RT-level ITC'99 benchmarks and first ATPG results

Fulvio Corno, Matteo Sonza Reorda, Giovanni Squillero

Politecnico di Torino

Dipartimento di Automatica e Informatica

Corso Duca degli Abruzzi 24

10129 Torino, Italy

http://www.cad.polito.it/

Abstract

Effective high-level ATPG tools are increasingly needed, as an essential element in the quest for reducing as much as possible the designer work on gate-level descriptions. We propose a new set of benchmark circuits targeted to researchers working in the area of RT-level automatic test sequence generation. The developed benchmarks share the characteristics of typical synthesizable blocks, are available as both RTL VHDL descriptions and gate level netlists, and allow the evaluation of the quality of test sequences generated from RT-level descriptions in terms of attained coverage of gate-level stuck-at faults. Exploiting these benchmarks, we analyzed the effectiveness of a prototypical ATPG tool (called ARTIST) suitable to generate test sequences starting from synthesizable RT-level VHDL descriptions. ARTIST overcomes several limitations inherent with previously proposed approaches, especially in terms of accepted descriptions and level of automation. Also, ARTIST was extremely useful in eliminating some bugs in the benchmarks. The results gathered on the new benchmark suite show that this ATPG was able to generate sequences whose quality is comparable with those generated by a state-of-the-art gate-level ATPG, thus showing the feasibility of RT-level test pattern generation. RT-level ATPGs will make it feasible for designers to evaluate the testability of their circuits before the synthesis step is performed, and to reduce the cost of the gate-level ATPG step.

1. Introduction

One of the crucial parameters for speeding-up and making more effective the evolution process in any technical research area is the availability of suitable and meaningful benchmarks, since they allow an easier and unbiased evaluation of new

ideas, accelerating the process towards the selection of those proposals that can most effectively be adopted in the industrial practice.

In the area of digital circuit testing, ISCAS'85 [BrFu85] and ISCAS'89 [BBKo89] gate-level benchmarks were introduced to evaluate combinational and sequential ATPG tools more than a decade ago. Afterwards, despite their initial purpose, they have been used for almost all applications in the field of test, and also for assessing the effectiveness of new methods in other areas, including logic optimization, power estimation, and partitioning. Today, this kind of benchmarks are still essential to develop, test and improve CAD algorithms and tools, but are gradually losing their importance, due to the introduction of new design techniques. For this reason, new benchmarks need to be developed and distributed, that more closely reflect current design needs.

Current and future CAD tools must face a new generation of systems-on-chip, whose complexity increases in several directions, such as:

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

Benchmark circuits donated by industrial entities contain all these issues, but are largely dissimilar in terms of description levels and styles, adopted libraries, test strategy, etc., thus creating difficulties for researchers to develop new techniques and test them over a series of circuit instances. While current design tools cannot ignore all the complexity aspects, for developing core algorithms a simpler set of benchmarks is more useful. ATPG algorithms, in particular, as it already happened for gate-level circuits, are easier to develop, optimize and improve when first applied to simplified circuits (i.e., 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 the new generation of test tools able to deal with RT-level descriptions (the last issue in the above list) represents a major conceptual problem, most of the former issues can be considered separately or added at a later stage. This paper therefore concentrates on benchmarks described at the RT-level and ignores the other complexity sources.

The need for test tools working on high-level descriptions has been reported since more than two decades ago, and it is now an increasingly accepted idea that the availability of effective high-level test tools would be very beneficial for improving the design process [ITC99a]. The main resistance to this concept comes from the unproven belief that the lack of structural information makes it impossible to generate effective test sequences starting from high-level descriptions. It is true that some fault effects can only be modeled at the lower abstraction levels, but this may be balanced by the advantages coming to the high-level ATPG process from working on more compact descriptions, containing also information about the functional behavior of the composing modules. Some structural faults may be missed, but a larger set of system configurations is more readily accessible [SGTT00]. Whether automatic test generation can effectively be performed on RT-level descriptions is therefore an open question, which strongly needs a widely accepted suite of neutral but representative benchmarks. Since the common practice in industrial testing is still to evaluate the goodness of generated sequences on gate-level descriptions and fault lists, the availability of both the RT- and the corresponding synthesized gate-level descriptions is crucial for the success of this suite.

