Proc. Estonian Acad. Sci. Engng., 1997, **3**, 4, 271–282
https://doi.org/10.3176/eng.1997.4.04

# MIXED-LEVEL TEST GENERATOR FOR DIGITAL SYSTEMS[1]

Marina BRIK, Gert JERVAN, Antti MARKUS, Priidu PAOMETS, Jaan RAIK, and Raimund UBAR

Department of Computer Engineering, Tallinn Technical University, Ehitajate tee 5, EE-0026 Tallinn, Estonia; e-mail: raiub@pld.ttu.ee

Received 30 May 1997, in revised form 9 October 1997

**Abstract.** This paper presents an automated mixed-level test pattern generator operating on register-transfer level and gate-level representations of digital systems. The system implements a novel test generation approach based on a uniform diagnostic model of a digital system for mixed-level representations in the form of alternative graphs (AGs). The model allows for defining a general fault model which uniformly covers high-level functional and low-level stuck-at fault models. The AG representation enables the application of standardized procedures for fault activation, fault propagation and line justification on both levels. The system consists of separate test generators for datapath and control parts, based on similar fault propagating procedures, which differ only in building general test plans. Experimental results show the efficiency of the approach used in the generator.

**Key words:** automatic test pattern generation, hierarchical test generation, mixed-level design representations, register-transfer level (RTL) circuits, gate-level circuits, binary decision diagrams (BDDs), alternative graphs (AGs), stuck-at fault model.

## 1. INTRODUCTION

Test generation for real life digital circuits on the gate-level is extremely complex. It has been shown that test generation for combinational circuits is a NP-complete problem [1]. However, algorithms developed so far can handle test generation for relatively large combinational circuits in a reasonable computing

[1] The research has been partially funded by the joint project COPERNICUS JEP 9624 (FUTEG), funded by European Community, Estonian Science Foundation grant 1850 and German-Estonian project EST 008-96 funded by BMFT Germany.

time [1,2]. Gate-level test generation for sequential circuits is even more complex and still remains unsolved in practice [3].

Test synthesis for digital systems encompasses three activities: selecting a description method, developing a fault model and generating tests to detect the faults derived from the fault model. The efficiency of test generation (quality, speed) is highly dependent on the description method and on the fault models chosen. The complexity of digital systems is increasing. As a result, the gate level test generation methods have become obsolete. Promising approaches are hierarchical methods which use mixed-level descriptions of systems. However, the drawback lies in the need of different languages and models for different levels.

As a possible solution for the problem, hierarchical test generation methods have evolved [4–11], which take advantage of a higher abstraction level (e.g. behavioural or RTL) information while generating tests for gate-level faults. Hierarchical test generation is based on the divide and conquer principles. A device under test is considered at different design abstraction levels, and test generation is performed on these levels by utilizing an appropriate test generation tool. In hierarchical testing, top-down and bottom-up strategies are known. In the *bottom-up approach*, tests generated at the lower level will be later assembled at the higher abstraction level [7]. The generality of this approach lies in the possibility of using *precalculated library tests* for components during the higher level test synthesis. On the other hand, high-level constraints cannot be considered in creating libraries of test patterns for components. This can be a reason why solutions cannot be found even if they exist. This paper represents a method which allows for implementing both, bottom-up and top-down approaches. By the top-down approach, the constraints extracted at a higher level [12] will be considered when deriving tests for components at the lower level. As the main theoretical basis for creating the test generator, AGs are used.

During the last ten years, *binary decision diagrams* [13,14] have spread in digital design and test. AGs were introduced as a generalization of BDDs to represent functions, structural properties and faults of complex digital systems [15–17]. AGs serve as a basis for a general theory of test design for mixed-level representations of systems, similarly to the Boolean algebra for the plain logical level. The fault model defined on AGs represents a generalization of the gate-level stuck-at fault model – the latter was defined for literals in Boolean expressions whereas the former is defined for nodes in graphs.

