A Survey of Hybrid Techniques for Functional Verification

Jayanta Bhadra and Magdy S. Abadir
Freescale Semiconductor

Sandip Ray
University of Texas at Austin

Li-C. Wang
University of California, Santa Barbara

Editor's note:
This article surveys recent advances in hybrid approaches for functional verification. These approaches combine multiple verification techniques so that they complement one another, resulting in superior verification effectiveness.

—Tim Cheng, Editor in Chief

The increasing size and complexity of industry hardware designs, along with stringent time-to-market requirements, have put a heavy burden on verification to ensure that designs are relatively bug free. Late detection of errors typically leads to higher costs due to associated delays and production losses. Although bug freedom remains an unfulfilled dream, in industry practice catching more bugs earlier in the design cycle is a top priority. Verification techniques that have matured over the years have addressed the verification bottleneck—that is, the bug detection problem—to various levels of satisfaction. A general theme successfully adopted by academia as well as several vendors is to apply multiple verification techniques so that they complement one another, resulting in an increase of the verification tool's overall effectiveness. (The "Commercial hybrid verification tools" sidebar lists some examples of commercially available tools offered by various vendors.) Such integration must be carried out delicately and precisely so that the overall technique becomes more than merely a sum of the techniques. In this article, we survey the research that has taken place in this area.

In industry practice, simulation remains the mainstay for most real-life verification issues. Simulation's scalability, along with its easy applicability to practically any design at almost every abstraction level, makes it useful for all verification tasks. When used as a stand-alone technique, simulation can detect simple and easy-to-find bugs. However, over time, its effectiveness in finding corner-case bugs significantly decreases because generating stimuli that target interesting corner cases is difficult. On the other hand, although traditional formal techniques (broadly, model checking and theorem proving) can, in principle, analyze and find all bugs in a design model, their applicability in practice is limited. The well-known state explosion problem limits model checking, and the cost of theorem proving is prohibitive because of the amount of skilled manual guidance it requires.

We define any verification techniques that don't fall under the *formal* category as *informal*. Although the capabilities of formal verification paradigms have been increasing over time, the need for an immediate practical solution has increased interest in hybrid techniques, which combine formal and informal techniques. The general goal of a typical hybrid technique is to address the verification bottleneck by enhancing coverage of the state space traversed.

Taxonomy of hybrid methods

In terms of computational complexity, the verification problem ranges from NP-hard to undecidable, depending on system class, desired properties, and formal-guarantee strength. Thus, we cannot hope to

0740-7475/07/$25.00 © 2007 IEEE Copublished by the IEEE CS and the IEEE CASS

solve all verification problems efficiently. Hybrid techniques combine at least two methods, trying to complement strengths and weaknesses effectively. Dill famously presented a convincing argument supporting hybridization.[1] Bartley, Galpin, and Blackmore made a strong case for combining formal verification and traditional simulation techniques.[2] Through comparative analysis, they showed that with the increasing complexity of the circuit models in question, the most effective way to deal with complexity is to combine the strengths of all the techniques. A major challenge is to ensure that the techniques complement rather than subvert each other when working in tandem. We classify hybrid functional-verification methods as follows:

- methods combining formal and informal techniques,
- methods combining two formal techniques,
- methods combining two informal techniques, and
- methods combining multiple verification techniques.

Figure 1 illustrates this taxonomy.

Methods combining formal and informal techniques

Generally, an informal verification technique's fundamental goal is to increase design space coverage and increase the chances of finding design errors. Because of the inherent incompleteness of informal techniques, combining one with formal techniques always yields a technique that is incomplete.

Commercial hybrid verification tools

Researchers have proposed a wide variety of hybrid techniques for functional verification. Most vendors have developed and deployed tools that use hybrid techniques. Here are some examples:

- Synopsys' Magellan and Formality: http://www.synopsys.com/products/solutions/discovery_platform.html.
- Cadence Design Systems' Incisive: http://www.cadence.com/products/functional_ver/index.aspx.
- Mentor Graphics' FormalPro: http://www.mentor.com/products/fv/product_indes.cfm.

As design size and complexity increase, industry will require further advances in functional-verification technology to keep pace.

Control space exploration

Some researchers have addressed the problem of finding bugs and increasing design space coverage through exploration of control circuits. Iwashita et al. use a formal finite-state model of microprocessor control logic to generate functional-test programs usable with simulation flows.[3] The technique enumerates all reachable states of a processor's pipeline and automatically generates instruction sequences covering them. The authors focus on pipeline microarchitecture and generate directed functional tests for interesting corner cases related to pipeline hazards. Horowitz et al. published a more generalized technique that works for a larger set of circuit types.[4] This technique targets error-causing interactions by auto-

