RT-level ITC'99 Benchmarks and First ATPG Results

Fulvio Corno

Matteo Sonza Reorda

Giovanni Squillero
Politecnico di Torino

New design flows require reducing work at the gate level and perfoming most activities before the synthesis step, including evaluatation of testability of circuits. We propose a suite of RT-level benchmarks that help improve research in high-level ATPG tools. First results on the benchmarks obtained with our prototype tool show the feasibility of the approach.

The availability of suitable and meaningful benchmarks influences the speed and effectiveness of any technical research area's evolution process. These allow an easier and unbiased evaluation of new ideas, accelerating the process towards selection of proposals that can be adopted most effectively in industrial practice.

In the area of digital circuit testing, the International Symposium on Circuits and Systems (ISCAS) 1985[1] and ISCAS 89[2] gate-level benchmarks were introduced to evaluate combinational and sequential automatic test-pattern generation (ATPG) tools. Despite their initial purpose, they have been used for almost all applications in the test field, as well as for assessing the effectiveness of new methods in additional areas, including logic optimization, power estimation, and partitioning. Today, this kind of benchmark is still essential to develop, test, and improve computer-aided design (CAD) algorithms and tools, but is gradually losing importance due to the introduction of new design techniques. For this reason, new benchmarks that reflect current design requirements need to be developed and distributed.

Current and future CAD tools must face a new generation of systems-on-chip, whose complexity increases in various ways, such as the

- incorporation of large portions of imported blocks, such as memories, processor cores, and intellectual property (IP) blocks;
- presence of particular architectural issues, such as internal buses, test access logic (boundary scan, scan chains) and test execution logic (built-in self test, or BIST), power management logic, multiple clock frequencies, and clock domains;
- increasing adoption of fault-tolerant structures, now essential even in consumer electronic products as circuit density increases; and
- higher level of abstraction, since in many cases most of the circuit is described at the register-transfer (RT) level in a hardware description language.

0740-7475/00/$10.00 © 2000 IEEE

Benchmark circuits donated by industrial entities encompass all these issues, but are largely dissimilar in terms of description levels and styles, adopted libraries, test strategy, and so on. This creates difficulties for researchers developing new techniques and testing them over a series of circuit instances. While current design tools cannot ignore all the complexity aspects, a simpler set of benchmarks is more useful for core algorithm development. ATPG algorithms, in particular, are easier to develop, optimize, and improve when first applied to simplified circuits (that is, single-clock, synchronous, with simple libraries) and later adapted to general circuits by means of both algorithm extensions and circuit/library transformations. In fact, while new test tools able to deal with RT-level descriptions (the last item listed above) are a major conceptual problem, most of the other issues can be considered separately or added later. Here we concentrate on benchmarks described at the RT level and will ignore the other complexity sources.

The need for test tools that work on high-level descriptions became apparent more than two decades ago. It's now increasingly accepted that the availability of effective high-level test tools would benefit the design process.[3] Resistance comes from an unproven belief that lack of structural information makes it impossible to generate effective test sequences starting from high-level descriptions. It's true that some fault effects can only be modeled at the lower abstraction levels, but this may be balanced by a higher efficiency of the high-level ATPG process that works on more compact descriptions, containing additional information about the functional behavior of the component modules. Some structural faults may be missed, but a larger set of system configurations is more readily accessible.[4] Whether automatic test generation can effectively be performed on RT-level descriptions remains an open question that requires a widely accepted suite of neutral but representative benchmarks. Since the common practice in industrial testing is still to evaluate the effectiveness of generated sequences on gate-level descriptions and fault lists, the availability of both the RT- and the corresponding synthesized gate-level descriptions proves crucial for the success of this suite.

The need for test tools that work on high-level descriptions became apparent more than two decades ago.

