An Error Simulation Based Approach to Measure Error Coverage of Formal Properties

P.Azzoni
Dipartimento di Informatica
Strada le Grazie, 15
37134 Verona, Italy
azzoni@sci.univr.it

A.Fedeli
STMicroelectronics
Via C. Olivetti, 2
20041 Agrate Brianza, Italy
andrea.fedeli@st.com

F.Fummi
Dipartimento di Informatica
Strada le Grazie, 15
37134 Verona, Italy
fummi@sci.univr.it

ABSTRACT

Many approaches have been proposed for digital system verification, either based on simulation strategies or on formal verification techniques. Both of them show advantages and drawbacks and new mixed approaches have been presented in order to improve the verification process. Specifically, the adoption of formal methods still lacks a coverage metrics to let the verification engineer get a measure of which portion of the circuit is already covered by the written properties that far and which parts still need to be addressed. The present paper describes a new simulation based methodology aimed at measuring the error coverage achieved by temporal assertions proved by model checking. The approach has been applied to the description of a protocol converter block, and some preliminary results are presented in the paper.

1. INTRODUCTION

The verification of a digital system at the various steps of its design life is an activity that can be performed adopting a simulation approach [11, 10, 24] or, a less popular, although extremely appealing for its promises of exhaustiveness, formal method [22, 6, 17, 2, 20]. Both strategies have strengths and weaknesses and the inherent complexity of the verification activity of non trivial systems leads to the need to budget time, human and hardware resources involved in the verification activity to be engaged up to the moment in which a reasonable confidence in system implementation correctness is reached. Growing systems complexity, combined with the cited promise of exhaustiveness of formal verification tools caused a proliferation of formal tools in the last years, in the academic world (VIS [14], SMV [22], NuSMV [8], STeP [21], ACL2 [19], Mocha [5], UPPAAL [23], Spin [15], Forte [17], Versys2 [20], just to name some) and, to a shorter extent, in the industrial world (RuleBase [6], FormalCheck [3], BlackTie [1], Formality [2], Prover [25]) where most of products fall in one of two categories: equivalence checking and property checking. Solutions for the equivalence checking problem are now available on the EDA market in different flavors and can be seen as belonging to an almost mature field, whereas tools for the model checking problem are still under refinement (from a commercial point of view) as capacity keeps being a strong limitation even for latest tool generation (actually, this is mainly what decreases the exhaustive promises to date). Model checking is used to verify a circuit description according to requirements formalized as *temporal* properties and constraints, in order to check the correctness of the implementation of the circuit. Model checking approach, although performing exhaustive verification under the expressed environment restrictions presents the following limitations:

- Model Checking may require long validation time since it can be reduced, on paths of a bounded length, to the NP-complete problem of satisfiability of boolean formulas.
- Formal Verification may not provide *complete* assurance that the device will work as intended, due to the possibly incompleteness or inconsistency of specification expressed by the system architect. This is the main consequence of the adoption of a natural language as a mean to express requirements and specification.
- It requires that assumptions are made on the device environment, and if they are incomplete or wrong the validation could be vacuous.
- It cannot provide a measure to understand when all design errors have been covered. Actually, property coverage is still an open issue and a not sedimented subject.

On the other hand, simulation based functional verification techniques [11, 10, 24] show the following problems:

- By this technique it is not possible to distinguish between hard to detect faults, which represent hard to verify states in the functional interpretation, and the real design errors because in both cases the faults result undetected. Thus one runs the risk of identifying not existent errors, or even worst, of ignoring existent design errors.

[0]This work is partially supported by the Medea+ A502 MESA European Project.

GLSVLSI'02, April 18-19, 2002, New York, New York, USA.

- It is impractical to obtain a complete coverage for all possible faults, since the space of test sequences is infinite.

