Learning Digital Test and Diagnostics via Internet

R.Ubar[1], A.Jutman[1], M.Kruus[1], E.Orasson[1], S.Devadze[1], H.-D.Wuttke[2]
[1] Tallinn University of Technology, Tallinn, Estonia
[2] Technical University of Ilmenau, Ilmenau, Germany

***Abstract*— An environment targeted to e-learning is presented for teaching design and test of electronic systems. The environment consists of a set of Java applets, and of web based access to the hardware equipments, which can be used in the classroom, for learning at home, in laboratory research and training, or for carrying out testing of students during exams. The tools support university courses on digital electronics, computer hardware, testing and design for testability to learn by hands-on exercises how to design digital systems, how to make them testable, how to build self-testing systems, how to generate test patterns, how to analyze the quality of tests, and how to localize faults in hardware. The tasks chosen for hands-on training represent simultaneously research problems, which allow to fostering in students critical thinking, problem solving skills and creativity.**

***Index Terms*— Tools for interactive learning and teaching, Web and Computer-based learning, Re-usable Learning Objects.**

I. Introduction

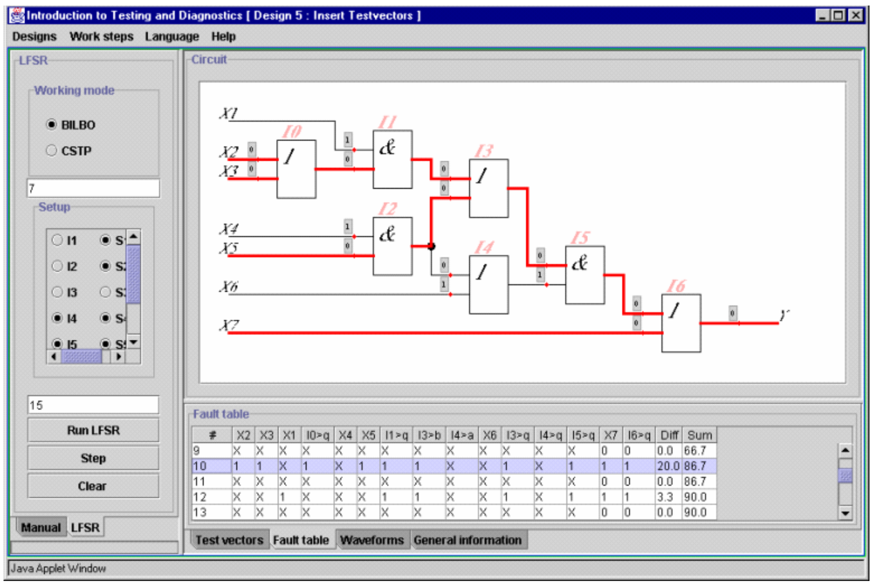
The increasing complexity of digital systems accompanied by entering the era of Systems-on-Chips (SoC) and Networks-on-Chips (NoC) has made testing and fault diagnosis in electronic systems one of the most complicated and time-consuming problems in electronics design and manufacturing. The more complex are getting electronics systems, the more important will be the problems of test and design for testability because of the very high cost of testing electronic products. At present, most system designers and electronics engineers know little about testing. This is because in the today's university curricula test issues are usually neglected. Students learn how to design electronic systems but not how to test them. The next generation of engineers involved with SoC and NoC technologies should be made better aware of the importance of test, and be trained much more in test technology to enable them to develop, design and produce high quality and defect-free products.

Today, in teaching digital design, mainly logic simulation methods are the objective and tool for investigating the behavior of systems. For analyzing defective microelectronic systems today diagnostic methods based on logic simulation only are not sufficient any more. The gigahertz operating frequencies and ultra high scale of integration in modern microelectronic systems make the physical level defects manifest themselves in new unusual and unconsidered ways. The good old logic level fault models such as stuck-at or even bridging fault models do not guarantee full defect coverage anymore [1]. Even rather easy to model defects like opens and zero-resistive shorts might behave in an unexpected way when their behavior is purely considered on logic level. Moreover, due to process variations and other phenomena, even the results of rather complicated electrical simulation methods sometimes do not match with the reality. This causes the necessity to create a possibility to learn also in related university courses more practically the effects of realistic defects in microelectronics systems.

In this paper, we present a conception and means to improve the skills of students educated for hardware and SoC design in test related topics. We present an interactive learning method based on using so-called *living pictures* [2]. The method presented here deals with the goal, to put interactive teaching modules to the Internet that can be used in a lecture as well as for individual self-studies. The modules can be accessed independent of time and place. On one hand, teachers can demonstrate different examples and procedures of test related topics using living pictures during the lessons. On the other hand, students can use the same simulations on their home computer, if the living pictures are available on the Internet. Finally, the same modules can be used during examination.

The core of the teaching concept presented are JAVA-applets (interactive modules) running on any browser connected to the Internet. These applets, tailored for educational purposes, support the test methods as tools for special tasks. In contrast to commercial available tools, they are easy to use because they support only the method that the teacher wants to teach and have a graphical user interface. That interface also visualizes details of the underlying algorithms (e.g. step-by-step execution). We call this type of applet "Living Pictures". By using interaction possibilities the students can produce input test stimuli, watch the behavior of the circuit in the fault-free mode and in different faulty modes. In the paper, different learning tasks and exercises are described, which make use of the applets. We developed two types of such applets: applets for investigating and learning test problems in simple gate level circuits [3], and applets for practicing design and test problems in digital systems that are more complex and consist of a control and a data path [4].