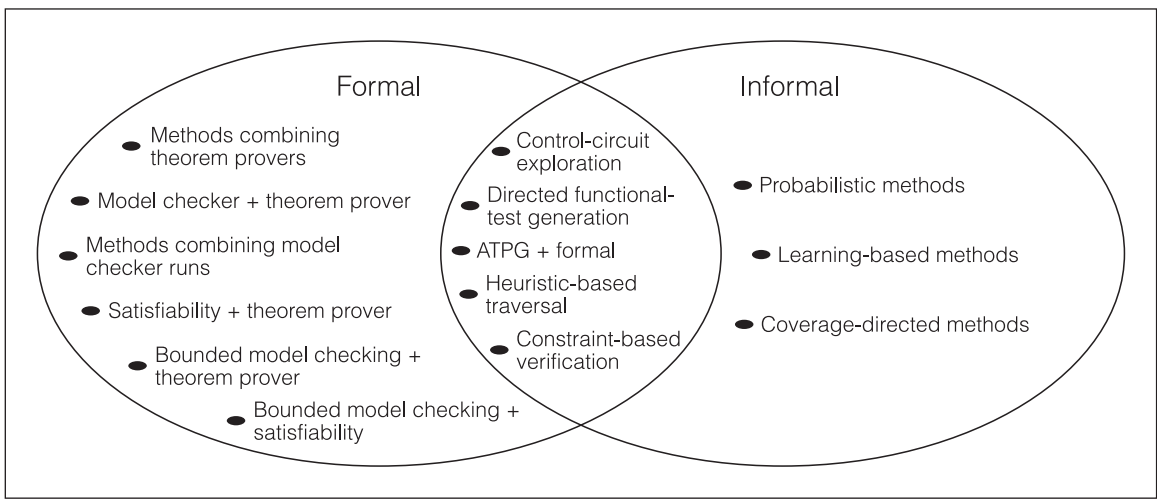

Figure 1. Hybrid verification techniques.

matically generating test vectors that make the processor exercise all control logic transitions in simulation. The technique doesn't target particular test cases but instead aims to enumerate the many improbable conditions in corner cases to maximize the probability of exposing bugs through simulation. The authors use techniques from formal verification to derive transition tours of a fully enumerated state graph of the processor's control logic. Their system works from a Verilog description of the original machine and has been used successfully to validate an embedded dual-issue processor in the node controller of the Stanford Flash Multiprocessor.

Moundanos, Abraham, and Hoskote also explored coverage-directed test generation using simulation and formal techniques,[5] in which user-provided inputs help a tool discover the interesting abstract control space behaviors of large designs. They presented the idea of covering an abstract control machine in order to generate manufacturing test suites that give better coverage of the original circuit. Ho and Horowitz present a similar approach.[6] They propose a new coverage analysis method based on projecting a minimized-control finite-state graph onto control signals for the design's data path. In essence, Moundanos, Abraham, and Hoskote as well as Ho and Horowitz propose using a state-and-edge coverage of the design's control state graph as coverage metrics to evaluate how well the original design has been tested. Both papers propose creating abstractions of the original design by extracting the design's control portion. In contrast, Geist et al. propose constructing an abstraction that models key features of the original design.[7] Although the two preceding approaches belong to the same domain—using abstract control models for test generation—they have a subtle difference: Ho and Horowitz hint that the control models could themselves be too large for analysis. To address the issue, they recognize that all control variables do not equally affect the design's behavior, and they define the concept of a control event to identify an interesting subset of the control variables. Sumners, Bhadra, and Abraham extend the work of Moundanos, Abraham, and Hoskote, proposing abstract control models that alleviate the state explosion problem by imposing a less constrained upper bound.[8]

Directed functional-test generation

An important related area of work is coverage-driven, directed functional-test generation. Typically, in an industry setting, verification teams have a set of conditions to meet before tape out. These conditions are usually based on a combination of several coverage metrics, known corner cases, and project complexity. In most industry design environments, the verification engineers are typically not required to write formal properties for system correctness but are expected to find design errors in the process of reaching this set of conditions. Directed functional verification plays an important role in reaching these conditions. Because of the ineffectiveness of pseudo-random test generation methods to cover specific corner cases, engineers generally cannot meet the needs of directed functional verification through simulation—especially under stringent time-to-market requirements. The pioneering work of Geist et al. in this area leveraged the strength of combining formal verification with simulation techniques.[7] The authors presented a study of a functional-verification methodology using the coverage of formal models to specify functional tests. They applied this methodology to a modern superscalar microprocessor and compared the resulting tests with tests generated by existing methods. The results showed that hybrid techniques can indeed improve functional verification.

An argument against some of the earlier methods[4–8] is that the step of converting abstract counterexamples obtained from abstract machines into nonspurious counterexamples of the original machine can be as complex as formally verifying the entire original machine. The designer might be able to suggest how to perform the expansion, but that makes the methods partly manual. Ganai et al. proposed a rarity-based metric for state prioritization that enabled an efficient directed search of a relatively large state space.[9] Later, Tasiran, Yu, and Batson proposed combining simulation and formal verification with an abstraction refinement technique using simulation runs in the large, original design model implementation.[10] They defined a refinement map that linked the simulation runs in the implementation to state transitions in the specification. Consequently, a model checker checks each specification-level state transition for consistency with the specification. Verification engineers can obtain error traces at the implementation level from the reverse map. The scheme also provides useful coverage information.

Although symbolic simulation is a powerful technique that can be effective in formal verification of design models, most real designs are too large and

complex for pure symbolic techniques. Yuan et al. used synthesis and verification concepts in a technique that basically combined formal verification and simulation to achieve better coverage while validating larger circuits.[11] They also showed that on designs for which formal verification is effective, hybrid techniques can find bugs far more quickly.

