46 WEB-BASED ENVIRONMENT FOR DIGITAL ELECTRONICS TEST TOOLS

Eero Ivask, Jaan Raik, Raimund Ubar, Andre Schneider*
Tallinn Technical University, ESTONIA, {ieero|jaan|raiub}@pld.ttu.ee
**Fraunhofer Inst. for Integr. Circuits, GERMANY, Andre.Schneider@eas.iis.fhg.de*

This paper describes a concept and implementation of the web-based environment for providing access over the Internet to existing stand-alone digital electronics test tools. On the user side only ordinary web browser is sufficient. The environment is built according to the client-server three-tier concept using HTML, Java applets/servlets and MySQL as database backend for user tracking and management tasks. In this paper we discuss integration of two software systems into web-based environment. The paper presents the workflows that can be executed over the Internet and gives the experimental results for estimating the efficiency of the hierarchical ATPG.

1. INTRODUCTION

Assuring the quality of modern System-on-Chip (SoC) technology is unthinkable without the use of efficient test methods. In addition to traditional logic level tools for built-in self-test, fault simulation, test generation and optimization, high-level and hierarchical test applications are required. However, not all the needed test tools may be available for a designer and installing a new one would be as an additional overhead.

The Internet opens a new dimension, and offers new chances using tools from different sources. The topic of this paper is web-based integration of tools and execution of programs over the Internet for testing digital circuits. For that purpose, a novel and efficient hierarchical ATPG tool called DECIDER was successfully integrated into new web-based environment to implement virtual laboratory for digital test. By this moment, no commercial systems for hierarchical test pattern generation are available on the market. An interesting feature of DECIDER is that its VHDL design interface allows to convert VHDL descriptions into decision diagram models, which provides a more general representation for digital hardware and can be used for fast simulation and test pattern generation (Ubar, 1998).

In addition, a set of diagnostic tools for the logic level test was made available over the Internet. The system is called Turbo Tester and it includes a powerful EDIF interface, which allows using the tools in cooperation with most of the available commercial EDA systems.

The rest of the paper is organized as follows: The general concept is specified in Section 2. The implementation details are given in section 3. The test generation systems integrated are presented in Section 4, and experimental results obtained by use of the new environment are shown in Section 5 and conclusions are given in 6.

2. GENERAL CONCEPT

A virtual environment to support the research and teaching of digital system testing is described below. User will be able to use test tools remotely over the Internet. System is based on an open architecture that allows easily add new tools later.

System core for remote tool usage has client-server concept similar to MOSCITO (Schneider, 2002) (MOSCITO). There is one master server, several application servers and arbitrary number of clients (see Figure 1). Master server holds the information about application servers, which provide service. On application server so called agents can be invoked. Agents encapsulate actual test tools (executables). User first accesses the master server and gets a list of services available. After selecting appropriate service, user is automatically re-directed to application server, and then the work with the actual tool can start. The big difference from MOSCITO is HTTP protocol based communication and use of Java applets and servlets. Also user tracking is unique. There is no need to install tools on the user's local computer. Therefore, user's effort for installation, configuration and maintenance of software will be drastically reduced. The system is implemented in Java and can therefore run on different computing platforms. Actual work tools must run on their native platform of course.

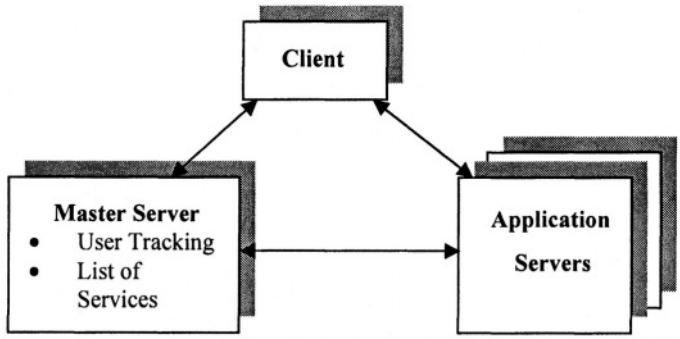

Figure 1 – General concept

Each tool will be wrapped into Java agent. Encapsulation of entire tools guarantees a uniform interface to the framework. There will be no need to reprogram existing tools. Only requirement for tool encapsulation is that tool is able to run from command line (in 'text mode'). All tool-specific details are stored in a description file. This allows automatically display appropriate tool configuration dialogs for end user.