For investigating realistic physical defects in microelectronic circuits, a novel web based hardware/ software (HW/SW) environment has been developed [5, 6]. The core of the environment is a special educational chip called “DefSim” where a large variety of shorts and


Introduction to Testing and Diagnostics [Design 5 : Insert Testvectors]
Designs Work steps Language Help
LFSR
Working mode
BILBO
CSTP
Setup
Run LFSR
Step
Clear
Manual LFSR
Circuit
Java Applet Window

Fault table

#	X2	X3	X1	I0>q	X4	X5	I1>q	I3>b	I4>a	X6	I3>q	I4>q	I5>q	X7	I6>q	Diff	Sum
9	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0.0	66.7
10	1	1	X	1	X	1	1	1	X	X	1	X	1	1	1	20.0	86.7
11	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0.0	86.7
12	X	X	1	X	X	X	1	1	X	X	1	X	1	1	1	3.3	90.0
13	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0.0	90.0

Test vectors | Fault table | Waveforms | General information

Figure 1. Living pictures for investigating logic circuits

opens can be physically inserted into a set of digital standard cells and small circuits [7]. This environment allows carrying out remote experiments via Internet with different realistic defects selected remotely in the DefSim.

We organized the paper in the following way: Section 2 presents the applets for e-learning of logic level diagnostics. Section 3 describes different learning scenarios for logic level testing and fault location. Section 4 is devoted to investigating the problems of higher-level test. In Section 5, we discuss the ideas and possibilities of remote experimenting with real physical defects in electronic circuits. Section 6 concludes the paper.

II. Environment of Web-based Learning Tools for Design and Test

Learning in practical situations and learning by doing is an efficient way to learn because a student will form mental pictures about the things to be learned, and they will be remembered better, too.

The e-learning software developed supports the action-based training in digital design and test via Internet. It offers a set of tools to inspect the different objectives of testing digital logic to be learned, access to multiple learning modules, a big library of examples and the possibility to generate new personal examples. It provides easy action and reaction (click and watch) by using "living pictures", the possibility of distance learning, and learning by doing. The core of that concept are Java-applets (the interactive modules) running over network, using standard browsers like Netscape and Internet Explorer with Java 1.2 runtime plug-in, or with Java 2 applet viewer.

Different types of applets have been created for learning design and test of electronic circuits or systems at the low logic level and at the higher register transfer or behavior levels.

One class of applets developed can be used for teaching and learning the basics of logic level digital test and testable design as illustrative tool explaining the problems of fault simulation, test generation, design for testability and fault diagnosis. The work window of this program (*Fig*.1) consists of a test pattern insertion panel, a view panel for design schematics, and a view panel for simulation results like waveforms, fault tables, diagnostic information etc. Test patterns can be inserted manually or generated automatically by different methods. The boxes at the lines on schematics are clickable for inserting or viewing signals during the test generation or fault diagnosis. The described environment is accessible in [3].

The applets can be used for carrying out different teaching or learning scenarios. In each scenario, they can be re-used and embedded into another context. Thus, they are a kind of re-usable learning objects (RLOs). For example, logic level research involves the following tasks:

- manual test generation for a given gate-level circuit,
- generating tests with automatic tools and analyze the quality of tests by fault simulation,
- generating diagnostic tables, and fault locating procedures,
- Finding a fault in a circuit, creating cost-effective procedures for fault diagnosis.

In *Fig*.1, a pattern is applied on the inputs of the circuit. On the upper view panel, the lines tested by the applied pattern are highlighted. On the lower panel, a fault table is shown. The rows correspond to the test patterns and the columns correspond to the lines of the circuit. The entry *x* in the table means that a particular line is not tested by the given pattern. The entry 0 or 1 means that a fault stuck-at-0 or stuck-at-1 of the particular line is tested by the given pattern. In the last column, also the percentages of faults detected by the test patterns, used up to the actual pattern, are shown.

The task of test generation consists of finding a set of test patterns, which is able to detect all the possible faults in the circuit. The students can try to minimize such a set of test patterns. Another learning scenario is to generate a set of test patterns, which can be used to locate any possible fault.

Some of the tasks can be organized in a gaming style or as a competition between students. For example, a fault can be inserted into a circuit by the teacher, and a competition between students will be thereafter carried out in a manner who is the first who can localize the fault i.e. who will be able to use the minimum search steps. This way of working with applets makes learning even more exciting.

III. LEARNING SCENARIOS OF LOGIC LEVEL TEST

We start working with the applet by selecting a circuit from a set of predefined ones. Then we can carry out different experiments with this circuit by selecting a proper working mode from the mode menu.

A. *References*

There are two methods possible for test vector generation using the applet:

- direct test vector insertion on inputs (on the vector insertion panel), and
- test generation by path activation in the circuit (on the schematics panel).

In the direct *test vector insertion mode* we can choose test vectors either automatically by using Linear Feedback Shift Register (LFSR) as a pseudorandom pattern generator [8], or by inserting vectors manually.

In the manual mode, we generate step-by-step input patterns, which are simultaneously simulated. The boxes at the lines on the schematics sub panel display the result of simulation – the values of internal signals on the connections. The waveforms can be viewed on the data sub panel.

When using LFSR, we have to specify the initial state of LFSR, to select a proper feedback structure of LFSR (different structures provide different quality of test patterns), and to specify the length of the desired test sequence. By changing the settings on the vector insertion panel, we can emulate different feedback structures of the chosen architecture.

The quality of the test patterns now can be analyzed by fault simulation. In the *fault simulation mode*, a fault table is generated and shown on the data panel for all the test vectors created at the given moment. For example, in the fault table on the lower panel of Fig. 1, the results of the fault simulation of five test patterns 9,10,11,12, and 13 are shown. By selecting a test vector in the fault table, all the detected faults will be highlighted by colors on the schematic panel. For example, on the upper schematic panel of *Fig*.1 we see the activated paths and all the detected faults for the vector number 10 selected in the fault table. The pattern itself can be viewed by clicking on "*Test vectors*" on the lower panel. The values in boxes at the lines of the circuit show the behavior of lines for the selected test vector. The value 0 (or 1) in the boxes means that the fault stuck-at-1 (or stuck-at-0) is activated and can be detected by the selected test pattern.

