Proceedings of the Estonian Academy of Sciences,
2014, **63**, 2, 151–162
doi: 10.3176/proc.2014.2.05
Available online at www.eap.ee/proceedings

COMPUTER
ENGINEERING

Environment for the analysis of functional self-test quality in digital systems

Raimund Ubar*, Sergei Kostin, Helena Kruus, Margit Aarna, and Sergei Devadze

Department of Computer Engineering, Tallinn University of Technology, Akadeemia tee 15a, 12618 Tallinn, Estonia

Received 3 February 2014, revised 24 March 2014, accepted 25 March 2014, available online 20 May 2014

Abstract. Dependability of computer architectures has become one of the most important engineering concerns. One of the possibilities to increase the dependability is to develop architectures with dedicated self-test capabilities which allow achieving high quality of testing in terms of fault coverage. We propose a new methodology for Built-in Self-Test (BIST), which combines the inherent functionality of the architecture with a small amount of pre-generated test data stored in the memory, and uses for monitoring of the test process a restricted number of test points, configured as a set of signature analysers. Contrary to the traditional scan-path based logic BIST, the proposed solution does not need additional hardware for test pattern generation, and will not have any impact on the working performance of the system. On the other hand, testing at normal working conditions allows exercising the system on-line and at-speed, facilitating the detection of dynamic faults like delays and crosstalks to achieve high test quality. The new self-test method is free from the negative aspect of over-testing, compared to the traditional logic BIST approaches. A method is presented to generate optimized test data for selected test routines, and to choose minimum set of test-points for response analysis. A tool framework is proposed to emulate self-testing architectures, and to carry out fault simulation for evaluating the test quality in terms of fault coverage.

Key words: self-test architectures, logic built-in self-test, software based self-test, faults, design for testability.

1. INTRODUCTION

Computer architectures and embedded processors are used in a wide range of application areas, from entertainment (smart phones, portable game consoles), to professional equipment (palmtops, digital cameras), and control systems in various fields (automotive, industry, telecommunications). Safety constraints in many of these areas require periodically checking whether the computing system is still correctly running, or if it is affected by a fault [1].

The technology advancements impose new challenges to testing systems-on-chip as device geometries shrink, and deep-submicron delay defects are becoming more prominent requiring more accurate tests than before [2]. Therefore testing of digital systems in dynamics by so-called at-speed testing has become a must. However, as the speed of microprocessors has reached GHz ranges, at-speed testing is increasingly difficult with traditional external test equipment.

The increasing size and complexity of microprocessor architectures directly reflects in more demanding test generation and application strategies. Modern designs implementing complex architectures, pipelined and superscalar designs have been demonstrated to be random pattern resistant [1]. Use of scan chains has proven to be often inadequate, producing overhead [3], excessive power dissipation during test [4] or leading to overtesting and yield loss [5].

A lot of research has been carried out to relieve the burden of external testers by introducing system self-test approaches like hardware-based Built-in Self-Test (BIST) [6] or Software-Based Self-Test techniques (SBST) [1,7].

In logic BIST [6], typical functions of external testers like test generation and response analysis are carried out on-chip, so that the tester should not handle high-speed signals externally and its role should remain only to send the test signals to the chip under test, and to receive the pass/fail signals. For example, scan-based and logic BIST solutions such as [8] relax the requirements on testers and considerably reduce the overall testing cost.

* Corresponding author, raiub@pld.ttu.ee

An increasingly popular solution to this challenge is based on developing suitable test programs, forcing the processors to execute them, and to check the produced results. This methodology, SBST [1,7] is particularly suitable for being applied at the end of manufacturing and in the field as well, to detect the occurrence of faults, caused by environmental stresses and intrinsic aging in embedded systems.

The question is whether a self-test sequence, running in the system, can adequately exercise its hardware components satisfying the targeted fault coverage requirements. Achieving the test quality target requires a proper test sequence generation, which is the focus of the current paper. It should also be pointed out that the quality of a test is measured not only by its fault coverage, but also by its code size, hardware overhead, and by the test execution time. The goal of the paper is to propose an approach, which combines the ideas of traditional logic BIST and processor based SBST to improve the test quality at less hardware overhead and avoiding performance loss compared to the traditional self-test approaches. We call this approach Functional BIST (FBIST), since the proposed scheme of BIST will use the inherent functionality of the circuit under test

The rest of the paper is organized as follows. In Section 2, an overview about the state-of-the-art of self-test techniques is given, Section 3 introduces a general scheme of the proposed functional self-testing architecture, followed in Section 4 by the framework of its synthesis. Section 5 describes a methodology for high-level Design for Testability (DFT) to improve the fault coverage of FBIST, and Section 6 presents an experimental low-level testability analysis set-up. Experimental results are presented in Section 7, and Section 8 concludes the paper.