Several tools can be started simultaneously. One servlet will serve many client applets in parallel. There is task queue management. Results reside initially on the

server computer where servlet is running. Each user has its own server-side workspace. In the database there is user's workspace folder name where results for certain task id can be found. It is possible to query on results, make statistics. Subsequently, general concept is elaborated in much greater detail.

3. IMPLEMENTATION

The environment for remote use of Test tools is built according to the client-server three-tier concept using HTML pages, Java applets/servlets and MySQL as database backend for user tracking and management tasks. General solution in details is given in Figure 2. Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies. Tomcat and servlets running on it play important role while gaining access to intranet resources on application servers and MySql database (platform independent open source DB).

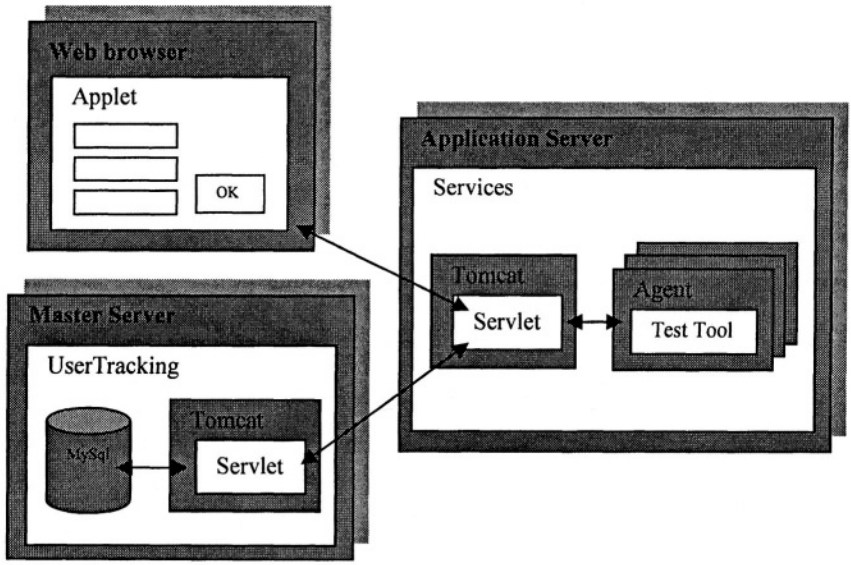

Figure 2 – Implementation details

3.1 Usage scenario

Below is described simplified scenario for end user (see Figure 2), we assume here that user has account already:

1. User logs in with login name and password. Information is sent to servlet which accesses database and verifies user. If user exists, then servlet sends confirmation message back to applet. Login screen is dismissed;
2. Applet displays tool's parameters dialog;

3. Tool's parameters are sent to servlet which launches appropriate program and makes a new entry to tracker database "Tasks" table;
4. Client applet will be notified about successful start or failure;
5. Client can/must time-to-time check status of his task(s). An e-mail could be sent to client when task is ready;
6. When status of the task is "completed" then user can see the results on his applet, can save them onto his computer.

3.2 Tool encapsulation

In order to integrate different tools, it is necessary to implement additional software layer. Each tool has to be wrapped into Java agent, which allows to adapt the input data to the embedded tool, convert the tool-specific data, simulation results (log files, test vectors, etc), map the control information to the embedded tool, transfer and convert status information (warning and error messages) to be submitted to the user.

Technically simplest way is to encapsulate tool as an entire program. Tool has to be able to run as a batch job. Integration of commercial tools is then also possible. Also embedding of a library (e.g. C, C++ routines) via the Java Native Interface (JNI) could be thinkable and also direct integration of Java-classes and applications (especially for Java software).

3.3 Communication

General communication is based on HTTP protocol. The tools on different computers and on different computing platforms (UNIX, Linux, Windows) can easily change data as serialized Java objects (datagrams). To minimize the implementation effort for parsers, translators and converters XML mark up language is used for configuration files and transmitted data. HTTP protocol allows us also easy firewall traversal as we can use default web server port and Java servlet extensions on web servers as sort of proxies in order to reach intranet resources. There is no need for opening extra ports in the firewall as it is the case in TCP/IP based communication.