In the *test generation* mode we choose a target fault in the schematic and create step by step proper activated paths in the circuit to activate the fault at his site and to propagate the error signals caused by the fault towards the output by clicking the needed values into boxes on the lines. From these values finally, an input vector will be deduced. The colors on lines help us to understand the

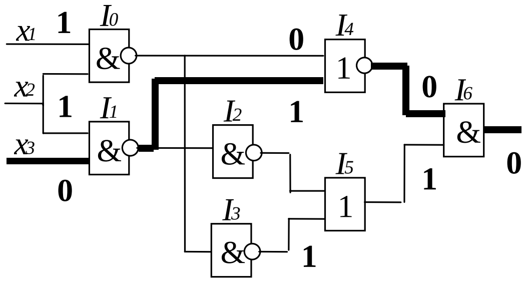


Figure 2. Test generation by path activation (fault free behavior)

current status of the task: red and green lines mark activated faults and activated paths. The inconsistencies of the signal values are highlighted by blue color. As the result of the procedure, it generates a test pattern. The detected faults are displayed also on the data panel in form of a row in the fault table.

For example, to generate a test pattern for the fault $x_3 \equiv 1$ in Fig. *2*, first, a signal with opposite value 0 to the faulty value 1 should be inserted to x_3 by clicking the box on the line x_3. Then, the faulty signal of x_3 should be propagated to the output of the circuit. By inserting the value 1 on the line x_2 the faulty signal from x_3 is propagated through the gate I_1. Next, by inserting the value 0 on the upper input of gate I_4 the faulty signal is propagated through the gate I_4. Finally, by inserting the value 1 on the lower input of gate I_6 the faulty signal is propagated through the gate I_6 to the output *y*. The activated path is shown in Fig.2 by bold lines. All the inserted values should be properly justified step by step by other signals moving towards the inputs. As the result, a test pattern will be created on the inputs. For this example, the test pattern $x_1x_2x_3 = 110$ will be found.

B. *Fault diagnosis*

In the *fault diagnosis mode*, we need at first to create a fault table by running the fault simulator for a set of previously generated test vectors. In the diagnosis mode, an unknown fault should be first introduced into the circuit.

The following diagnosis strategies chosen from the scenarios' menu can be investigated: combinational and sequential diagnosis.

For learning the *combinational* diagnostic strategy, a single vector or a subset of vectors can be selected and applied to the erroneous circuit (by imitating test experiments). The applet shows the results of testing, and displays the subset of suspected faults. If the subset contains a single fault, the diagnostic task is solved. If the subset contains more than one fault, the diagnostic resolution should be improved. To improve the resolution, additional test vector(s) may be generated and used in the repeated test experiment.

Sequential diagnosis (guided-probe testing) is based on the guided probing strategy. A test pattern is applied and the expected behavior of the circuit is displayed. The principle of guided-probe testing is to back-trace an error from the output where it has been observed to its source (faulty gate). By clicking on the connection boxes, the real values of signals of the faulty circuit can be measured. A faulty gate is located if it has been found that the signal on

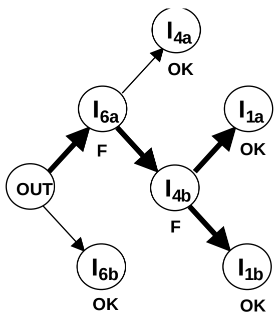


Figure 3. Fault diagnosis scenario

the output of the gate is faulty, while only correct expected signals are observed at its inputs.

The main didactically point in learning of the both diagnostic strategies is to try to localize the fault by as few as possible test vectors (in the combinational approach) or by as few measurements (in the case of sequential approach) as possible. In this task, students can carry out a competition: The student who needs fewer measurements to localize the fault will be the winner.

As an example, let us see the procedure of sequential fault localization by pinpointing the signals in the circuit for the case of test pattern $x_1x_2x_3$ = 110 represented in *Fig*.2. Suppose the gate I_1 is faulty and produces a wrong signal 0 instead of the expected 1. The erroneous signal is propagated by this test pattern up to the output (as a wrong signal 1 instead of the expected 0) and the error can be observed there.

Now three possible fault location procedures can be imagined for this particular case covered by the diagnostic tree in *Fig*.3. The nodes in the graph in *Fig*.3 mean signal measurements on the inputs of the gates I_{jk}. The inputs of a gate I_j are denoted from above down by $k = a, b$.

First, we may use a trivial backtracing procedure of erroneous signals shown as the full search tree in *Fig*.3. In the worst case, we may click to measure the signals on all of the 6 nodes of the tree. Starting with the node I_{6b} we observe a correct signal. Then, we try the next input I_{6a} of the gate I_6 where the error is detected. We continue now backtracing in the node $I_{4\,a}$. Then, we try the next input I_{4b} of the gate I_4 where again the error is detected. Now we backtrace to the inputs of the gate I_1 where no errors are found. This means that we have located the erroneous gate I_1 by 6 measurements.

Secondly, we may analyze the fault activation conditions on the inputs of gates in the back-trace tree for local optimization (at each gate) of the search process. For example, based on the input signals of the gate I_6 we can reason that if an erroneous signal has been propagated through the gate, it can originate only from the input I_{6a}. In other words, we can skip pinpointing of the node I_{6b} because as the result of reasoning we can learn that an error at the input I_{6b} of the gate I_6 would not be able to cause a change of the signal on the output of the gate I_6. In the same way, we realize that the measurement of I_{4a} is also not needed. As the result, the backtracing procedure will cost only 4 measurements (see the bold lines in *Fig*.3).