Mishra and Dutt proposed a hybrid technique addressing functional coverage-directed test generation for pipelined processors.[12] They made three important contributions. First, they proposed a general graph-theoretic model that captures the structure and behavior of a wide variety of pipelined processors. Second, they proposed a functional fault model that defines functional coverage for pipelined architectures. Finally, they presented test generation procedures that accept the architecture's graph model as input and generate directed test programs to detect all of the functional fault model's faults. A later technique used model checking to generate functional tests for microarchitectural descriptions of processor pipelines. The technique used an abstract symbolic-model verifier (SMV) model of a simple MIPS pipeline to generate directed tests through hybrid verification. Target corner cases found from pipeline hazard conditions were written in the form of linear temporal logic (LTL) properties. Then, the negated version of the properties was used on the SMV model to get a witnessing-instruction sequence exhibiting the target pipeline hazard. Using model checking is admittedly unrealistic for industrial-strength designs. Therefore, Koo and Mishra developed a theory of composition for pipeline models and LTL properties and applied it through satisfiability (SAT)-based bounded model-checking (BMC) methods to generate functional tests.[13]

ATPG and formal techniques

Researchers have explored the hybridization of ATPG and formal techniques. Traditionally, ATPG avoids state space explosion by dual justification and propagation techniques that localize the search instead of dealing with the entire state space at once. Because formal techniques can address the inherent incompleteness of ATPG, the two complement each other effectively.

Jones and Privitera proposed an important hybrid ATPG-formal technique in the form of automatic generation of manufacturing test vectors for functional verification, which gives the advantages of both random and directed testing.[14] The authors showed that formal specifications can be used as inputs to a test generator. Successful in dealing with commercial designs, the technique is a stepping-stone toward practical formal verification. Ganai et al. experimented with an effective hybridization of symbolic manipulation and automatic manufacturing test pattern generation. They proposed a tool called SIVA (Simulation Verification with Augmentation), which is useful for coverage-directed state space search on circuit models.[9] The tool successfully integrates simulation with symbolic techniques for efficient state space search. The main algorithms combine ATPG and binary decision diagrams (BDDs) to generate directed validation test vectors. Researchers also presented methods for automatically generating *lighthouses*, which guide the search toward interesting, hard-to-reach coverage goals.[15] Experimental results showed that the hybrid technique achieved better coverage than symbolic techniques or simulation in isolation.

Researchers have also used sequential ATPG for verifying circuit properties. Its main benefit is that it requires no explicit storage of states at each time frame. Boppana et al. suggested using sequential ATPG for model checking.[16] They verified safety properties and studied the efficiency of sequential ATPG algorithms for state space exploration. Huan and Cheng proposed combining structural, word-level, sequential ATPG techniques with modular, arithmetic, constraint-solving techniques to check safety properties.[17] They transformed the problem into a counterexample generation problem solved by sequential ATPG. Hsiao and Jain used simulation-based ATPG along with genetic algorithms to verify certain safety properties.[18] They made the important observation that although only value justification is necessary for checking safety properties, the incomplete but useful information learned from value propagation can improve ATPG performance for property checking. Abraham, Vedula, and Saab proposed an approach for formally verifying safety and bounded liveness properties using sequential ATPG.[19] Their approach automatically converts properties into a monitor circuit with a target fault so that finding a test for the fault corresponds to formally establishing the property. Sequential ATPG becomes ineffective on large, complex circuits. To address this problem, Vedula, Townsend, and Abraham used a well-known technique called program slicing to reduce the module under verification, accelerating sequential ATPG performance in solving the BMC problem.[20]

State space traversal through heuristics

Researchers have explored hybrid techniques that address efficient state space traversal through heuristics quite extensively. Yang and Dill used heuristics to guide search in relatively large state spaces.[21] Their work's basic motivation was that formal verification engines are most useful when they find bugs quickly. Among their interesting heuristics is target enlargement—enlarging the error or target state(s) to provide a larger goal state set. They also used tracks (approximate preimages), guideposts (designer hints), and Hamming distance as search metrics. Experimental results showed a one-third reduction in the state space analyzed. Hu et al. proposed a similar technique, which uses a form of overapproximated symbolic image computation to guide simulation and reach coverage goals.[22] Bergmann and Horowitz combined formal and informal verification techniques quite successfully to mitigate the state space exploration problem.[23] They showed that simple, informal artifacts such as incremental invariants, depth charts, state charts, and various other path analysis techniques can make state space search more effective for large circuits.

Shyam and Bertacco proposed a tool (Guido) based on a hybrid verification technique that uses formal methods to guide simulation toward a target.[24] It has two unique features. First, it uses circuit structure to compute a cost function to guide the simulation. Second, it has a fine-grained sequence controller that monitors and controls the direction of the simulation through a combination of randomization and controlled forward movement. Using circuit structure to compute a cost function is a better way to guide simulation than using Hamming distance as the cost function.[25] Although Hamming distance is easy to compute, two states that could be close in Hamming distance could potentially be far from each other in the state space, thus misleading the simulation. Wagner, Bertacco, and Austin proposed a tool (StressTest) and a related technique that leverage user inputs in the form of key activity points that must be stressed and monitored throughout the simulation.[26] The idea is similar to that of earlier work.[5,8] StressTest can be effective in finding corner-case design bugs as well as performance problems. It uses a Markov chain model based on activity monitors. StressTest is based on an abstract representation of the input circuit model. This implies that it is independent of the circuit implementation. This feature makes the tool more flexible and portable; but it also burdens the verification engineer with ensuring that the abstract model is correct. Otherwise, the tests generated by StressTest can be incorrect.

Constraint-based verification

Yuan, Pixley, and Aziz invented a set of constraint-based verification techniques that proved useful in industrial-strength verification problems.[27] In a constraint-driven random simulation methodology, the user provides constraints that characterize correct interaction between the design under test and its environment. Using those constraints, tools generate random stimuli for the design under test. The resultant stimuli can mimic a legal environment. The user can also provide biases that cause important corner-case behaviors to be exercised more thoroughly during simulation. The beauty of constraint-based verification is that it can be used in both formal and informal techniques. Its application base is broad because it works at the module, block, and unit levels of a design. Constraints also formally document interfaces to the design under verification in a machine-readable manner.