These considerations motivated us to develop a set of RT-level benchmarks and to contribute it to the ITC 99 benchmark set.[5] These benchmarks lack many characteristics of industrial circuits, but offer a wide set of test cases, of different complexity, with uniform characteristics. At the same time, we started developing an ATPG environment able to generate test sequences starting from synthesizable RT-level descriptions. Our goal was to prove that test sequences generated by RT-level ATPGs could reach a stuck-at fault coverage on the corresponding gate-level circuits at least comparable with the one obtained by traditional gate level tools. RT-level ATPGs can assist designers because they allow for identification of hardly testable circuits (or circuit components) early in the design flow (such as before the logic synthesis step). This check is now performed at the gate level only, requiring redesign in cases of negative result. Moreover, for some circuits the test sequences generated at the RT level are so effective that the gate-level ATPG step can be completely avoided. For the purpose of this work, we assume that the circuits do not exploit any design-for-testability mechanism (full or partial scan, for example).

We introduce the suite of RT-level benchmarks developed, providing general information about characteristics and standards we adopted for their description. In addition, we outline the RT-level prototypical ATPG tool we developed, called Automatic RT-level Input Sequence generator for Test purposes (Artist), and report a first set of experimental data gathered using its prototypical implementation. These results can serve as a reference point for those working in the area of high-level ATPG algorithms and should demonstrate the feasibility of RT-level ATPG and design validation.

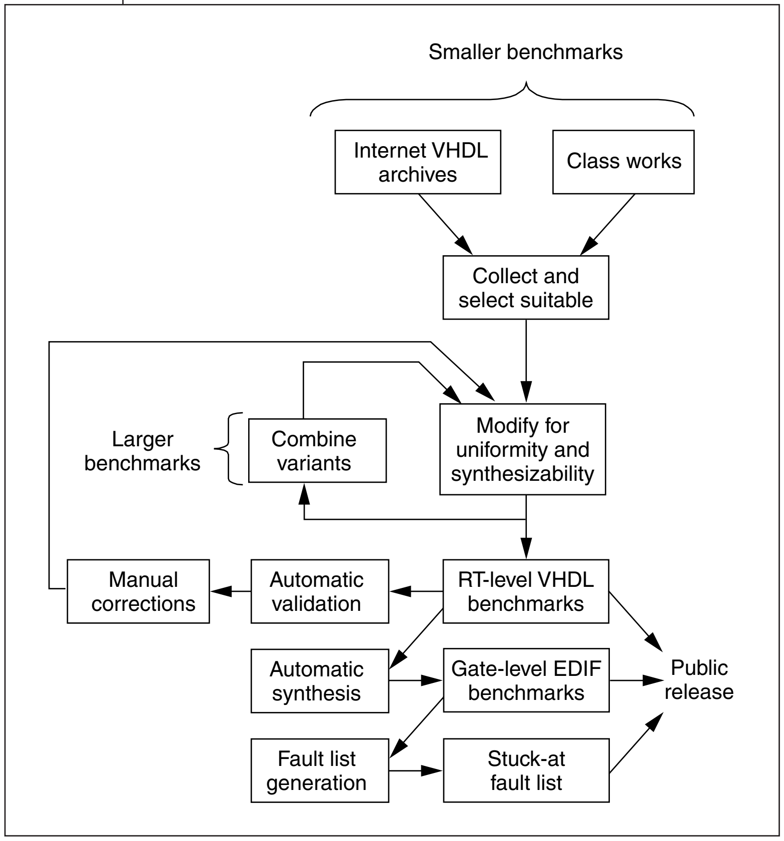

Figure 1. Benchmark development process.