There is a third possibility to analyze the situation for a global optimization of the searching process. We can find out by reasoning the simulated state of the circuit that the fault should locate somewhere on the bold lines in *Fig*.2. We can use now the well-known *divide and conquer* approach. By measuring the value of I_{4b} in the middle of the activated path, we can divide all the possible faults into two equal groups. In the case of correct signal at I_{4b}, we have to proceed towards the output and pinpoint the value of I_{6a} to determine which of the gates I_4 or I_6 is faulty. In the case of erroneous signal, we have to continue towards the inputs and measure the value of x_3 to determine if either the input x_3 or the gate I_1 is faulty. In both cases, we need only 2 measurements to locate the fault instead of the 6 measurements of the full search.

With this little example, we managed to show that the fault diagnosis process could be regarded as a demanding mental experiment. A competition can be organized between students to make the learning procedure an exciting event.

IV. LEARNING HIGHER LEVEL TEST

Entering the SoC era with its new concepts involves teaching the design of electronic systems on higher levels of abstraction like register transfer level (RTL), instruction set architecture or behavioral levels. We have developed another type of Java applets for learning high-level design and test in control intensive digital systems. Students can exercise RTL implementations of more complex functionalities represented by data flow graphs or micro-programs (like multiplication, division, signal processing algorithms etc.).

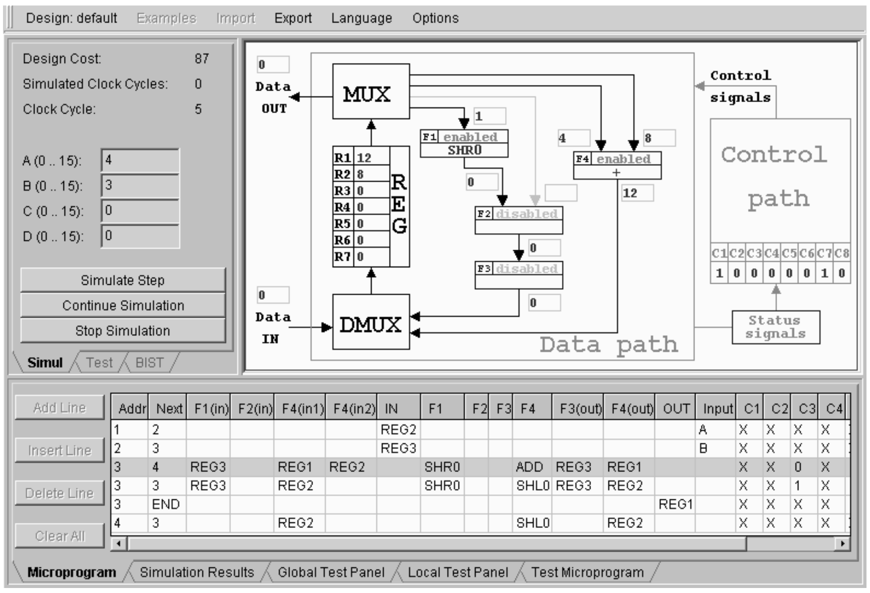
Such topics as design of data-flows and microprograms of computing algorithms, investigation of tradeoffs between speed and hardware cost in digital design, RTL simulation, design for testability, test generation, built-in self-test (BIST), diagnostic analysis and other related problems are covered by these applets.

A. Description of the RTL design and test applet

The work window of the applet (*Fig*.4) consists of three parts – a control panel, a view panel for design schematics, and a panel for microprogram development and viewing simulation results. The described environment is also accessible at [4].

A structure of the *data-path* and the needed functional units for implementing particular microoperations can be found from the library. Different architectures can be explored and experimented for implementation of a given set of functions or algorithms.

Each *functional unit* (FU) of the data path *F*1 … *F*4 in *Fig*.4 contains a number of microoperations (functions: unary and binary), which are labeled by corresponding control signals activating a chosen function. There is an overlap between possible functions of *F*4 and of *F*1, *F*2 and *F*3 to allow a parallelization of a given algorithm. The student can select one or more microoperations for each unit of the data path when implementing his own algorithm (like multiplication, division etc.). Each micro-operation has a gate-level implementation, and the number of gates determines its cost and at the end, the final hardware cost of the system. The student can select, thus, a particular implementation of his algorithm meeting either the cost or timing requirements, or he can compare different hardware solutions for the given algorithm and find out the design tradeoffs. For any chosen architecture, the system calculates the cost of the hardware, and the



Design Cost: 87
Simulated Clock Cycles: 0
Clock Cycle: 5

A (0 .. 15): 4
B (0 .. 15): 3
C (0 .. 15): 0
D (0 .. 15): 0

Addr	Next	F1(in)	F2(in)	F4(in1)	F4(in2)	IN	F1	F2	F3	F4	F3(out)	F4(out)	OUT	Input	C1	C2	C3	C4
1	2					REG2								A	X	X	X	X
2	3					REG3								B	X	X	X	X
3	4	REG3		REG1	REG2		SHR0			ADD	REG3	REG1			X	X	0	X
3	3	REG3		REG2			SHR0			SHL0	REG3	REG2			X	X	1	X
3	END												REG1		X	X	X	X
4	3			REG2						SHL0		REG2			X	X	X	X

Figure 4. Applets for investigating digital systems

speed of processing (number of clock cycles needed by the developed micro-program) can be measured by simulation. The simulation is supported by an RT-level model of the system as a whole and by gate-level models of each microoperation in each FU.

The *control path* is a microprogrammed controller, which implements a Mealy FSM (Final State Machine). The controller consists of a microprogram table and an interpreter. The microprogram is developed by the user to realize a given algorithm based on the selected resources of the data path. The user fills in the rows of microprogram table, which contains information about the address of the current and the next microinstruction, multiplexer (MUX) and de-multiplexer (DMUX) configurations, Data IN values, selection of functions in FUs (*F*1 to *F*4) at each microinstruction, and status signal configuration.

The *Fig*.4 presents an example of algorithm of multiplication of two operands A and B. The result of the multiplication is stored in REG1 and fed out to the data output.

