MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
KHARKOV NATIONAL UNIVERSITY OF RADIOELECTRONICS

ISBN 966-659-113-8

Proceedings of IEEE East-West Design & Test Workshop (EWDTW'05)

Odessa, Ukraine, September 15 – 19, 2005

CONTENTS

ASSERTIONS-BASED MECHANISM FOR THE FUNCTIONAL VERIFICATION OF THE DIGITAL DESIGNS

Vladimir Hahanov, Oleksandr Yegorov, Sergiy Zaychenko, Alexander Parfentiy, Maryna Kaminska, Anna Kiyaschenko
Kharkov National University of Radioelectronics,
61166, UKRAINE, Kharkiv, Lenin av. 14, tel. (+38)-057-7021326,
E-mail: hahanov@kture.kharkov.ua

Abstact. The verification model for the digital system-on-chip (SoC) designs is offered. This model is based on extending design with software-based code in the form of assertions, which allow to essentially decrease the design prototyping time. Theoretical statements and practical examples, which prove the efficiency of the assertions mechanism for solving tasks of the projects validation on the system level are listed.
Keywords: verification, validation, assertion engine, PSL.

1. Introduction

According to [1] the verification cost of the digital devices, designed on the base of ASIC, IP-core, SoC technologies, takes up to 70% of the overall design cost. Similarly, up to 80% of the project source code implements a testbench. Reducing these two mentioned parameters minimizes time-to-market, and this is one of the main problems for the world-leading companies in the area of Electronic Design Automation (EDA). The goal of the verification tasks is to eliminate all design errors as early as possible to meet the requirements of the specification. Passing the error through the subsequent design stages (from a block to a chip, and later to a system) each time increases the cost of it's elimination. Validation – a higher-level verification model – confirms the correctness of the project against the problems in the implementation of the major specified functionality.

The *goal of this paper* is to noticeably decrease the verification time by extending the design with software-based redundancy – the assertions mechanism [2-5], which allows to simply analyze the major specified constraints during the device simulation process and to diagnose the errors in case of their detection.

To achieve the declared goal it is necessary to solve the following *problems*:

1. To formalize the assertions-based product verification process model.
2. To develop the software components for synthesis and analysis of the assertions for the functionality, blocks and the entire system.
3. To get experimental confirmation of the benefits from using assertions to reduce time-to-market or, in other words, to noticeably reduce verification and overall design time.

2. Model of design verification process

The model of design verification process can be introduced as generalized equation $T \oplus F = L$, or, in more detailed form (with the components):

$$\{T_w, T_t\} \oplus \{F_s, F_v\} = \{L_f, L_p, L_s\}, \qquad (1)$$

where $\{T_w, T_t\}$ – represents operational (working) and test stimulus with the expected responses; $\{F_s, F_v\}$ – the main completely specified design model and additional model for verification/validation; $L = \{L_f, L_p, L_s\}$ – the lists of detected functional violations (conflicting conditions, defects), functional paths and states.

From the equation (1) we can define, that testing verification on the base of a classic testbench (HDL-description of test or operational stimulus [1,6,7]):

$$T_t \oplus F_s = L_{ts}, \qquad (2)$$

assumes building the checking sequences for the detection of all possible defects, or a complete testing of the design (block) behavior using the specified fault-free model. However, synthesis of a complete test for the considered defect class (NP-complete problem [1]) for each Unit Under Test (UUT) is one of the most expensive design stages, which involves large time and engineering expenses. If these tests are not supposed to be re-used on the level of system integration, the complexity of the tests generation might not become profitable. The functional verification is cheaper from the point of view of single-time additional overheads, as it uses only operational stimulus from the designer, which are not supposed to be built for the final UUT:

$$T_w \oplus \{F_s, F_v\} = L_{w\{s,v\}}. \qquad (3)$$

So, accordingly to (2) and (3), there are 2 possible ways to perform the diagnostic experiment upon the project models:

$$\begin{aligned} &1)\ T_t \oplus F_s = L_{ts}; \\ &2)\ T_w \oplus \{F_s, F_v\} = L_{w\{s,v\}}. \end{aligned} \qquad (4)$$

