Review on Productivity Improvement of SoC Hardware Verification

LIU Qiang and MA Jian-guo

(School of Electronic Information Engineering, Tianjin University Naikai Tianjin 300072)

Abstract As system on chip (SoC) design complexity explodes, the gap between chip design and verification has been widened. How to improve verification productivity represents a great challenge to the IC industry. Recent efforts in addressing this challenge are reviewed and then a new verification paradigm, verification-while-designing, is proposed. This methodology combines hierarchical design and recursive verification, aiming at releasing the verification challenge. The fundamental points enabling the methodology are also outlined.

Key words formal verification; incremental verification; simulation; SoC hardware verification

提高SoC硬件系统验证效率方法的综述

刘 强，马建国

(天津大学电子信息工程学院 天津 南开区 300072)

【摘要】随着片上系统(SoC)设计复杂度的增加，芯片设计和验证之间的差距逐渐拉大。如何提高验证效率成了集成电路业面临的一个巨大挑战。该文回顾了近年来面向这一挑战的国内外研究工作，提出了一个新的验证范例——边设计边验证。该方法结合结构化设计和递归式验证，用于缓解验证的负担。同时，还提出了实现该方法的基本要素，用于指导未来的研究。

关 键 词 形式验证; 递归式验证; 仿真; SoC硬件验证

中图分类号 TP301.6 文献标志码 A doi:10.3969/j.issn.1001-0548.2013.02.001

Recent advances in semiconductor technologies and design methodologies allow more and more functions to be integrated in a single chip—system on chip (SoC), to meet the diversity requirements from various applications. While improving performance and system size, this integration has brought great challenges to verification, which ensures that the SoC hardware designs with millions of transistors work correctly and meet the design specifications, under the time-to-market constraint. The 2010 Wilson Research Group Functional Verification Study[1] reveals that in the past three years the number of verification engineers in the industry has increased up to 58% and the time of designers spent on verification is 50%. These clearly show the shift of the central issue in SoC designs from design to verification, and it is highly desirable to improve verification productivity.

A simplified standard SoC hardware design flow, as used in existing commercial design tools, is shown in Fig.1. This flow consists of four types of verification techniques deployed in multiple steps.

Simulation (Sim) is the widely used dynamic verification technique and usually is performed after RTL coding (RTL simulation) and after place&route (post-simulation) to ensure the compliance of the designs with the hardware specifications. It is easy to use in practice and can verify various metrics, such as functionality, timing, power, etc.. However, the effectiveness of simulation depends on the stimulus input. Enumerating all possible inputs is infeasible for

Received date： 2013 – 02 – 15

收稿日期： 2013 – 02 – 15

Foundation item: Supported by the National Natural Science Foundation of China under Grant(61204022)；the Natural Science Foundation of Tianjin China under Grant(12JCYBJC30700)

基金项目：国家自然科学基金(61204022)；天津市自然科学基金(12JCYBJC30700)

Biography：LIU Qiang was born in 1978, and his research interests include digital IC design and EDA .

作者简介：刘强(1978 –)，男，博士，副教授，主要从事数字集成电路和EDA方面的研究.

complex designs under the time constraint, while finding a reduced set of stimuli targeting corner-cases is difficult[2].