In the hierarchical test generation approach discussed in this paper, different design abstraction levels (RTL and gate-level) of the system under test are represented by AGs. This feature provides a uniform model representation and an application of standardized procedures for all levels of abstraction. As different from known methods, both control path and data path are handled here by uniform fault models and uniform fault activation and fault propagation methods. Uniform approaches for gate-level and higher level descriptions easily afford to adopt gate-level fault propagation and line justification methods, algorithms and tools for higher level ones.

In this paper, section 2 defines the concept of AG, section 3 explains the generality of the fault model defined on AGs, section 4 describes the structure of the test generation system, and in section 5, experimental results are discussed.

## 2. ALTERNATIVE GRAPHS

In general, an AG is defined as a non-cyclic directed graph whose nodes are labelled by variables. There are different finite sets of values the variables can take. In special cases, variables can also be substituted by constants or by algebraic expressions. The graph has only one starting node (root). The number of terminal nodes, i.e. nodes without any successor nodes, is not limited. For each value from a set of predefined possible values of a non-terminal node variable (or expression), there exists one and only one output branch from the node. Different branches of the same node may lead into the same successor node.

Consider a situation where all node variables are fixed to some value. By these values, for each non-terminal node, a certain output branch will be chosen which enters its corresponding successor node. Let us call such connections between nodes *activated branches* under the given values. Succeeding each other, activated branches form, in turn, *activated paths*. For each combination of values for all node variables, there exists always a corresponding activated path from the starting node to a terminal node. Let us call this path the *main activated path*. For each combination of values for all node variables, there exists one and only one value equal to the value of the variable (or expression) at the terminal node of the main activated path. This relationship describes a mapping from a Cartesian product of the sets of values for variables to all nodes to the joint set of values for variables in the terminal ones. Therefore, by AGs it is possible to represent arbitrary digital functions $Y = F(x)$, where $Y$ is the variable whose value will be calculated on the AG, and $x$ is the vector of all variables which belong to the labels of the nodes in the AG.

When using AGs to describe complex digital systems, at the first step, we have to represent the system by a suitable set of interconnected components (combinational or sequential ones). On the second step, we have to describe these components by their corresponding functions which can be represented by AGs. Examples of representing the RTL description of a digital system by AGs are depicted in Figs.1 and 2.

The finite state machine (FSM) of the control part is described by an AG where non-terminal nodes represent the current state and inputs for the control part, and terminal nodes are for representing the next state and control signals leading to the datapath. Figure 1 shows an example of a fragment of a FSM state table and the corresponding AG representation. In this example, $q$ denotes the next state variable and $q'$ is the current state. The variables out1, out2, out3 and out4 denote the output variables of the FSM. The graph in Fig.1 describes the behaviour of the FSM at the current state s5.

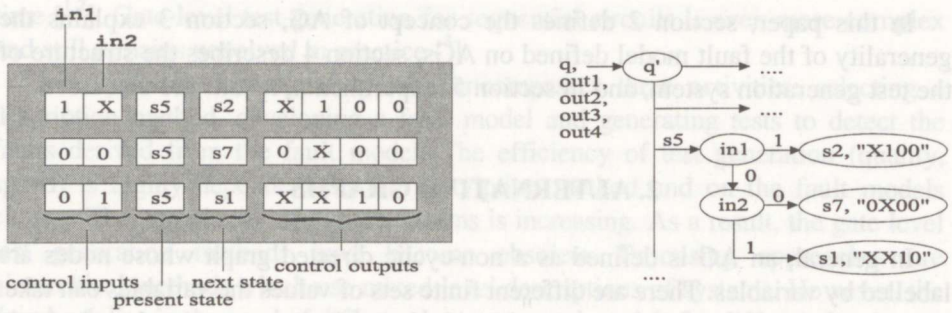


Fig. 1. Represention of a control part of a digital system by an alternative graph.