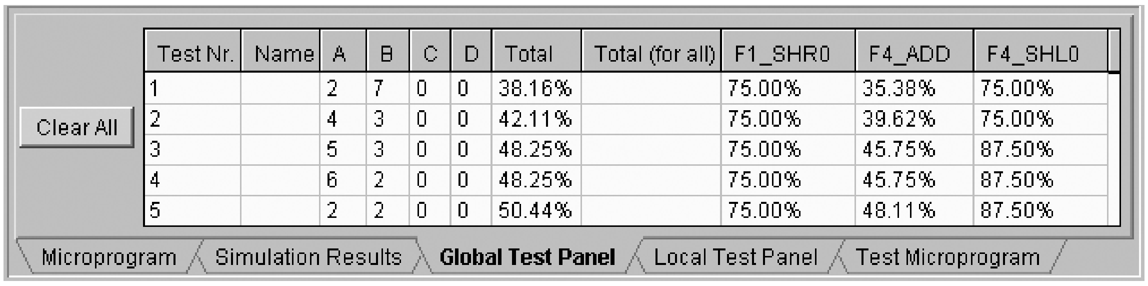
The *RT-Level simulation* is carried out at the higher level by using corresponding to functional units Java subroutines which are activated according to condition values by the control signals in the order given in the microprogram table. The simulation data is stored in the Simulation Results sub panel. This data reflects the states of all the registers, outputs of all the functional blocks, data input and output of the device, current states at each clock cycle and the condition signals. The simulation data can be used by the student as a debugging info as well as for the improving the efficiency: the speed or the cost of the system.

With this applet we are aimed at showing a variety of different modern testing techniques including functional and deterministic testing, a number of Built-in Self-Test (BIST) solutions.

B. Learning test by the Register Transfer Level Applet

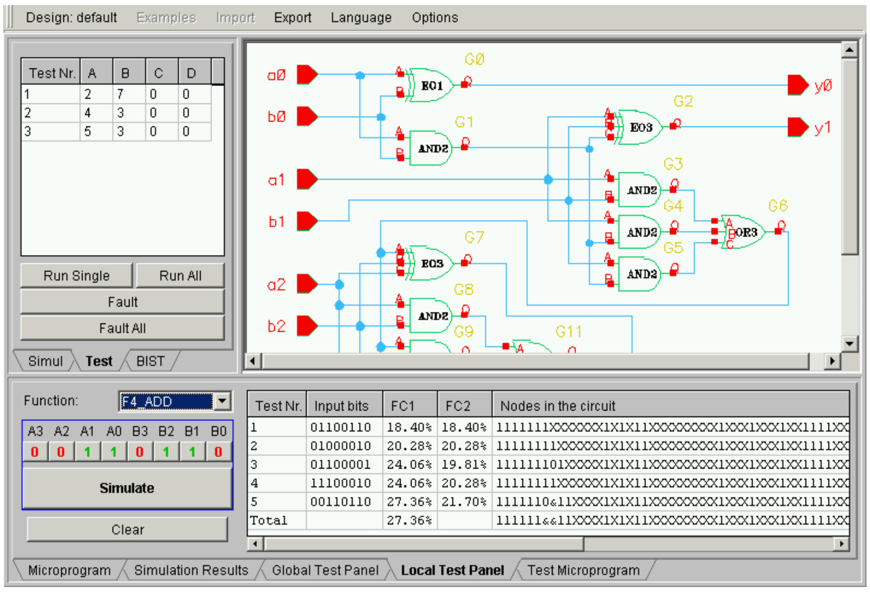
Functional Test. In this mode the cheapest test technique is investigated, which does not require designing special test programs and embedding of special test structures into the system. The required level of fault coverage must be achieved by a smart selection of input data. The sole checkpoint allowed for catching the fault is the data path primary output. Moreover, it only can be observed at the time when the microprogram outputs the final result.

The fault simulation information is presented at the Global Test Panel (*Fig*.5). The input operands (A, B, C, D) are specified first. The same microprogram is used then repeatedly for fault simulation for all the input data. The fault coverage is calculated for each selected FU and for the whole system as well. The cumulative fault coverage for each input vector is provided in the Global Fault Coverage table (*Fig*.5).



Test Nr.	Name	A	B	C	D	Total	Total (for all)	F1_SHR0	F4_ADD	F4_SHL0
1		2	7	0	0	38.16%		75.00%	35.38%	75.00%
2		4	3	0	0	42.11%		75.00%	39.62%	75.00%
3		5	3	0	0	48.25%		75.00%	45.75%	87.50%
4		6	2	0	0	48.25%		75.00%	45.75%	87.50%
5		2	2	0	0	50.44%		75.00%	48.11%	87.50%

Figure 5. Global Fault Coverage table



Test Nr.	A	B	C	D
1	2	7	0	0
2	4	3	0	0
3	5	3	0	0

A3	A2	A1	A0	B3	B2	B1	B0
0	0	1	1	0	1	1	0

Test Nr.	Input bits	FC1	FC2	Nodes in the circuit
1	01100110	18.40%	18.40%	1111111XXXXXX1X1X11XXXXXXXXX1XXX1XXX1XX1111XX
2	01000010	20.28%	20.28%	11111111XXXXX1X1X11XXXXXXXXX1XXX1XXX1XX1111XX
3	01100001	24.06%	19.81%	111111101XXXX1X1X11XXXXXXXXX1XXX1XXX1XX1111XX
4	11100010	24.06%	20.28%	11111111XXXXX1X1X11XXXXXXXXX1XXX1XXX1XX1111XX
5	00110110	27.36%	21.70%	1111110&11XXX1X1X11XXXXXXXXX1XXX1XXX1XX1111XX
Total		27.36%		111111&&11XXX1X1X11XXXXXXXXX1XXX1XXX1XX1111XX

Figure 6. Deterministic test pattern generation

The primary task of the student during investigation of functional testing is the selection of good operands (A, B, C, D) in order to achieve the target fault coverage as fast as possible. For simpler designs, this technique can be feasible. However, for structures that are more complicated we have to use something more sophisticated.