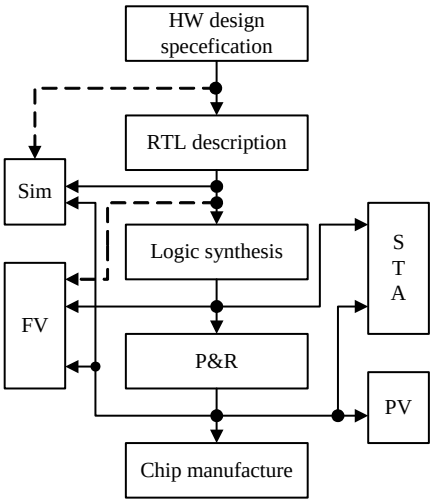

Fig.1 SoC hardware design flow

In contrast, static verification methods, formal verification(FV), and static timing analysis(STA), do not need external stimuli and in theory can achieve 100% bug coverage. Formal verification checks the logical equivalence of designs at the different abstract levels by equivalence checking[3-4], and checks property satisfiability of designs by model checking (or property checking)[5]. Both procedures involve converting designs to Boolean expressions and use binary decision diagram (BDD) and satisfiability (SAT) solvers to find counterexamples. Due to the increasing size and complexity of SoC designs, the storage and time required for the solver algorithms in the formal verification are becoming unaffordable[2]. Static timing analysis calculates the delay of all paths in a design and finds the timing violations. It is executed after synthesis and place&route to estimate the timing performance of designs and see if the user-specified timing goal is satisfied.

Physical verification (PV) is the last verification step before taping out. DRC (design rule check) and LVS (layout vs schematic) are performed in this step to improve manufacturability and check the layout netlist with the logic netlist.

In the design flow shown in Fig.1, various verifications are carried out seven times and even more in practice. Whenever a bug is found during the verifications, designers have to go back to the early design stages to locate and fix the bugs. It is the individual verification burden and the iterative design flow that makes the verification a timing- and labor-consuming task for complex hardware designs.

This paper reviews the recent efforts in simulation and formal verification techniques, which address the verification problems, and proposes a new verification paradigm for verification productivity improvement. The rest of this paper is organized as follows. In Section 1, the recent development in simulation stimulus generation is briefly described. Section 2 lists formal verification techniques tackling the complexity issue, and Section 3 shows the hybrid verification approaches. In Section 4, the new trend in verification, design for verification, is shown. Section 5 presents our idea of verification while designing, followed by the conclusion drawn in Section 6.

1 Automatic generation of stimulus

The simulation based verification needs a large amount of stimulus inputs to drive a design to run through all possible states, so that hidden errors can be revealed. Therefore, automatically generating sufficient and representative stimuli becomes a key to the high coverage simulation.

Constraint-based stimulus generation is a widely used and efficient approach, where constraints are the formal specifications of design properties. By solving the constraints, a set of valid stimuli meeting the specifications is generated. Traditionally, the constraints are represented as Boolean formulas which can then be solved by satisfiability (SAT) solvers[6], or the constraints form binary decision diagrams (BDDs) which are traversed based on branch probability to generate stimuli[7]. These two approaches exploit specific constraints or branch probability control to cover corner-cases.

Recent development of automatic stimulus generation are shown in [8-11]. In [8], a extended finite state machine (FSM) was used to specify the properties of interface protocols. The extended FSM considered not just the system states, but also the internal variables related to state transitions, in order to more accurately describe the behaviors of interfaces.

Stimulus vectors were then generated automatically from the extended FSM. A self-tuning approach was proposed in [9] to generate uniformly distributed random stimuli. In an iterative mode, distribution analysis results of generated stimuli were fed back to a SAT solver by adding new constraints, such that the stimuli generated in the next iteration were in the different regions of the search space. In [10], simulation-based verification was applied to the floating-point division design. Various operational modes for the floating-point division were modeled to cover specific cases and describe constraints for operands in each case. The developed tool FPgen solved the constraints and generated stimuli for floating-point division. In [12], multiple processor SoCs were verified at the system level using simulation with pseudorandom stimuli. Simulation traces, recording the information of instruction execution on processors and bus, were passed to a C++ model of the SoCs to generate expected results. Then the simulation results were compared with the expected results to check the correctness of the systems. In [11], transfer-resource graph was used to model the interaction between the components of a SoC and generate test cases for verification of resource contention.