Benchmark circuits development

Among the ITC'99 benchmark suite, our team at Politecnico di Torino contributed 22 circuits for developing algorithms working on RT-level circuits described in VHDL. They consist of VHDL sources in a standardized format and of the corresponding gate-level netlists and fault lists. The benchmarks are representative of typical circuits, or circuit parts, that current tools can automatically synthesize as a whole. They share the following properties:

- Circuits are described in synthesizable VHDL at the RT level. When different synthesizers required different styles, we followed Synopsys Design Compiler description styles. However, we explicitly avoided any compiler-specific directive. We used no VHDL packages except IEEE standard logic and arithmetic packages. The code is mainly behavioral, with one or more concurrent processes, but some circuits also contain structural code. For simpler parsing, no concurrent statements appear outside processes.
- Gate-level descriptions are available, in simplified flattened electronic data interchange format (EDIF) format and in ISCAS 89 bench format. We obtained them with logic synthesis over a library compatible with ISCAS 89 elementary gates.
- Fault lists for single stuck-at faults (complete and collapsed) are available, generated by an industry-standard tool in an easily readable format.
- Circuits behave in a purely synchronous way: one single-phase clock signal goes directly to all memory elements without intervening logic. This constraint simplifies the timing model under which the circuits operate, but also increases the predictability of the synthesis process.
- A global reset signal allows a trivial initialization sequence for all benchmarks. Different from gate-level circuits, RT-level VHDL descriptions always require an initialization sequence to simulate properly. (If simulated from an undetermined state, the VHDL simulation semantics for synthesizable behavioral statements doesn't correctly propogate undefined values.) Thus the presence of a global reset signal is simply for syntactic simplification.
- No internal memories (except register banks), three-state busses, or wired connections are present.
- The 22 circuits cover wide size and complexity ranges: from 1 to 37 primary inputs (plus the ubiquitous clock and reset signals), from 1 to 97 primary outputs, from 1 to 33 VHDL processes, from 68 to 1,613 VHDL lines, from 4 to 3,320 flip-flops, and from 46 to 68,752 combinational gates. One of the circuits (b16) is parametric and can be synthesized to different sizes. The largest circuit (b18), with its 68K gates and 430K faults (190K collapsed), is much larger than the largest ISCAS 89 benchmark.

The benchmark development process appears in Figure 1. We started from a set of VHDL descriptions collected from various sources and modified them to comply with

Table 1. Benchmark characteristics.

Circuit			VHDL		Gate-Level		Fault List		Random
Name	PI	PO	No. of Lines	No. of Processes	No. of Gates	No. of Flip-flops	Complete	Collapsed	Testability %
b01	2	2	110	1	46	5	258	127	98.45
b02	1	1	70	1	28	4	150	64	99.33
b03	4	4	141	1	149	30	822	382	74.82
b04	11	8	102	1	597	66	3,356	1,477	84.24
b05	1	36	332	3	935	34	5,552	2,553	33.48
b06	2	6	128	1	60	9	302	151	93.71
b07	1	8	92	1	420	49	2,404	1,120	58.28
b08	9	4	89	1	167	21	918	439	98.15
b09	1	1	103	1	159	28	900	417	87.33
b10	11	6	167	1	189	17	1,054	468	91.56
b11	7	6	118	1	481	31	2,868	1,308	91.81
b12	5	6	569	4	1,036	121	6,084	2,777	21.22
b13	10	10	296	5	339	53	1,818	835	32.45
b14	32	54	509	1	4,775	245	28,990	12,643	81.39
b15	36	70	671	3	8,893	449	55,568	23,316	13.05
b16	M+1	1	68	n	f(N,M)	N	f(N,M)	f(N,M)	f(N,M)
b17	37	97	810	15	24,194	1,415	152,808	65,324	8.81
b18	36	23	1,424	33	68,752	3,320	429,712	188,458	1.34
b20	32	22	1,085	3	9,419	490	57,794	25,274	81.91
b21	32	22	1,089	3	9,803	490	60,052	26,516	85.20
b22	32	22	1,613	4	15,071	735	92,536	40,200	81.30

desired characteristics. The initial VHDL files came from public archives over the Internet and from student work at our institution. Structurally combining copies or slight variations of smaller benchmarks created larger ones. Modifications were aimed at guaranteeing synchronicity and at increasing syntactical uniformity. They included adding reset signals, moving or removing wait statements, adding and/or joining clock signals, eliminating redundant hierarchy levels, eliminating references to external packages, and other minor corrections. In this process, we sacrificed the original behavior in favor of uniformity of description. Finally the circuits were synthesized with Synopsys Design Compiler version 1998.08 over a library composed of 1- to 5-input simple gates and D-type flip-flops.