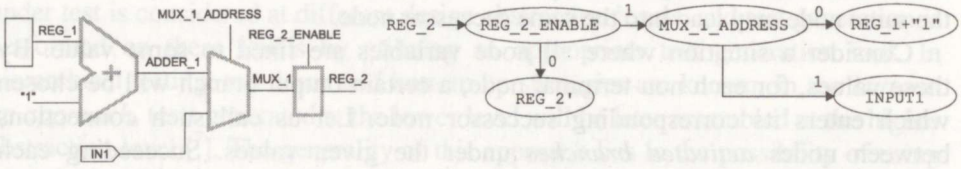


Fig. 2. Represention of a data part of a digital system by an alternative graph.

The datapath is described as a set of AGs. An AG corresponds to each register and each signal fanout. Here, non-terminal nodes represent the control signals coming from the control part or inputs and terminal nodes represent signals of the datapath, i.e. primary inputs, registers, constants or data manipulation expressions. Figure 2 shows a datapath fragment and its corresponding AG model.

## 3. FAULT MODEL

The main idea of the fault model defined for AGs is the relationship of faults to nodes of graphs, in a similar way as logical level faults stuck-at-1 and stuck-at-0 are related to Boolean variables (or literals) in the corresponding logical expressions. In the simplest case, at the logical level, each node in AGs is labelled by a Boolean variable and, therefore, is related to the stuck-at faults s-a-1 and s-a-0 of this variable. This relationship between nodes in AGs and faults in circuits can be generalized in comparison to the Boolean level.

Each path in an AG describes the behaviour of the system in a specific mode of operation. The faults, having effect on the behaviour, can be associated with the nodes along the given path. A fault causes an incorrect leaving the path activated by a test. From this point of view, we introduce the following abstract fault model for the nodes $m$ with node variables $z(m)$ in AG-representations of digital systems:

1. The output branch at $z(m) = i$ of a node $m$ is always activated; notation: $z(m)/\varnothing \Rightarrow i$.

2. The branch at $z(m) = i$ of a node $m$ is broken; notation: $z(m)/i \Rightarrow \varnothing$.

3. Instead of the given branch at $z(m) = i$ of a node $m$, another branch at $z(m) = j$ or a set of branches $\{j\}$ is activated; notation: $z(m)/i \Rightarrow \{j\}$.

The fault model defined on AGs is directly related to nodes and gives a clear description of the faulty behaviour of the system in terms of the model. In this fault model, first, the situation how an AG is activated is described (a value of the node variable or node function is given), and, secondly, the new faulty activated situation is discussed.

Different fault models for different representation levels of digital systems can be covered by this uniform node fault model defined on AGs. The physical meaning of the faults associated with a particular node depends on the meaning of the node. Depending on the adequacy of representation of the structure of the system, the fault model proposed for AGs can cover a wide class of structural and functional faults introduced for digital circuits and systems [19,20]. For example, the fault model for the nodes labelled by Boolean variables $z \in \{0,1\}$ covers the stuck-at fault model $z/0$ ($z/1$) for gate-level circuits, the fault model for the nodes labelled by integer variables represents widespread functional fault models for decoders, multiplexers, instruction decoding units of microprocessors [19], case constructions in procedural models of systems [11,20], etc.

Denote each fault $F_i$ in a digital system represented by AGs as a quadruple: $F_i = (G, N, C, V)$, where $G$ is the graph where the fault is defined, $N$ is the node affected by the fault, $C$ is a condition needed to activate the fault (not always formally derivable from the AG-model itself), and $V$ is the action of the fault, i.e. the identification of the faulty activation of the outputs of the node. For example, the faults in a graph $G$ at a node $m$, defined above, will have the following general notation:

$$z(m)/\varnothing \Rightarrow i: \quad (G, m, \{z(m) = \varnothing\}, i);$$
$$z(m)/i \Rightarrow \varnothing: \quad (G, m, \{z(m) = i\}, \varnothing);$$
$$z(m)/i \Rightarrow \{j\}: \quad (G, m, \{z(m) = i\}, \{j\}).$$

In Fig. 3, examples of some typical faults in a digital circuit and in the corresponding AG-model are depicted:

1) $F1$: (OUT, 5, $x_1 = 0$, 1) – stuck-at fault at a line $x_1$ in the circuit.

