Internet-based Collaborative Test Generation with MOSCITO

A. Schneider[1], E. Ivask[2], P. Miklos[3], J. Raik[2], K.H. Diener[1], R. Ubar[2], T. Cibáková[3], E. Gramatová[3]
[1]Fraunhofer Institute for Integrated Circuits (IIS/EAS), Germany
[2]Tallinn Technical University (TTU), Estonia
[3]Institute for Informatics (IIN), Slovakia

Abstract

This paper offers an Internet-based environment for enhancing problem-specific design flows with test pattern generation and fault simulation capabilities. Automatic Test Pattern Generation (ATPG) and fault simulation tools at structural and hierarchical levels available at geographically different places running under the virtual environment using the MOSCITO system are presented. These tools can be used separately, or in multiple applications, for test pattern generation of digital circuits. In order to link different tools together and with commercial design systems, respectively a set of translators was developed. The functionality of the integrated design and test system was verified by several benchmark circuits.

1 Introduction

The quality of testing and the speed of test generation depend on the system description, the fault models and the used Test Pattern Generation (TPG) tools. Hierarchical test pattern generation approaches can considerably improve the quality of testing, in particular, for complex digital systems. To find an optimal test set for a Circuit Under Test (CUT) with a high defect coverage, generally, several TPG systems are needed.

The Internet opens a new dimension, and offers new chances using tools from different sources. The basic idea of this paper aimes at exploiting an Internet-based tool integration. For that purpose several TPG tools implemented at geographically different places were successfully integrated into the new virtual environment MOSCITO [1]. The essential features due to this integration environment were experimentally proved. The results obtained are presented also.

The paper is organized as follows. The MOSCITO system is described in section 2. The tool environment and the tools description are given in section 3 and section 4 respectively. Experimental results are shown in section 5.

2 MOSCITO

Starting from the idea to connect tools via the Internet to form an appropriate workflow for solving dedicated design problems the MOSCITO system was developed and implemented. The main emphasis was put on the following aspects:

- Encapsulation of design tools and adaptation of the tool-specific control and data input/output to the MOSCITO framework (MOSCITO agent, see below).
- Communication between the tools for data exchange to support distributed, Internet-based work.
- Uniform graphical user front-end program for the configuration of the tools, the control of the whole workflow and the visualization of result data.

Moreover, an important goal is to provide the functionality of a tool (e.g. fault simulator, a test pattern generator, a netlist translator, ...) to a potential user as a service in a local area network (LAN). This approach is similar to the Application Service Provider (ASP) idea or the recent approach of Web Services. In the present system the following tools have been integrated in MOSCITO:

- several translators for EDIF, ISCAS, and VHDL design description formats,
- Turbo-Tester tools for logic level fault simulation and test generation [2],
- DECIDER - a hierarchical test pattern generation for digital systems [3], [4],
- DefGen - ATPG for I_{DDQ} and voltage testing of digital circuits [5], [6],
- Tst2Alb - a data converter between ATPG tools
- ALB - an automatic fault library builder [7].

All the tools can act as MOSCITO agents and each of them provides a demanded service. The user are empowered to combine all the services to a problem-specific workflow. That means, the needed tools have not to be installed on the users local computer. Due to that fact the user's effort for installation, configuration and maintenance of software will be drastically reduced. Furthermore, specialized tools

can be executed on their native platform with a high performance (e.g. supercomputer with fast CPUs and large memory, Workstation-Cluster). So the entire workflow will speed up. To facilitate remote computing in this way is important for application with huge amount of computing time: e.g. fault simulation as well as test pattern generation.

The MOSCITO framework was implemented in JAVA and can run on different computing platforms. The only prerequisit is an installed Java Virtual Machine. At the moment MOSCITO is used on SUN workstation (Solaris) and on PCs (Microsoft Windows and LINUX).

2.1 Software architecture

MOSCITO mainly consists of three software layers:

- kernel layer,
- interface layer,
- extensions.