The circuits have been publicly available since 1997.[6] However, in early 1999 they were revised following discovery of semantic errors were found during application of an automatic validation tool.[7] Such errors weren't evident in the first release, for in many cases they led to out-of-range conditions on some signals that occur only in particular conditions.

Table 1 reports, for each benchmark, the number of primary inputs (PI), primary outputs (PO), VHDL lines, and VHDL processes. The number of PIs doesn't include the clock and reset input signals existing in all the benchmark circuits. Results after synthesis are given in terms of combinational gates and flip-flops. Intended for the evaluation of purely sequential ATPGs, the circuits don't include any design-for-testability structure (such as full or partial scan), although scan may be inserted easily thanks to the simple clocking structure. To allow the generation of independent and quantitative measures about circuit testability, we generated complete and collapsed stuck-at fault lists with the "faultlist" tool of the Synopsys Testgen package version 3.0.2. The fault lists, whose size appears in Table 1, are distributed with the

Table 2. Original functions.

Circuit Name	Original Function
b01	Finite state machine (FSM) comparing serial flows
b02	FSM that recognizes binary coded decimal (BCD) numbers
b03	Resource arbiter
b04	Compute minimum and maximum
b05	Elaborate the contents of a memory
b06	Interrupt handler
b07	Count points on a straight line
b08	Find inclusions in sequences of numbers
b09	Serial-to-serial converter
b10	Voting system
b11	Scramble string with variable cipher
b12	1-player game (guess a sequence)
b13	Interface to meteo sensors
b14	Viper processor (subset)
b15	80386 processor (subset)
b16	Hard-to-initialize circuit (parametric)
b17	Three copies of b15
b18	Two copies of b14 and two of b17
b20	A copy of b14 and a modified version of b14
b21	Two copies of b14
b22	A copy of b14 and two modified versions of b14

benchmarks. The last column reports the inherent testability of the benchmarks, expressed as the fault coverage attained by the application of 10,000 pseudorandom patterns at the circuit PIs.

To help researchers understand their results with the benchmarks, Table 2 hints at the circuit function of the original VHDL description. However, while the benchmarks are syntactically correct and their simulation doesn't produce errors, their development process offers no guarantee that current VHDL descriptions are functionally meaningful.

Since the first appearance of the benchmark set, people from more than 200 institutions (estimated from the different domain names on the Web access log) have already downloaded it from our Web site, and published results are beginning to appear.[5] We encourage researchers to download[8] and use the circuits, and perhaps generate new ones with the same characteristics to increase the availability of test cases to the research community. Furthermore, if authors notify us of the publication of results concerning the benchmarks, we will maintain an ongoing, publically available reference list.

The Artist ATPG system

The goal of Artist is to implement an RT-level ATPG, a tool able to generate input sequences starting from a synthesizable RT-level description. This should attain a high fault coverage with respect to the standard stuck-at fault list when simulated on the corresponding gate-level description. The reported experimental results, together with the research results recently presented in the literature, support the claim that RT-level test sequence generation is now feasible.[9]

Although other proposals exist in the literature to attack similar problems, Artist adopts an original solution compared to previous approaches. Most previously published papers on the subject[10-12] are much less general in terms of accepted circuit descriptions and much more complex to use. Due to the approach it's based on, Artist can produce sequences for more general synthesizable VHDL description, with few limitations in size, complexity, or characteristics, and doesn't require any effort from the designer for remodeling the circuit or extracting special information from it. Artist shares some common ideas with the RAGE tool CPSR 97 (Corno, Prinettoand Sonza Reorda) but, when compared to RAGE, Artist proves much more powerful both in terms of accepted descriptions (most of the limitations existing in RAGE have been removed) and quality of the results produced.