Methods combining formal techniques

Researchers in formal methods have widely recognized the importance of providing a way to combine disparate tools. Effectively combining theorem proving and model checking has long been a great challenge to the research community. Even with continuing advances in model-checking technology, industrial-scale verification efforts immediately encounter limits on model-checking capacity. Using theorem proving to compose verification results offers the possibility of ameliorating some of these limits, without decomposing the next-state functions used by the model checker. Therefore, the associated composition theory is relatively simple. Although the basic idea is arguably simple, implementing this mutually complementary technology is quite challenging.

Trajectory evaluation and theorem proving

Joyce and Seger experimented with combining trajectory evaluation with theorem proving. They used trajectory evaluation as a decision procedure for the higher-order logic (HOL) proof system.[28] They observed that most user interaction occurs with the model checker, not the proof system. Therefore, using a model checker as the decision procedure in a proof

system does not result in an effective hardware verification environment. Eventually, Hazelhurst and Seger developed VossProver as an experiment in implementing a lightweight proof tool on top of trajectory evaluation. They used symbolic trajectory evaluation (STE) to prove a circuit's low-level properties, and combined these properties to prove the top-level specification through a mechanical theorem prover. The technique's usefulness was demonstrated when Hazelhurst and Seger verified a 64-bit integer multiplier and later when Aagaard and Seger verified a pipelined, IEEE-compliant floating-point multiplier.[29,30] Aagaard, Jones, and Seger verified an instruction-length marker against an implementation-independent specification of the IA-32 architecture instruction lengths.[31]

Later, Forte, a formal verification environment that combines STE with lightweight theorem proving in HOL, showed that this methodology can be useful in an industrial-scale verification environment.[32] The methodology is tightly integrated with FL (a strongly typed, higher-order, general-purpose functional programming language), enabling the verification environment to be customized and large proof efforts to be organized and scripted effectively. Additionally, FL serves as a specification language at a level well above the temporal-logic specification used in the STE runs.

Combining theorem provers and decision procedures

A proposal for so-called interface logics discusses some early ideas for combining different theorem provers.[33] The goal was to connect automated reasoning tools by defining a single logic *L* such that the logic of each individual tool can be viewed as sublogics of *L*. More recently, with the success of model checkers and Boolean SAT solvers, there has been significant interest in connecting theorem provers with decision procedures as well. Modern theorem provers such as the Prototype Verification System (PVS),[34] Isabelle,[35] HOL,[36] and ACL2 (A Computational Logic for Applicative Common Lisp)[37] implement connections with external deduction tools. PVS provides connections with model checkers and SAT solvers.[38] Isabelle uses external tools as oracles for checking formulas as theorems during a proof search, and it has been used to integrate model checkers and arithmetic decision procedures.[39,40]

Connecting external tools with the HOL family of theorem provers is one of the goals of the Prosper Project, which uses the HOL98 theorem prover as a uniform, logic-based coordination mechanism between several verification tools.[41] HOL4, the latest incarnation of the family, uses an external oracle interface to decide large Boolean formulas through connections to state-of-the-art BDD and SAT-solving libraries.[42] It also uses an oracle interface to connect to ACL2.[43] There has been independent research on building sound connections between ACL2 and model checkers and SAT solvers. Ray, Matthews, and Tuttle integrate ACL2 with SMV.[44] Reeber and Hunt connect ACL2 with the Zchaff SAT solver.[45] Sawada and Reeber provide a connection between ACL2 and IBM's general-purpose transformation-based verification tool, SixthSense,[46] to verify an industry floating-point multiplier design.[47] Manolios and Srinivasan connect ACL2 with Uclid.[48]

In implementing connections between two formal tools, soundness guarantees provided by their combination are of obvious importance. Most of the interfaces just mentioned involve a form of trust tag that indicates that the validity of results certified by the combined tools relies on the soundness of all the individual tools and their integration. In HOL and Isabelle, the tag is a logical construct introduced as a hypothesis of each certified formula.[49] ACL2 implements tagging at the level of definition and theorem files.[50] There has also been work on using an external tool to search for a proof that the theorem prover can check without assistance from the tool. Hurd describes such an interface that connects HOL with first-order logic.[51] McCune and Shumsky present a system called Ivy that uses the Otter theorem prover to search for first-order proofs of formulas in equational theories, and then invokes ACL2 to check the proof objects.[52]

Composition of model-checking runs

Other notable work on hybrid formal techniques involves composition of several model-checking runs. Camilleri used a theorem prover in conjunction with a model checker to verify a cache coherence protocol.[53] Separate tools were used to verify different properties, but the results were not combined mechanically. Jang et al. used computation tree logic (CTL) model checking to verify a set of properties on an embedded microcontroller.[54] The proof of the top-level specification was achieved through a compositional argument using the properties but was not mechanized through a theorem prover.