3.4 User tracing

User management module is described in this section. Without proper user management anybody in the Internet could possibly use valuable computer resources. Better practice would be to allow registered users access the resources. User tracking system allows us to monitor and control the usage of services. It may allow also billing the business customers. Main goal here was to provide sufficient set of basic functions to allow support user registration, tracking and management. User tracking is database based. Tool execution and data base access over Internet is carried out via Java servlet technology. Below the implementation specifics are given.

As we know, web-based http communication is stateless. This means that we have to keep track about all necessary information. As work tools tend to run long, then normal user's http session is not valid for such time period and result data is

lost. We want to provide a possibility for user to come back online later to check his results. Therefore we need to identify (track) users and save all their relevant data. Using so called "cookies" could be one solution, but database approach offers many advantages like powerful SQL query mechanism, speed, reliability, consistency of data and ease of use.

User tracing module has open architecture, general API (application programming interface). With slight modifications it is also usable for any similar web-based system, where user tracking is needed. It has 3 layers: presentation layer (user tier), business logic tier (data base queries, etc.), physical database (MySQL-platform independent open source DB).

First two layers are implemented in Java programming language. User is accessing database via presentation layer, not directly. This makes architecture open. User tier consists several functions to run business layer queries. For example, we could have different user interfaces for different applications. Then if database structure or business logic changes, we don't have to change our user interfaces. More over- it is easy to introduce user tracking facility to new applications. It is much easier to invoke appropriate function (command), than construct a new query every time a new application needs one.

3.5 User interface

Graphical User interface (GUI) is based on collection of Java applets, which can be integrated into HTML page when needed (e-learning solutions). GUI applet reads the layout properties (field names, default values, etc) from initialization file. It would be easy for non-qualified Java programmer to introduce new tools into web-based environment by modifying initialization fail only. Features of GUI are following:

- Reading in problem description including data from project file;
- Selecting a service (tool) from the set of available services;
- Buttons to start and stop the tools;
- A console window collects all messages from the running tools;
- The visualization of all results (test vectors, statistic information);
- Downloading results: user clicks appropriate button which displays html page containing appropriate link;

4. TEST GENERATION SYSTEMS

Below the toolsets integrated into proposed web environment are presented.

4.1 Hierarchical ATPG DECIDER

DECIDER (Raik, 2000) 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) for testing the given FU is created by implementing 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 TPG and the FU under test.

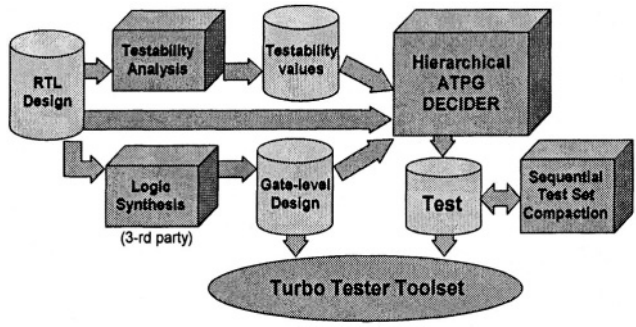

Figure 3 – Hierarchical ATPG workflow

If the filter does not allow finding 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.

The general structure of the hierarchical ATPG system is shown in Figure 3. In addition to the test pattern generator it contains tools for testability analysis and test set compaction. Design interfaces from VHDL and EDIF are available (not shown in the figure).

4.2 Logic-Level ATPG System Turbo Tester

The Turbo Tester (TT, shortly) Automatic Test Pattern Generator software (Turbo Tester) developed in Tallinn Technical University consists of a set of tools (see Figure 4) for solving different test related tasks at the gate-level. The test generation and fault simulation tools of TT are used also by hierarchical ATPG described in section 4.1.