Some attempts have been made to combine formal verification and simulation. FoCs [4], for instance, derives a VHDL description to be attached to the design under test from a property written in RCTL. Functional verification by simulation is in that case driven by formal definitions of expected properties; Foster [13], whose work is focused on equivalence checking, proposes a methodology that exploits rigor of formal verification and high performance of simulation and ATPG. However, both techniques do not investigate the relation between model checking and error simulation, neither the possibility of using model checking as a test pattern generator to detect errors.

The present work combines model checking with functional verification based on simulation in order to:

- Measure the error coverage achieved by the set of proved formal properties, with respect to a design under test and one particular functional error model.
- Use this error coverage as an estimation of the completeness of the properties with respect to the design itself.
- Increase this error coverage by using a functional test pattern generator.
- Identify a new set of properties, with respect to the undetected errors, in order to guide the verification engineer in the process of defining a sufficient set of properties able to verify if the implementation satisfies the specification.

This paper focuses on the first two objectives and proposes a new approach to investigate the significance of a practical metrics for the model checking coverage measure problem. That approach has a more practical appealing than those, more theoretical, proposed by Chockler et al. [7] and Katz et al. [18]: [7] uses the underneath circuit transition relation and its composition with formulas and automata to define structure, tree and node coverage, whereas [18] describes an operational procedure which assumes the availability of complete specifications in a formalized format.

The paper is organized as follows. Section 2 explains the proposed methodology with the integration of model checking and error simulation. Section 3 describes how properties are written and witnesses and counterexamples are generated. Section 4 describes the role of error simulation. Section 5 presents a case study and the experimental results obtained using the methodology. Finally, in section 6 we draw our conclusions and possible future work directions.

2. METHODOLOGY

The goal of the proposed methodology is achieved by the actions represented in Figure 1 and summarized in the following phases.

Property checking A set of properties is written and they are formally verified by a property checking tool. Confuted rules point out erroneous behaviors that must be fixed since they represent a discrepancy between specification and implementation. In most cases, for every rule to be checked the property checker is able to generate at least one trace (a *witness*) that satisfies it, or at least one trace (a *counterexample*) that refutes it. Those traces are collected in order to be used in the simulation error step. Note that at the end of this phase all the collected traces are actually witnesses, since all the identified design errors are removed.

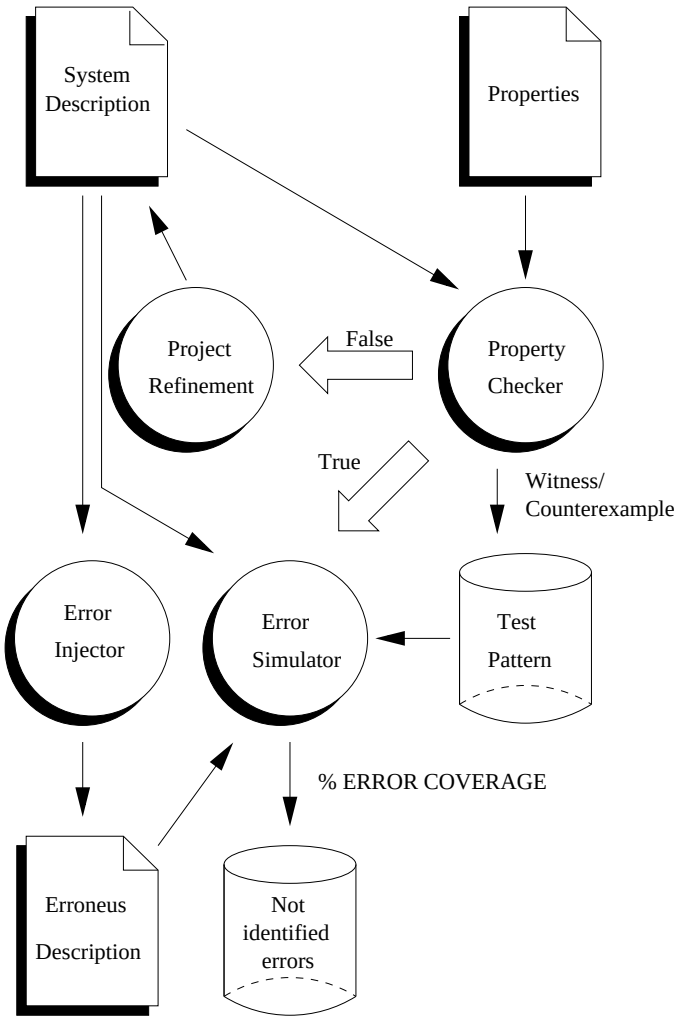

Figure 1: Control data flow of the methodology.