System overview

The Artist system implements a simulation-based approach, inspired by the success of gate-level ATPG based on genetic algorithms (GAs). Some (initially random) input sequences are simulated, and their coverage characteristics are iteratively improved by analyzing the simulation trace.

The system can process structural and behavioral synthesizable VHDL descriptions at the RT level. It consists of four main components (Figure 2):

- A GA whose goal is to cultivate test sequences, improving their value under the selected metric. In this context, a test sequence is a series of vectors, to be applied at consecutive clock cycles.

- A commercial VHDL simulator that simulates the sequences computed by the GA.
- An analyzer that examines the VHDL control and data dependencies to identify basic blocks (that is, jump-free consecutive sequences of statements); to compute control and data dependencies, and correlation probabilities; and to generate the list of high-level faults to cover.
- A code instrumentation tool that modifies the original VHDL description by inserting always-false "assert" statements, which allow the GA to determine the actual sequence of executed basic blocks.

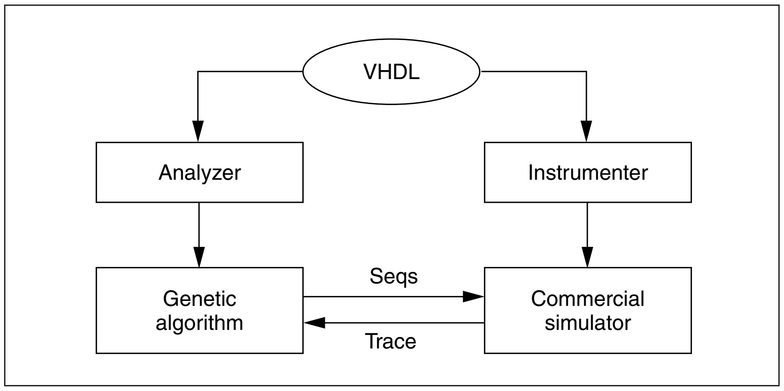

Figure 2: Artist structure

To improve sequences, the GA analyzes the trace of the executed instructions and computes a fitness function that quantifies the effectiveness of each sequence. Such a fitness function is based on the list of executed basic blocks and on the correlation probabilities among basic blocks.

Artist adopts a testability metric derived from statement coverage. We enhanced the metric for observability to lead the ATPG first to excite (execute) each basic block in the description, then to observe it (propagate to some primary output the values assigned in the block). We detail the criteria used for observability later. The Artist algorithm implements two different phases:

- In the first phase all basic blocks are considered simultaneously. The goal is to generate a set of sequences *S* that activate most of the blocks. They are used to initialize the genetic population in the second phase and aren't necessarily included in the final test set. The fitness function is the ratio of activated blocks:

$$\text{fitness}(S) = \frac{\text{activated\_blocks}(S)}{tot\_blocks}$$

- In the second phase each block is targeted separately. For each target block *T*, the goal is to generate a sequence *S* able to test it. In this phase the activated blocks are weighted by their correlation probability[9] with the target (term "+*O*" takes into account observability and is described in the next section):

$$\text{fitness}(S,T) = \sum_{b \in \text{activated\_bb}(S)} \text{correlation}(b,T) + O$$

The algorithm stops when all target basic blocks have been considered.

Observability issues

When RT-level test pattern generation targets design validation tasks, traditional branch coverage metrics are usually considered satisfactory. Even if moving to more complex metrics, such as path coverage, the main goal during design verification is just to excite all behaviors. Their observation is guaranteed, thanks to the simulation environment whereby all internal values of the design can be inspected.

However, when dealing with production testing, but also for black box verification, the effects of observability can't be neglected[13]—all activated instructions must be observed at the circuit primary outputs. In the Artist framework, the generated sequences must observe a block after having executed it. To lead the GA to this goal, we added the term "+*O*", which measures how close the sequence is to observing the target, to the fitness equation.

The exact computation of this term would require complete integration with an RT-level fault simulator. Since satisfactory commercial fault simulation solutions do not exist yet, and given the inherent overhead introduced by fault simulation of VHDL code, we approxi-

Table 3. Gate-level quality of RT-level generated sequences.