The MOSCITO kernel provides functionality for basic object and data management, file handling, XML processing, and communication. Due to the fact that MOSCITO is an open system a special interface layer provides programming interfaces for integration of new tools (e.g. test pattern generators, simulators, translators), new workflows (chain or cycle of a certain number of tools), and appropriate viewers such as for diagrams, plain text and images. Each interface is represented by a Java class which contains the basic functionality. The user only needs to extend this class and can implement its own extension. A large number of templates and example implementations helps the user to integrate a new tool or workflow in less than one or two days.

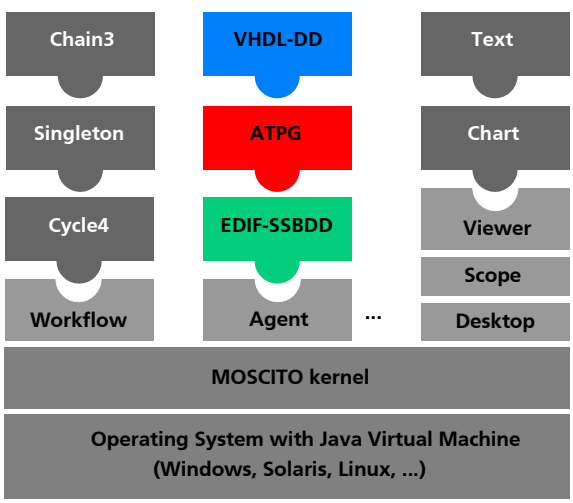

Figure 1. MOSCITO software architecture.

2.2 Tool encapsulation

For the integration of design tools (e.g. fault simulators, test pattern generators, netlist translators) with MOSCITO a sophisticated agent interface (MOSCITO agent) was introduced. Thus a tool, e.g. TPG, is embedded into a MOSCITO agent for:

- adaptating the input data to the embedded tool,
- converting the tool-specific data (simulation results, logfiles, test vectors),
- mapping the control information to the embedded tool and the transfer and conversion of status information (warning and error messages) to be submitted to the user.

For embedding programs into a MOSCITO agent there are three ways:

- Integration of the entire program: the software has to be run capable as a batch job (e.g. ATPG). In this way the integration of a lot of commercial tools is possible.
- Embedding of a library via the Java Native Interface (JNI): e.g. C, C++ or FORTRAN routines can be embedded.
- Direct integration of Java-classes and applications, respectively, in particular for software written in JAVA.

Encapsulation of the tools as a MOSCITO agent guarantees an uniform interface to the framework. All tool-specific details are aggregated in a special agent description file. This file is necessary to create tool-specific dialogs for the configuration of the tool via the front-end program.

2.3 Communication

The implementation of the tool communication is based on TCP/IP-sockets. Thus the tools can be executed on different computers or on different computing platforms (e.g. UNIX, Windows). All we need for communication is a LAN or Internet access. In this way, a lot of problems caused by the limited availability of the tools (e.g. incompatible computing platform, insufficient resources) can be prevented.

Usually, it is necessary to adapt/convert input as well as output data for each tool. To minimize the implementation effort for parsers, translators and converters, the format for all data transmitted in MOSCITO was set to a special XML format, the Moscito Markup Language (MoscitoML).

2.4 Graphical User Interface (GUI)

To offer a uniform and consistent concept for the user interaction the MOSCITO system has been provided with a graphical front-end with the following functionality:

- The problem description including all data (models, specification, initial values, configuration) can be read

in from a MOSCITO project file.

- Workflows can be chosen from a set of predefined flows for the specific problem.
- A browser supports the choice of agents (tools) needed for the solution of the problem from the set of available services.
- With buttons for start, pause, resume and stop the workflow can be controlled by the user.
- A console window collects all messages from the running tools and allows the observation of the proper operation or trouble shooting, respectively.
- The visualization module MOSCITO Scope supports the display of all result data (test vectors, statistic information).

The graphical front-end aims at using design tools via the Internet in a simple and efficient manner. Actually, the front-end is available as a JAVA application and has to be installed together with the MOSCITO software.