Functional error simulation All collected witnesses are used as input sequences for a functional error simulator. An automatic tool, AMLETO [12], is used to generate an erroneous description of the design under test based on the *bit coverage* error model [10]. This error model has been selected, since it has been proved in [9] that it is suited to reveal design errors. This simulation estimates the achieved error coverage. Some injected errors are usually untested by the witnesses. They can be:

- *untestable errors*, which are symptoms of some mistakes in the design, e.g.: over-specified functionalities or wrong modules interconnection.
- *undetected errors* because the number of applied

witnesses is too low to verify all aspects of the implementation. Thus, a larger number of properties should be written.

At this point, an automatic test pattern generator could be used in order to increase the error coverage by removing some untested errors.

Furthermore, a feedback connection of the ATPG phase and the model checker would allow to check either if already written rules can generate other witnesses to identify undetected injected errors, or if some new ad-hoc rules can be added to disambiguate between hard to verify injected errors and redundant errors; these interactions will be part of our future work and will not be developed subsequently in this paper.

3. PROPERTIES AND WITNESSES GENERATION

Let us examine the relationship between properties and functional errors. Different properties can be generated to verify the correct implementation of a specification. Such properties can be summarized as follows:

- **Safety**: to prove the correctness of the design under test checking that *the system must not evolve in a configuration which violates the specification.*
- **Liveness**: to prove the reactivity of the design under test by considering the fact that, after the right stimuli are applied, *sooner or later the expected result, should be obtained.*

```
rule no_retract_grant {

  envs default_req_eop_lck_opc;
  envs fair_request_granted;

  assign req_slow := 0;

  formula {
    AG((!RESET & GNT & !REQ) -> AX(GNT))
  }
}
```

Figure 2: Example of a property.

From the proposed verification methodology perspective, properties written using the "for all" path quantifier (A-formulas, A(p)) should be preferred with respect to properties written using the "existential" path quantifier (E-formulas, E(p)). This is both a direct consequence of the implication $A(p) \rightarrow E(p)$, and a consequence of the significance of counterexamples generated for A-formulas compared to those of E-formulas: counterexamples for E-formulas should show an infinite number of paths, which is obviously not feasible, and addressed restricting the counterexample shown to the shortest common prefix shared by all the counterexamples. In the error simulation phase, the proposed methodology simulates as many counterexamples and witnesses as possible in order to measure the error coverage obtained by property checking, thus the possibility of generating many witnesses is preferable. Note that, counterexamples became witnesses at the end of the property checking phase.

As reported in Table 1, the property checker supplies at least one witness when a property results true; what is actually relevant for the current work is that property checkers are able to generate (finite) sets of witnesses for the same property. In fact, as the approach we are proposing combines the meaningfulness of witnesses as test sequences for an error simulator, it is a preeminent matter to have a highly variegate set of witnesses, so that the biggest number of injected errors is detected.

As said at the end of section 2, if there were injected errors not detected by any witness produced so far, it should be understood if such condition derives from the errors being redundant or from the set of witnesses being incomplete; if the first case happened, we would have identified a design mistake, otherwise, we should find out if a witness for an already written rule is missing or not, and in this latter case we should write a new rule, aimed at trying to generate a new witness. This will be addressed by future development of our present work.