Predicate abstractions

Another hybrid formal approach is the use of predicate abstraction to prove invariants of system implementations.[55] Predicate abstraction is a form of abstract interpretation in which, given a set of predicates P, an abstract transition relation is constructed that stipulates how each predicate in P is updated at each concrete transition.[56] The abstract system is a conservative approximation of the concrete design. The method combines formal tools in the following manner. Determining the transition relation involves the use of validity checks, typically with a theorem prover or SAT solver, and exploring the abstract system reduces to a reachability analysis problem for a model checker to perform. The research challenges include discovering an appropriate set of predicates and reducing the number of validity checks for constructing the abstract transition relation. Researchers have addressed predicate discovery with refinement guided by counterexamples and effective use of search.[57] One promising approach, developed by Namjoshi and Kurshan, involves computing a fixpoint over the weakest liberal precondition starting with an initial set of predicates.[58] This method has been used with indexed predicate discovery in Uclid,[59] and also forms the basis of a rewrite-based predicate abstraction approach implemented in ACL2.[60] Researchers have reduced validity checks through effective representation of the abstract models[61] and the use of expressive, quantified predicates.[59,60]

Combining two model-checking techniques

Hazelhurst et al. proposed a tool (MIST) that hybridizes two model-checking techniques. The tool enables a handshake between STE and SMC—either BDD- or SAT-based.[62] Model checking is effective in proving that a property holds on a circuit model but is generally capacity-constrained by the state explosion problem. MIST addresses SMC's state initialization problem and is especially useful for circuit instances that have complex initialization sequences. MIST uses STE to automatically obtain these complex, symbolic initial states. STE is a natural choice for this application because of the inherent abstraction provided by the antecedent of the initial formula used, even when the original circuit is large and complex. Once the initial set of states is obtained with STE, the tool can use SMC to check the property on the original circuit model. MIST enhances SMC's capacity and performance; helps the debugging process by letting the verifier focus on critical, error-prone areas; and makes the initialization process more efficient.

Methods combining informal techniques

Kuehlmann et al. reported a guiding algorithm that uses probabilistic techniques.[63] It assigns each design state a probability based on the likelihood of the state's leading to a target state. The algorithm allocates a set of ranks to the design states according to the assigned probabilities. Guided-search algorithms use the ranking system to find a path from the starting states to one of the target states. These algorithms could act as good complements to existing hybrid techniques for state space search. However, because the probability values are assigned by approximate analysis, there is no apparent mechanism to avoid dead-end states. Yuan et al. introduced the concept of input biasing, which can be considered a probabilistic constraint.[64] Input biasing makes it easier to cover interesting corner cases. The researchers proposed using constraints and biasing to form a simulation environment instead of using an explicit testbench in hierarchical functional verification. The method unified the handling of biases and constraints through BDDs.

Researchers have also proposed several learning-based techniques. Various coverage-directed functional-verification schemes from IBM have proven effective for addressing large-scale verification problems. One method uses computer learning and Bayesian networks.[65] Shimizu and Dill describe coverage-directed informal methods that use formal descriptions for collecting coverage information and deriving simulation inputs.[66] The description is a list of interface properties describing a bus protocol. The cache coherence protocol specification is cycle accurate and is in the form of RTL interface specifications. However, the drawback of these techniques is that the properties described are localized in time; for example, properties cannot express constraints on bus protocol transactions.

Tasiran et al. proposed a novel coverage metric—tag coverage—which addresses a major weakness of the code coverage metric by augmenting it with an observability measure.[67] The tag coverage metric considers a code segment in the model to be covered only when it has executed, and the execution's effect is recorded at one of the points in the circuit under observation. The authors use the tag coverage measure to guide a semiformal functional-test-genera-

tion algorithm as it selects probability distributions for biased random-input-pattern generation covering targeted portions of the state space. The algorithm is based on an approximate analysis of the circuit modeled as a Markov chain in the steady state.

Methods combining multiple techniques

The first successful attempt to combine several disparate but cooperative verification techniques into a single hybrid technique was Ketchum by Ho et al.[68] This tool improved traditional simulation techniques by using capabilities such as test pattern generation and nonreachability analysis. Ketchum combines simulation with multiple formal methods, including symbolic simulation, SAT-based BMC, symbolic fixpoint computation, and automatic abstraction. It also addresses the design engineer's requirements by enabling simulation monitors. Nonreachability analysis helps design engineers focus on coverage issues far earlier in the design cycle. The tool interleaves random simulation with symbolic simulation to expose buggy behavior in deep circuit blocks. Additionally, by performing reachability analysis on an abstract design, the tool rules out unreachable configurations, thus pruning the explored state space. Ho et al. reported that Ketchum's effective hybrid technique provides a tenfold capacity enhancement compared to previous results. The hybrid functional-test-generation algorithms of Ketchum and SIVA[9] are generally considered to be in the same domain because both tools interleave simulation and formal engines to reach coverage goals. However, there are some differences. First, SIVA uses ATPG and symbolic image computation, whereas Ketchum uses symbolic simulation and SAT-based BMC. Second, SIVA maximizes toggle coverage, whereas Ketchum can take advantage of arbitrary simulation checkers. Third, because SIVA doesn't use a single simulation trace but rather computes a search tree, Ketchum has an advantage over SIVA in the simulation phase.

FUTURE AVENUES for increasing tool capacity and accuracy might include developing new verification procedures better optimized to use adaptive techniques, and seamlessly addressing various subproblems to synergistically solve the overall verification challenge. This can be achieved through hybrid tools that perform finer-grained handshakes between various tools to provide features such as circuit preanalysis for activating various verification engines, on-the-fly interaction between verification techniques, and realistic user guidance. ■

Acknowledgments

Sandip Ray is partially supported by DARPA and the National Science Foundation under grant CNS-0429591.