2.5 Internet-based usage

At first it is necessary to start one MOSCITO server on each host belonging to a domain of services. After that an administrator has to register one or more MOSCITO agents (e.g. VHDL-DD translator, Sysnopsys agent for synthesis, ATPG) so that they are available as remote services via LAN or Internet. Now a user can start the MOSCITO front-end program (GUI) and can browse through registered agents, can select, configure, and initialize the appropriated workflow and the needed agents. MOSCITO automatically calls remote tools and establishes direct connections between the tools for data transfer. Furthermore, the GUI allows the user to control and observe the data processing provided by a certain workflow. Result data are transmitted to the front-end and displayed by appropriate viewers (text, diagrams, images). Finally MOSCITO closes the connections between all remote tools and organizes correct termination of them.

3 Tool environment

To validate the MOSCITO system and to collect experiences while using it for real-life applications an experimental tool environment for test pattern generation (shown in **Fig. 3**) was designed and mapped to a MOSCITO workflow. In the following chapters the functionality of the tools themself will be explained in detail.

Design information can be generated in different ways, by VHDL files to be processed by commercial or experimental high-level or logic synthesis systems, or provided manually by schematic editors. The gate-level design is presented in the EDIF format. In university research practice, ISCAS benchmark families with a dedicated ISCAS presentation format are widely used. For linking together test generation and fault simulation tools with all the needed formats, different translators are developed.

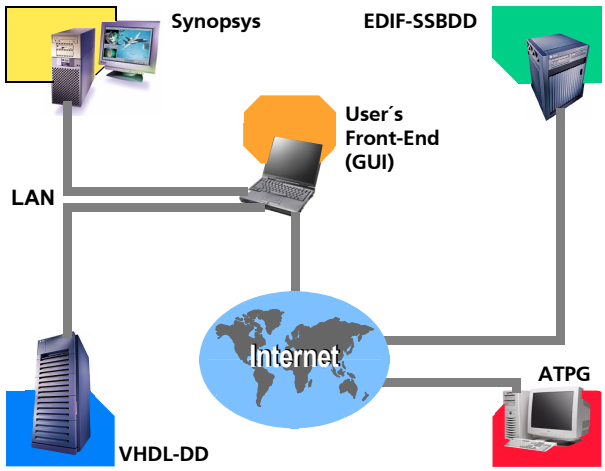

Figure 2. Distributed workflow: four MOSCITO agents are running on different remote hosts controlled by the user´s Front End.

For exchanging information between different tools (e.g. test libraries), an appropriate exchange interface was developed.This interface makes possible to generate test patterns in one geographical site and to analyze the quality of patterns in another site. In such a way, joint experiments were carried out in the field of defect-oriented test [9] where defect level analysis was performed in Poland, and logic level defect oriented fault simulation and test generation were carried out in Slovakia and Estonia.

4 Tool descriptions

4.1 Logic-level ATPG tools

The Turbo Tester ATPG software (block 6 in Fig.1) consists of a set of tools for solving different test related tasks by different methods and algorithms:

- test pattern generation by deterministic, random and genetic algorithms
- test optimization (test compaction)
- fault simulation and fault grading for combinational and sequential circuits
- defect-oriented fault simulation and test generation
- multi-valued simulation for detecting hazards and analyzing dynamic behaviour of circuits
- testability analysis and fault diagnosis.

All the Turbo Tester tools operate on the model of Structurally Synthesized Binary Decision Diagrams (SSBDD) [10]. The tools of Turbo Tester run on the structural level.

Two possibilities are available - gate-level and macro-level. In the second case, the gate network is transformed into macro network where each macro represents a tree-like subnetwork. Using the macro-level helps to reduce the complexity of the model and to improve the performance of tools. The fault model in the Turbo Tester is the traditional stuck-at model. However, the fault simulator and test generator can be run also in the defect-oriented mode, where defects in the library components can be taken into account. In this case, additional input information about defects in the form of defect tables for the library components is needed [9].