4. FUNCTIONAL ERROR SIMULATION

After the property checking step, all the collected witnesses and counterexamples must be used to simulate the DUT in order to obtain the error coverage with respect to the *bit coverage* error model. Bit coverage states:

- occurrences of variables and signals used in the description of the system under test are considered as a sequence of bits, thus every bit can be stuck-at 0 or stuck-at 1 to represent one particular error;
- conditions included in the description of the system under test can be stuck-at true or stuck-at false independently from their real value.

In order to simulate the DUT considering the bit coverage model, AMLETO can be used to obtain the erroneous description that is able to represent the behavior of the system in presence of stuck-at errors. The error simulation phase proceeds by comparing the output between the error free DUT and the erroneous DUT for every witness and for every injected error. At the end of the simulation, the error coverage achieved by the previously proved properties is computed.

It is worth to note that to avoid problems due to the reset sequence of asynchronous design under test, an extension to AMLETO has been made, since in its first release it has been developed for synchronous systems only. To obtain the erroneous description of the design under test, AMLETO substitutes every occurrence of every signal and every condition of control flow statements (as `if`, `case` and `while` statements) with the `inject_error` function. This function returns the faulty value of the occurrence when the relative error is activated. The activation of every error depends on the `error_port` signal added to the entity of the erroneous design. During the error simulation phase, when one error is detected, the erroneous description and the error free description must be reset to proceed with the simulation of the next error. In particular, it is necessary that all internal signals of the erroneous description do not keep the old value imposed by the previous injected error. If internal signals are not reset to their default values, the error simulation

	Existential path quantifier (E)	For All path quantifier (A)
True Property	At least one witness	At least one witness
False Property	The property checker generates only the common root to the inherently infinite family of counterxample traces	At least one counterexample

Table 1: Number of witnesses or counterexample generated by the property checker with respect to a property written by using the "existential" or the "for all" quantifier.

process can produce wrong results. The assignments of internal signals of synchronous systems is executed at every clock cycle, but this is not true for asynchronous systems where not all processes are sensitive to a clock signal. To avoid this problem AMLETO has been modified by inserting the `error_port` in the sensitivity list of all processes. In this way, when the value of the `error_port` changes, all processes are reactivated and all internal signals are reinitialized (see Table 2), allowing a correct error simulation.

5. CASE STUDY

The proposed methodology has been applied to test a VHDL description of a protocol converter composed of two modules: the *requestor* and the *response*. First of all, a set of properties have been written and proved by using a commercial SMV [22] (see Figure 2) based property checker. Some of these properties resulted false and thus some project refinements of the description was needed in order to satisfy all the properties. So far a certain number of witnesses have been collected.

In this way, after the property checking phase the error simulation has been applied as reported in Figure 3. A C based error simulator has been interfaced with the two VHDL descriptions (the error free and erroneous one) of the DUT by using a FLI (Foreign Language Interface) library.

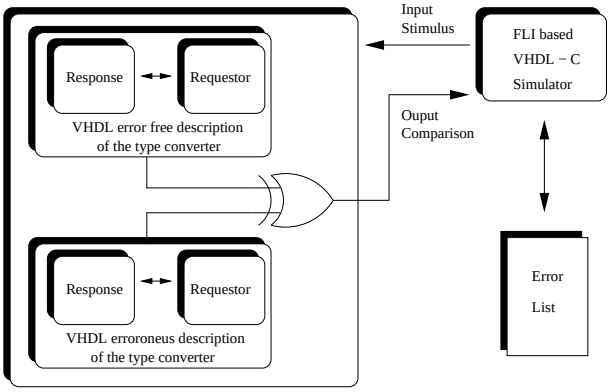

Figure 3: Architecture of the case study.