	Artist			Gate-Level ATPG		
Circuit	CPU Time	Fault Coverage %	Test Length	CPU Time	Fault Coverage %	Test Length
b01	4,118	100.00	1,061	< 1	100.00	129
b02	1,731	99.33	940	< 1	99.33	60
b03	5,131	74.33	374	5,356	74.82	245
b04	6,905	89.42	427	2,359	91.51	558
b05	33,393	33.50	2,800	51,467	33.38	223
b06	2,315	97.02	62	< 1	97.35	118
b07	2,251	57.53	461	33,415	57.28	148
b08	2,106	86.27	329	12	98.15	582
b09	9,054	81.33	1,187	3,624	90.56	967
b10	10,851	90.42	586	919	92.22	416
b11	5,092	85.98	532	24,198	81.00	228
b12	67,575	45.99	5,541	77,297	21.17	276
b13	43,450	68.37	4,538	23,625	59.19	300
b14	55,240	79.65	4,743	14,014	95.04	7,728
b15	60,990	31.96	2,733	50,822	16.26	66
b17	14,475	15.50	1,197	38,245	2.07	16
b18	278,338	1.50	279	> 500,000	0.62	10
b20	128,193	79.99	7,825	29,961	26.57	112
b21	74,845	82.61	6,376	46,202	55.14	148
b22	149,544	71.59	2,582	31,323	55.79	102

mate the observability term with a computational cost as close as possible to that of simulating the fault-free system. Fallah et al[13] provide a good example of this approach where analysis of data dependencies across conditional statements helps estimate the set of signals affected by an assignment. In Artist, we implemented an observability strategy more approximate than those proposed earlier[13,14] due to the looser integration with the simulator in our case.

Artist explicitly traces the set of variables and signals to which the target fault has been propagated by analyzing the simulation trace, with the knowledge of the data transfers performed in each basic block.

This analysis is exact, except for dependencies where a variable is assigned depending on a conditional. There we optimistically assume that variables in conditional expressions are observed on all signals and variables assigned within the conditioned blocks.

Finally, the value of the term "+*O*" is the weighted sum of the execution counts of the statements containing the variables or signals to which the target fault effects have been propagated. Weights are determined as the distance between the statement and the nearest PO. A more detailed discussion of the observability term in Artist appears elsewhere.[12]

Implementation details

We implemented the above-described method in a prototypical environment consisting of a mix of commercial and in-house developed tools. The preliminary VHDL code analysis process exploits the GraphGen option of the LEDA VHDL System (LVS) toolkit. For simulating RT-level descriptions we resort to the V-System 5.3 VHDL simulator by Model Technology. The code instrumentation process uses the reverse analysis option of the LVS toolkit. The implementation consists of about 4,700 lines of C code for VHDL code analysis and instrumentation, linked to the LEDA Procedural Interface (LPI), and of 3,500 lines of C code for the GA and the interface to the simulator.

Experimental results for ATPG

The genetic population consists of 50 individuals. In each generation 30 new sequences are generated, then selection is performed on the whole set of individuals. Individuals are selected for reproduction using their linearized fitness. In 30% of the cases, the new individual is built by mutating a single parent: the original sequence can be shortened, or enlarged, or some bits may be flipped. In 70% of the cases, the new individual is built by mating two different parents: the offspring sequence can inherit the beginning from one parent and the end from the other, or entire bit columns from each parent.

Table 3 reports the results obtained by running Artist on a Sun Ultra 5 working at 333 MHz with 256 Mbytes of memory. For easy evaluation of the generated sequences' effectiveness, we also report the results obtained by running a state-of-the-art commercial gate-level ATPG on the gate-level version of the same benchmarks. The first column reports the CPU time of the ATPG run in seconds. The second column shows the fault coverage attained at the gate level using the stuck-at fault model. When Artist is considered, this means that first the tool is run on the RT-level description, then the sequences produced are fault simulated on the corresponding gate-level description to obtain their percent stuck-at fault coverage. The third column reports the test length in clock cycles.