Deterministic Test. This mode is aimed at a gate-level test generation and fault simulation for each selected FU separately. For learning these activities, the applets described in Section 3 can be used. The simulation results for FUs are provided in the fault table at the Local Test Panel (*Fig*.6). For each vector, the fault coverage (FC2) is calculated and the information on tested nodes is given. The cumulative fault coverage (FC1) is also shown for each simulation step. The hierarchical RT-level fault simulation is applied in order to evaluate the global fault coverage of those vectors for the data path as a whole. For this purposes a hierarchical test program should be composed for each selected FU. The simulation data will be reflected in the Global Test Panel (*Fig*.5) in the same way as it is done in the Functional Test mode.

In order to help the user to generate gate-level test vectors for FUs, the gate-level schematic of the currently selected FU is displayed. The user selects a target fault and generates a test vector. After pressing the “Simulate” button this vector is fault simulated at the gate level and the results (local fault coverage) are added into the fault table. At the same time, the same vector is sent to RT-level hierarchical fault simulator in order to fill in the Global Test Panel. This panel shows the vector as two decimal operands. The test microprogram, used for RT-level fault simulation, must provide a good access to the selected FU. A simple version of such a program is generated automatically. It can be used as a template by a student in order to develop a more sophisticated test program if needed.

The Built-in Self-Test (BIST) Mode. The Deterministic Test mode is one of the most efficient ways of testing. However, it does not provide access to internal signals of the system under test. This problem is addressed by various BIST solutions. Usually it is a scan-path with a random test pattern generator (TPG) and one or more signature analyzers (SA). By the scan-path technology, the inputs and the outputs of the combinational blocks in the data path are directly accessible by TPGs and SAs [8].

Our teaching system allows reconfiguration of internal registers into the BIST mode. Depending on the chosen BIST method, some of them can perform functions of TPG, SA or both. When the configuration is chosen, the fault simulation will be performed and the results are displayed in the way similar to the one used in Functional and Deterministic test modes.

The modes described above help to illustrate the way of operation of different BIST structures and show how their efficiency depends on the TPG configuration. The selection of a good configuration for each selected FU is the main problem to solve by the student. Another task is the selection of such a single TPG configuration that allows testing all of the FUs in the shortest possible time.

Different training and research scenarios of testing the design can be exercised by the applet. Generating high-level or hierarchical test programs and analyzing their quality develops real skills in the student, which are needed in testing and diagnosing electronic products in the industry.

V. WEB-BASED ENVIRONMENT FOR EXERCISING REAL DEFECTS

The gigahertz operating frequencies and high integration levels of modern microelectronic systems make the physical level defects manifest themselves in a new unusual way. The traditional logic level fault models like stuck-at or even bridging ones do not guarantee full defect coverage anymore [1]. Even rather easy to model defects like opens and zero-resistive shorts might behave in an unexpected way when their behavior is purely considered on logic level. Moreover, due to process variations and other phenomena, even the results of rather complicated electrical simulation methods sometimes do not match with the reality.

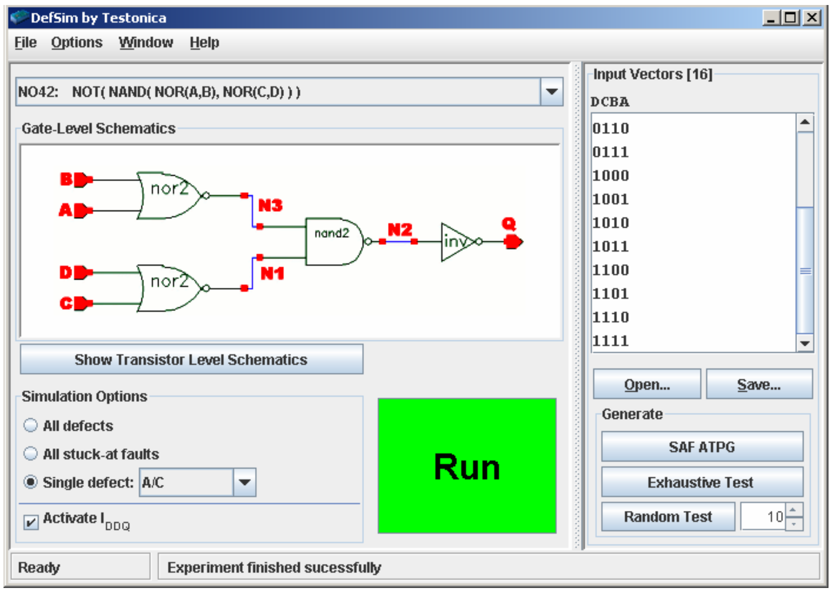


Figure 7. Graphical User Interface to DefSim

These facts fully agree with the results one can observe using our measurement environment for real CMOS defects – DefSim [5, 6]. The central element of the DefSim environment is the IC with a large variety of shorts and opens physically inserted into a set of digital standard cells and small circuits. The IC is attached to a dedicated measurement box serving as an interface to the computer. The box supports two measurement modes – voltage and I_{DDQ} testing [9].The communication between the DefSim hardware and software goes through the USB port.

The DefSim IC has three main structural parts: a matrix of simple digital circuits, addressing mechanism, and a measurement circuitry [7]. Each circuit from the range is implemented in many copies where one of them is correct and all the others are intentionally defective. All such defects have different locations within a corresponding copy of the circuit. During chip operation, only one such copy can be active at a time.

