IP Based Design 2002

Session : .....

INTERNET–BASED TESTABILITY DRIVEN TEST GENERATION IN VIRTUAL ENVIRONMENT MOSCITO

A. Schneider, K.-H. Diener, G.Elst
Fraunhofer Institute for Integrated Circuits (IIS/EAS)
Dresden Germany

E.Ivask, J.Raik, R.Ubar, Tallinn Technical University
Tallinn Estonia

Abstract :

The paper describes an environment for an Internet-based co-operation in the field of design and test of digital systems. A VLSI design flow is combined with an Internet-based hierarchical automated test pattern generation (ATPG). A novel hierarchical ATPG driven by testability measures is presented. Both, the register-transfer (RT) and the gate level descriptions are used, and decision diagrams are exploited as a uniform model for describing systems at both levels, for calculating testability measures and for test generation. The ATPG and testability analyzer can be run at geographically different places under the virtual environment MOSCITO. The interfaces between the integrated tools and also the commercial design tools were developed and implemented. The functionality of the integrated design and test system was verified in several co-operative experiments over Internet by partners in different geographical sites. The experimental results have shown the advantages of using structural tests generated by ATPG compared to using functional test sequences created by designers.

1 INTRODUCTION

In the field of digital design, the system-on-chip (SoC) technology is becoming state-of-the-art. In order to come up with innovative electronic systems in time and with competitive cost, a lot of EDA problems should be solved: HW/SW codesign, high-level synthesis, testability evaluation, test pattern generation. Usually, not all the needed EDA tools are available for a designer in his working site.

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, a novel and very efficient testability driven ATPG tool was successfully integrated into the new virtual environment MOSCITO [1] to implement a Internet-based cooperative design environment.

The paper is organized as follows: The MOSCITO system is specified in Section 2. The description of the new approach to testability calculation is given in Section 3. The testability driven ATPG is presented in Section 4, and experimental results obtained by the use of the MOSCITO environment are shown in Section 5.

2 MOSCITO

The software developed at IIS/EAS offers a Client-Server concept. There is one Master Server, several Slave servers and arbitrary number of clients. The requested service is provided by Slave servers. So-called Agents are attached to each Slave server. The Agents encapsulate service providing program executables. The communication is based on TCP/IP-sockets. The main emphasis of the tool integration 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
- Communication between tools, data exchange to support distributed, Internet-based work
- Graphical user interface to configure the tools, control the workflow and visualize result data

An important goal is to provide the functionality of a tool to a potential user as a service in LAN or WAN. This approach is similar to the Application Service Provider (ASP) idea or the recent approach of Web Services. The following tools have been integrated in MOSCITO:

- High-level synthesis (1) with RTL output [2]
- Interface from RTL VHDL (4) to ATPG

- Interfaces from EDIF (5) and ISCAS (6) formats to ATPGs and fault simulators
- Hierarchical ATPG (7) DECIDER [3]
- Logic level ATPG (8) Turbo-Tester [4].

The listed tools can act as MOSCITO agents and each of them supply a demanded service. The user can combine all the services to a problem-specific workflow. The needed tools have not to be installed on the users local computer. User's effort for installation, configuration and maintenance of software will be drastically reduced.

The MOSCITO was implemented in JAVA and can run on different computing platforms. The only prerequisite is an installed Java Virtual Machine. At the moment MOSCITO is used on SUN workstations (Solaris) and on PCs (Microsoft Windows and LINUX). MOSCITO consists of three software layers: kernel layer, interface layer, extensions. The kernel provides the functionality for basic object and data management, file handling, XML processing, and communication. Since MOSCITO is an open system, a special interface layer offers programming interfaces for integration of new tools, new workflows 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 has 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 few days.