4.2 Hierarchical ATPG

In addition to the gate-level tools, a hierarchical test generation system DECIDER [3], [4] has been developed and linked to MOSCITO. DECIDER includes a Register-Transfer Level (RTL) VHDL interface for importing high-level design information, and also an EDIF interface for importing gate-level descriptions of logic

The ATPG uses a top-down approach, with a novel method of combining random and deterministic techniques. Tests are generated for each functional unit (FU) of the system separately. First, a high-level symbolic test frame (test plan) is created for testing the given FU by deterministic search. As the result, a symbolic path for propagating faults through the network of components is activated and corresponding constraints are extracted. The frame will adopt the role of a filter between the random TPG and the FU under test. If the filter does not allow to find a random test with 100% fault coverage for the component under test, another test frame will be chosen or generated in addition to the previously created ones. In such a way, the following main parts in the ATPG are used alternatively: deterministic high-level test frame generator, random low-level test generator, high-level simulator for transporting random patterns to the component under test and low-level fault simulator for estimating the quality of random patterns.

These test patterns are the input stimuli for the RTL design. Since the test generation implements also high-level fault models, we do not know the precise gate-level stuck-at fault coverage of these tests. Therefore, the test patterns have to be converted in order to correspond to the stimuli for the gate-level netlist of the entire design. This is required for gate-level fault simulation in order to measure the quality of generated tests.

4.3 Defect-oriented ATPG system

The DefGen ATPG system (block 10 in Fig.1) is a hierarchical ATPG system for combinational circuits for I_{DDQ} and/or voltage testing [5], [6]. The random, deterministic TPG algorithms and a fault simulator are involved in the ATPG system. The TPG process uses the functional fault model and runs over the functional test set specified for