References

1. D.L. Dill, "What's Between Simulation and Formal Verification?" *Proc. 35th Design Automation Conf.* (DAC 98), ACM Press, 1998, pp. 328-329.
2. M.G. Bartley, D. Galpin, and T. Blackmore, "A Comparison of Three Verification Techniques: Directed Testing, Pseudo-Random Testing and Property Checking," *Proc. 39th Design Automation Conf.* (DAC 02), ACM Press, 2002, pp. 819-823.
3. H. Iwashita et al., "Automatic Test Program Generation for Pipelined Processors," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design* (ICCAD 94), IEEE Press, 1994, pp. 580-583.
4. M.A. Horowitz et al., "Architecture Validation for Processors," *Proc. 22nd Ann. Int'l Symp. Computer Architecture* (ISCA 95), IEEE CS Press, 1995, pp. 404-413.
5. D. Moundanos, J.A. Abraham, and Y.V. Hoskote, "Abstraction Techniques for Validation Coverage Analysis and Test Generation," *IEEE Trans. Computers*, vol. 47, no. 1, Jan. 1998, pp. 2-14.
6. R.C. Ho and M.A. Horowitz, "Validation Coverage Analysis for Complex Digital Designs," *Proc. Int'l Conf. Computer-Aided Design* (ICCAD 96), IEEE CS Press, 1996, pp. 146-153.
7. D. Geist et al., "Coverage-Directed Test Generation Using Symbolic Techniques," *Proc. 1st Int'l Conf. Formal Methods in Computer-Aided Design*, LNCS 1166, Springer-Verlag, 1996, pp. 143-158.
8. R. Sumners, J. Bhadra, and J. Abraham, "Automatic Validation Test Generation Using Extracted Control Models," *Proc. 13th Int'l Conf. VLSI Design* (VLSID 00), IEEE CS Press, 2000, pp. 312-320.
9. M. Ganai et al., "SIVA: A System for Coverage-Directed State Space Search," *J. Electronic Testing: Theory and Applications*, vol. 17, no. 1, Feb. 2001, pp. 11-27.
10. S. Tasiran, Y. Yu, and B. Batson, "Linking Simulation with Formal Verification at a Higher Level," *IEEE Design & Test*, vol. 21, no. 6, Nov.–Dec. 2004, pp. 472-482.
11. J. Yuan et al., "On Combining Formal and Informal Verification," *Proc. 9th Int'l Conf. Computer-Aided*

Verification, LNCS 1254, Springer-Verlag, 1997, pp. 376-387.

12. P. Mishra and N. Dutt, "Functional Coverage Driven Test Generation for Validation of Pipelined Processors," *Proc. Design, Automation and Test in Europe* (DATE 05), IEEE CS Press, vol. 2, 2005, pp. 678-683.
13. H.-M. Koo and P. Mishra, "Test Generation Using SAT-Based Bounded Model Checking for Validation of Pipelined Processors," *Proc. 16th ACM Great Lakes Symp. VLSI* (GLSVLSI 06), ACM Press, 2006, pp. 362-365.
14. K.D. Jones and J.P. Privitera, "The Automatic Generation of Functional Test Vectors for Rambus Designs," *Proc. 33rd Design Automation Conf.* (DAC 96), ACM Press, 1996, pp. 415-420.
15. P. Yalagandula, A. Aziz, and V. Singhal, "Automatic Lighthouse Generation for Directed State Space Search," *Proc. Design, Automation and Test in Europe* (DATE 00), IEEE CS Press, 2000, pp. 237-242.
16. V. Boppana et al., "Model Checking Based on Sequential ATPG," *Proc. 11th Int'l Conf. Computer Aided Verification*, LNCS 1633, Springer, 1999, pp. 418-430.
17. C.-Y. Huan and K.-T. Cheng, "Using Word-Level ATPG and Modular Arithmetic Constraint-Solving Techniques for Assertion Property Checking," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 3, Mar. 2001, pp. 381-391.
18. M. Hsiao and J. Jain, "Practical Use of Sequential ATPG for Model Checking: Going the Extra Mile Does Pay Off," *Proc. 6th IEEE Int'l High-Level Design Validation and Test Workshop* (HLDVT 01), IEEE CS Press, 2001, pp. 39-44.
19. J.A. Abraham, V.M. Vedula, and D.G. Saab, "Verifying Properties Using Sequential ATPG," *Proc. Int'l Test Conf.* (ITC 02), IEEE CS Press, 2002, pp. 194-202.
20. V.M. Vedula, W.J. Townsend, and J.A. Abraham, "Program Slicing for ATPG-Based Property Checking," *Proc. 17th Int'l Conf. VLSI Design* (VLSID 04), IEEE CS Press, 2004, pp. 591-596.
21. C.H. Yang and D.L. Dill, "Validation with Guided Search of the State Space," *Proc. 35th Design Automation Conf.* (DAC 98), ACM Press, 1998, pp. 599-604.
22. A.J. Hu et al., "Approximate Reachability with BDDs Using Overlapping Projections," *Proc. 35th Design Automation Conf.* (DAC 98), ACM Press, 1998, pp. 451-456.
23. J.P. Bergmann and M.A. Horowitz, "Improving Coverage Analysis and Test Generation for Large Designs," *Proc. Int'l Conf. Computer-Aided Design* (ICCAD 99), IEEE CS Press, 1999, pp. 580-583.
24. S. Shyam and V. Bertacco, "Distance-Guided Hybrid Verification with GUIDO," *Proc. Design, Automation and Test in Europe* (DATE 06), European Design and Automation Assoc, vol. 1, 2006, pp. 1211-1216.
25. M.K. Ganai, A. Aziz, and A. Kuehlmann, "Enhancing Simulation with BDDs and ATPG," *Proc. 36th Ann. Design Automation Conf.* (DAC 99), ACM Press, 1999, pp. 385-390.
26. I. Wagner, V. Bertacco, and T. Austin, "StressTest: An Automatic Approach to Test Generation via Activity Monitors," *Proc. 42nd Design Automation Conf.* (DAC 05), ACM Press, 2005, pp. 783-788.
27. J. Yuan, C. Pixley, and A. Aziz, *Constraint-Based Verification*, Springer, 2006.
28. J.J. Joyce and C.H. Seger, "Linking BDD-Based Symbolic Evaluation to Interactive Theorem Proving," *Proc. 30th Design Automation Conf.* (DAC 93), ACM Press, 1993, pp. 469-474.
29. S. Hazelhurst and C.-J.H. Seger, "A Simple Theorem Prover Based on Symbolic Trajectory Evaluation and BDDs," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 4, Apr. 1995, pp. 413-422.
30. M.D. Aagaard and C.-J.H. Seger, "The Formal Verification of a Pipelined Double-Precision IEEE Floating-Point Multiplier," *Proc. Int'l Conf. Computer-Aided Design* (ICCAD 95), IEEE CS Press, 1995, pp. 7-10.
31. R.B. Jones, C.-J.H. Seger, and M. Aagaard, "Combining Theorem Proving and Trajectory Evaluation in an Industrial Environment," *Proc. 35th Design Automation Conf.* (DAC 98), ACM Press, 1998, pp. 538-541.
32. C.-J.H. Seger et al., "An Industrially Effective Environment for Formal Hardware Verification," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 9, Sept. 2005, pp. 1381-1405.
33. J.D. Guttman, *A Proposed Interface Logic for Verification Environments*, tech. report M-91-19, Mitre, 1991.
34. S. Owre, J. Rushby, and N. Shankar, "PVS: A Prototype Verification System," *Proc. 11th Int'l Conf. Automated Deduction* (CADE-11), LNCS 607, Springer, 1992, pp. 748-752.
35. T. Nipkow, L. Paulson, and M. Wenzel, *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, LNCS 2283, Springer-Verlag, 2002.
36. M.J.C. Gordon, and T.F. Melham, eds., *Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic*, Cambridge Univ. Press, 1993.
37. M. Kaufmann, P. Manolios, and J. Moore, *Computer-Aided Reasoning: An Approach*, Kluwer Academic, 2000.