Currently two types of defects are implemented in DefSim: opens and hard shorts in conducting layers. These defects are located both inside logic gates and upon (or between) signal lines outside the gates. It is possible to select any defect of interest by addressing a corresponding copy of the circuit. Then the user can apply an arbitrary input test sequence and measure the circuit's response to it in terms of both the binary logic values and current levels (IDDQ). It is also possible to compare its behavior over the correct copy of the same circuit.

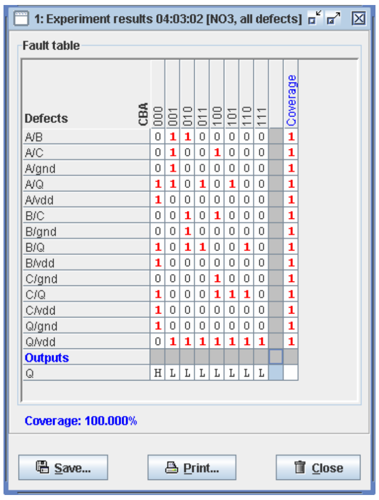
The DefSim software package provides a very convenient access to the features of the IC and ensures a smooth way of going through educational scenarios for students [6].

As the first step, the user has to select a target circuit to work with. Then, the list of implemented defects for this circuit becomes available. The user can work either with a single specific defect or with a group of defects simultaneously. A random defect can be selected too, if defect localization is the intended task.

From the didactical point of view, the DefSim environment targets (but it is not limited to) two main areas of expertise: defect modeling and defect observability. First, the user gets a chance to compare the efficiency of different logic level fault models in terms of their capability to cover all possible shorts and opens in a CMOS circuit. The students will learn in practice that some simple defects represent a real challenge especially from the diagnostics (defect localization) point of view.

Since DefSim environment supports both voltage and I_{DDQ} testing, the user can compare the fault detection efficiency of those test methods as well. In most cases, their effectiveness is noticeably different.

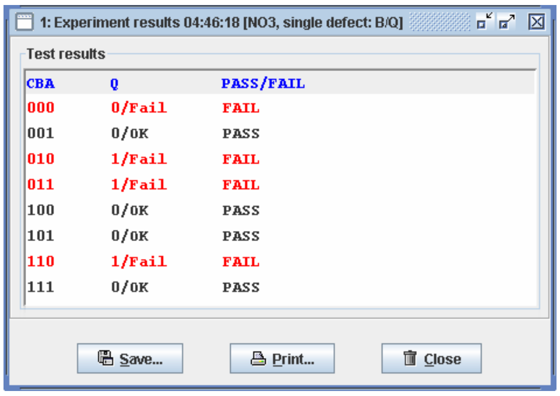
Besides the fault list, two types of schematics are available for each circuit: the logic level scheme and transistor level one. The necessary test patterns can be generated either by the user using these schematics or automatically by the software. Then, the prepared test patterns are sent to the IC and applied to either the selected defect or a group of defects, and consequently, the circuit responses are recorded. The results of measurements are displayed in several different forms (Fig.8). For instance, the user can observe the truth table



Defects / CBA	000	001	010	011	100	101	110	111		Coverage
A/B	0	1	1	0	0	0	0	0		1
A/C	0	1	0	0	1	0	0	0		1
A/gnd	0	1	0	0	0	0	0	0		1
A/Q	1	1	0	1	0	1	0	0		1
A/vdd	1	0	0	0	0	0	0	0		1
B/C	0	0	1	0	1	0	0	0		1
B/gnd	0	0	1	0	0	0	0	0		1
B/Q	1	0	1	1	0	0	1	0		1
B/vdd	1	0	0	0	0	0	0	0		1
C/gnd	0	0	0	0	1	0	0	0		1
C/Q	1	0	0	0	1	1	1	0		1
C/vdd	1	0	0	0	0	0	0	0		1
Q/gnd	1	0	0	0	0	0	0	0		1
Q/vdd	0	1	1	1	1	1	1	1		1
Outputs										
Q	H	L	L	L	L	L	L	L		

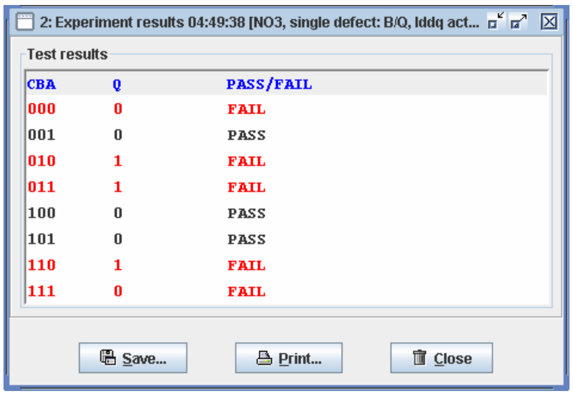
Coverage: 100.000%

Figure 8. Measurement results: defect table



```
CBA   Q        PASS/FAIL
000   0/Fail   FAIL
001   0/OK     PASS
010   1/Fail   FAIL
011   1/Fail   FAIL
100   0/OK     PASS
101   0/OK     PASS
110   1/Fail   FAIL
111   0/OK     PASS
```

a)


2: Experiment results 04:49:38 [NO3, single defect: B/Q, Iddq act...
Test results
Save...
Print...
Close

```
CBA   Q   PASS/FAIL
000   0   FAIL
001   0   PASS
010   1   FAIL
011   1   FAIL
100   0   PASS
101   0   PASS
110   1   FAIL
111   0   FAIL
```

b)

Figure 9. Measurements results windows

for a certain defect, or a fault (defect) table for a group of defects, or IDDQ value info. Such a defect table for a simple NO3 cell (3-input NOR-gate) is given in Fig. 8.

Basically, all the exercises on the DefSim environment can be divided into two groups: less advanced and more advanced ones. Here we will consider a simpler flow as an example. It is targeted on students whose main specialization is not digital test but general microelectronics. A bit more advanced exercises are considered in [5].