These experiments show noticeable difference between the preparation and evaluation time, as the generation time of the classic testbench T_t is much larger (up to hundred times) than the time to prepare the functional stimulus T_w. Besides, T_t is oriented on 100% defect coverage, which might occur during the design and manufacturing proc-

ess, while T_w is supposed to check only the operational modes, which are listed in the specification. So, we may assert, that $T_w \subseteq T_t$, but it means that the lowered defect coverage has to be compensated for the T_w. To do it, the number of the observed lines (the monitored model paths) could be increased. Such extension is actually provided by the assertions mechanism, which plays a part as a model extension, and is compensating the relatively low defect coverage of the input sequences from T_w. "Decreasing the test length by increasing number of the observed lines, preserving the checking sequences quality, should be used, when the size of the test is a critical parameter" [8].

Let's estimate the expenses on the preparation and evaluation of the model experiment according the (4). On the preparation stage engineering cost is the major component: $[Q(T_t) >> Q(T_w)] \& [Q(F_s) \approx Q(F_s, F_v)]$, assuming that $Q(F_s) >> Q(F_v)$. On the experiment stage computation cost is the largest portion: $Q(T_t \times F_s) >> Q(T_w \times \{F_s, F_v\}$. Hence we can define the efficiency of the assertions-based method relatively to the classic testbench-based verification as:

$$\varphi = \frac{Q(T_t) + Q(F_s) + k \times Q(T_t) \times Q(F_s)}{Q(T_w) + Q(F_s) + Q(F_v) + k \times Q(T_w) \times [Q(F_s) + Q(F_v)]}, \quad (5)$$

where k – is a reduction coefficient between the engineering and computation cost, which depends on the cost of the designer's time and workstations.

The biggest impact on the relation (5) is made by the $Q(T_t), Q(F_s), Q(T_w)$ components. So, the assertions-based verification efficiency depends on the time of development or generation of the testbench:

$$\varphi = \frac{Q(T_t) + Q(F_s)}{Q(T_w) + Q(F_s) + Q(F_v)} \approx \frac{Q(T_t)}{Q(T_w)} \approx 3 \div 10 . \quad (6)$$

Besides, if computation time for the verification takes relatively long time (tens of hours), reducing it in 3-5 times could make a serious impact on the overall time-to-market. Therefore the efficiency of the assertions-based analysis method should be also considered as a part of a model experiment, which according to (5) can be estimated as:

$$\varphi = \frac{Q(T_t) \times Q(F_s)}{Q(T_w) \times [Q(F_s) + Q(F_v)]}, \quad (7)$$

where $Q(T_t) \times Q(F_s)$ – is a simulation time of a classic testbench for the defects or fault-free behavior checking on the completely specified model; $Q(T_w) \times [Q(F_s) + Q(F_v)]$ – analysis time for the operational test upon the design extended with the assertions model.

Practice also shows an additional benefit of the assertions-based design verification mechanism, which, assuming (7), reduces the time-to-market [9]: «First 20% of the input stimulus usually detect up to 80% of the defects and check approximately 80% of the system functionality. The remaining 80% of the test sequences are supposed to check only 20% of the errors».

For a better understanding of the relations between the key terms, such as verification, validation, assertion, let's introduce the following definitions [10, 11].

Definition 1. Verification – is an analysis process of a system or it's components to ensure the correctness of the model transformations, while going through the current design stage.

Definition 2. Validation – is a process of checking operability of the system or it's components by testing it against the major requirements of the specification after each design stage.

Definition 3. Certification – is a written guarantee, that the system or it's components completely meet the requirements of the specification and are applicable for operational usage.

Definition 4. Assertion – is a system-level statement, defining the correctness of the transformations against the input description on the current design stage or specification requirements.

Following this definition, let's declare assertion as a predicate $L_i = f(L_{i1}, L_{i2}, ..., L_{ij}, ..., L_{in}) = Y = \{0,1\}$, which takes "true – 1" or "false – 0" values on the set of the input variables. According to (1), assertions could be classified on the 3 types $L = \{L_f, L_p, L_s\}$ – for checking faults, functional paths and possible UUT states. Such classification allows testing the computation mechanism and control logic within the software model. Usually this leads to usage of the transition graphs and involves checking the reachability of all paths and transitions [12-15].