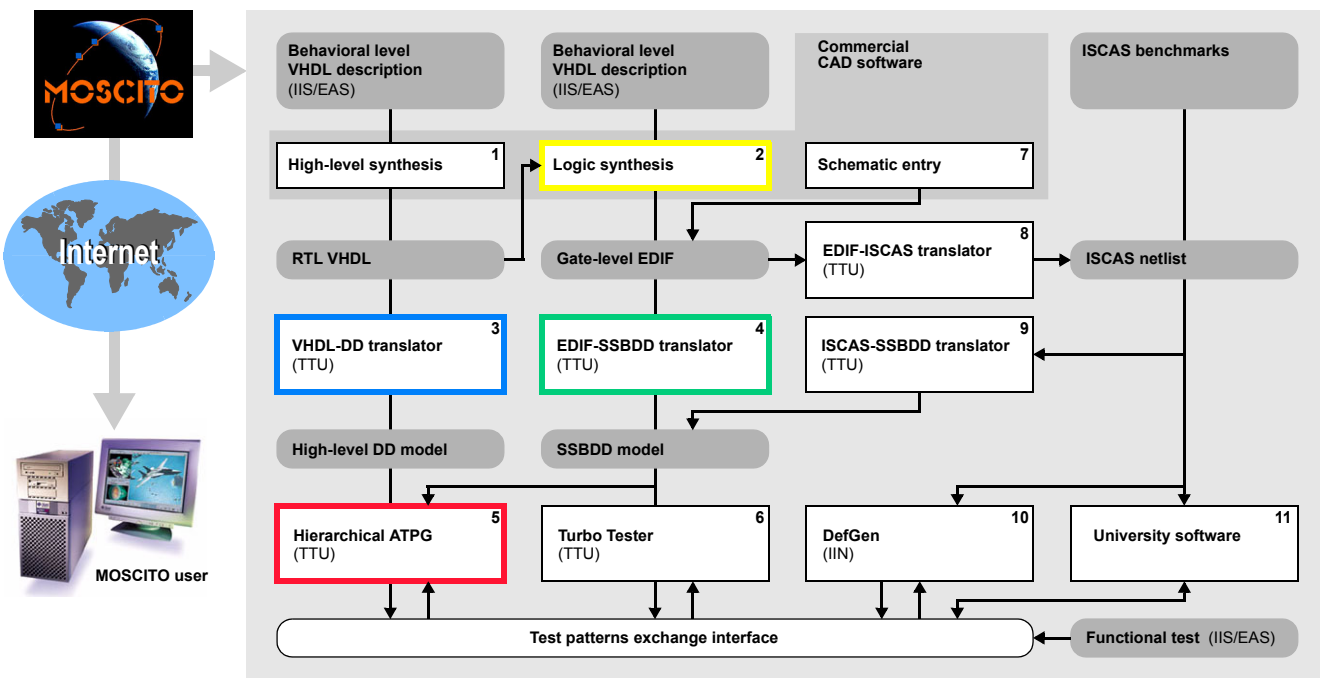

Figure 3. Integrated design and test flow

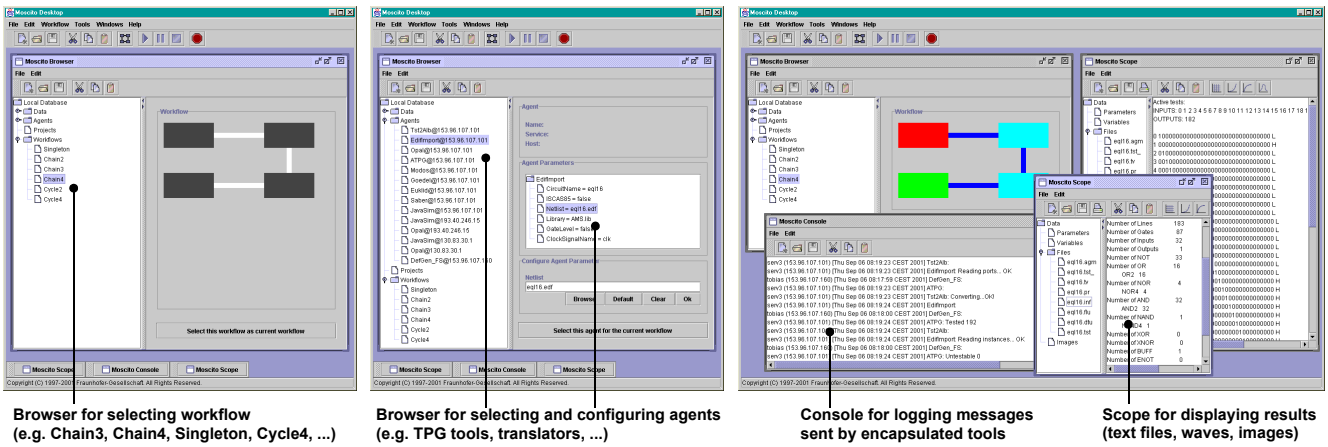

Figure 4. MOSCITO Desktop - the graphical user interface.

each functional cell of a CUT structure. The deterministic TPG techniques are based on justification and propagation of the predefined test patterns for each cell in a cicuit description. The functional test set for each cell is named a list of fault conditions and it is a part of the fault conditions library for DefGen. These lists can be created e.g. from a defect analysis of circuits cell at the low level or can be specified by the designer with regards to the used fault model for the investigated cells. The input format for circuit description is the language from ISCAS'85 benchmark circuits. The EDIF-ISCAS translator from Turbo tester (block 8 in Fig.1) can be used as the interface to DefGen.

An Automatic Fault Library Builder (ALB) has been developed and implemented for finding an optimal functional patterns for cells in the CUT structure [14]. The patterns are generated from different defect/fault tables for selected cell. Then, the received list of functional patterns are involved into the fault conditions library of the DefGen ATPG system. Some defect tables have been created for several combinational standard gates (e.g. from the 0.8 µm CMOS library) and lists of optimal patterns have been generated by ALB for the ATPG experiments.