The results show that

- Artist generates test sequences whose gate-level fault coverage is generally comparable with that obtained by the gate-level ATPG. A few circuits exist for which the latter performs substantially better (for example, b08, b09, and b14), but for others the reverse is true (such as b11, b12, b13, b15, b17 and b20 through b22). By analyzing those benchmarks that proved critical for Artist, we found that their test requires very specific sequences hard to find using a genetic approach such as the Artist one.
- For the smallest circuits, Artist has much higher CPU time requirements than the gate-level ATPG, while for the largest benchmarks the CPU time requirements of the two tools are comparable.
- Sequences generated by Artist are longer than those generated by the gate-level ATPG. This occurs mainly because no test compaction mechanism is currently implemented, not even a basic fault dropping one.

According to these results we can conclude that, starting from RT-level descriptions, Artist generally can produce test sequences whose quality is comparable with that of the sequences generated by a state-of-the-art gate-level ATPG, thus reducing the cost of running a gate-level ATPG step.

Exploiting RT-level sequences for design validation

At the RT level, a strong link exists between test pattern generation for physical faults and pattern generation for functional validation. The fault detection metric used in Artist subsumes branch coverage, frequently used for design validation and software testing.[15] Moreover, Artist takes into account observability, which has proved crucial for effective test bench generation.[16] For these reasons, sequences generated by ARTIST can be exploited during design validation. They also prove very effective in pinpointing critical sections in the VHDL code, from the syntactical and semantic points of view.

At the RT level, a strong link exists between test pattern generation for physical faults and pattern generation for functional validation.

In particular, Artist can, when generating test patterns, identify the VHDL entities, processes, or statements that are poorly controllable or observable, and those for which it couldn't generate any test pattern. This information is vital for design validation. In many cases, while testing Artist, we found that the statements that it couldn't cover were effectively unreachable—we discovered some design errors in our benchmarks while generating test patterns. As an example, we found that a common design error corresponds to missing or incorrect bounds checking. In this case Artist found some overflow conditions (mainly sums or increments not truncated or wrapped to the allowed range of values) that escaped manual simulation. These conditions allowed us to correct the ITC'99 VHDL benchmarks, which in their previous release contained those bugs. Further details about the usage of the Artist system for design validation appear elsewhere.[7]

The Politecnico di Torino circuits belonging to the ITC'99 benchmark suite mainly support research in the area of high-level ATPG. To ease development of new test tools at the RT level, we standardized the benchmarks in terms of description styles and language. They don't contain rarely used statements or system-level structures. We adopted the VHDL language, and for every circuit the corresponding gate-level description and fault list are also available.

Exploiting these circuits, we evaluated the RT-level ATPG Artist. Experimental results on the benchmarks demonstrate that RT-level ATPG is now feasible. Designers can exploit it to evaluate the testability of their descriptions before the synthesis step, significantly improving the whole design flow.

Artist can be extended to the case of partial-scan circuits or to circuits including some BIST portion. Moreover, the proposed ATPG technique can be successfully adopted for other purposes outside the test area, such as for generating test benches for design validation or verifying properties of the design (model checking). We hope to improve the ATPG algorithm by supporting more effective fault models and by reducing the required CPU time.