Definition 5. Assertions mechanism – is a complete system of the assertion statements $L = (L_i, L_i, ..., L_i, ..., L_{ik})$ and their analysis tools, which are intended to verify and validate the design process, and to diagnose the operational violations.

The part of the structural design process model, when the validation and verification processes collaborate, is shown on the fig. 1. Its necessary here to notice, that the verification is applied to the component (stage) of the design process, while validation – to the entire system. However, these conceptions are very relative, as the system could be always considered as a component of a larger

system, and, backwards, component with necessary refinement becomes a system (structure).

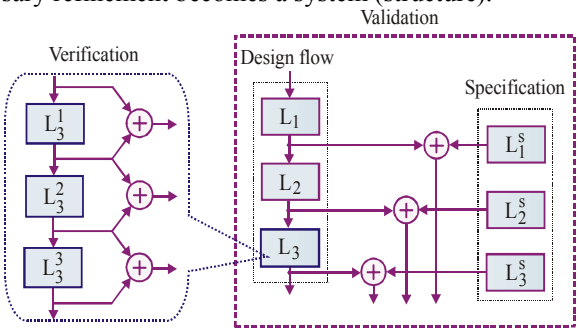

Fig.1. Hierarchical collaboration of the design validation and verification processes

Particularly, the verification and validation processes might have the same analysis points for building the assertions (fig. 2). Such regular structure is typical for the verification of a system, consisting of the ready-to-use blocks or IP-cores, when the assertions are built on the base of the component's input description and specification. In general case, the number of paths, being monitored by the assertions, should be minimized, but without loss of the verification quality. At the same time, the length of the test tends to grow.

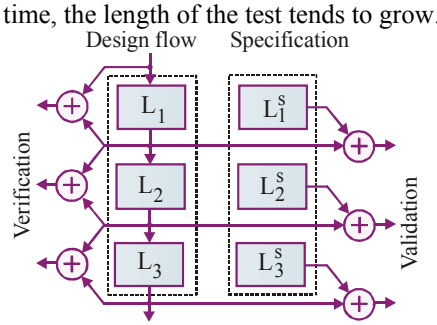

Fig. 2. Two-dimensional collaboration of the design validation and verification processes

Taking the given definitions and explanations into a count, the model of a design verification macro process consists of the following components: 1) Testbench, written on one of the system languages or HDLs; 2) completely specified model of the UUT within the simulation environment; 3) assertions model represented by a specialized description language, placed into the assertions analysis mechanism. Rough verification process structure is shown on fig. 3.

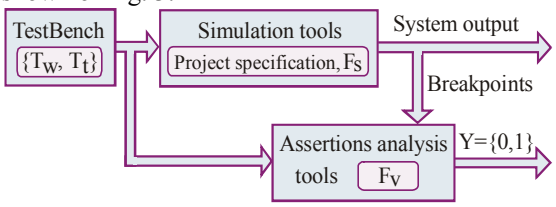

Fig. 3. Assertions-based design verification process

However, such verification process model is incomplete, as it does not consider the issues of the UUT diagnosis and operability recovery. Its necessary to say, that the diagnostic experiment (fig. 3) should be completed knowing the exact diagnosis (detected location and type of the defect) in case of the assertion violation (fig. 4). After determining the diagnosis, the operability recovery phase should be executed – manually or automatically by the testbench built-in mechanisms. Such verification model (fig. 4) influences the object of diagnostics (testbench, formal project specification) with a feedback. Here the assertions analysis module plays a part of a diagnosis tool working under control of experimental input signal vector $Z^d = (Z_1^d, Z_2^d, .., Z_j^d, ..., Z_k^d), Z_j^d = \{0,1\}$ .Ultimately, if the testability conditions are satisfied [16], the input vector can unambiguously identify the location and type of defect on the base of the previously prepared faults dictionary table [17].