All the TT tools operate on the model of Structurally Synthesized Binary Decision Diagrams i.e. SSBDD (Ubar, 1998). The tools run on the structural logic level. Two possibilities are available - gate-level and macro-level. In the latter, the gate network is transformed into macro network where each macro represents a tree-like sub-

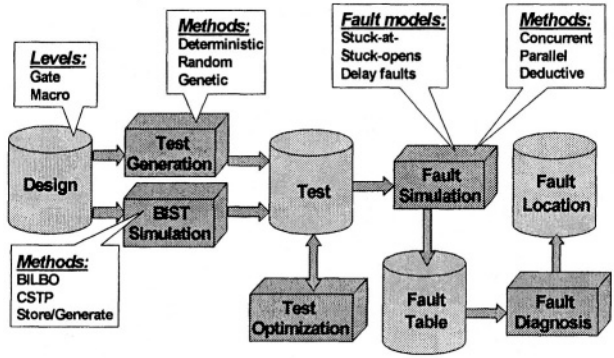

Figure 4 – Turbo Tester workflow

network. 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.

5. EXPERIMENTS

Table 1 shows the comparison of the hierarchical ATPG and state-of-the-art tools GATEST (Rudnick, 1994) and HITEC (Niermann, 1991) on six examples. As the experiments showed, the ATPG integrated to new web environment is considerably faster than other tools, obtaining higher fault coverage for the larger designs in the benchmark set. The experiments were run on a 366 MHz SUN UltraSPARC 60 server with 512 MB RAM under SOLARIS 2.8 operating system. Actual stuck-at fault coverage of the test patterns generated by all the three tools were measured by the fault simulator of Turbo Tester.

Table 1 – Comparison of sequential circuit test generation tools

circuit	faults	HITEC (2)		GATEST (4)		DECIDER (3)	
		f.c.,%	time, s	f.c., %	time, s	f.c., %	time, s
Gcd	454	81.1	169.5	91.0	75	89.9	129.8
Sosq	1938	77.3	728.4	79.9	739	80.1	129.6
mult8x8	2036	65.9	1243	69.2	822	74.7	93.7
Ellipf	5388	87.9	2090	94.7	6229	95.0	1258.9
Risc	6434	52.8	49,020	96.0	2459	96.5	150.5
Diffeq	10,008	96.2	13,320	96.4	3000	97.1	453.7

6. CONCLUSIONS

We have presented the solution for web based remote use of standalone command line programs. Although this solution was particularly implemented for digital electronics test tools, we believe that current solution in general works for other domains with different standalone programs as well. Although, special attention is probably needed for interoperability of the single tools when subsequent execution of the tools is needed. Here, tool interaction is simpler since they belong to one family and they naturally have common representation format.

The web environment was built using the client-server three-tier concept. There was one master server, several application servers, and arbitrary number of clients. For users just ordinary web browser is sufficient. HTML pages and Java applets were used for user GUI. Java servlets were proposed to execute programs on application servers and open source database MySQL as database backend for user tracking and simultaneous access management. Database connection library was designed reusability in mind. Proposed Java servlets are multithreaded and there exists database connection pool to speed up simultaneous access. There is also task queue management. Firewall traversal should not be problem usually for proposed system.

7. ACNOWLEDGEMENTS

The research has been partly funded by European projects REASON (IST-2000-30193), Evikings II (IST-2001-37592) and ESF grants G5637, G4300, G5649 and by mobility grant of Estonian Information Technology Foundation EITSA.

8. REFERENCES

1. MOSCITO. http://www.eas.iis.fhg.de/solutions/moscito
2. Niermann TM, Patel J. "HITEC: A test generation package for sequential circuits", Proc. European Conf. Design Automation (EDAC), pp.214-218,1991
3. Raik J, Ubar R. "Fast Test Pattern Generation for Sequential Circuits Using Decision Diagram Representations.", JETTA, Kluwer Academic Publishers. Vol. 16, No. 3, pp. 213-226, June, 2000
4. Rudnick EM, Patel J, Greenstein GS. T.M.Niermann: Sequential Circuit Test Generation in a Genetic Algorithm framework. DAC., pp. 698-704, 1994
5. Schneider A et. al. Internet-based Collaborative Test Generation with MOSCITO. Proc. of DATE'02, Paris, France, March 4-8, 2002, pp.221-226.
6. Turbo Tester. http://www.pld.ttu.ee/tt
7. 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