2. STATE-OF-THE-ART OF SELF-TEST TECHNIQUES

In traditional logic BIST, test pattern generation is mostly performed by Linear Feedback Shift Registers (LFSR) [6], cellular automata [9], or multifunctional registers like BILBO (Built-in Logic Block Observer) [10] to apply pseudorandom patterns to the Circuit Under Test (CUT) and to analyse its output responses.

Unfortunately, many circuits contain Random-Pattern-Resistant (RPR) faults which limit the fault coverage that can be achieved by using traditional BIST, based on pseudorandom patterns. They demand as well very long test sequences and long test application times in addition to increased area overhead. Improvements have been achieved by modifying the CUT by either inserting test points [11–13], using Weighted Pseudo-random Sequences (WPS) [14], or by redesigning CUT to improve fault coverage [15]. The drawback of these techniques is that they generally involve additional logic to circuitry that can degrade performance.

Another method to improve fault coverage is to use "mixed mode" or hybrid approaches [16–22], where pseudorandom data are combined with deterministic ones to improve detection of RPR faults, and compared to WPS less additional hardware is required. The pseudorandom and deterministic data may be combined in different ways like using ROM compression [16], LFSR reseeding [14] either by bit-flipping [17] or bit-fixing [18], multi-polynomial scheme [19], or embedding deterministic patterns [20].

However, in most of these approaches the architecture is extremely tailored to the CUT, and any change in the CUT requires re-synthesis of the complete BIST hardware. Another drawback of traditional BIST is the use of special hardware for test pattern generation on chip, which causes area overhead and performance degradation.

To overcome these problems, FBIST methods have been proposed which exploit specific functional units like adders, multipliers, Arithmetic Logic Units (ALU) or processor cores for on-chip pseudorandom test generation and test response evaluation [21–24]. These units are available in data-path architectures used in traditional general purpose processors and in digital signal processing units. The term FBIST describes a test method to control functional modules so that they generate a test set, which targets structural faults within other parts of the system.

Usually these ALU-based FBIST methods are called Arithmetic BIST (ABIST), since they essentially adopt the additive congruential generation scheme of pseudo-random numbers [25]. ABIST, along with the accumulator-based response compaction scheme [26] facilitates the BIST strategy for high-performance data path architectures that use the functionality of existing hardware, is entirely integrated with CUT, and results in at-speed testing with no performance degradation and with little area overhead.

The drawback of using ABIST is the same as is with traditional LFSR based BIST – selected test pattern sequences are not capable to detect all RPR faults, which in turn may lead to low fault coverage. In Fig. 1, a range of all possible and different patterns, generated by a BIST, starting with the first pattern P_1 up to the last one of the cycle P_n in a pseudorandom order, is shown. For LFSR with length m the number of all different patterns in this range will be $2^m - 1$. Because of the huge number of patterns the BIST is able to generate, a smaller window with length $N << 2^m$ will be typically used, which however is not able to cover specific patterns needed for detecting RPR faults. The latter remain outside of the window.

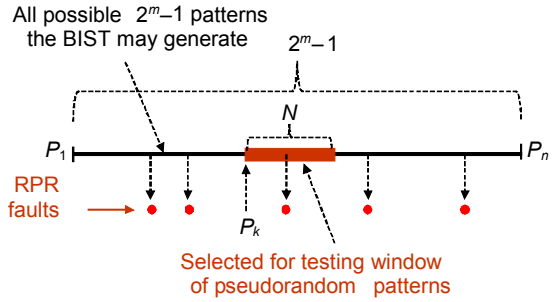

Fig. 1. Random-pattern-resistant faults detection problem.

Software Based Self-Test as another special case of FBIST is an approach that has gained increasing acceptance for testing processor cores and using processors for testing other components in Systems-on-Chip (SoC) [27–31]. SBST moves the test functions also from external testers to on-chip resources whereas the test patterns are produced by the processors, using their native instructions. Usually, in this approach, the test programs and associated test data are first, loaded into on-chip memories, and subsequently, these test programs are executed by the processor at actual/full speed (at-speed).