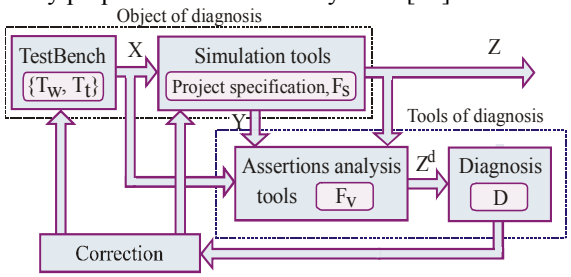

Fig. 4. Assertions-based verification and diagnosis model

Accordingly to the suggested structure (fig. 4), the refined assertions-based diagnosis model for UUT can be presented as:

$$M = \{T, F_S, F_V, Z^d, D\}, \quad (8)$$

where $Z = f(T, F_S)$; $Z^d = g(T, F_S, F_V)$; $D = h(T, F_S, F_V)$ represent operational outputs (project functionality), verification outputs (forming the constraints for checking correctness within the main specified model) and diagnosis outputs (localization and classification of the defect). At the same time, the generalized diagnosis process model can be defined as:

$$(T \oplus F_S \vee T \oplus F_V = 1) \rightarrow D = h(T, F_S, F_V), \quad (9)$$

when detection of the non-zero assertion signals activates the defects diagnosis and the design operability recovery.

3. Verification model implementation and practical results

Software implementation of the design verification and diagnosis process, based on the assertions analysis, consists of the architectural components, which are interacting as shown on the fig. 5.

Verification and diagnosis system contains the following components:

1) Assertions input component – a translation module for one of the industry standard assertions notations;

2) Internal language-level model for storing and handling the assertions constructions, being repre-

sented with the temporal relations and diagnosis algorithms (verification directives);
3) Internal assertions analysis model for checking the given formal temporal constraints within the simulation process;
4) Module for synthesis of the optimized model for the assertions analysis from the given language-level constructions;
5) Data structures and components, organizing the interaction and compatibility between the assertions and the external design model during compilation and simulation time.

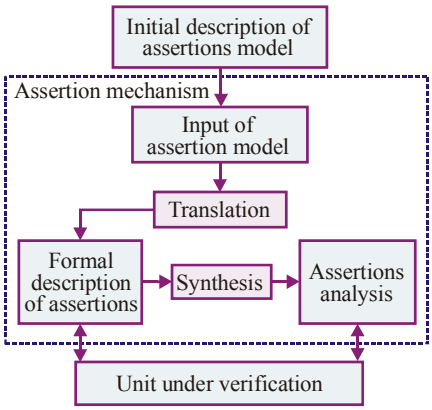

Fig. 5. Software implementation of the verification and diagnosis system

The strategy for building assertions input module totally depends on the usage of particular notation. If industry standard description languages, like PSL (Property Specification Language) [18,19], OVA (OpenVera Assertions) [20], System Verilog [19,21], are used for the description of the formal model, it is required to have a complete translation component, including preprocessor, lexical analyzer, parse and semantic handlers, which are performing syntax and semantic analysis of the input language constructions. Each of these languages has its own unique features for the binding to the main model of the UUT.

Computation complexity of the verification task and its impact on the overall design time are shown on the fig. 6. Determining the extremum point DV on the line 3 – the accumulation result of the lines 1 and 2 – is in the scope of designer's interest. For each particular case, depending on the testability, controllability and observability measurements, this point will have its own unique value. Finding this value is not only a scientific, but also practical goal for the designers. Verification line 1 determines the usability level of adding the assertions (software-based redundancy) into design, in other words, effect of the inserting additional monitoring points, which could noticeably decrease the overall time-to-market. Dependence 2 defines a well-known function of the design cost relatively to the system complexity.

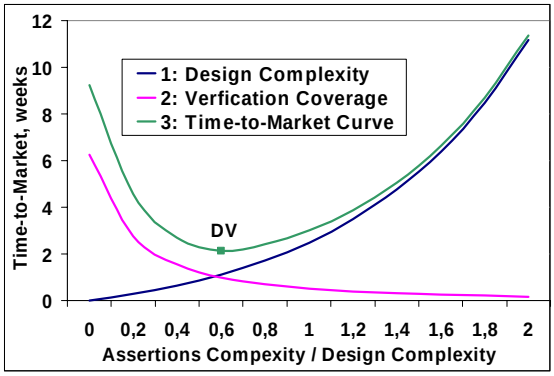

Fig. 6. Time-to-Market and design (+ assertions) computation complexity

4. Conclusions

The scientific novelty of the results, obtained in this paper, is in the definition of mathematical model of assertions-based design verification and diagnosis process, which allows to easily localize and classify the operational violations (defects) within the simulation process.