2) $F2$: (OUT, 8, $\{x_3 = 1, x_2 = 0\}$, 0), or (OUT, 6, $\{x_3 = 0, x_2 = 1\}$, 0) – bridging (or crosstalk) fault between leads $x_2$ and $x_3$; the condition $C$ (the values of $x_3$ and $x_2$) is needed to reveal the bridge by observing the signal value on one of the two leads involved in the fault.

3) $F3$: (OUT, 2, $I = 2,\{2, 3\}$) – functional fault in the decoder (instead of the active output 2, two outputs 2 and 3 are simultaneously active); this fault can be caused, for example, by a bridge or crosstalk between the leads 2 and 3 at the output of the decoder. For microprocessors, a similar class of functional faults

has been introduced for functions like instruction decoding, source and destination register decoding in [19].

4) $F4$: (OUT, 3, $\{R = 76, \text{IN} = 4\}$,*) – another type of functional faults is considered at the high-level primitive block $F$ (e.g. ALU) that is related to the hierarchical approach in test generation; the condition $\{R = 76, \text{IN} = 4\}$ represents a local test pattern for $F$, generated at the lower level. Asterisk * in the place of $V$ means that the faulty value in the output of $F$ is not determined.

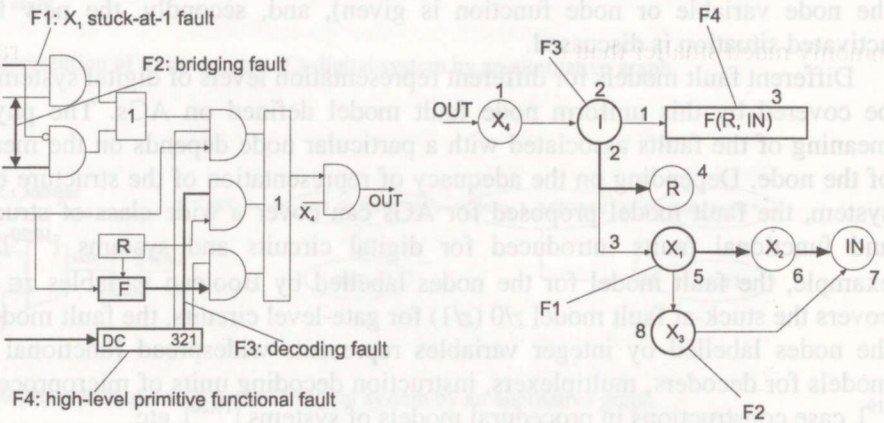


Fig. 3. Represention of faults of digital circuits in alternative graphs.

From above, it follows that the fault model defined on AGs can be regarded as a generalization of the gate-level stuck-at fault model for higher level representations of digital systems. The stuck-at fault model is defined for Boolean variables (literals), whereas the generalized new fault model is defined for nodes of AGs. As nodes with Boolean labels represent only a special class of nodes in AGs, the logical level stuck-at fault model also represents special class of faults in AGs only. In [21], a general notation of the fault model is introduced for representing a wide class of faults in digital systems like stuck-at, bridging, crosstalking, and general functional faults.

## 4. STRUCTURE OF THE TEST GENERATOR

Input data for the test generator is a register-transfer level VHDL description of the system. Such representations are provided by various high-level synthesis tools where behavioral descriptions are compiled into RTL ones. Figure 4 shows the place of the test generator in the design process.