The positive features of the SBST technique that have supported its introduction into a typical microprocessor test flow are: non-intrusiveness (no need for any processor modification), no extra power consumption compared to the normal operation mode, at-speed testing (at the processor's actual speed) which enables screening of delay defects that are not observable at lower frequencies, and no over-testing compared to the scan-path testing approach. Self-test programs developed for manufacturing test can be reused in the field throughout product lifetime.

The problem with SBST is still in generation of high quality test data – operands to be used by the instructions which build up the test program. Another problem is with observability of test responses. Differently from ABIST, where the responses of the tested blocks are captured at each clock cycle, in the case of SBST the results of the processor instructions as responses of the test are registered only in the end of the testing instruction (in the end of the sequence of microinstructions). This may result in a lot of fault masking cases, which reduces the fault coverage.

In this paper we propose a functional BIST, which combines clock cycle based response collection as used in hardwired ABIST with software based flexibility to extend the restricted application area of ABIST from specific data-path architectures to a larger class of processor architectures. The clock cycle based observation technique allows to avoid fault masking, and selecting proper instruction sequences supported by properly generated test data allows to achieve higher fault coverage. The clock cycle based test response observation is carried out using built-in Signature Analysers (SA). The places for inserting SA flip-flops will be found by profiling of test programs or microprograms of testing instructions to find out the most frequently visited nodes in CUT. This helps to capture maximum amount of information from the test process and to achieve high fault coverage.

Another novelty is to combine test program generation with testability improvement regarding RPR faults by inserting optimized set of test points into the hardware. The combination of test data generation with DFT improvement allows to explore different trade-offs between testing cost and quality. Differently from [27,28], the sequences of component test patterns are not needed to store in the chip, they will be generated online by the resources of the system. At the same time, at-speed testing guarantees high fault coverage.

3. GENERAL SCHEME OF THE FUNCTIONAL BIST

The main idea of the proposed FBIST concept includes the use of activated on-chip functional processes as test pattern generators for a selected CUT and monitoring the behaviour of CUT by a Multiple Input Signature Analyser (MISR). MISR is the only additional hardware needed for the implementation of FBIST. The functionality of the processor is used to apply the test patterns to each component at-speed. The tests are delivered by processor instructions and unfolded by microinstructions from local control units.

Consider a data-path of a processor in Fig. 2 with ALU as a CUT. The data path consists of a register block for temporary storing of the data, which

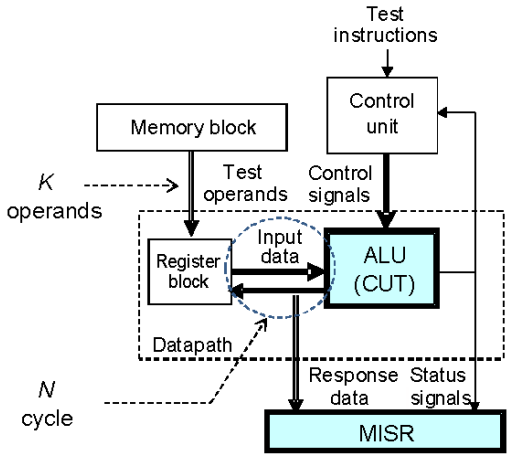

Fig. 2. Functional BIST of a digital system.

participates in the operation carried out in ALU. For example, during the operation of fractional number division the register block will store the dividend, divisor, all intermediate results of division, the quotient, and the counter of cycles needed for controlling the whole process of division. The input data from the register block and the control signals from the control unit are interpreted as input test patterns for CUT. The output data from ALU sent back to the register block and the status signals as feedback to the control unit as the most frequently visited nodes during the tested operation are interpreted as responses of CUT, and are registered in MISR.

As the result of N clock cycles of the division operation, N functional test patterns are generated on-line, and consequently, N responses of ALU will be compressed in the MISR, which is monitoring the whole division process. The whole microinstruction level test process is launched by a division instruction, which includes two operands – the dividend and divisor.