38. N. Shankar, "Using Decision Procedures with a Higher-Order Logic," *Proc. 14th Int'l Conf. Theorem Proving in Higher-Order Logics* (TPHOLs 01), LNCS 2152, Springer, 2001, pp. 5-26.
39. O. Müller and T. Nipkow, "Combining Model Checking and Deduction of I/O Automata," *Proc. 1st Workshop Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1019, Springer-Verlag, 1995, pp. 1-16.
40. D. Basin and S. Friedrich, "Combining WS1S and HOL," *Frontiers of Combining Systems 2*, D.M. Gabbay, and M. de Rijke, eds., Research Studies Press/Wiley, 2000.
41. L.A. Dennis et al., "The PROSPER Toolkit," *Proc. Int'l Conf. Tools and Algorithms for Constructing Systems* (TACAS 00), LNCS 1831, Springer-Verlag, 2000, pp. 78-92.
42. M.J.C. Gordon, "Programming Combinations of Deduction and BDD-Based Symbolic Calculation," *London Mathematical Society J. Computation and Mathematics*, vol. 5, Aug. 2002, pp. 56-76.
43. M.J.C. Gordon et al., "An Integration of HOL and ACL2," *Proc. Formal Methods in Computer-Aided Design* (FMCAD 06), IEEE CS Press, 2006, pp. 153-160.
44. S. Ray, J. Matthews, and M. Tuttle, "Certifying Compositional Model Checking Algorithms in ACL2," *Proc. 4th Int'l Workshop ACL2 Theorem Prover and Its Applications*, ACL2 Steering Committee, 2003, http://www.cs.utexas.edu/users/moore/acl2/workshop-2003.
45. E. Reeber and W.A. Hunt Jr., "A SAT-Based Decision Procedure for the Subclass of Unrollable List Formulas in ACL2 (SULFA)," *Proc. 3rd Int'l Joint Conf. Automated Reasoning* (IJCAR 06), LNCS 4130, Springer, 2006, pp. 453-467.
46. H. Mony et al., "Scalable Automated Verification via Expert-System Guided Transformations," *Proc. 5th Int'l Conf. Formal Methods in Computer-Aided Design* (FMCAD 04), LNCS 3312, Springer, 2004, pp. 159-173.
47. J. Sawada and E. Reeber, "ACL2SIX: A Hint Used to Integrate a Theorem Prover and an Automated Verification Tool," *Proc. Formal Methods in Computer-Aided Design* (FMCAD 06), IEEE CS Press, 2006, pp. 161-170.
48. P. Manolios and S.K. Srinivasan, "Refinement Maps for Efficient Verification of Processor Models," *Proc. Design, Automation and Test in Europe* (DATE 05), IEEE CS Press, vol. 2, 2005, pp. 1304-1309.
49. E.L. Gunter, "Adding External Decision Procedures to HOL90 Securely," *Proc. 11th Int'l Conf. Theorem Proving in Higher-Order Logics* (TPHOLs 98), LNCS 1479, Springer, 1998, pp. 143-152.
50. M. Kaufmann et al., "Integrating External Deduction Tools with ACL2," *Proc. 6th Int'l Workshop Implementation of Logics* (IWIL 06), CEUR Workshop Proceedings, 2006, pp. 7-26.
51. J. Hurd, "An LCF-Style Interface between HOL and First-Order Logic," *Proc. 18th Int'l Conf. Automated Deduction* (CADE-18), LNCS 2392, Springer, 2002, pp. 134-138.
52. W. McCune and O. Shumsky, "Ivy: A Preprocessor and Proof Checker for First-Order Logic," *Computer-Aided Reasoning: ACL2 Case Studies*, P. Manolios, M. Kaufmann, and J.S. Moore, eds., Kluwer Academic, 2000, pp. 217-230.
53. A. Camilleri, "A Hybrid Approach to Verifying Liveness in a Symmetric Multi-Processor," *Proc. 10th Int'l Conf. Theorem Proving in Higher-Order Logics* (TPHOLs 97), LNCS 1275, Springer, 1997, pp. 33-48.
54. J.-Y. Jang et al., "Formal Verification of FIRE: A Case Study," *Proc. 34th Design Automation Conf.* (DAC 97), ACM Press, 1997, pp. 173-177.
55. S. Graf and H. Saidi, "Construction of Abstract State Graphs with PVS," *Proc. 9th Int'l Conf. Computer-Aided Verification*, LNCS 1254, Springer-Verlag, 1997, pp. 72-83.
56. P. Cousot and R. Cousot, "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Approximation or Analysis of Fixpoints," *Proc. 4th ACM SIGACT-SIGPLAN Symp. Principles of Programming Languages* (POPL 77), ACM Press, 1977, pp. 238-252.
57. S. Das and D.L. Dill, "Counter-Example Based Predicate Discovery in Predicate Abstraction," *Proc. 4th Int'l Conf. Formal Methods in Computer-Aided Design* (FMCAD 02), LNCS 2517, Springer, 2002, pp. 19-32.
58. K.S. Namjoshi and R.P. Kurshan, "Syntactic Program Transformations for Automatic Abstraction," *Proc. 12th Int'l Conf. Computer-Aided Verification* (CAV 00), LNCS 1855, Springer, 2000, pp. 435-449.
59. S. Lahiri and R.E. Bryant, "Indexed Predicate Discovery for Unbounded System Verification," *Proc. 16th Int'l Conf. Computer-Aided Verification* (CAV 04), LNCS 3114, Springer, 2004, pp. 135-147.
60. S. Ray and R. Sumners, "Combining Theorem Proving with Model Checking through Predicate Abstraction," *IEEE Design & Test*, vol. 24, no. 2, Mar.-Apr. 07, pp. 132-139.
61. S.K. Lahiri, R.E. Bryant, and B. Cook, "A Symbolic Approach to Predicate Abstraction," *Computer-Aided Verification*, LNCS 2725, LNCS, 2003, pp. 141-153.
62. S. Hazelhurst et al., "A Hybrid Verification Approach: Getting Deep into the Design," *Proc. 39th Design Automation Conf.* (DAC 02), ACM Press, 2002, pp. 111-116.
63. A. Kuehlmann, K.L. McMillan, and R.K. Brayton, "Probabilistic State Space Search," *Proc. Int'l Conf. Computer-Aided Design* (ICCAD 99), IEEE CS Press, 1999, pp. 574-579.