The efficiency of simulation was discussed in [13-14]. In [13], functional test generation proceeded in three steps. First, given a graph representation of the target hardware design and possible fault models, properties were automatically generated, each corresponding to one possible fault in the design. Second, the generated properties were partitioned into disjoined sets, each containing properties with similarities in structure, texture, influence range and Boolean expression. Finally, each property from a property set was checked by a SAT solver one by one to generate counterexamples which were used as tests to the original designs. During SAT solving, the wrong search decisions taken for the previous properties were avoided for the followings to reduce the total search time for the set and accelerate test generation. In [14], an approach to filter redundant tests was proposed, such that the diversity of tests within the simulation time was increased. A machine learning technique, support vector analysis, was used to cluster tests. A new test falling in one of the existing test clusters was ignored, with respect to the argument that similar tests would cover the similar functionalities of a design.

2 Formal verification techniques targeting complexity

To deal with the complexity of large designs, various techniques have been developed, so that existing formal verification techniques can effectively work on them.

Model checking checks if a given property was satisfied by traversing all possible paths from the initial states to every reachable state in the state space of a design[15]. Given a property, abstraction techniques have been applied to reduce the state space that has to be searched in model checking, by removing parts of the design which do not affect the property. In [15], predicates, which are Boolean expressions were used to convert Verilog designs to symbolic Boolean formula. The formula were solved to generate an abstract model, where the number of state transitions expressed by means of the predicates was reduced. There were two problems in using model abstraction for formal verification. First, abstraction model generation could be time-consuming. In [15], predicates were partitioned into subsets for generating multiple small abstract models and then these models were together to approximate the original design space. Second, the abstraction based model checking might generate counterexamples which were not existing for the original designs, due to the fact that the abstraction model was usually conservative and introduced some transitions which were not in the original designs. This was because some state variables were removed in the abstraction model. Therefore, abstraction model refinement was needed. In [15], additional constraints and new predicates were added to the symbolic formula, when false counterexamples occured, to generate more precise abstraction model. In [16], a learning approach was proposed to find the minimum set of additional state variables which were omitted in the previous abstracting and added them to the refined

abstraction to eliminate the false counterexamples.

Bounded model checking is another widely used technique for the state space reduction. The principle is to unfold a finite state system for only k (the bound) time frames to avoid the state space explosion and then check the given properties[5]. Since the bounded model checking only verifies the property within the k time steps, induction-based verification[17] was proposed to induce that the property was also satisfied at time step $k + 1$. The root of the induction comes from the logic equivalence and implications of circuit signals. By means of the induction and the incremental SAT, the efficiency of bounded model checking was improved[17]. In addition, to improve the coverage of bounded model checking, an automatic analysis approach was proposed in [5] to check the correspondence between circuit outputs and specified properties. If an output behavior was not described by any given properties with the corresonding input and state assignments, then there existed a scenario not covered by the verification. Based on the analysis, the properties corresponding to the missing scenario was added and the verification coverage was increased.

In [18], a different methodology was proposed. The abstraction model, state machines, was built and verified incrementally. That is, a basic state machine was first built for the core functionality and a theorem prover was used to verify the state machine; then the state machine was extended to include more advanced functions and the extended model was verified without the need of verifying previous proved properties. In this way, the verification complexity was reduced. A similar but simple approach was used in [19] for Intel internal usage.

A compositional verification approach was proposed in [20], where the whole circuit system correctness was based on the verification of individual system modules. One of challenges in the compositional verification was to generate an accurate and simplified environment description for each module, representing the interaction of this module with the rest of the system. This work first generated a state graph for each module with an estimated environment; then it applied autonomous failure path removing, interface invisible state transition compacting, and redundant state transitions and states removing to reduce the state graph; and finally used an iterative algorithm to refine the interface constraints of modules alternatively to make the approximated environment more accurate.

Sequential equivalence checking is usually applied in a block wise[3]. Consistent design partitioning in different design abstract levels was recommended in [3] to facilitate the equivalence checking of the different levels. In [4, 21], the verification complexity problem was partitioned vertically during a high-level synthesis flow. The synthesis procedure from the system level to the RTL level was partitioned into three phases and equivalence checking was applied in each of the phases. As a result, the task in each verification step was simplified.