The practical benefit is in the noticeable (in several times) reduction of the complete design verification time, achieved by the extending the simulation system with an assertions analysis mechanism. Main point of the suggested method is to move away from building the classic testbench for checking fault-free behavior and defects – to the creation of the operational input stimulus for checking major functionality. The overhead of such simplification is in the insertion of additional monitored and analyzed points, which make the reduced functional test closer to 100% coverage.

In comparison with the traditional verification process [1], based on the complex testbenches and completely specified models, the suggested method has the following advantages:

1) Usage of the assertions strongly reduces the amount of human-written testbench code, which in 2-3 times decreases the capacity of the manual work for a skilled designer – the most expensive part of any project. Apart from that, the minimization of the code size with assertions model decreases the probability of making bugs in the testbenches.

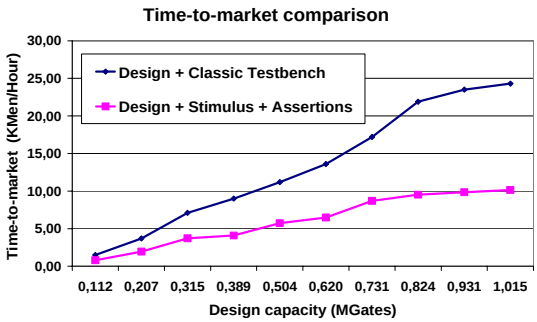

Fig. 7. Comparison of verification methods

2) Because the simulation and assertions analysis tools are interacting, the monitors are instantiated directly within the design model components, so the access to the values of interested internal signals is automatically provided. This allows to localize and classify the violations on-the-fly, and to simplify the detection of the bugs and regressions, while project goes through the stages of the top-down design process.
3) The assertion violation message directly specifies the hierarchical path within the design, where the error was found, and also the exact moment of the time, when the error has occurred. There is no need in any extra effort for the localization of the caught problem.
4) The moment of the assertion violation could be used by the verification environment as an additional breakpoint, or for launching user-defined error handling scripts. This feature improves the diagnosis depth and/or raises the verification procedures to higher level of abstraction.
5) Assuming the assertions model appropriately and completely represents the functional specification of the device, the statistics of the simulation process could be used to analyze the operational coverage of the testing stimulus.
6) Special verification libraries, like OVL (Open Verification Library) [23] were developed, which generalize typically necessary parameterized formal constraints (verification-IP), which could be re-used for verification of the wide-range of digital devices.
7) Unlike the simple debugging tools, which are oriented on the basic monitoring of internal state of design components, simulation with assertions activates error identification and handling scenarios only in case of their occurrence.
8) The assertions mechanism allows building self-testing models of designed components, which simplify detection of the operational violations, defects and regressions.

References:

1. *Bergeron Janick*. Writing testbenches: functional verification of HDL models. Boston: Kluwer Academic Publishers. 2001. 354 c.

2. *Annette Bunker, Ganesh Gopalkrishnan, Sally Mckee*. Formal Hardware Specification Languages for Protocol Compliance Verification // ACM Transactions on Design Automation of Electronic Systems. Vol. 9. No. 1. 2004. P. 1–32.

3. *C. Eisner, D. Fisman, J. Havlicek, Y.Lustig, McIsaac, and D. Van Campenhout.* Reasoning with temporal logic on truncated paths // 15th International Conference on Computer Aided Verification (CAV'03), LNCS 2725, Boulder, CO, USA, July 2003. Springer-Verlag. P. 27-40.

4. *S. Maisniemi, J. Kalinainen*. Assertion-Based Verification with PSL Integrated with an Existing RTL Verification Environment// PSL/SUGAR Consortium Meeting DATE 2004. P. 1-5.