Test Pattern Generation (TPG) technique at higher levels of abstraction rests upon a functional fault model and physical defect - functional fault relationships in the form of a defect coverage table at the lower level. Each table (one for a given cell) includes the following information:

- list of all possible faults (e.g. shorts, bridges between nonequipotential conducting paths resulting in a short circuit - which are caused by physical defects);
- erroneous logical functions performed by the faulty gate;
- list of input test patterns detecting possible physical defects.
- (optional) probabilities of occurrence of the physical defects.

The lists of erroneous functions and test patterns can be obtained by electrical simulations at the transistor level or calculated using Boolean algebra. Probabilities of defects occurrence can be calculated by layout probabilistic analysis at the physical level taking into account defect density and size distribution.

Some experiments have been performed with the ATPG tools running at the Tallinn Technical University and the Institute of Informatics of the Slovak Academy of Sciences using defect tables created at the Warsaw Technical University separately [15]. These systems have been integrated under the MOSCITO system developed and provided by the Fraunhofer Institute for Integrated Circuits for testing their functionality in the new virtual environment.

5 Experiments

The described environment has been tested in the frame of European project VILAB by the partners IIS/EAS (Germany), IIN (Slovakia), LIU (Sweden), TTU (Estonia) for several designs according to the following general algorithm (the reference to exploited tools in Fig. 1 is given in parentheses):

1. The user evaluates the quality of his own functional test for the new design (2) by using Turbo Tester fault simulator (4,6). Alternatively, he can also make use of other university fault simulator (10,11).
2. If the results are acceptable (test has obtained the demanded quality) go to END, else go to Step 3.
3. The user can work with the implemented ATPGs 5, 6 or 10. If the I_{DDQ} or defect oriented testing is demanded he uses DefGen (8,10).
4. If the stuck-at-fault model is accepted, the user can work with the ATPGs 5, 6 or 10. If the circuit is a simple sequential or combinational one (e.g. only FSM without data-path) go to 5. If the circuit consists of the

control- and data-paths the user can work with the hierarchical ATPG (3,4,5). In this case, both the RTL description from high-level synthesis system (1) and the gate-level description from logic synthesis system (2) are needed. If the results are acceptable (test has the needed quality), go to END, else go to Step 5.

5. The user can work with the gate-level ATPG (4,6). If the results are acceptable (test has the needed quality), then go to END, else go to Step 6.
6. The testability should be now improved by redesign. Some flip-flops can be included, for example, into the scan-path. For testing the new full or partial scan-path design the user can work again with the gate-level ATPG (2,4,6).
7. Depending on the results (the quality of test reached), the step 6 can be repeated till the demanded test quality has been obtained.
8. END.

This environment has been utilized for research purposes. For example, the performance of the hierarchical ATPG (5) was compared against the existing university tools GATEST and HITEC [11], [12]. For that the translator 8 was necessary. The results of comparison of different ATPGs are given in the table below.

	DECIDER		GATEST		HITEC	
	Fault cover %	Time s	Fault cover %	Time s	Fault cover %	Time s
gcd	91.0	3.4	92.2	89.8	89.3	195.6
mult8x8	79.4	13.6	77.3	1585.0	63.5	1793.0
diffeq	95.8	15.8	96.0	9720.0	95.1	N.A.

6 Summary

In the paper an Internet-based environment based on MOSCITO system is presented. The environment is focused on providing high-level and logic level design flows with test pattern generation and fault simulation operational activities. The main effort was put on linking together test generators and fault simulators with varying functionalities and diverse fault models available at geographically different sites. The system provides interfaces and links to commercial design environments (such as Synopsys) and also to other university tools. The functionality of the integrated design and test system was verified by several benchmark circuits and by different design and test flows.

Furthermore, authors believe that the MOSCITO architecture is powerful enough to solve similar problems in other application areas of automated system design. Future work will go in this direction.