These considerations motivated us to develop a set of RT-level benchmarks and to contribute it to the ITC'99 benchmark set [ITC99b]. 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. The goal of this effort was to prove that test sequences generated by RT-level ATPGs can 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 be useful to designers mainly because they allow to identify hardly testable circuits (or circuit components) early in the design flow (i.e., before

the logic synthesis step), while this check is now performed at the gate-level only, thus requiring a design re-cycle in case 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 (e.g., full- or partial-scan).

The main portion of this paper introduces the suite of RT-level benchmarks we developed, providing general information about their characteristics and about the standards we adopted for their description. As a second contribution, this paper outlines the RT-level ATPG we developed, called ARTIST, and reports a first set of experimental data, that have been gathered using its prototypical implementation; these results can represent a reference point for those working in the area of high-level ATPG algorithms, and should contribute to demonstrate that RT-level ATPG and design validation are feasible.

The paper is organized in the following way: Section 2 describes the Politecnico di Torino benchmarks, and reports information about both the circuits and the process followed for their construction. Section 3 first outlines the ARTIST algorithm and the environment implementing it; then it provides experimental results assessing its effectiveness on the previously introduced benchmarks. Section 3 also describes the use we made of the sequences generated by ARTIST for debugging the benchmarks themselves. Section 4 finally draws some conclusions.

2. Benchmark circuits development

Among the ITC'99 benchmark suite, 22 circuits were contributed by our team at Politecnico di Torino; these circuits are aimed at researchers developing algorithms working on RT-level circuits described in VHDL, and 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 can be automatically synthesized as a whole with current tools, and share the following properties:

- Circuits are described in synthesizable[1] VHDL at the RT-level. No VHDL packages are used, except IEEE *standard logic* and *arithmetic* ones. The code is in prevalence 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 EDIF format and in ISCAS'89 bench format, and have been obtained via logic synthesis over a library compatible with ISCAS'89 elementary gates.
- Fault lists (complete and collapsed for single stuck-at faults) are available, generated by an industry-standard tool and in an easily readable format.
- Circuits behave in a purely synchronous way: only one single-phase clock signal is present, that 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 is available, that allows a trivial initialization sequence to be valid for all benchmarks. Differently 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 behavioral statements does not allow to propagate undefined values as one would expect while maintaining synthesizability), so the presence of a global reset signal acts just as a 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, from 46 to 68,752 combinational gates. One of the circuits (b16) is parametric and can be synthesized to different sizes. The largest

[1] when different synthesizers required different styles, we followed Synopsys Design Compiler description styles. However, we explicitly avoided any compiler specific directive.

circuit (b18), with its 68K gates and 430K faults (190K collapsed) is much larger that the largest ISCAS'89 benchmark.

The development process of the benchmarks is shown in Fig. 1. We started from a set of VHDL descriptions *collected* from various sources, and we *modified* them to comply with our desired characteristics. The initial VHDL files were taken from public archives over the Internet and from class works developed by students at our institution. Larger benchmarks were obtained by structurally combining smaller ones, or slight variations of smaller ones. Modifications aimed at guaranteeing sincronicity and at increasing syntactical uniformity, and 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, the original behavior has been sacrificed in favor of uniformity of description. The circuits have finally been synthesized with Synopsys Design Compiler v. 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, and first results were published in [CPSo97]. However, in early 1999 they have been revised, since some semantic errors were found thanks to the application of an automatic validation tool [CMPS00]; such errors were not evident in the first release, since in many cases they led to out of range conditions on some signals, that occurred only on very particular conditions. More details about the use of a high level ATPG for design validation are given in Section 3.5.