5. *Components* of a complete assertions-based verification solution. Cadence. 2005.
www.cadence.com/products/functional_ver/abv_dt.aspx

6. *IEEE* Std 1076-2002, IEEE Standard VHDL Language Reference Manual. 124 p.

7. *IEEE* Std 1364-2001, IEEE Standard for Verilog Hardware Description Language. Electrical and Electronics Engineers, Inc.USA. 2003. P. 1-23.

8. *Hahanov V. I., Kaminskaya M. A., Egorov A. A., Pobezhenko A. A., Pobezhenko I. A.* "Povyshenie kachestva testa na osnove technologii Boundary Scan" (in Russian – "Improving test quality on the base of Boundary Scan technology") // Radioelectronics and informatics, № 3, 2004, Kharkiv, Ukraine, 2004, pp.85-90.

9. *Grinthal E.T.* Tutorial: Software Quality Assurance for CAD // 22nd Design Automation Conference. 1985. P. 555-561.

10. *IEEE* 610.12-1990 Standard Glossary of Software Engineering Terminology. IEEE Society Press. 1990. 84 p.

11. *The* IEEE Standard Dictionary of Electrical and Electronics Terms, Sixth Edition. // Institute of Electrical & Electronics Engineers. 1997. 1278 p.

12. *R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-haim, E. Singerman, A. Tiemeyer, M. Y. Vardi, and Y. Zbar*. The ForSpec Temporal Logic: A new temporal property-specification language. In Tools and Algorithms for Construction and Analysis of Systems. // Lecture Notes in Computer Science. Vol. 2280. Springer-Verlag. Berlin. Germany. 2002. P. 296–211.

13. *C. Eisner, D. Fisman, J. Havlicek, McIsaac, and D. Van Campenhout*. The definition of a temporal clock operator // 30th Int. Colloq. Aut. Long. Prog. (ICALP'03) July 2003. Springer-Verlag. P. 857-870.

14. *I. Beer, S. Ben-David, C. Eisner, and A. Landver*. RuleBase: An industry-oriented formal verification tool // 33rd Annual Conference on Design Automation. ACM Press. New York. 1996. P. 655–660.

15. *T. Arons, E. Elster, L. Fix, S. Mador-Haim, M. Mishaeli, J. Shalev, E. Singerman, A. Tiemeyer, M. Y. Vardi, L. D. Zuck*. Formal Verification of Backward Compatibility of Microcode // Computer Aided Verification. 2005. P. 185-198.

16. *Abramovici M., Breuer M.A. and Friedman A.D.* Digital systems testing and testable design. Computer Science Press. 1998. 652 p.

17. *Bondarenko M.F., Krivulia G.F., Riabtsev V.G., Fradkov S.A., Hahanov V.I.* "Proektirovanie i diagnostika kompjuternych sistem i setej" (in Russian – "Designing and diagnosis of the computer systems and networks"), Kiev, NMC VO, Ukraine, 2000, 306p.

18. *Property* Specification Language, Reference Manual//Accellera Organization, USA, June 2004. 131 p.

19. J. Havlicek, Y. Wolfsthal. PSL and SVA: two standard assertion languages addressing complementary engineering needs//Design and Verification Conference and Exhibition. 2005.P. 1-7.

20. *OpenVera*™ Language Reference Manual: Assertions, version 1.4, Synopsys Inc., April 2004, 136p.

21. *System* Verilog 3.1a Language Reference Manual, Accellera, 2004, 586p.

22. *Forczek M, Hrynkiewicz K*. "Formal properties evaluation", Proceedings of Programmable Devices and Systems Workshop 2003 (PDS-2003), pp.305-311.

23. *Open* Verification Library: Assertion Monitor Reference Manual, v. 3.10.14, Accellera, 2004, 228p.

Camera-ready was prepared in Kharkov National University of Radio Electronics
by Dr. Svetlana Chumachenko and Volodymyr Obrizan
Lenin ave, 14, KNURE, Kharkov, 61166, Ukraine

Approved for publication: 05.09.2005. Format 60×841/8.
Relative printer's sheets: 33,4. Order: without cash transfer. Circulation: 100 copies.
Published by SPD FL Stepanov V.V.
Ukraine, 61168, Kharkov, Ak. Pavlova st., 311