Differently from the known approaches, where the instructions are regarded as test patterns and the results of the instructions are regarded as test responses, in the proposed case all the input patterns of CUT during each clock cycle of the instruction are regarded as test patterns with immediate monitoring of the responses of CUT in each cycle by MISR. As the result, we have achieved a multiplication effect of N times in the number of test patterns when moving the test access from the instruction level to the microinstruction level. Denote by L the number of bits in the data operands (dividend and divisor), and by l the number of bits on the inputs of ALU. Then the reduction in the test data volume to be stored in the memory through the compression of test data in the described FBIST scheme is equal to $R = Nl/2L$.

In this scheme, the functional patterns produced directly on the inputs of ALU have the similar role as pseudorandom test patterns in classical BIST schemes. To improve the fault coverage of FBIST, the same operation can be carried out with different operands. The problem to be solved is the choice of the best operands to minimize the length of the whole test procedure. Similarly to the pseudorandom test, the functional test patterns may not be able to cover random-pattern-resistant faults, which limit the fault coverage that can be achieved with the pure functional BIST approach. However, the possibility of repeating the same (division) program with different operands gives the possibility to exercise different windows of pseudorandom patterns as explained in Fig. 1 to target the RPR faults. Another possibility to improve the fault coverage is to use DFT approach, i.e., to insert additional test points whereas the observation test points can be integrated with MISR.

4. FUNCTIONAL BIST SYNTHESIS FRAMEWORK

In Fig. 3, the methodology and framework are shown for generating functional BIST for processor architectures,

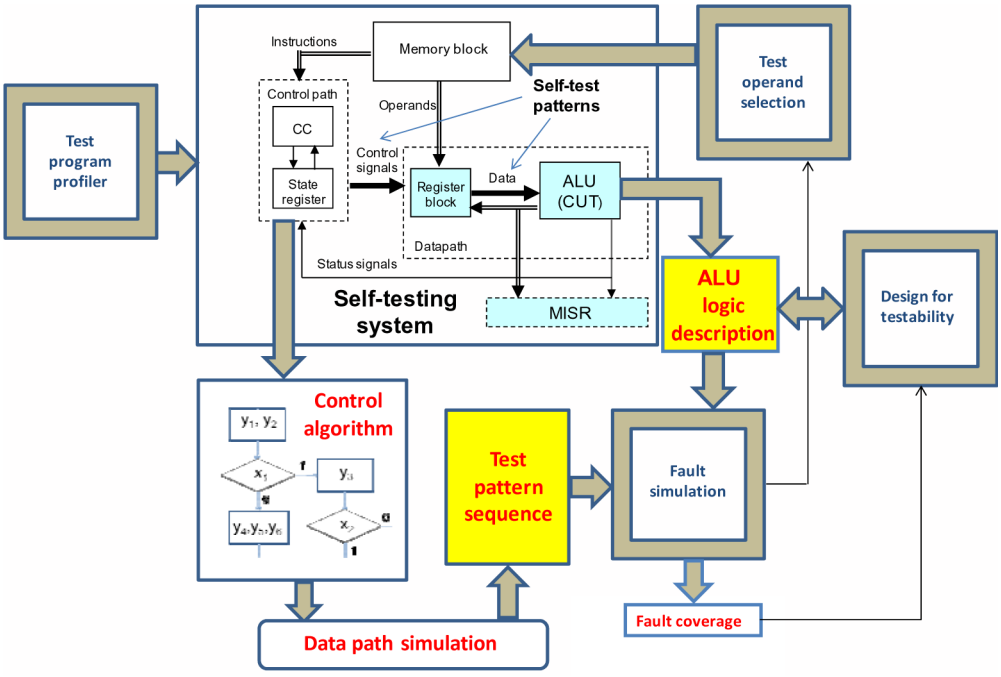

Fig. 3. Functional BIST synthesis methodology for a digital system.

which uses the inherent functionality of the processor (i.e. instruction set or selected working routines) for implementing test programs. The goal of the framework is to partition the hardware into components to be tested (i.e. into a set of CUT), to generate for each CUT the test data for the related test program to be stored in the memory, and to improve the testability of CUT to achieve higher fault coverage of testing.

In the general case, CUT can be specified by profiling the test programs to find out the most frequently visited nodes in the hardware exercised by the test program. These nodes will serve as the best places for inserting MISR. In a particular case, different well defined subcircuits or components of the data-path like adders, multipliers and ALU can serve as CUT and the outputs of these CUT will be used to place MISR.

The framework consists of the following tools: test program profiler, data path simulator, fault simulator, test operands generator, and design for testability advisor.