Ref.[22] took the advantages of both model abstraction and partitioning in model checking. They sequentially applied an abstraction and refinement approach-Craig interpolants, circuit-based quantification and SAT enumeration to convert circuit representations to state representations with reduced memory utilization. When the monolithic abstraction was not possible, e.g. the memory requirement was too large, a divide-and-conquer approach was applied and partial state sets were generated through reachable states associated with a certain set of primary input assignments or a set of specific properties. The verification was then carried out on the conjunction of multiple partial state sets.

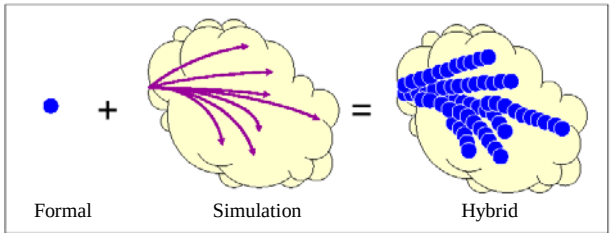

Fig. 2 A hybrid verification, where formal verification performs based on simulation traces [29]

3 Hybrid functional verification

To overcome the drawbacks of simulation and formal verification, the hybrid functional verification techniques were developed. In the survey[2], various hybrid approaches were presented, such as methods

combining formal approaches, combing informal approaches, and combing formal and informal approaches. Here we focus on the approaches combining simulation and formal verification.

Simulation can be used to bring a system to some particular states and then formal verification is applied, as shown in Fig. 2. Ref.[23] dealt with the verification problem during design transformation. They locally verified the modified system modules by abstracting properties of the modules, generated stimuli according to the property constraints, and then performed symbolic model checking. In [24], a system design was first partitioned into functional modes, directed simulation was then applied to set the system to desired states, unrelated signals during the functional modes and states were removed, and finally formal verification was performed. In [25-26], simulation traces were analyzed and modeled to extract properties, and then the properties were formally checked. Todman et al. exploited symbolic simulation and equivalence checking to verify reconfigurable hardware designs at word level in [28].

The formal verification technique can also guide simulation to improve coverage. In [27], the verification process was a two-phase iteration. In the first phase simulation was executed and when the coverage could not be further improved model checking was performed in the second phase to find the state space unvisited in the first phase. The design under verification was then set to one of the unvisited states and the verification process went back to the first phase to start a new simulation into that unvisited space.

Commercial verification tools support the hybrid verification. Ref.[30] integrated a simulator VCS to reach deep into designs and used formal verifications to check whether or not the designs satisfy a set of properties specified by users. Cadence provided the Incisive Verification Kit[31] which supported hybrid verification at both IP level and SoC level. Mentor Graphics's Questa verification platform[32] integrated simulation and formal verification tools.

4 Design for verification

Although the great efforts, described in the previous sections, have been made to improve the verification efficiency, the increasing design complexity makes the verification still a massive challenge faced by industry. A promising solution to address this problem is to shift the burden from verification to design, such that the bugs in the designs are easy to find and can be identified early to reduce the design iteration cycles.

Design-for-verification fulfills this idea and leverages designers' knowledge and experiences to facilitate verification. So far, assertion is the widely used means enabling design-for-verification. Assertion has been used in software programming as a true-false statement to check if certain conditions expected by programmers occur. It can efficiently help programmers to reason out the origins of bugs. This capability well matches to the requirement of complex SoC verification. In [33], the assertions were added into various abstract levels of designs to describe design assumptions, properties, and interface constraints, which designers think might result in errors. These assertions were then checked during simulation and formal verification, and all suspected cases were verified purposely in stead of aimlessly searching. In addition, the assertions can describe constraints to conduct stimulus generation and can be used as metric to evaluate the coverage of verification[33].

To enable the assertion-based verification methodology, three questions have to be addressed[34]: what to assert, how to write assertions and how to activate assertions. Research in academia and industry has been carried out to provide solutions to these issues.