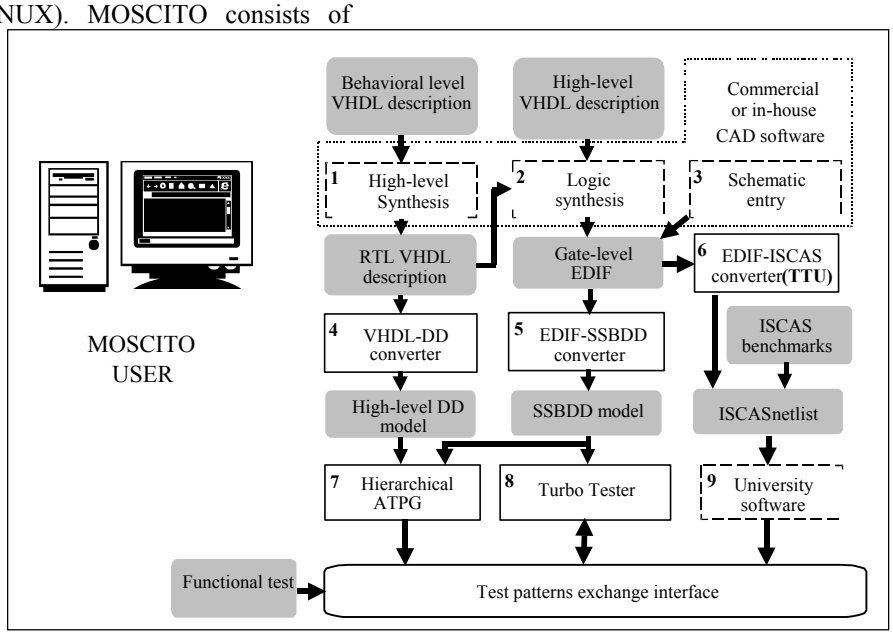


Fig. 1. Integrated tool environment

2.1 TOOL ENCAPSULATION

The Agent interface was introduced to integrate different tools with MOSCITO. The embedding of a tool into a MOSCITO agent allows:

- adapting the input data to the embedded tool,
- converting the tool-specific data (simulation results, logfiles, test vectors),
- passing the control information to the tool, to transfer and convert the status information to be submitted to the user.

The embedding is possible in three ways:

- Integrating the entire program: it has to be capable running as a batch job. Integration of many commercial tools is possible that way
- Embedding a library (e.g. C, C++ routines) via Java Native Interface (JNI)
- Direct integration of Java classes and applications; in particular for JAVA software.

The encapsulation of the tools as a MOSCITO agent guarantees a uniform interface to the framework. All tool-specific details are aggregated in a special agent description file to create tool-specific configuration dialogs in user GUI. To minimize the implementation effort for parsers, translators and converters, a special XML format-the Moscito Markup Language is used for all transmitted data.

2.4 GRAPHICAL USER INTERFACE

- The problem description, including all data 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.
- The visualization module displays all the result data (test vectors, statistic information).

- A console window collects all messages from the running tools and allows the observation of the proper operation or trouble shooting

The front-end is a JAVA application and has to be installed together with the MOSCITO software.

2.6 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 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. The MOSCITO system 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. Finally MOSCITO closes the connections between all remote tools and organizes correct termination of them.

2.7 FIREWALL TRAVERSAL

A firewall can be regarded as filter which allows certain type of communication (e.g. TCP/IP protocol based) "go through" configurable chock points (called ports)[13]. A firewall is implemented for example as a specialized software running on a well secured computer. The Internet and intranet are accessible only via that computer. Opening a port in a firewall means just configuring filter rules. In a case of restrictive firewall there are only few ports left open for incoming internet connections (like port 80 for http web server). In order to comply with firewall requirements, the major MOSCITO communication schema was modified. Since firewall disables direct connections between subcomponents, all the communication has to be organized through predetermined communication ports. Random port numbers are not allowed. Simplified communication schema in a firewall protected environment is shown in Fig. 2

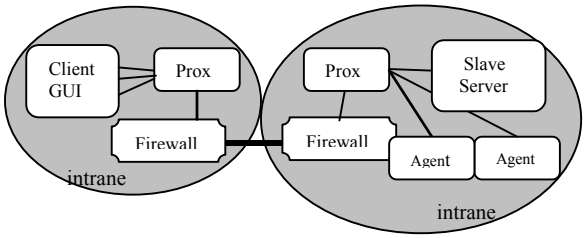


Fig. 3. Communication between Client and Agents via proxy

To solve the firewall traversal problem a MOSCITO proxy is to implement as Java application (Fig.3). Here, again MOSCITO socket based communicationis is used.

Proxy mechanism enables hosts in one side of proxy server to gain full access to hosts in the other side of the proxy server without requiring direct IP reachability. It works by redirecting connection requests from hosts in one side to hosts in the other side to a proxy server that authenticates and authorizes the requests, establishes a proxy connection and passes data back and forth.