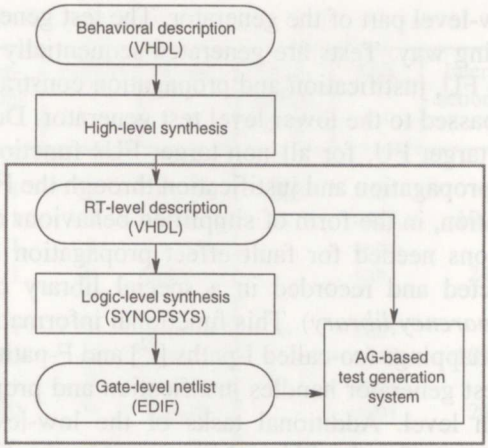


Fig. 4. The test system in a design process.

Figure 5 shows the general structure of the test generation environment. It consists of a hierarchical datapath test generator, a control part test generator, and a high-level AG model generator. From RT-level VHDL description, the high-level AG generator generates high-level AG model, which will be applied as an input of the datapath and control part test generators.

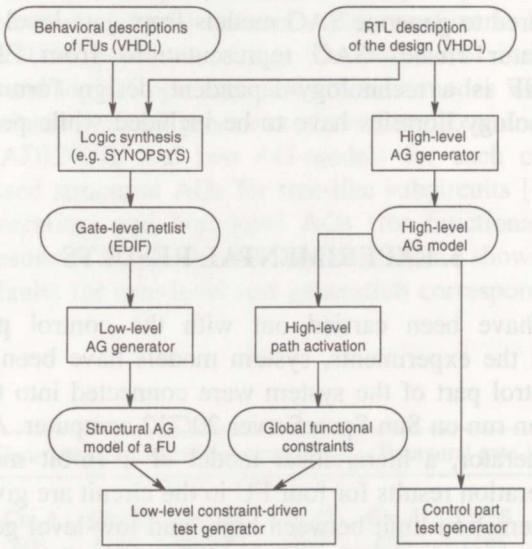


Fig. 5. The AG-based test synthesis system.

The datapath test generator has a hierarchical structure. The high-level part of the generator performs symbolic path activation at RT-level. During the path activation, functional constraints are extracted and applied to a CSP algorithm

contained in the low-level part of the generator. The test generation process takes place in the following way. Tests are generated sequentially for each functional unit (FU). For each FU, justification and propagation constraints are extracted at the high level and passed to the lower level test generator. During extracting, the constraints for the target FU, for all non-target FUs functional information are applied to perform propagation and justification through the FUs at the functional level. Such information, in the form of simplified behaviour of the block under a few simple conditions needed for fault effect propagation and justification, is preliminarily extracted and recorded in a special library of test modes (also referred to as *transparency library*). This functional information will consist of a set of input/output mappings (so-called I-paths [22] and F-paths [23]).

The low-level test generator handles justification and propagation constraints derived at the high level. Additional tasks of the low-level program are to generate gate-level tests for the FU and to assemble the final test set for the datapath. It is a random gate-level AG-model test generator. The input data at the target module boundary will be obtained by applying random vectors to the inputs of the reduced constraints driven model and performing logical simulation on this reduced model. In order to test the FUs, gate-level models of the FUs must be synthesized. The current system uses the Design Compiler by Synopsys Inc. for the logic-level synthesis. The RT-level VHDL description and a VHDL library of FUs, containing generic bit-width behavioral descriptions of the FUs are the input for logic-level synthesis. Due to the fact that the low-level test generator operates with structural AG (SAG) representations, the low-level AG generator is required to generate SAG models from gate-level netlists. The low-level AG generator creates SAG representations from EDIF 2.0.0 netlist descriptions. EDIF is a technology-dependent design format, and therefore, appropriate technology libraries have to be included while performing EDIF to SAG conversions.

## 5. EXPERIMENTAL RESULTS

Experiments have been carried out with the control part and datapath generators. In all the experiments, system models have been used where both datapath and control part of the system were connected into the whole system. All tests have been run on Sun SparcServer 20/712 computer. As an input for the datapath test generator, a hierarchical model of a 16-bit multiplier has been chosen. Test generation results for four FU in the circuit are given in Table 1. By increasing the interaction limit between high- and low-level generators, three of the FU have been tested at 100 per cent efficiency. For one of the FU, the transparent paths could not be activated at the high level. The results of test generation for control parts (FSM) of three different digital systems (multiplier *s344*, differential equation solver *diffeq*, and greatest common divisor *gcd*) are shown in Table 2.