The goal of the test program profiler is to find out in the system the best observation places to capture maximum amount of information from the test process for achieving as high fault coverage as possible.

The data path simulator is used for finding the functional test pattern sequences applied to the inputs of CUT, and is produced by the given sequence of test instructions with related test operands. The fault simulator is used for measuring the quality of the sequence of functional test patterns. If the quality of test is not satisfied, it will be extended by selecting additional test instructions or operands. Such a modification of the test program will be repeated till the test quality is satisfied.

The bottleneck of the whole process of BIST synthesis is the speed of the fault simulator, since it is used for evaluating the test programs and test data in the process of searching best solutions. The second role of the fault simulator is to evaluate the decisions for inserting test points to improve the testability. We updated our fault simulator, developed in [32], to cover the needs of the described BIST synthesis framework. The experimental results of using the simulator are presented in Section 7.4.

For generating test operands, the methods of random search or genetic algorithms can be used. In this paper, we have used a genetic test operand generator based on using the Java Genetic Algorithms Package (JGAP) [33].

5. IMPROVEMENT OF THE TESTABILITY

The blocks in the subsystems may contain RPR faults, which are difficult to detect with selected self-test subroutines, and the generated test data. We call these blocks as difficult testable ones. To improve the testability of the subsystem we have two possibilities: to improve either observability or controllability. For this purpose proper test points should be inserted. Examples of improving the testability in a given subsystem are shown in Fig. 4.

Assume that there are a not well observable Block 1, and a not well controllable Block 2 in the subsystem. SA is for collecting the response signals from the subsystem under test. A dedicated test signal T is used for switching the system into the test mode, e.g., to reconfigure selected registers or flip-flops into signature analyser, and to allow specific test related control over the subsystem.

Insertion of a test point OP allows making the Block 1 directly observable in SA, whereas the test point CP is inserted for dedicated control over a selected input of the Block 2. In the normal work, test signal T is low to select the upper channel of the multiplexer for direct connecting the Block 1 with Block 2, whereas in the test mode, signal T will switch the input of Block 2 to the lower channel of MUX for sending the control signal CP to Block 2. The control signal can be generated in different ways from other parts of the subsystem, e.g., from the Block 1.

To minimize the cost of hardware needed for improving the testability of CUT, the number of test points to be inserted either for observation or control should be minimized. The minimum set of test points should be selected on the basis of not detected faults.

To each node of CUT a weight can be assigned, measured by the amount of information it provides about not detected faults. The nodes with highest weights can be selected as test points.

The amount of information in general case can be measured as entropy:

$$I = -p \log_2 p - (1-p) \log_2 (1-p), \qquad (1)$$

where p is the probability that a message is chosen from all possible choices in the message space. In our case, the message space consists of two messages: either at the given test point a not detected fault can be detected or no faults can be detected at this test point.

Assume, that the Blocks 1 and 2 in Fig. 4 are the only blocks in the subsystem. Assume also that the

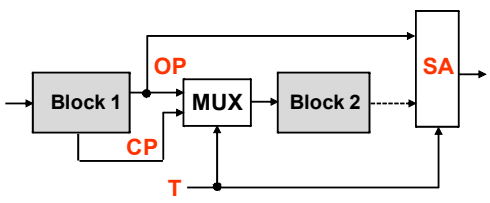

Fig. 4. Insertion of test points into a subsystem.

Block 1 contains n_1, and the Block 2 contains n_2 undetected faults. Then, for calculating I for the output of Block 1 we have in the formula (1):

$$p = \frac{n_1}{n_1 + n_2}. \quad (2)$$

Example 1. Consider a subsystem in Fig. 5 consisting of 4 blocks 1, 2, 3, and 4, which contain 4, 12, 28, and 20 undetected faults, respectively. For the outputs of the blocks we will have the probabilities:

$$p_1 = 4/64 = 0.0625,\ p_2 = 16/64 = 0.25,\ p_3 = 0.5,\ p_4 = 1,$$

and the corresponding amounts of information

$$I_1 = -0.0625\log_2 0.0625 - 0.9375\log_2 0.9375 = 0.325,$$
$$I_2 = 0.8, \quad I_3 = 1, \quad \text{and} \quad I_4 = 0.$$

This result suggests to select as the first test point the output of the Block 3 with $I_3 = 1$, since on the output of it, if made directly observable, exactly half of the undetected faults can be detected.