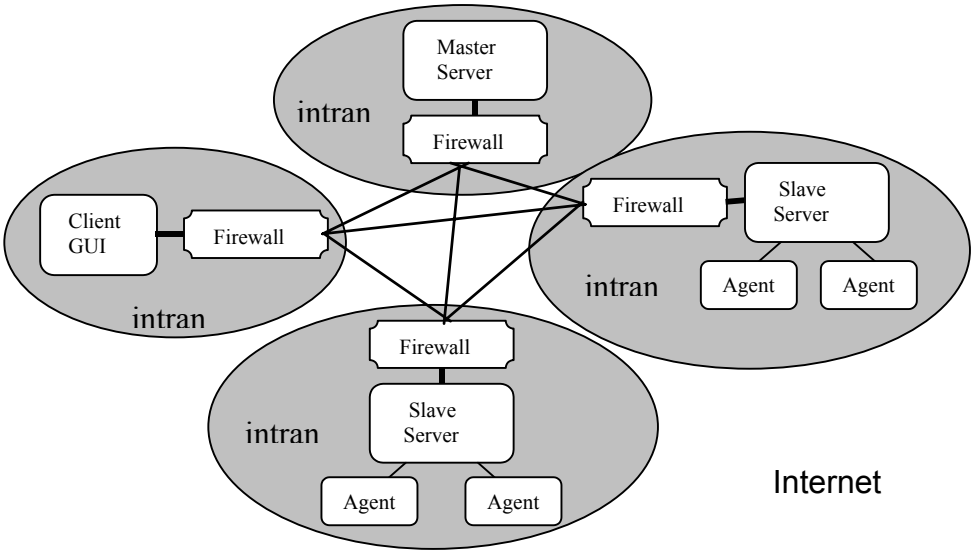


Fig. 2. Communication between firewall protected MOSCITO subsystems: connections are allowed only between dedicated communication ports

2.8 TOOL ENVIRONMENT

An experimental tool environment for design and test pattern generation (Fig.1) was developed and mapped to a MOSCITO workflow. In the following the functionality of the tools 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 format are widely used. For linking test generation and fault simulation tools with all the needed formats, different translators and interfaces were developed (Blocks 4,5,6 in Fig.1). The interfaces make possible to design a circuit in one geographical site, generate test patterns in another site, and to analyze the quality of patterns in a third site.

3 TESTABILITY DRIVEN TEST GENERATION

The test generation requires two basic steps: excitation of the fault, and propagation of erroneous values to primary outputs. Both steps might involve backtracking. Controllability and observability measures can be used to reduce the number of backtracks. A lot of methods have been proposed for determining testability measures [5]. The first ideas of measuring testability in gate-level circuits [6] have been extended also to digital systems [7]. The known testability calculation methods use different approaches for gate- and RT levels. Basically, the well-known methods are not easily usable because for each component its own formula of testability calculation has to be given. In the present approach Decision Diagrams (DD) [8] are exploited for both, the gate level and the RT level testability calculation. The same models and similar path tracing procedures utilized as for test generation, and no particular libraries are needed for testability calculation.

3.2 RT LEVEL DECISION DIAGRAMS (DDs)

In general case of RT level DDs the values of variables at nonterminal nodes (and functional expressions at terminal nodes are not binary.

Fig.4 presents an example of a word-level DD for a control part of a simple control part of a digital system.

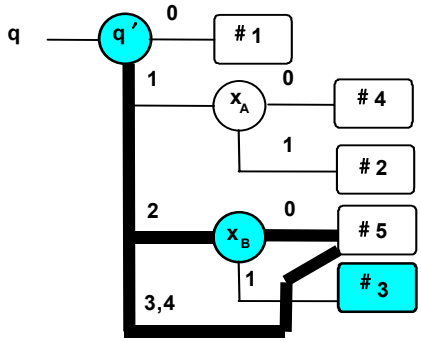


Fig. 4. RT level DD for a control part

The variables of the DD have the following meaning: q denotes the next sate, q' indicates the current state, x_A and x_B identify input signals, and the constants in terminal nodes labelthe next state value. Tracing the path activated by given values of q', x_A and x_B we reach a terminal node which gives us the next state value.

An example of a register-level data-path and of his compressed DD is depicted in Fig.5. The DD in Fig.5 is the superposition of the components' DDs of the given data path.