*Table 1*

| Inter-action limit | FUs | Inter-actions | Vectors | Result | Inter-action limit | FUs | Inter-actions | Vectors | Result |
|---|---|---|---|---|---|---|---|---|---|
| 1 | add1 | 1 | 7 | success | 1000 | add1 | 1 | 7 | success |
| | add2 | 1 | 0 | failure | | add2 | 1000 | 76 | partial |
| | and1 | 1 | 3 | success | | and1 | 1 | 3 | success |
| | sub1 | 1 | 0 | failure | | sub1 | 244 | 0 | failure |
| 100 | add1 | 1 | 7 | success | ∞ | add1 | 1 | 7 | success |
| | add2 | 100 | 0 | failure | | add2 | 1093 | 78 | success |
| | and1 | 1 | 3 | success | | and1 | 1 | 3 | success |
| | sub1 | 100 | 0 | failure | | sub1 | 244 | 0 | failure |

*Table 2*

| Circuit | Faults | Coverage, % | Vectors | Time, s |
|---|---|---|---|---|
| *s344* | 90 | 84.44 | 11 | 0.26 |
| *diffeq* | 104 | 91.35 | 16 | 0.46 |
| *gcd* | 142 | 93.66 | 40 | 0.80 |

To investigate the adequacy of the mixed-level fault model defined for AGs and to investigate the possibility of achieving high quality gate-level tests by using high-level descriptions only, experiments were carried out with a restricted class of digital systems – with benchmarks based on a family of *n*-bit simplified RISC processors Cir_n (the number of instructions was 8). Only arithmetical and logical operations in different modes were implemented, and only combinational parts of processors have been examined. Circuits were synthesized by CADENCE, and two AG-models for each circuit have been created – compressed structural AGs for tree-like subcircuits [16] (for structural gate-level test generation) and high-level AGs (for functional high level test generation). The results of test generation experiments are shown in Table 3. The number of target faults for gate-level test generation corresponds to the case of compressed model.

*Table 3*

| Item | Functional high-level test generation | | | | Structural gate-level test generation | | | |
|---|---|---|---|---|---|---|---|---|
| Circuit name | Cir_4 | Cir_8 | Cir_16 | Cir_32 | Cir_4 | Cir_8 | Cir_16 | Cir_32 |
| Number of gates | 603 | 1195 | 2379 | 4747 | 603 | 1195 | 2379 | 4747 |
| Test length | 126 | 126 | 126 | 126 | 111 | 140 | 169 | 274 |
| Fault coverage, % | 99.86 | 99.93 | 99.96 | 99.98 | 99.93 | 99.97 | 99.98 | 99.99 |
| Test gener. time, s | 0.1 | 0.4 | 1.2 | 4.3 | 31.1 | 109.8 | 440.0 | 2584.6 |
| Gate-level faults | 2728 | 5360 | 10624 | 21152 | | | | |
| Target faults | | | | | 1522 | 2970 | 5866 | 11657 |

In the low-level random constraints driven test generator, the critical path tracing fault simulation method was implemented. High efficiency of simulation resulted from the compaction of the model by replacing the gate-level AGs with macro-level AGs (compressed AGs). In Table 4, gate- and macro-models are compared for their simulation speed. Simulation time per pattern is given for both models. Because of fault collapsing and decreasing the model complexity, simulation time decreases from 2.6 to 9.1 times for the given set of benchmark circuits.