Designers at some degree know where the assertions should place, such as the interface between the testbench and the design, the interface between modules of the design, and the internal points of the modules[35]. However, the sufficiency and completeness of the manual insertion of assertions, in general, is difficult to achieve. Zocalo Tech developed

a tool Zazz[34] to automatically find important signals which should be asserted. To address the second question, assertion languages and libraries have been developed and standardized, such as property specification language (PSL)[36-37], SystemVerilog assertions (SVA)[38] and Open Verification Library (OVL)[39]. These languages and libraries can directly be used by designers to define assertions in their designs, and are supported by the commercial verifications tools[30-32]. Ref.[35] presented the testbench generation algorithms producing testbenches which can quickly reach assertions, according to the assertion placements and property definitions. In [40], the simulation trace was monitored and when new transactions appeared assertions were generated correspondingly. The assertions were then used in the subsequent formal verification to prove complete transactions.

The assertion-based verification methodology has significantly improved the verification productivity and thus has been widely adopted by industry. In 2010, 69% of industry have used assertions as their verification technique [1].

Ref.[41] extended the idea of design-for-verification to synthesis-for-verification. As named, the approach considered how to synthesize designs to benefit the following verification. Specifically, the approach investigated the impacts of high level synthesis on the model checking of designs, and used the lessons to guide the high-level synthesis of a system-level model to generate a RTL model, which with no implementation details released the burden of model checking. The verified RTL model was then used as a golden model to compare with another synthesized RTL model from the same system-level model, which was optimized for performance and usually needed much more efforts to verify. By equivalence checking the later RTL model was verified with reduced costs.

5 Verification while designing

The design-for-verification methodology changes the traditional separate design and verification to design-considering-verification. However, as the design complexity explodes, verifying a hardware design as a whole from scratch becomes unwise.

To increase verification productivity and further extend the idea of design-for-verification, that is identifying and fixing errors earlier and breaking the iteration flow, verification can be performed in parallel with design, i.e. verification-while-designing.

Modern SoC designs are based on the top-down hierarchical structure. Design specifications are described from the system level down to the module level to the block level. This hierarchy facilitates the design, such as design team work and IP-reuse. However, as passing through each stage in the design flow from abstraction to realization, the designs are gradually flattened and the hierarchical information gets lost. The verification afterwards has to work on the designs as a whole. Although design partitioning has been increasingly used, the partitioning process, consisting of choosing right module size and boundary, design breaking up, and adding constraints to manage the separated modules, is complicate and time-consuming[42].

Therefore, a systematic methodology combining the hierarchical design and recursive verification is desired. The principle behind of this methodology is:

1) A system component is verified when it is designed, including functionality and performance. If the component design does not meet the specifications and constraints, the found errors will be fixed locally.

2) When the design of a system component is verified to be correct, an abstract model of the component is generated with the preserved interface and functionality while implementation details are removed.

3) The abstract models of the components are used to verify other components and interface protocols at the higher system levels.

4) As the design proceeds in a hierarchy, verification follows until the system level.

The procedure of the methodology is illustrated in Fig.3. This methodology can handle the design complexity issue, since it verifies the functional and performance details of individual system components separately and then checks the communications

between components using their abstract counterparts at the higher levels, with respect to the corresponding specifications. In this way, the costs of existing verification techniques, such as simulation and model checking, can be reduced and controlled. This methodology enables early error identification in the components and in the interfaces between the components at various system levels, because the verification is carried out with design coding. In addition, assertions are encouraged to be used with this methodology to leverage the benefits of existing verification efforts. The assertions can be inserted into the abstract models to help verification.