The design for testability may consist of several steps. After each step of the improvement either of the observability or of the controllability, the amounts of information for the nodes of the circuit should be recalculated to find the next best place for inserting a test point.

Consider in Fig. 6 a subsystem, which contains undetectable faults in all three blocks. Assume that all the faults in Block 1 can be detected if the block is made directly observable. On the other hand, assume that the Blocks 1 and 2 will still both contain undetectable faults

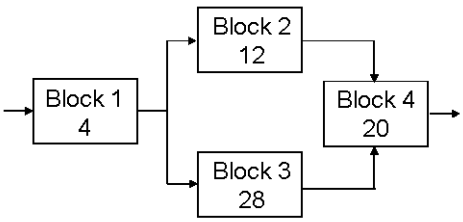

Fig. 5. Subsystem of 4 blocks with undetected faults.

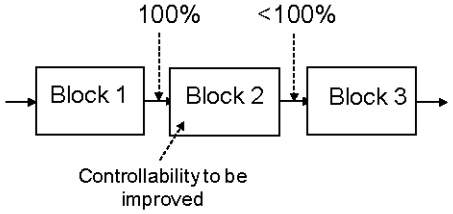

Fig. 6. Subsystem of 3 blocks with untested faults.

if the Block 2 was made directly observable. In this case, we have to improve the controllability of the Block 2. If now all the faults in Blocks 1 and 2 will be detectable on the output of Block 2, the problem with Blocks 1 and 2 is solved. However, if the faults in Block 1 remain still not detected through Block 2, we have to make the output of Block 1 directly observable or improve its input controllability.

Now we have to test again the full system. If Block 3 will still contain undetectable faults, we have to improve its controllability. If after that the Block 2 will still have undetected faults, it must be done directly observable.

To summarize, after finding a test point TP with the highest amount of information about the undetected faults in CUT, we have, first, to solve the testability problem in the part of CUT, which feeds the test point TP, as explained above on the basis of Fig. 6. Thereafter, we have to recalculate the probabilities and information quantities for the remaining part of CUT, find the best place for the next test point and repeat the procedure.

6. EXPERIMENTAL TESTABILITY ANALYSIS SET-UP

Exploration of testability improvement solutions is a very costly procedure, since it needs a lot of design modifications and evaluation of each modification by measuring its testability with fault simulation which itself is a time consuming procedure. We developed an experimental set-up for testability analysis which considerably speeds up the exploration procedure.

The set-up consists of two tables: simulation table (ST) and fault table (FT). They will be created for the given circuit by fault simulating the given test pattern set. This set-up will be the basis for design explorations in search for the optimum testability. Instead of direct circuit modifications, we simulate the circuit indirectly by modifying only ST, and instead of the fault simulation of the whole new modified circuit, we simulate only this part of the circuit, which is influenced by the injected circuit change.

Example 2. Consider the circuit in Fig. 7 which represents the smallest member of the ISCAS'85 benchmark suite [34]. The test, applied to the inputs of the circuit, consists of a set of 5 test patterns. The results of simulation and fault simulation are depicted in Tables 1 and 2, respectively. The columns in both tables correspond to the 15 nodes w_j in the circuit (including 5 inputs and 2 outputs), and the rows i correspond to the 5 test patterns T_i.

The entries $c(i, j)$ in Table 2 mean the following: $c(i, j) = 0$, if the test pattern T_i detects the stuck-at fault $w_j \equiv 0$; $c(i, j) = 1$, if the test pattern T_i detects the

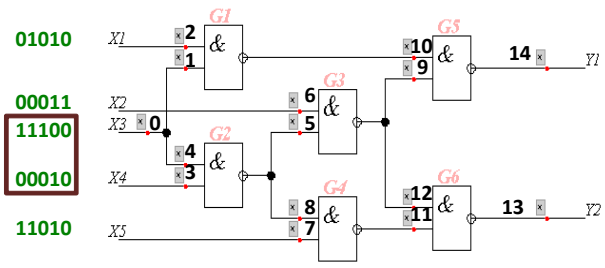

Fig. 7. ISCAS circuit c17.