Some interesting observations have been established during the error simulation phase:

- The model checker can increase the length of the test sequences generated starting from one witness. That is, it can produce a witness W of length L and other witnesses, for the same rule, which include W, but are longer than L. This extra effort in the generation of further witnesses is useless since, such longer test sequences never increased, in our experiments, the error coverage.
- Witnesses generated by the model checker describe only the track of the input signals needed to prove the property. However, error simulation needs the values for all input signals, thus it is fundamental to fix the unspecified values to their default values (i.e., 0 for integer, 'U' for std_logic, '0' for bit, etc.). This implies the use of at least a 3-value error simulator.
- By considering all the checked properties, a total number of **1627** witnesses have been extracted. They are composed of **45550** test vectors. Their simulation detects **1454** errors on **3850** injected errors, thus achieving **37.7%** error coverage. This result has been obtained by fixing the unspecified values to their default values. On the other hand, the error coverage achieved by randomly completing the unspecified values is **56.7%**.
- On the contrary, the error coverage obtained by considering only one property is **25.2%** with default completion. This is due to the local scope of the witnesses generated by one single property. The error coverage raised from **25.2%** to **50.4%** by randomly completing the unspecified values.
- A so low property error coverage could be related either to the low number of properties checked or to many hard to detect injected errors. For this reason, we simulate the system with **1627** randomly generated sequences composed of **74** test vectors each. The error coverage raises to **73.6%**.

Such preliminary results imply the difference between: *property coverage* and *witness error coverage*. The proposed verification approach aims at measuring the *property coverage* by using the measure of the *witness error coverage*. Model checkers usually supply only one witness for every proved property, but most of the true properties could have an infinite number of witnesses. Thus, computing error coverage based on just one witness is not the right approach in order to measure the error coverage for some kinds of properties, particularly such properties that prove some rules about the data signals. In fact, in this way only the error coverage of the considered witness is obtained, which is a significant underestimation of the real *property coverage*. For instance, the analyzed example is a protocol converter which highlights this behavior since the majority of errors have been injected on data, which are simply transmitted from the *requestor* to the *response*. Such data requires a large number of test sequences to be tested, but very few properties to be verified.

Before transformation	After transformation
process (state_ctrl)	process (state_ctrl, error_port)
begin	begin
...	...
state_ctrl_next <= state_ctrl;	state_ctrl_nect <= inject_error(state_ctrl, error_port, ...);
...	...
end process;	end process;

Table 2: Processes transformation by using AMLETO.

For this reason, we are currently working on the extension of the proposed verification approach. The extension will take a witness, supplied by the model checker, as a seed for the automatic generation of other witnesses, where signals that are relative to the control part of the DUT remain unchanged while signals that represent data are assigned in a random or genetic way. In this way, the *witness error coverage* will better approximate the *property coverage*.

6. CONCLUDING REMARKS

In this paper we proposed a new simulation based approach for measuring the error coverage of formal properties and a case study has been presented to show the obtained experimental results. The work is a first step towards the integrations of formal methods and simulation based strategies in order to describe a new methodology for digital system verification as early reported in section 1.

7. ADDITIONAL AUTHORS

G.Pravadelli (Dipartimento di Informatica, Strada le Grazie 15, 37134 Verona Italy, email: `pravadelli@sci.univr.it`), U.Rossi (STMicroelectronics, Via C. Olivetti 2, 20041 Agrate Brianza Italy, email: `umberto.rossi@st.com`, F.Toto (STMicroelectronics, Via C. Olivetti 2, 20041 Agrate Brianza Italy, email: `franco.toto@st.com`).

8. REFERENCES

[1] Verplex black-tie functional checker. http://www.verplex.com/BlackTie_DataSheet_082701.PDF.

[2] Formality reference user's manual. Synopsys Inc., 1998–2001.

[3] Formal check user's manual. Cadence, 1999–2002.

[4] Y. Abarbanel, I. Beer, L. Gluhovsky, S. Keidar, and Y. Wolfsthal. Focs - automatic generation of simulation checkers from formal specifications. In *CAV2000: 12th International Conference on Computer Aided Verification, LNCS*, volume 1855, pages 538–542. Springer-Verlag, 2000.