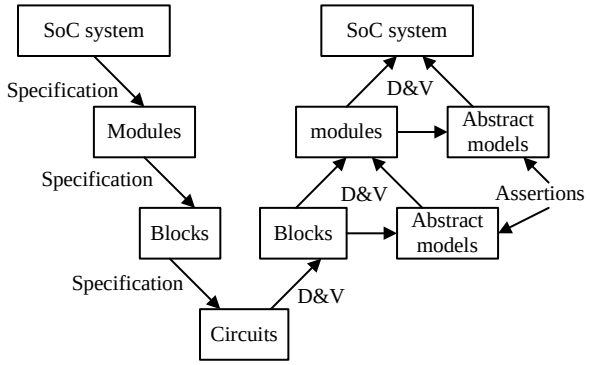

Fig.3 Illustration of the flow of verification-while-designing. D&V: Design and verification

The verification-while-designing does not mean to replace the present verification context in the design flow, but the cost and time of verification, which at present take more than half the cost of SoC hardware design, could be reduced considerably.

The foundation of this verification-while-designing methodology includes the following points:

1) Precise and detailed specifications which should clearly and completely describe the functionalities of the system components and the intricacies of signal protocols at interfaces at different levels, as well as various performance constraints such as timing, power and resources.

2) Interaction mechanism between design and verification. The innovative point of this methodology is all about verification during design. When to trigger the verification is the key. Realization of the incremental verification is promising as design proceeds. The mechanism should allow the trade-off between the incremental step and the cost of verification which could affect the design process.

3) Abstract model and its generation. The abstract model should mimic the behavior of the corresponding system components without the internal implementation details. The model should also act in compliance with specifications. Automation of building such abstract models from the verified system components is also important with regard to the verification productivity.

There have been research works targeting one piece or another of the above problems. The idea of integrating design and verification has been implemented in [43]. Transformations performed at various design levels were verified locally by equivalence checking. In [44], microarchitecture composing blocks were modeled and abstracted with parameters capturing timing and functionality. The abstracted blocks facilitated the system modeling and communication verification between blocks. Incremental verification was exploited in [45]. During physical synthesis, the changes between the new design and the old design after each circuit optimization were monitored. If a change was significant, in terms of similarity factor defined in the work, equivalence checking was performed to find the potential errors immediately. This way can quickly isolate the change that introduces errors. In [46], properties captured at the HDL level were translated into the netlist level, and thus the high-level power management strategy and the low-level circuit control signals could be verified at the RTL level. In other words, the impact of the high-level designs on the low-level circuits was known early. Recursive verification was applied to mixed-signal design verification[47], where an analog cell, needing long simulation time, after verification was replaced with an abstract model for verifying other related cells.

However, all these efforts have been done not specifically targeting verification-while-designing. Therefore, a systematic methodology and related tools are needed to enable this verification paradigm, based on the recent development in verification techniques.

6 Conclusion

This paper surveys the recent efforts in SoC hardware verification techniques. Various techniques,

methodologies and paradigms have been proposed to deal with the efficiency and quality of hardware verification. Although improvement is achieved, the speed is not matched to the SoC design trend, and industry faces the verification productivity challenge. Based on the review, we propose a new idea of verification, which is to perform verification in parallel with design of individual system modules. We outline the key points of the idea and problems which we need to address to realize verification-while-designing.

References

[1] FOSTER H. (2011) Prologue: The 2010 wilson research group functional verification study[EB/OL].(2011-03-30). http://blogs.mentor.com/verificationhorizons/blog/2011/03/30/prologuethe-2010-wilson-research-group-functional-verification-study/.

[2] BHADRA J, ABADIR M S, WANG L C, et al. A survey of hybrid techniques for functional verification[J]. IEEE Des Test, 2007, 24(2): 112-122.

[3] MATHUR A, FUJITA M, CLARKE E, et al. Functional equivalence verification tools in high-level synthesis flows [J]. Design Test of Computers, IEEE, 2009, 26(4): 88-95.

[4] KARFA C, SARKAR D, MANDAL C. Verification of datapath and controller generation phase in high-level synthesis of digital circuits[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2010, 29(3): 479-492.

[5] GROSSE D, KUHNE U, DRECHSLER R. Analyzing functional coverage in bounded model checking[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2008, 27(7): 1305-1314.