Table 1. Simulation table ST

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	0	0	1	1	0	1	1	1	1	0	1	1	0
1	1	1	0	1	1	0	1	1	1	0	0	1	1	1
1	1	0	0	1	1	0	0	1	1	1	1	1	0	0
0	0	1	1	0	1	1	1	1	0	1	0	0	1	1
0	0	0	0	0	1	1	0	1	0	1	1	0	1	1

Table 2. Fault table FT

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	X	X	1	1	X	X	1	0	0	0	0	1	X	0	1
2	0	0	0	1	X	X	X	0	0	X	1	1	X	0	0
3	X	X	1	X	X	X	1	1	X	0	0	0	0	1	1
4	1	X	X	X	1	0	0	X	X	1	X	X	X	0	0
5	X	X	X	X	X	0	0	X	X	1	X	X	1	0	0
	&	0	&	1	1	0	&	&	0	&	&	&	&	&	&

stuck-at fault $w_j \equiv 1$; $c(i,j) = \mathrm{X}$, if the test pattern T_i does not detect any fault at the node w_j. The last row with entries s_j in the fault table summarizes information in the columns in the following way: $s(j) = 0$, if there is at least one entry $c(i,j) = 0$ in the column j, and no entry $c(i,j) = 1$; $s(j) = 1$, if there is at least one entry $c(i,j) = 1$ in the column j, and no entry $c(i,j) = 0$; $s(j) = \&$, if there is at least one entry $c(i,j) = 0$ and at least one entry $c(i,j) = 1$, and finally, $s(j) = \mathrm{X}$, if all entries in the column are $c(i,j) = \mathrm{X}$.

Assume, these 5 test patterns form the whole test set applied to the circuit during a test program activated in the system. From FT we see that 5 faults $w_1 \equiv 1$, $w_3 \equiv 0$, $w_4 \equiv 0$, $w_5 \equiv 1$, and $w_8 \equiv 1$ are not detected by the given test, which gives the fault coverage only 83.3% (25 stuck-at faults from 30 are detected).

To improve the quality of the given test, we may improve the testability of the circuit by inserting test points in a similar way as we did at the higher level in Section 5.

It is easy to see that in this example it is not possible to test the faults $w_3 \equiv 0$, $w_4 \equiv 0$, $w_5 \equiv 1$, and $w_8 \equiv 1$ by adding test points for observing the values on the outputs of faulty gates, because the faults at the inputs of the gates cannot propagate through the gates by the given test patterns. For example, the nodes 5 and 8 have during all the test patterns continuously the value 1 (see the columns 5 and 8 in Table 1), which means that the faults $w_5 \equiv 1$ and $w_8 \equiv 1$ are never activated (to activate a fault $w_j \equiv 1$ we need to apply on the node the opposite value $w_j = 0$). The conclusion of the described case is: we have to improve the controllability of the related gates.

To minimize the number of test points needed we have to start to exercise with multiplexers at the nodes closest to the inputs, taking into account the fact that after making a node w_j controllable, the subsequent nodes, having a path from w_j, may get controllable as well. This is the case with the given circuit. After making the node w_3 controllable, the nodes w_4, w_5, and w_8 will be controllable as well, and all the faults at the nodes w_3, w_4, w_5, and w_8 will be detected. The only not detectable fault is now $w_1 \equiv 1$. This fault can be detected by making the output of the gate, i.e. the node 10 observable. After inserting the two test points, the fault coverage of the given test set will be 100%.

Experimental results of DFT for more complex circuits are presented in the next section.

7. EXPERIMENTAL CASE STUDIES

We have carried out experiments with synthesis of functional BIST for two data-paths: (1) restoring 16-bit integer divider, and (2) non-restoring 16-bit signed integer divider. The goal of the experiments was to design a FBIST by improving the testability of a system with as few added test points as possible. The experiments were carried out for two approaches: random and genetic generation of test operands.

7.1. Experiments with restoring integer divider

The data path of the system consists of 3 registers, cycle counter and 16-bit ALU as the CUT. ALU has 53 inputs and 17 outputs. To improve the testability, 4 test points were inserted which were connected via XOR gate to a single additional output. Test operands were found by random search. The shortest test with 100% fault coverage, which was found, consists of 3 operands which produce 197 direct input patterns to ALU.

Table 3 illustrates the test coverage as the function of the number of test operands used by the instruction. For each number of operands, 1000 random experiments were carried out for finding the best combination of operands, and the average, best and worse results are shown. The same experiments are illustrated as well in Fig. 8.