■ References

1. F. Brglez, and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran", *Proc. IEEE Int'l Symp. Circuits and Systems*, IEEE Computer Soc. Press, Los Alamitos, Calif., June 1985, pp. 695-698**.**
2. F. Brglez, D. Bryan, and K. Kozminski, "Combinatorial Profiles of Sequential Benchmark Circuits," *Proc. IEEE Int'l. Symp. Circuits and Systems*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1989, pp. 1929-1934.
3. "High Time for High-Level Test Generation," *Panel at ITC99: Int'l Test Conf.*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1999, pp. 1112-1119.
4. M.B. Santos, et al, "RTL-based Functional Test Generation For High Defect Coverage in Digital SoCs," *IEEE European Test Workshop*, Cascais (P), IEEE Computer Soc. Press, Los Alamitos, Calif., May 2000, pp. 99-104.
5. "ITC'99 Benchmark Circuits?Preliminary Results," *Panel at ITC'99: Int'l Test Conf.*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1999, pp. 1125-1130.
6. F. Corno, P. Prinetto, and M. Sonza Reorda, "Testability analysis and ATPG on behavioral RT-level VHDL," *Proc. IEEE Int'l Test Conf.*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1997, pp. 753-759.
7. F. Corno, et al, "Automatic Test Bench Generation for Validation of RT-level Descriptions: an Industrial Experience," *IEEE Design, Automation and Test in Europe*, Paris (F), IEEE Computer Soc. Press, Los Alamitos, Calif., March 2000, pp. 385-389.
8. Politecnico di Torino ITC'99 benchkmarks, downloadable at the URL http://www.cad.polito.it/tools/itc99.html.
9. F. Corno, M. Sonza Reorda, and G. Squillero, "High-Level Observability for Effective High-Level ATPG, VTS-2000," *18th IEEE VLSI Test Symp.*, Montreal (CA), IEEE Computer Soc. Press, Los Alamitos, Calif., May 2000, pp. 411-416
10. D. Moundanos, J.A. Abraham, and Y.V. Hoskote, "A Unified Framework for Design Validation and Manifacturing Test," *Proc. IEEE Int'l Test Conf.*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1996, pp. 875-884.
11. K.-T. Cheng, and A.S. Khrishnakumar, "Automatic Generation of Functional Vectors Using the Extended Finite State Machine Model," *ACM Transactions on Design Automation of Electronic Systems*, Vol. 1, No. 1, Jan. 1996, pp. 57-79.
12. F. Ferrandi, F. Fummi, and D. Sciuto, "Implicit Test Generation for Behavioral VHDL Models," *Proc. IEEE Int'l. Test Conf.*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1998, pp. 436-441.
13. F. Fallah, P. Ashar, and S. Devadas, "Simulation Vector Generation from HDL Descriptions for Observability-Enhanced Statement Coverage," *Proc. 35th Design Automation Conf.*, ACM press, 1999, pp. 666-671.
14. S. Devadas, A. Ghosh, and K. Keutzer, "An Observability-Based Code Coverage Metric for Functional Simulation," *Proc. IEEE/ACM Int'l Conf. on Computer Aided Design*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1996, pp. 418-425.
15. B. Beizer, *Software Testing Techniques (second edition)*, Van Nostrand Rheinold, New York, 1990.
16. P.A. Thaker, V.D. Agrawal, and M.E. Zaghloul, "Validation Vector Grade (VVG): A New Coverage Metric fo Validation and Test," *Proc. 15th IEEE VLSI Test Symp.*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1999, pp. 182-188.

Fulvio Corno is an assistant professor at the Politecnico di Torino. His research interests include CAD for VLSI design, test of digital systems, high-level design, genetic algorithms, and symbolic techniques. He received his MS degree in electronic engineering from the Politecnico di Torino, Italy, in 1991 and his PhD degree in computer science from the same institution in 1994.

Matteo Sonza Reorda is an associate professor in the Department of Computer Science and Automation at Politecnico di Torino, Italy. His research interests include automatic test pattern generation, built-in self-test, and fault tolerant design. He received his MS in electronics and PhD in computer science from the Politecnico di Torino, Italy, in 1986 and 1990, respectively.

Giovanni Squillero is working on his PhD on approximate techniques for test and verification of digital systems described at a high level at the Politecnico di Torino, Italy. His research interests also include evolutionary algorithms. He. received his MS degree in computer science engineering in 1996 from the Politecnico di Torino.

■ Send questions and comments to Fulvio Corno, Politecnico di Torino, Dipartimento di Automatica e Informatica, Corso Duca degli Abruzzi 24, 10129 Torino, Italy; e-mail fulvio.corno@polito.it.