*Table 4*

| Circuit | c432 | c499 | c880 | c1355 | c1908 | c2670 | c3540 | c5315 | c6288 | c7552 |
|---|---|---|---|---|---|---|---|---|---|---|
| ms/pattern (Gate) | 8.5 | 45.0 | 17.5 | 37.0 | 62.0 | 110 | 185 | 360 | 700 | 750 |
| ms/pattern (Macro) | 2.2 | 6.5 | 3.4 | 12.0 | 9.6 | 17 | 21 | 43 | 275 | 83 |
| Time ratio (M/G) | 3.9 | 6.9 | 5.2 | 3.1 | 6.5 | 6.5 | 8.8 | 8.4 | 2.6 | 9.1 |

*Table 5*

| Circuit | Gates | Universal simulator | | | Logical simulator | |
|---|---|---|---|---|---|---|
| | | RT-L | Macro-L | Gate-L | Macro-L | Gate-L |
| RISC_8 | 907 | 0.32 | 14 | 28 | 2.3 | 6.3 |
| RISC_16 | 1735 | 0.32 | 29 | 99 | 4.7 | 11.5 |
| RISC_32 | 3373 | 0.32 | 75 | 270 | 9.5 | 22.0 |

Table 5 shows simulation results in sec per 10 000 test patterns for the benchmark family of RISC-processors given as RT- and gate-level AG-models. The universal simulator is capable of uniformly simulating both high- and low-level models. On the other hand, the dedicated logical simulator is faster. Two types of binary AGs were used – gate and compressed macro ones.

## 6. CONCLUSIONS

This paper has described a hierarchical test generation system based on using alternative graphs. Differently from known methods, both higher and lower level design abstraction levels, and both control path and data path were handled by uniform fault models and uniform fault propagation methods. The fault model defined for an AG can be regarded as a generalization of the gate-level stuck-at fault model for higher level representations of digital systems. Uniform basis for gate- and higher level descriptions easily allow us to adopt and generalize gate-

level simulation and test generation methods to higher level ones. As a result, the complexity of the problem reduces, and the efficiency of test generation and fault simulation increases. In comparison with gate-level test generation, the high efficiency of the mixed-level test generator described in this paper has been shown by experiments with the family of benchmark circuits of simplified RISC processors.

## REFERENCES


1. Fujiwara, H. *Logic Testing and Design for Testability*. MIT Press, Cambridge, MA, 1985.
2. Abramovici, M., Breuer, M. A., and Friedman, A. D. *Digital Systems Testing and Testable Design*. Computer Science Press, 1990.
3. Ghosh, A., Devadas, S., and Newton, A. R. *Sequential Logic Testing and Verification*. Kluwer Acad. Publishers, 1992.
4. Anirudhan, P. N. and Menon, P. R. Symbolic test generation for hierarchically modeled digital systems. *Proc. Int. Test Conf.*, Washington, DC, USA, Sept. 1989, 461–469.
5. Bhattacharya, D. and Hayes, J. P. A hierarchical test generation methodology for digital circuits. *J. Electronic Testing: Theory and Application*, 1990, **1**, 103–123.
6. Karam, M., Leveugle, R., and Saucier, G. Hierarchical test generation based on delayed propagation. *Proc. Int. Test Conf., Nashville, TN, USA, 1991.* Nashville, 1991, 739–747.
7. Lee, J. and Patel, J. H. ARTEST: An architectural level test generator for data path faults and control faults. *Proc. Int. Test Conf., Nashville, TN, USA, Oct. 1991.* Nashville, 1991, 729–738.
8. Rao, S. R., Pan, B. Y., and Armstrong, J. R. Hierarchical test generation for VHDL behavioral models. *Proc. EDAC*, Paris, Febr. 1993, 175–183.
9. Corno, F., Prinetto, P., Sonza Reorda, M., Gläser, U., and Vierhaus, H. T. Improving topological ATPG with symbolic techniques. *IEEE VLSI Test Symposium, Princeton (NJ), USA, April 1995.* Princeton, 1995, 338–343.
10. Sallay, B., Petri, A., Tilly, K., and Pataricza, A. High-level test pattern generation for VHDL circuits. *Proc. 1st IEEE European Test Workshop, Montpellier, June 12–14, 1996.* Montpellier, 1996, 200–204.
11. Gramatova, E., Bezakova, J., and Fisherova, M. BX-TPG algorithm at the behavioral level implemented under LEDA VHDL system. *Proc. 2nd IEEE European Test Workshop, Cagliari, May 28–30, 1997.* Cagliari, 1997, 18–19.
12. Krupnova, H. and Ubar, R. Constraints analysis in hierarchical test generation for digital systems. *Proc. Baltic Electronisc Conference, Tallinn (Estonia), Oct. 9–14, 1994.* Tallinn, 1994, 313–318.
13. Minato, S. *Binary Decision Diagrams and Applications for VLSI CAD*. Kluwer Acad. Publishers, 1996.
14. Sasao, T., Fujita, M., et.al. *Representations of Discrete Functions*. Kluwer Acad. Publishers, 1996.
15. Ubar, R. Vektorielle alternative Graphen und Fehlerdiagnose für digitale Systeme. *Nachrichtentechnik/Elektronik*, 1981, **31**, 1, 25–29.
16. Ubar, R. Test synthesis with alternative graphs. *IEEE Design and Test of Computers*, Spring 1996, **13**, 1, 48–57.
17. Raik, J., Ubar, R., Jervan, G., and Krupnova, H. A constraint-driven gate-level test generator. *Proc. 5th Baltic Electronics Conference, Tallinn (Estonia), Oct. 7–11, 1996.* Tallinn, 1996, 237–240.
18. Ubar, R. and Brik, M. Multi-level test generation and fault diagnosis for finite state machines. *Dependable-Computing – EDCC-2*. Lecture Notes in Computer Science No. 1150. Springer Verlag, 1996, 264–281.