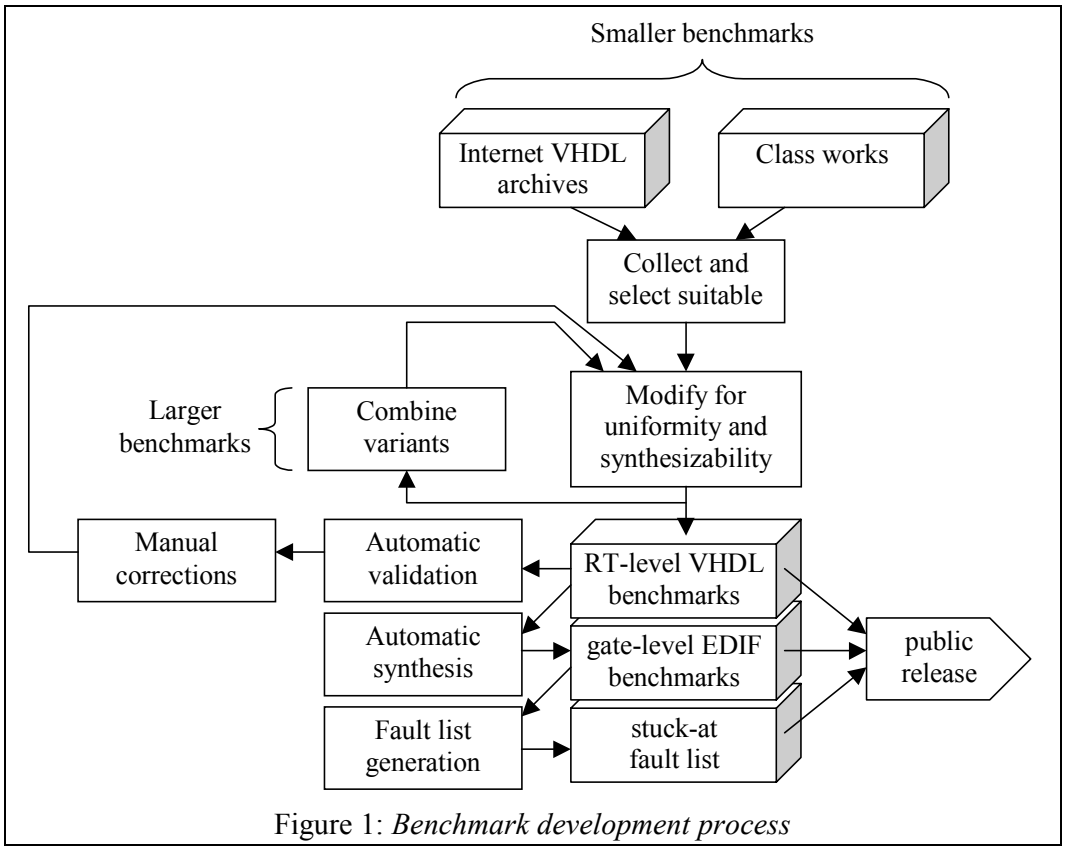

Figure 1: *Benchmark development process*

Tab. 1 reports, for each benchmark, the number of primary inputs (PI), primary outputs (PO), VHDL lines, and VHDL processes. The number of PIs do not 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. Being intended for the evaluation of purely sequential ATPGs, the circuits do not include any Design for Testability structure (e.g., full- or partial-scan), although scan can be easily inserted 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 v. 3.0.2. The fault lists, whose size is also indicated in Tab. 1, are distributed with the benchmarks. The last column finally reports some indication about the inherent testability of the benchmarks, expressed as the average fault coverage attained by the application of 10 sequences of 10,000 pseudo random patterns each at the circuit primary inputs.

Circuit			VHDL		Gate-level		Fault list		Random
name	**PI**	**PO**	**#line**	**#proc**	**#gate**	**#FF**	**complete**	**collapsed**	**testability**
b01	2	2	110	1	46	5	258	127	98.95
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	85.88
b05	1	36	332	3	935	34	5,552	2,553	33.44
b06	2	6	128	1	60	9	302	151	95.09
b07	1	8	92	1	420	49	2,404	1,120	58.28
b08	9	4	89	1	167	21	918	439	98.17
b09	1	1	103	1	159	28	900	417	87.49
b10	11	6	167	1	189	17	1,054	468	90.59
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	21.21
b13	10	10	296	5	339	53	1818	835	56.55
b14	32	54	509	1	4,775	245	28,990	12,643	81.25
b15	36	70	671	3	8,893	449	55,568	23,316	13.69
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	8.83
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

Table 1: *Benchmark characteristics*

To help researchers understand their results with the benchmarks, Tab. 2 shows a hint about the circuit function of the original VHDL description. However, while the benchmarks are syntactically correct and their simulation does not produce any error, due to their development process, and there is no guarantee that current VHDL descriptions are functionally meaningful.