[5] R. Alur, T. A. Henzinger, F. Mang, S. Qadeer, S. K. Rajamani, and S. Tasiran. Mocha: Modularity in model checking. In *CAV'98: 10th International Conference on Computer Aided Verification, LNCS*, volume 1427, pages 521–525. Springer-Verlag, 1998.

[6] I. Beer, S. Ben-David, C. Eisner, and A. Landver. Rulebase, an industry-oriented formal verification tool. In *ACM/IEEE Design Automation Conference*, pages 655–660, 3–7 July 1996.

[7] H. Chockler, O. Kupferman, R. P. Kurshan, and M. Y. Vardi. A practical approach to coverage in model checking. In *CAV2001: 13th International Conference on Computer Aided Verification, LNCS*, volume 2102, pages 66–77. Springer-Verlag, 2001.

[8] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. Nusmv: a new symbolic model verifyer. In *CAV'99: 11th International Conference on Computer Aided Verification, LNCS*, volume 1633, pages 495–499. Springer-Verlag, July 1999.

[9] F. Ferrandi, F. Fummi, L. Gerli, and D. Sciuto. Symbolic functional vector generation for vhdl specifications. In *Proc. IEEE Design Automation and Test in Europe Conference (DATE)*, pages 442–446, 9–12 March 1999.

[10] F. Ferrandi, F. Fummi, and D. Sciuto. Implicit test generation for behavioral vhdl models. In *Proc. IEEE International Test Conference*, pages 587–596, 20-22 October 1998.

[11] A. Fin and F. Fummi. A vhdl error simulator for functional test generation. In *Proc. ACM/IEEE Design Automation Conference*, 4–8 March 2000.

[12] A. Fin, F. Fummi, and G. Pravadelli. Amleto: A multi-language environment for functional test generation. In *Proc. IEEE International Test Conference*, 2001.

[13] H. Foster. Applied boolean equivalence verification and rtl static sign-off. In IEEE [16], pages 6–15.

[14] T. V. Group. Vis: A system for verification and synthesis. In *CAV'96: 8th International Conference on Computer Aided Verification, LNCS*, volume 1102, pages 428–432. Springer-Verlag, July 1996.

[15] G. J. Holzmann. The spin model checker. In *IEEE Trans. on Software Engineering*, volume 23, n. 5, pages 279–295. Springer-Verlag, 1997.

[16] IEEE, editor. *IEEE Design & Test of Computers*, volume 18, issue 4, July-August 2001.

[17] R. B. Jones, J. W. O'Leary, C.-J. H. Seger, M. D. Aagaard, and T. F.Melham. Practical formal verification in microprocessor design. In IEEE [16], pages 16–25.

[18] S. Katz, D. Geist, and O. Grumber. Have i written enough properties?*A method of comparison between specification and implementation.* In *10^{th} CHARME, LNCS*, volume 1703, pages 280–297. Springer-Verlag, 1999.

[19] M. Kaufmann, P. Manolios, and J. S. Moore. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, Kluwer Academic Publishers, 2000.

[20] N. Krishnamurthy, M. S. Abadir, and A. K. Martin. Design and development paradigm for industrial

formal verification. In IEEE [16], pages 26–35.

[21] Z. Manna and the STeP group. Step: The stanford temporal prover. In *TAPSOFT'99: Theory and Practice of Software Development, LNCS*, volume 915, pages 793–794. Springer-Verlag, May 1995.

[22] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell Massachusetts, 1993.

[23] P. Pettersson and K. G. Larsen. UPPAAL2k. *Bulletin of the European Association for Theoretical Computer Science*, 70:40–44, Feb. 2000.

[24] D. Saab, Y. Saab, and J. A. Abraham. Automatic test vector cultivation for sequential vlsi circuits using genetic algorithms. In *Proc. IEEE Transaction Computer Aided Design*, volume 15, issue 10, pages 1278–1285, October 1996.

[25] M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a sat-solver. In *FMCAD2000: Formal Method in Computer Aided Design, LNCS*, volume 1954, pages 108–125, 2000.