19. Thatte, S. and Abraham, J. Test generation for microprocessors. *IEEE Trans. on Comp.*, June 1980, 429–441.
20. Ward, P. C. and Armstrong, J. R. Behavioral fault simulation in VHDL. *ACM/IEEE 27th Design Automation Conf.*, Orlando, FL, USA, 1990, 587–593.
21. Ubar, R., Markus, A., Jervan, G., and Raik, J. Fault model and test synthesis for RISC-processors. *Proc. 5th Baltic Electronics Conf., Tallinn (Estonia), Oct. 7–11, 1996.* Tallinn. 1996, 229–232.
22. Abadir, M. S. and Breuer, M. A. A knowledge based system for designing testable VLSI chips. *IEEE Design & Test*, Aug. 1985, 56–68.
23. Freeman, S. Test generation for data path logic: The F-path method. *IEEE J. Solid State Circuits*, Apr. 1988, **23**, 421–427.

## DIGITAALSÜSTEEMIDE MITMETASANDI TESTIGENERAATOR

Marina BRIK, Gert JERVAN, Antti MARKUS, Priidu PAOMETS, Jaan RAIK ja Raimund UBAR

On kirjeldatud automaatset mitmetasandi testigeneraatorit, mis opereerib digitaalsüsteemide registri- ja ventiilitasandil. Süsteem realiseerib uue lähenemisviisi, mis baseerub ühtsel mitmetasandi digitaalsüsteemide diagnostilisel mudelil – alternatiivsel graafil (AG). Seesugune lähenemisviis võimaldab defineerida üldise veamudeli, mis üheselt katab kõrgtasandi funktsionaalsed ja mädaltasandi konstantrikked. AG-de kasutamine lubab rakendada standardseid protseduure rikete aktiviseerimiseks, rikete levitamiseks ja loogikavõrrandite lahendamiseks mõlemal tasandil. Süsteem koosneb eraldiseisvatest testigeneraatoritest andme- osa ja juhtosa jaoks. Mõlemal on sarnased vea levitamise protseduurid ja nad erinevad teineteisest ainult üldiste testiplaanide ülesehituse poolest. Eksperimendid kinnitavad käsitletud lähenemisviisi efektiivsust.