Since their first appearance, people from over 170 institutions[2] already downloaded the benchmarks set from our web site, and some published results are beginning to appear [ITC99b]. We encourage researchers to download [Benc99] and use the circuits, and possibly to generate new ones with the same characteristics in order to increase the availability of test cases to the research community. Furthermore, if authors notify us of the publication of some result concerning the benchmarks, we will maintain and publish a list of pointers to such references.

[2] estimated from the different domain names on the web access log

Circuit name	Original function
b01	Finite state machine (FSM) comparing serial flows
b02	FSM that recognizes BCD numbers
b03	Resource arbiter
b04	Compute min and max
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 2 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

Table 2: *Original functions*

3. The ARTIST ATPG system

In this Section we briefly overview the ARTIST (*Automatic RT-level Input Sequence generator for Test purposes*) test pattern generation system. The goal of ARTIST is to implement an RT-level ATPG, i.e., a tool able to generate input sequences, starting from a synthesizable RT-level description, that attain a high fault coverage with respect to the standard stuck-at fault list when fault 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.

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 (see for example [MAHo96] [ChKr96] [FFSc98]) are

much less general in terms of accepted circuit descriptions, and much more complex to use. Due to the approach it is based on, ARTIST can produce sequences for *more general* synthesizable VHDL description, with few limitations in size, complexity, or characteristics, and does not require any effort to the designer for re-modeling the circuit or extracting special information from it. ARTIST shares some common ideas with the RAGE tool [CPSR97], but, when compared to RAGE, ARTIST is much more powerful both in terms of accepted descriptions (most of the limitations existing in RAGE have been removed) and of quality of the produced results. The reader interested in further details about the ARTIST algorithm can refer to [CSSq00]

3.1. System Overview

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

The system is able to process structural and behavioral synthesizable VHDL descriptions at the RT-level and is composed of four main components (Figure 2):

- a *Genetic Algorithm* (GA) whose goal is to cultivate test sequences, improving their value under the selected metric. In this context, a *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* (i.e., jump-free consecutive sequences of statements), to compute control and data dependencies, 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, that allow the GA to determine the actual sequence of executed basic blocks.

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

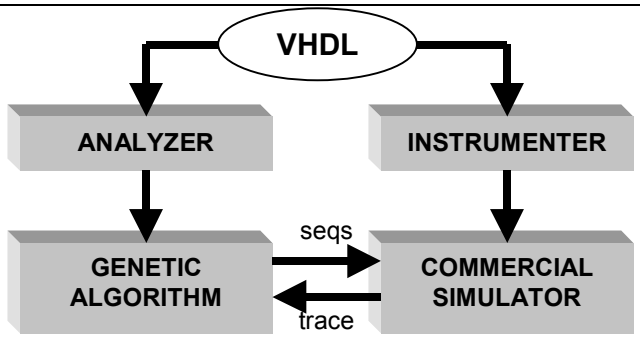

Figure 2: *ARTIST structure*

ARTIST adopts a testability metric derived from statement coverage. The metric was enhanced for observability to lead the ATPG first to *excite* (i.e., execute) each *basic block* in the description, then to *observe* it (i.e., propagate to some primary output the values assigned in the block). The criteria used for observability are detailed in Section 3.2. 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 are not 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 with the target (term "+*O*" takes into account observability and is described in Section 3.2):

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

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

3.2. Observability issues

When RT-level test pattern generation is targeted at 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, since their observation is guaranteed thanks to the simulation environment, where the designer can inspect all internal values of the design.

However, when dealing with production testing, but also for black box verification, the effects of observability can not be neglected [DGKe96]: 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 add to the fitness equation the term "+*O*," that measures how close the sequence is to observe the target.