- Getting a truth table of good (without defects) CMOS simple and complex standard gates.
- Getting a truth table of good (without defects) small combinational circuits (C17 or CB1).
- Repeating steps 2 and 3 but with a given defect of a certain type in order to observe how the circuit's function is modified by the defect.
- Getting basic knowledge of voltage and current testing principles.

The first and the second step of this flow are illustrated in Fig. 8 based on a simple cell NO3. The last row of this table named "Q" gives correct output values of this cell. The measurement results of a defected circuit instance (short B/Q) are illustrated in *Fig*.9a. There one can see the binary value at the output Q and the fault detection information ("PASS/FAIL") provided for each test vector ABC.

It is not that hard to notice that normally the short B/Q is detected when there are opposite logic levels on the corresponding lines B and Q, where B appears to be a stronger driver than Q. The only exception is the last pattern 111, where 0-value on Q doesn't change despite line B tries to drive it to logic 1. In all other cases 1 on B appears to be stronger than 0 on Q, which actually contradicts with simple and commonly used logic-level models for short defects like Wired-AND and Wired-OR [8]. This is an example supporting the need for accurate and real implementation of CMOS defects in silicon for getting correct understanding of their real behavior.

The last step in the flow for investigating the impact of IDDQ testing is illustrated in *Fig*.9b. It shows the measurement results of static supply current levels on the output of NO3 cell. One can see that in this case the last pattern '111' comes up with the fault detection. Obviously, this is the result of a different degree of observability a particular testing method is featuring.

VI. Conclusions

An environment consisting of web-based applets for hands-on training is presented for improving the skills of students to be educated for electronics design in test related topics. The novelty of this environment is in supporting by a uniform web-based tool-set learning of design and test problems in digital systems. It is also in starting with an investigation of real physical defects at the very low transistor level, reasoning of test and diagnostic related problems at the traditional logic level, and learning how to generate complex test programs and localize faults at higher functional levels of complex digital systems. Based on the described environment, an e-learning conception using simple applets in a form of "living pictures" can be introduced into study programs at technical universities for teaching courses on digital electronics, design of digital systems, testing and design for testability. The applets may be used for solving real research and engineering problems in the field of electronics design and test, which allow fostering in students critical thinking and creativity in an exciting working environment and stimulating atmosphere. This environment has been successfully tested at Tallinn University of Technology.

Acknowledgment

This work was supported by the Thuringian Ministry of Science, Research and Art (Germany), by the Enterprise Estonia funded ELIKO Technology Development Center, and by the Estonian Science Foundation grants G5649 and G5910. The DefSim GUI and server software were developed by Testonica Lab (http://www.testonica.com).

References

[1] D.Kasprowicz, W.A.Pleskacz, "Improvement of Integrated Circuit Testing Reliability by Using the Defect Based Approach," Microelectronics Reliability, PERGAMON – Elsevier Science, vol. 43/6, June 2003, pp. 945-953.

[2] R.Ubar, H.-D.Wuttke, "The DILDIS-Project – Using Applets for More Demonstrative Lectures in Digital Systems Design and Test", Proceedings of the 31st ASEE/IEEE Frontiers in Education Conference, FIE'2001, Oct. 10-13, 2001, Reno, NV, USA, pp.SIE-2-7.

[3] http://www.pld.ttu.ee/applets/td/

[4] http://www.pld.ttu.ee/applets/rtl/

[5] W.A.Pleskacz, T.Borejko, A.Walkanis, V.Stopjakova, A.Jutman, R.Ubar. DefSim: CMOS Defects on Chip for Research and Education. 7th IEEE Latin-American Test Workshop, March 26-29, 2006, Buenos Aires, Argentina, pp.74-79.

[6] http://www.pld.ttu.ee/defsim/

[7] W.A.Pleskacz, T.Borejko, T.Gugala, P.Pizon, V.Stopjakova, ”DefSim – The Educational Integrated Circuit for Defect Simulation,” Proc. of IEEE Int. Conf. on Microelectronic Systems Education, Anaheim, USA, June 2005, pp. 121-122.

[8] M.L. Bushnell, V.D. Agrawal. Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits. Kluwer Academic Publishers, 2000, 690 pp.

[9] V. Stopjakova, H. Manhaeve: “CCII+ Current Conveyor Based BIC Monitor for I_{DDQ} Testing of Complex CMOS Circuits,” *Proc. of European Design & Test Conference*, pp. 266-270, Paris, France, March 1997.

AUTHORS

R. Ubar is with the Computer Engineering Department, Tallinn University of Technology, Raja 15, 13511 Tallinn, Estonia (e-mail: raiub@pld.ttu.ee).

A. Jutman is with the Computer Engineering Department, Tallinn University of Technology, Raja 15, 13511 Tallinn, Estonia (e-mail: artur@pld.ttu.ee).

M. Kruus is with the Computer Engineering Department, Tallinn University of Technology, Raja 15, 13511 Tallinn, Estonia (e-mail: kruus@cc.ttu.ee).

E. Orasson is with the Computer Engineering Department, Tallinn University of Technology, Raja 15, 13511 Tallinn, Estonia (e-mail: elmet@pld.ttu.ee).

S. Devadze is with the Computer Engineering Department, Tallinn University of Technology, Raja 15, 13511 Tallinn, Estonia (e-mail: serega@pld.ttu.ee).

H. D. Wuttke is with the Faculty of Informatics and Automation, Ilmenau Technical University, Post box 100565, D-98684 Ilmenau, Germany (e-mail: dieter.wuttke@tu-ilmenau.de).

Manuscript received 20 May 2006. This work was supported by the Thuringian Ministry of Science, Research and Art (Germany), Enterprise Estonia funded ELIKO Technology Development Center, and by the Estonian Science Foundation grants G5649 and G5910. The DefSim GUI and server software were developed by Testonica Lab (http://www.testonica.com).