[6] FALLAH F, DEVADAS S, KEUTZER K. Functional vector generation for HDL models using linear programming and boolean satisfiability[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2001, 20(8): 994-1002.

[7] YUAN J, SHULTZ K, PIXLEY C. Modeling design constraints and biasing in simulation using BDDs[C]// IEEE/ACM International Conference on Computer-Aided Design. Piscataway: IEEE Press, 1999: 584-589.

[8] SHIH C H, HUANG J D, JOU J Y. Automatic verification stimulus generation for interface protocols modeled with non-deterministic extended FSM[J]. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 2009, 17(5): 723-727.

[9] ZHAO Y, BIAN J, DENG S, et al. Random stimulus generation with self-tuning[C]//13th International Conference on Computer Supported Cooperative Work in Design. Piscataway: IEEE Press, 2009: 62-65.

[10] GURALNIK E, AHARONI M, BIRNBAUM A, et al. Simulationbased verification of floating-point division[J]. Computers, IEEE Transactions on, 2011, 60(2): 176-188.

[11] XU X, LIM C C. Using transfer-resource graph for software-based verification of system-on-chip[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2008, 27(7): 1315-1328.

[12] BHADRA J, TROFIMOVA E, ABADIR M. Validating power architecture technology-based MPSoCs through executable specifications[J]. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 2008, 16(4): 388-396.

[13] CHEN M, MISHRA P. Functional test generation using efficient property clustering and learning techniques[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2010, 29(3): 396-404.

[14] GUZEY O, WANG L C, LEVITT J R, et al. Increasing the efficiency of simulation-based functional verification through unsupervised support vector analysis[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2010, 29(1): 138-148.

[15] JAIN H, KROENING D, SHARYGINA N, et al. Word-level predicate-abstraction and refinement techniques for verifying RTL verilog[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2008, 27(2): 366-379.

[16] HE F, SONG X, HUNG W, et al. Integrating evolutionary computation with abstraction refinement for model checking[J]. Computers, IEEE Transactions on, 2010, 59(1): 116-126.

[17] CABODI G, NOCCO S, QUER S. Strengthening model checking techniques with inductive invariants[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2009, 28(1): 154-158.

[18] BOHM P. Incremental and verified modeling of the PCI express protocol[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2010, 29(10): 1495 -1508.

[19] BEERS R. Pre-RTL formal verification: an intel experience[C]//Proceedings of the Design Automation Conference. New York: ACM, 2008: 806-811.

[20] YAO Y, ZHENG H, Automated interface refinement for compositional verification[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2009, 28(3): 433-446.

[21] KARFA C, SARKAR D, MANDAL C, et al. An equivalence-checking method for scheduling verification in high-level synthesis[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on,. 2008, 27(3): 556-569.

[22] CABODI G, GARCIA L, MURCIANO M, et al. Partitioning interpolant-based verification for effective unbounded model checking[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2010, 29(3): 382-395.

[23] RAUDVERE T, SANDER I, JANTSCH A. Application and verification of local nonsemantic-preserving transformations in system design[J]. Trans Comp-Aided Des Integ Cir Sys, 2008, 27(6): 1091-1103.

[24] TIWARI P, MITRA R S. Hybrid verification of protocol bridges[J]. IEEE Design & Test of Computers, 2007, 24(2):