Acknowledgements: *This work has been supported by the European Community under the INCO Copernicus project CP977133 VILAB: Microelectronics Virtual Laboratory for Cooperation in Research and Knowledge Transfer. Partially this work has been supported also by Estonian Science Foundation (Grants No 3658 and 4300) and by the Slovak VEGA project grant 2/6091/20 - Behavioural and real Defet Test Generation for Digital Circuits and Systems.*

References

[1] P. Schneider, S. Parodat, A. Schneider, P. Schwarz: A modular approach for simulation-based optimization of MEMS. Design, Modeling, and Simulation in Microelectronics, 28-30 November 2000, Singapore, pp 71-82, SPIE Proceedings Series Volume 4228

[2] J. Raik, R. Ubar: Feasibility of Structurally Synthesized BDD Models for Test Generation. Proc. of the IEEE European Test Workshop, Barcelona (Spain), May 27-29, 1998, pp.145-146.

[3] J. Raik, R. Ubar: Sequential Circuit Test Generation Using Decision Diagram Models. IEEE Proc. of DATE. Munich, March 9-12, 1999, pp. 736-740.

[4] J. Raik, R. Ubar: Fast Test Pattern Generation for Sequential Circuits Using Decision Diagram Representations. Journal of Electronic Testing: Theory and Applications. Kluwer Academic Publishers. Vol. 16, No. 3, pp. 213-226, 2000.

[5] E. Gramatova, T. Cibakova, P. Miklos: Defect Oriented TPG for combined I_{DDQ} - Voltage Testing of Combinational Circuits. Proc. of ETW'2000.

[6] E. Gramatova, J. Gaspar, T. Cibakova: Fault Simulation for Combined I_{DDQ} - Voltage Tesing of Combinatorial Circuits, Proc. of DDECS'00, pp. 52-58.

[7] T. Cibáková, E. Gramatová, W. Kuzmicz, W. Pleskacz, J. Raik, R. Ubar: Defect-Oriented Library Builder and Hierarchical Test Generation. Proceedings of DDECS'2001, Gyor, Hungary, pp.163-167.

[8] G. Jervan, P. Eles, Z. Peng, J. Raik, R. Ubar: High-Level Test Synthesis with Hierarchical Test Generation. IEEE 17th NORCHIP Conference, Oslo, Nov. 8-9, 1999, pp.291-296.

[9] M. Blyzniuk, FT. Cibakova, E. Gramatova, W. Kuzmicz, M. Lobur, W. Pleskacz, J. Raik, R. Ubar: Hierarchical Defect-Oriented Fault Simulation for Digital Circuits. IEEE European Test Workshop, Cascais, Portugal, Mai 23-26, 2000, pp.151-156.

[10] R. Ubar: Multi-Valued Simulation of Digital Circuits with Structurally Synthesized Binary Decision Diagrams. Overseas Publishers Assotiation N.V. Gordon and Breach Publishers, Multiple Valued Logic, Vol.4 pp. 141-157, 1998.

[11] E. M. Rudnick, J. H. Patel, G.S. Greenstein, T.M. Niermann: Sequential Circuit Test Generation in a Genetic Algorithm framework. Design Automation Conf., pp. 698-704, 1994.

[12] T. M. Niermann, J. H. Patel: HITEC: A Test Generation Package for Sequential Circuits. European Conf. Design Automation, pp. 214-218, 1991.

[13] T. M. Niermann, W. T. Cheng, J. H. Patel: PROOFS: A Fast, Memory Efficient Sequential Circuit Fault Simulator. 27th ACM/IEEE Design Automation Conference, pp. 535-540, 1990.

[14] T. Cibakova, E. Gramatova, W. Kuzmicz, W. Pleskacz, J.Raik, R.Ubar: Defect-Oriented Library Builder and Hierarchical Test Generation, Proc. of DDECS'01, Gyor (Hungary), April 18-20, 2001, pp. 163-168.

[15] T. Cibakova, M. Fischerova, E. Gramatova, W. Kuzmicz, W. Pleskacz, J. Raik, R. Ubar: Defect-oriented Test Generation Using Probabilistic Estimation. Proc. of MIXDES'01, Zakopane (Poland), June 21-23, 2001, pp. 131-136.

[16] MOSCITO: http://www.eas.iis.fhg.de/solutions/moscito