3.3 CONTROLLABILITY FOR RT LEVEL DDs

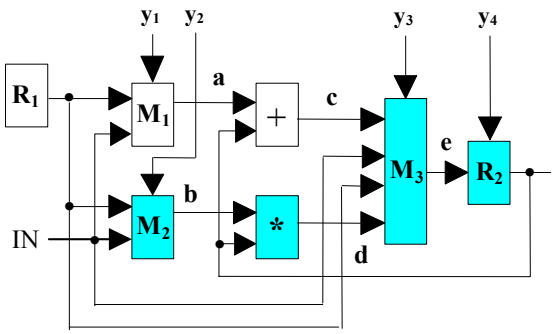
For controlling a given working mode of the system, the control variables have to be assigned specific values. Controllability $C(y=k)$ of a multivalued control signal y in a digital system to a specific value k can be regarded as the probability $P(y=k)$ that y will take the value k. The computation of $P(y=k)$ is based on traversing paths in the DD for y and using the formula:

$$P(y=k) = \sum_{L_i \in L(k)} \prod_{x \in X_i} P(x=e) \qquad (1)$$

where e is the value of the variable x needed for activating the path $L_i \in L(k)$ and $L(k)$ is the set of all possible paths which enters into the terminal node with constant value k.

As an example, we calculate the controllability of $P(q=5)$ for DD in Fig.4 as follows:

$$P(q=5) = P(q=2)\, P(x_B=0) + P(q=3) + P(q=4).$$



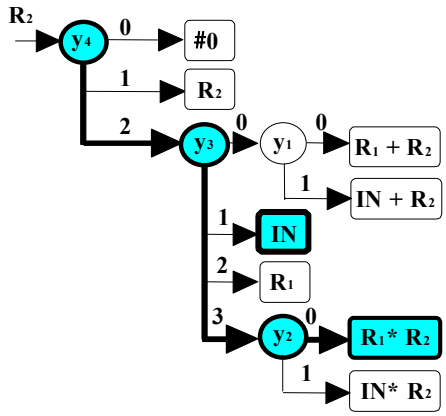


Fig. 5. RT level DD for a data part

For data variables (buses, registers) in hierarchical test generation we usually have to assign them symbolic values along some activated paths either from inputs of the system or from easily controlled internal points (like scan-path registers). For that case we need to calculate the controllabilities for creating such activated paths. Controllability C(R) of a data word R in a digital system can be regarded as the probability P(R=IN) that R can directly be loadable from a given control point IN. The computation of P(R=IN) is based on traversing paths in the DD and using the formula:

$$P(R=IN) = \sum_{L_i \in L(IN,R)} \prod_{x \in X_i} P(x=e) \qquad (2)$$

where $L_i \in L(IN,R)$ is the set of all possible paths to control the value of R from the control point IN.

As an example, we calculate the controllability of P(R_2=IN) in G_{R2} in Fig.5 as follows:

$$P(R_2=IN) = P(y_4=2)\, P(y_3=1).$$

3.4 OBSERVABILITY FOR RT LEVEL DDs

For terminal nodes m^T in a DD we define the observability O($z(m^T)$) of the functional expression $z(m^T)$) as the probability P(y=$z(m^T)$) that y will have the value equal to $z(m^T)$. The computation of P(y=$z(m^T)$) is based on traversing paths in the DD G_y and using the formula:

$$P(y=z(m^T)) = \sum_{L_i \in L(m_0,m^T)} \prod_{x \in X_i} P(x=e) \qquad (3)$$

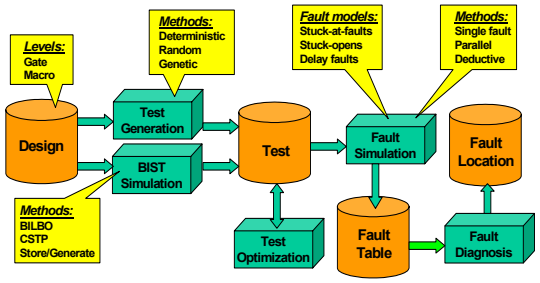
where $L_i \in L(m_0,m^T)$ is the set of all possible paths from the initial node m_0 of the graph to the terminal node m^T. To highlight the procedure, we calculate the observability of P(R_2= R_1+ R_2) in G_{R2} in Fig.5 as follows:

$$P(R_2 = R_1 * R_2) = P(y_4=2)\, P(y_3=3)\, P(y_2=0).$$

3.5 HIERARCHICAL ATPG (block7 in Fig.1) 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. The search is guided by the testability measures calculated by a testability analyzer. As the result of the search process, a symbolic path (a test frame) for propagating faults through the network of components is activated and corresponding constraints are extracted. The test frame will adopt the role of a filter between the random patterns to the component under test and low-level fault simulator for estimating the quality of random patterns. The mentioned low-level tools belong to Turbo Tester software package in Fig.6 (also block 8 in Fig.1).

