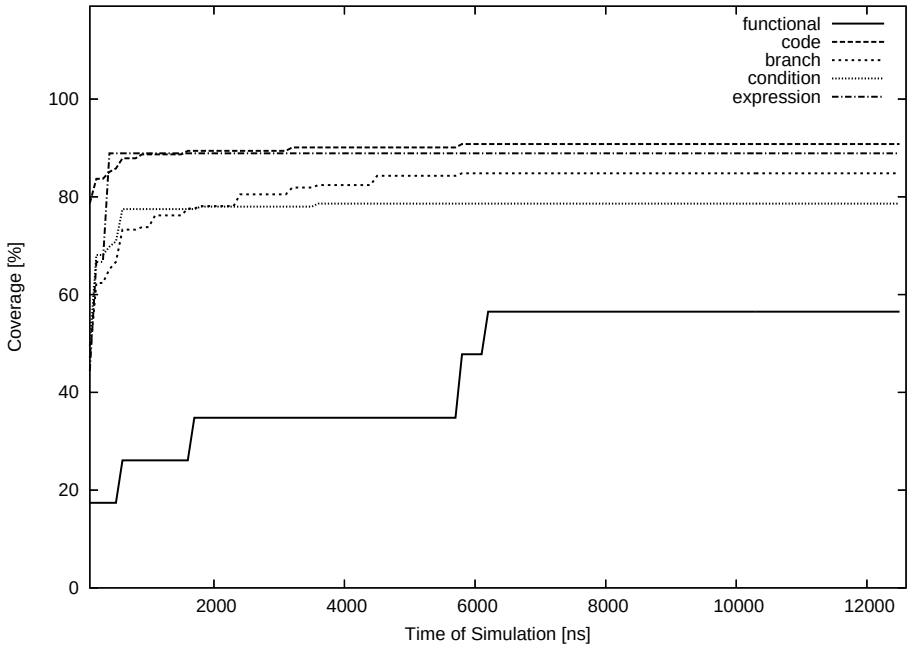

FIGURE 8: Functional Coverage using Microproperties vs. Traditional Coverage for an AMBA Master

8. RELATED WORK

The embedding of assertions into VHDL implementations is as well known as the embedding of assertions into software implementations. Some simulation environments for digital hardware provide the possibility to simulate PSL properties within a simulation run. This requires special tools as it has not yet been included in the VHDL standard. PSL shall be integrated in the next VHDL-2008/VHDL-200x standard. Our presented approach using VHDL assertions allows the use of all simulators supporting the current VHDL standard.

Publications which quantitatively evaluate testbenches starting from and referring to a complete property-set are not known to the authors.

The normalization of temporal properties was used, e.g., by Fisher [Fis91] in order to reason formally about LTL properties.

9. CONCLUSIONS

In this paper, we presented a method allowing to quantitatively relate formal properties of a module with a testbench simulating a concrete implementation of the whole system. We have shown that the use of microproperties leads to a more detailed analysis which parts of the system's functionality were covered by the testbench than using the original (non-normalized) set of formal properties. Our functional coverage metric is defined as the percentage of activated microproperties. It was shown that this metric leads to a more significant assessment of testbenches than other coverage metrics. The necessary transformation steps from arbitrarily written formal ITL properties to microproperties were described. Microproperties build the basis for our generation of VHDL assertions and PSL `cover`-statements, respectively. Common methods like assertion coverage can be used during simulation to calculate coverage metrics so that no modification of existing simulators is required.

If the degree of determination of the formal properties was calculated before, the defined formal coverage degree provides an objective view over the quality of the testbench. The presented method makes additional use of the verification properties which were written for formal verification. This formal specification allows to have a machine analysable coverage model.

In the near future we want to apply this method to verification processes in industry which used formal and non-formal methods separated from each other. We hope to bring simulation and property checking - the distinct areas of verification - together.

REFERENCES

[Acc04a] Accellera Organization, Napa, CA, USA. *SystemVerilog 3.1a Language Reference Manual, Accellera's Extensions to Verilog*, 2004.

[Acc04b] Accellera Organization Inc., Napa, CA, USA. *Property Specification Language - Reference Manual, Version 1.1*, June 2004.

[Agl07] Federico Aglietti. AHB System Generator. www.opencores.org, 2007.

[And05] Thomas L. Anderson. Coverage is the heart of verification. *EE Times*, 2005.

[CK93] Kwang Ting Cheng and A. S. Krishnakumar. Automatic Functional Test Generation Using The Extended Finite State Machine Model. In *DAC '93: Proceedings of the 30th ACM/IEEE Design Automation Conference*, pages 86–91, New York, NY, USA, 1993. ACM Press.

[CKV03] Hana Chockler, Orna Kupferman, and Moshe Y. Vardi. Coverage Metrics for Formal Verification. In *Correct Hardware Design and Verification Methods (CHARME)*, pages pp. 111–125, 2003.

[CKV04] Hana Chockler, Orna Kupferman, and Moshe Vardi. Coverage Methods for Formal Verification, 2004.

[Cla07] Koen Claessen. A Coverage Analysis for Safety Property Lists. In *Proc. of Conference on Formal Methods for Computer Aided Design (FMCAD)*. IEEE, November 2007.

[Eme90] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume B: Formal Models and Semantics*, pages 995–1072. Elsevier, Amsterdam, 1990.

[Fis91] Michael Fisher. A Resolution Method for Temporal Logic. In *In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI*, pages 99–104. Morgan Kaufman, 1991.

[HKHZ99] Yatin Vasant Hoskote, Timothy Kam, Pei-Hsin Ho, and Xudong Zhao. Coverage Estimation for Symbolic Model Checking. In *DAC*, pages 300–305, 1999.

[HMA95] Yatin V. Hoskote, Dinos Moundanos, and Jacob A. Abraham. Automatic Extraction of the Control Flow Machine and Application to Evaluating Coverage of Verification Vectors. In *ICCD '95: Proceedings of the 1995 International Conference on Computer Design*, pages 532–537, Washington, DC, USA, 1995. IEEE Computer Society.

[HYHD95] Richard C. Ho, C. Han Yang, Mark A. Horowitz, and David L. Dill. Architecture Validation for Processors. In *ISCA*, pages 404–413, 1995.

[One08] OneSpin Solutions GmbH. *User Documentation: OneSpin 360^{TM}- Version 5.0*. OneSpin Solutions GmbH, 2008.

[OSE07] Martin Oberkönig, Martin Schickel, and Hans Eveking. A Quantitative Completeness Analysis for Property-Sets. In *FMCAD '07: Proceedings of the Formal Methods in Computer Aided Design*, pages 158–161, Washington, DC, USA, 2007. IEEE Computer Society.

[Pnu77] Amir Pnueli. The Temporal Logic of Programs. *Symposium on Foundations of Computer Science*, 0:46–57, 1977.

[SNBE06] Martin Schickel, Volker Nimbler, Martin Braun, and Hans Eveking. On Consistency and Completeness of Property-Sets: Exploiting the Property-Based Design-Process. In *Proc. of FDL*, 2006.

[SNBE07] Martin Schickel, Volker Nimbler, Martin Braun, and Hans Eveking. CandoGen - A Property-Based Model Generator. In *University Booth, DATE'07*, 2007.