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 due to the inherent overhead introduced by fault simulation of VHDL code, we approximate the observability term with a computational cost as close as possible to that of simulating the fault-free system. A good example of this approach is in [FADe99], where analysis of data dependencies across conditional statements helps estimating the set of signals affected by an assignment. In ARTIST, we implemented an observability strategy more approximate than the one proposed in [DGKe96] and [FADe99] 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, where 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 variables or signals to which the target fault effects have been propagated.

Weights are determined as the distance between the statement and the nearest primary output. A more detailed discussion about the observability term in ARTIST can be found in [CSSq00].

3.3. Implementation Details

The above described method has been implemented in a prototypical environment which consists of a mix of commercial and in-house developed tools.

The preliminary VHDL code analysis process exploits the GraphGen option of the LEDA 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 LPI interface, and of 3,500 lines of C code for the Genetic Algorithm and the interface to the simulator.

3.4. Experimental Results for ATPG

In this sub-section we report experimental data gathered by running ARTIST on the benchmarks and by evaluating the ability of the generated sequences in covering the stuck-at faults on the corresponding gate-level descriptions.

The parameters of the Genetic Algorithm in both phases were set as follows. The genetic population is composed of 50 individuals and in each generation 30 new sequences are first 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 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 mating two different parents: the offspring sequence can inherit the beginning from one parent and the end from the other, or some entire bit column from each parent.

Table 3 reports the results obtained by running ARTIST on a Sun Ultra 5 working at 333 MHz with 256MB memory. For allowing an easy evaluation of the

effectiveness of the generated sequences, we reported in the table also 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. The second column shows the attained fault coverage 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 produced sequences are fault simulated on the corresponding gate-level description to obtain their percent stuck-at fault coverage. The third column reports the length of the test 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; there are a few circuits for which the latter performs substantially better (e.g., b08, b09, and b14), but there are others for which the reverse is true (e.g., b11, b12, b13, b15, b17 and b20 through b22). By analyzing the benchmarks which proved to be critical for ARTIST, we found that their test requires very specific sequences, that can hardly be found 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.
- The length of the sequences generated by ARTIST is higher than those generated by the gate-level ATPG. This is mainly due to the fact that 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 is generally able to 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.

Circ	ARTIST			Gate-level ATPG		
	CPU	FC%	LEN	CPU	FC%	LEN
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

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

3.5. Exploiting RT-level sequences for design validation

At the RT-level, there is a strong link between test pattern generation for physical faults and pattern generation for functional validation. The fault detection metric used in ARTIST subsumes branch coverage, that is frequently used for design validation and software testing [Beiz90]. Moreover, ARTIST takes into account observability, which has been proved to be crucial for effective test bench generation [TAZa99]. For these reasons, sequences generated by ARTIST can fruitfully be exploited during design validation, and they proved to be very effective in pinpointing critical sections in the VHDL code, from the syntactical and semantic points of view.

In particular, ARTIST is able, when generating test patterns, to identify the VHDL entities, processes, or statements that are poorly controllable or observable, and those for which it was not able to generate any test pattern: this information is vital for design validation. In many cases, while testing ARTIST, we found that the statements that it was not able to 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 that were not truncated nor wrapped to the allowed range of values) that escaped manual simulation. These conditions allowed us to correct the ITC'99 VHDL benchmarks, that in their previous release contained those bugs. Further details about the usage of the ARTIST system for design validation can be found in [CMPS00].

4. Conclusions

We presented the Politecnico di Torino circuits belonging to the ITC'99 benchmark suite. These circuits are mainly intended to support the research in the area of high-level ATPG, and correspond to synthesizable RT-level descriptions of different size, complexity, and type. For ease of development of new test tools at the RT-level, the benchmarks have been standardized in terms of description styles and language, and do not contain rarely used statements nor system-level structures. The VHDL language has been adopted, and for every circuit the corresponding gate-level description and fault list are also available.