The TPG and the FU 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

Fig.6. A set of low-level ATPG tools Turbo-Tester



	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
Mult 8x8	79.4	13.6	77.3	1585	63.5	1793
Diffe g	95.8	15.8	96.0	9720	95.1	N.A.

Table 1. Exp. results for hierarchical ATPGs

4 EXPERIMENTS

The MOSCITO based environment has been utilized for research purposes. The performance of the hierarchical ATPG (3) was compared against the existing university tools GATEST [10] and HITEC [11]. For that the translator 6 was necessary. The results of comparison of different ATPGs are given in Table 1. Actual stuck-at fault coverages of the test patterns generated by all the

circuit	# of high-level tests	worst known		average random		HLS metrics [7]		current approach	
		# tested	coverage	# tested	coverage	# tested	coverage	# tested	coverage
gcd	19	10	52.6 %	11.5	60.5 %	19	**100.0** %	18	94.7 %
mult8x8	20	3	15.0 %	10.1	50.5 %	15	**75.0** %	15	**75.0** %
diffeq	45	13	28.9 %	23,7	52.7 %	13	28.9 %	28	**62.2** %
risc_32	97	57	58.8 %	71.4	73.6 %	75	77.3 %	92	**94.8** %
average test coverage:		38.83 %		59.33 %		70.30 %		**81.62** %	

Table 2. Evaluation of testability measures in RTL test generation using DECIDER

three tools were measured by the same fault simulator. The Table 2 compares two different testability approaches: this one proposed in this paper with average result of random testability measures and the worst testability ordering known to us. The column '# of high level tests' shows the total number of high-level tests set up by the ATPG. Columns '# tested' show the number of these tests passed by DECIDER. Columns 'coverage' show the percentage of the RTL tests passed. Note, that this is not the actual gate-level stuck-at fault coverage but rather an RT-level assessment. However, as our experience and previous research [10, 11] have shown, this number is very close to the gate-level fault coverage. As it can be seen from the Table 2, average test coverage for the benchmark set is 60 %. However, if inconvenient node ranking is used this number can drop as low as 40 %. The experiments showed, however, that the method published in [7] can increase the test coverage to 70 %, while the one proposed in this paper raises the coverage to roughly 80 %.

6 CONCLUSIONS

In the paper an Internet-based test environment supported by MOSCITO system [13] is presented. The environment is focussed on providing high-level and logic level design flows with testability analysis, test pattern generation and fault simulation at register-transfer and gate level operational activities. The main effort was put on linking together test generators and fault simulators with varying functionalities available at geographically different sites. A novel approach of testability driven hierarchical test generation was developed and experimented. Differently from known ATPGs, the testability measures calculated at RT level were used for guiding high-level path activation search. The system provides interfaces and links to commercial design environments 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 continue in this direction.

REFERENCES

[1] A.Schneider et. al Internet-based Collaborative Test Generation with MOSCITO. *Proc. of DATE'02*, Paris, France, March 4-8, 2002, pp.221-226.

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

[3] J.Raik, R.Ubar: Fast Test Pattern Generation for Sequential Circuits Using DD Representations. *J. of Electronic Testing: Theory and Applications. Kluwer Publ*. Vol. 16, No. 3, pp. 213-226, 2000.

[4] J.Raik, R. Ubar: Feasibility of Structurally Synthesized BDD Models for Test Generation. *Proc. of the ETW,* Barcelona 1998, pp.145-146.

[5] M.L.Bushnell, V.D.Agrawal. Essentials of Electronic Testing. Kluwer Acad. Publ., 2000.

[6] Goldstein L.HControllability/observabili analysis of digital circuits. *IEEE Trans. Circuit*. Syst., CAS-26, No.9, 1979, pp. 685-693.

[7] Gu X., Kuchcinski K., Peng Z. Testability Analysis and Improvement from VHDL Behavioral Specifications. EURO-DAC, 1994.

[8] Ubar R. Test Synthesis with alternative graphs. IEEE Design & Test of Computers. Spring 1996, pp.48-57.

[9] Ubar R. Multi-Valued Simulation of Digital Circuits with Structurally Synthesized BDDs. *OPA N.V. Gordon & Breach Publ, Multiple Valued Logic,* Vol.4, pp. 141-157, 1998.

[10] E.M.Rudnick et. al Sequential Circuit Test Generation in a Genetic Algorithm framework. *DAC.,* pp. 698-704, 1994.

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

[12] M. L. Flottes, R. Pires, B. Rouzeyre, "Analyzing Testability from Behavioral to RT Level", Proc. ED&TC, pp.159-165, 1997.

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