64. J. Yuan et al., "Modeling Design Constraints and Biasing in Simulation Using BDDs," *Proc. Int'l Conf. Computer-Aided Design* (ICCAD 99), IEEE CS Press, 1999, pp. 584-590.
65. S. Fine and A. Ziv, "Coverage Directed Test Generation for Functional Verification Using Bayesian Networks," *Proc. 40th Design Automation Conf.* (DAC 03), ACM Press, 2003, pp. 286-291.
66. K. Shimizu and D.L. Dill, "Deriving a Simulation Input Generator and a Coverage Metric from a Formal Specification," *Proc. 39th Design Automation Conf.* (DAC 02), ACM Press, 2002, pp. 801-806.
67. S. Tasiran et al., "A Functional Validation Technique: Biased-Random Simulation Guided by Observability-Based Coverage," *Proc. IEEE Int'l Conf. Computer Design* (ICCD 01), IEEE CS Press, 2001, pp. 82-88.
68. P.-H. Ho et al., "Smart Simulation Using Collaborative Formal and Simulation Engines," *Proc. Int'l Conf. Computer-Aided Design* (ICCAD 00), IEEE CS Press, 2000, pp. 120-126.

The biographies of **Jayanta Bhadra**, **Magdy S. Abadir**, and **Li-C. Wang** are on p. 111 of this issue.

Sandip Ray is a postdoctoral fellow in the Department of Computer Sciences of the University of Texas at Austin. His research interests include formal methods (particularly the effective combination of theorem proving and algorithmic decision procedures to increase the capacity of formal verification for large-scale systems), distributed systems, complexity theory, algorithm design, model checking, and logic. Ray has a BS in computer science from Jadavpur University, Calcutta, India, an MS in computer science from the Indian Institute of Science, Bangalore, India, and a PhD in computer science from the University of Texas at Austin.

■ Direct questions and comments about this article to Jayanta Bhadra, Freescale Semiconductor, 7700 W. Parmer Lane, MD PL34, Austin, TX 78729; jayanta.bhadra@freescale.com.