Exploiting these circuits, we evaluated a new RT-level ATPG named ARTIST, based on a Genetic Algorithm interacting with a commercial VHDL simulator. Experimental results on the benchmarks show that the generated sequences reach a percent coverage of the gate-level stuck-at faults comparable with that obtained by a state-of-the-art gate-level ATPG. This means that RT-level ATPG is now feasible, and designers can exploit it to evaluate the testability of their descriptions before the synthesis step is performed, thus significantly improving the effectiveness of the whole design flow.

Although ARTIST has been developed under the hypothesis that no Design for Testability technique is adopted (i.e., a purely sequential ATPG approach is followed) the method can be fruitfully 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, e.g., for generating test benches for design validation or for verifying properties of the design (model checking).

5. References

[BBKo89] F.Brglez, D.Bryan, K.Kozminski: *Combinatorial Profiles of Sequential Benchmark Circuits*, Proc. IEEE Int. Symposium on Circuits and Systems, 1989, pp. 1929-1934

[Beiz90] B. Beizer, *Software Testing Techniques* (2nd ed.), Van Nostrand Rheinold, New York, 1990

[Benc99] Politecnico di Torino ITC'99 benchkmarks, downloadable at the URL http://www.cad.polito.it/tools/itc99.html

[BrFu85] F. Brglez, H. Fujiwara: *A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran*, Proc. IEEE Int. Symposium on Circuits And Systems, June 1985

[ChKr96] K.-T. Cheng, A.S. Khrishnakumar, "Automatic Generation of Functional Vectors Using the Extended Finite State Machine Model," ACM Trans. On Design Automation of Electronic Systems, Vol. 1, No. 1, Jan. 1996, pp. 57-79

[CPSo97] F. Corno, P. Prinetto, M. Sonza Reorda, "Testability analysis and ATPG on behavioral RT-level VHDL," *Proceedings IEEE International Test Conference*, 1997, pp. 753-759

[CMPS00] F. Corno, A. Manzone, A. Pincetti, M. Sonza Reorda, G. Squillero, *Automatic Test Bench Generation for Validation of RT-level Descriptions: an Industrial Experience*, DATE-2000: IEEE Design, Automation and Test in Europe, Paris (F), March 2000, pp. 385-389

[CSSq00] F. Corno, M. Sonza Reorda, G. Squillero, *High-Level Observability for Effective High-Level ATPG*, VTS-2000: 18th IEEE VLSI Test Symposium, Montreal (CA), May 2000, pp. 411-416

[DGKe96] S. Devadas, A. Ghosh, K. Keutzer, "An Observability-Based Code Coverage Metric for Functional Simulation," *Proceedings IEEE/ACM International Conference on Computer Aided Design*, 1996

[FADe99] F. Fallah, P. Ashar, S. Devadas, "Simulation Vector Generation from HDL Descriptions for Observability-Enhanced Statement Coverage," *Proceedings 35th Design Automation Conference*, 1999, pp. 666-671

[FFSc98] F. Ferrandi, F. Fummi, D. Sciuto, "Implicit Test Generation for Behavioral VHDL Models," *Proceedings IEEE International. Test Conference*, 1998

[ITC99a] *High Time for High-Level Test Generation*, Panel at ITC99: International Test Conference, 1999, pp. 1112-1119

[ITC99b] *ITC'99 Benchmark Circuits - Preliminary Results*, Panel at ITC99: International Test Conference, 1999, pp. 1125-1130

[MAHo96] D. Moundanos, J. A. Abraham, Y. V. Hoskote, "A Unified Framework for Design Validation and Manifacturing Test," *Proceedings IEEE International Test Conference*, 1996, pp. 875-884

[SGTT00] M. B. Santos, F. M. Gonçalves, I. C. Teixeira, J. P. Teixeira, *RTL-based Functional Test Generation For High Defect Coverage in Digital SoCs*, IEEE European Test Workshop, Cascais (P), May 2000

[TAZa99] P. A. Thaker, V. D. Agrawal, M. E. Zaghloul, "Validation Vector Grade (VVG): A New Coverage Metric fo Validation and Test," *Proceedings 15th IEEE VLSI Test Symposium*, 1999, pp. 182-188