124-131.
[25] SEN A, GARG V. Formal verification of simulation traces using computation slicing[J]. Computers, IEEE Transactions on, 2007, 56(4): 511-527.
[26] ROGIN F, KLOTZ T, FEY G, et al. Advanced verification by automatic property generation[J]. Computers Digital Techniques, IET, 2009, 3(4): 338-353.
[27] KARLSSON D, ELES P, PENG Z. Model validation for embedded systems using formal method-aided simulation[J]. Computers Digital Techniques, IET, 2008, 2(6): 413-433.
[28] TODMAN T, LUK W. Verification of streaming designs by combining symbolic simulation and equivalence checking [C]//International Conference on Field Programmable Logic and Applications. Piscataway: IEEE Press, 2012: 203-208.
[29] 郭炜, 郭筝, 谢憬. SoC设计方法与实现[M]. 北京: 电子工业出版社, 2011.
GUO Wei, GUO Zheng, XIE Jing. SoC: design methods and realize[M]. Beijing: Electronic Industry Press, 2011.
[30] SYNOPSYS INC. Magellan: Hybrid RTL formal verification[EB/OL].[2013-02-01].http://www.synopsys.com/Tools/Verification/FunctionalVerification/Pages/Magellan.aspx.
[31] ADENCE DESIGN SYSTEMS INC. Cadence Incisive VerificationKit[EB/OL].[2013-02-01].http://www.cadence.com/products/fv/ivkit/pages/default.aspx.
[32] MENTOR GRAPHICS CORPORATIO. Questa verification platform[EB/OL]. [2013-02-01]. http://www.mentor.com/products/fv/questa-verification-platform.
[33] SCHUTTEN R, FITZPATRICK T. Design for verification: blueprint for productivity and product quality[EB/OL]. (2003-08-07).http://www.synopsys.com/Tools/Verification/Documents/dfv wp.pdf.
[34] ZOCALO TECH. Enabling assertion based verification [EB/OL]. [2010-11-01]. http://www.zocalo-tech.com/files/assertionverification_whitepaper.pdf.
[35] PAL B, BANERJEE A, SINHA A, et al. Accelerating assertion coverage with adaptive testbenches[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2008, 27(5): 967-972.
[36] MORIN-ALLORY K, BOULE M, BORRIONE D, et al. Validating assertion language rewrite rules and semantics with automated theorem provers[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2010, 29(9): 1436-1448.
[37] CIMATTI A, ROVERI M, TONETTA S. Symbolic compilation of PSL[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2008, 27(10): 1737-1750.
[38] BUSTAN D, KORCHEMNY D, SELIGMAN E, et al. Systemverilog assertions: past, present, and future SVA standardization experience[J]. Design Test of Computers, IEEE, 2012, 29(2): 23-31.
[39] FOSTER H. Introduction to the new accellera open verification library[EB/OL]. [2013-02-01]. http://www.eda.org/ovl/pages/pdfs/dvcon06 foster.pdf.
[40] DEORIO A, BAUSERMAN A, BERTACCO V, et al. Inferno: streamlining verification with inferred semantics[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2009, 28(5): 728-741.
[41] GANAI M K, MUKAIYAMA A, GUPTA A, et al. Synthesizing “verification aware” models: Why and how? [C]//Proceedings of the 20th International Conference on VLSI Design. [s.l.]: [s.n.], 2007: 50-56.
[42] MITRA R. Strategies for mainstream usage of formal verification[C]//ACM/IEEE Design Automation Conference. New York: ACM, 2008: 800-805.
[43] SEGER C. Integrating design and verification-from simple idea to practical system[C]//IEEE/ACM International Conference on Formal Methods and Models for Co-Design. [S.l.]: IEEE, 2006: 161-162.
[44] CHATTERJEE S, KISHINEVSKY M, OGRAS U. Xmas: quick formal modeling of communication fabrics to enable verification[J]. Design Test of Computers, IEEE, 2012, 29(3): 80-88.
[45] CHANG K H, PAPA D, MARKOV I, et al. Incremental verification with error detection, diagnosis, and visualization[J]. Design Test of Computers, IEEE, 2009, 26(2): 34-43.
[46] HAZRA A, GOYAL S, DASGUPTA P, et al. Formal verification of architectural power intent[J]. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 2013, 21(1): 78-91.
[47] SHI C J. Mixed-signal system-on-chip verification using a recursively verifying-modeling (RVM) methodology[C]// IEEE International Symposium on Circuits and Systems. [S.l.]: IEEE, 2010: 1432-1435.

编 辑 蒋 晓