Table 3. Fault coverages of test sequences

Number of operands	Fault coverage, %		
	Average	Best	Worse
1	87.88	93.76	63.68
2	95.04	98.65	85.85
3	97.58	100	90.74
4	98.63		93.86
5	99.19		95.53
6	99.49		96.46
7	99.65		97.92
8	99.72		98.34
9	99.77		97.81
10	99.82		98.54

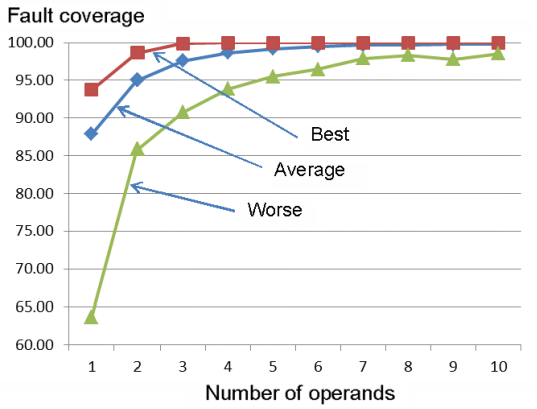

Fig. 8. Fault coverage as the function of test length.

Figure 9 demonstrates statistically how fast it would be to generate for the circuit a test with 100% fault coverage by pure random search. 1000 experiments were carried out, and in each experiment random operands were added to the test till the 100% fault coverage was achieved. Frequency means how many times from 1000 random experiments the 100% fault coverage was achieved for the given number of operands.

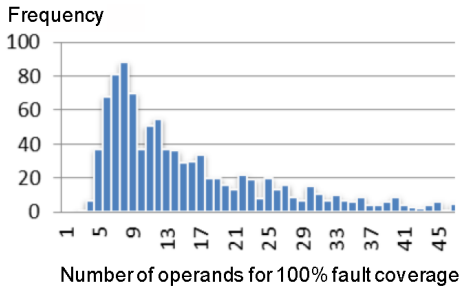

Fig. 9. Frequency ranges of random test lengths.

7.2. Experiments with non-restoring divider

In this experiment the operands were generated by genetic test operand generator.

To have an understanding about the difficulty of generating high quality test operands, an experiment was carried out with 10 000 and 100 000 random samples, and the fault coverage for each test operand was measured. The 10 best results are included in Table 4. In Table 5, the results of comparing the random and genetic test operand generation approaches are depicted.

Whereas in Table 4 we were measuring the fault coverage for independently generated random 1-operand experiments then in Table 5 we were interested in getting the best final fault coverage with as less as possible number of operands. In the best experiment the first operand chosen randomly happened to be with very low fault coverage 76.17%, compared to the high fault coverage numbers in Table 4. However, the subsequent random sequence of the next 4 operands increased the cumulative fault coverage up to the maximum value 99.77, achieved by the random approach.

In case of the genetic algorithm, the shortest 100% test generated included 4 operands. However, by adding a single additional test point to improve the testability of the circuit, it was possible to reduce the number of operands up to three, as in the case of the restoring divider.

Table 4. Best fault covers for single operands

Best selected operands	Fault coverage, %	
	10 000 samples	100 000 samples
1	83.29	84.46
2	83.18	84.46
3	83.06	84.11
4	82.94	84.00
5	82.94	83.76
6	82.83	83.76
7	82.83	83.64
8	82.83	83.64
9	82.83	83.64
10	82.71	83.52

Table 5. Comparison of test synthesis methods

Number of operands	Random approach	Genetic approach	
		Without DFT	With DFT
1	76.17	83.29	83.29
2	95.09	97.76	97.76
3	97.78	99.53	100
4	98.71	100	
5	99.77		

Table 6. Parameters of genetic experiments

Experiment No.	Evolutions	Population	Test points	Coverage, %	Time, s
1	50	250	4	99.53	495
2	100	250	4	99.53	1003
3	50	500	2	99.76	1024
4	50	1000	3	99.65	1876
5	70	1000	1	99.88	2500

To minimize the number of test points, and to demonstrate the possibility of trade-off between hardware cost, test synthesis time and fault coverage, several experiments with the genetic algorithm were carried out. The columns 2–6 in Table 6 mean, the numbers of evolutions, the population size, the numbers of test points needed for adding into the circuit to achieve 100% fault coverage with 3-operand tests, the fault coverage achieved by genetic algorithm without adding test points, and the time in seconds used for test synthesis by using the genetic algorithm, respectively. The 5th experiment shows that we need only a single added test point to achieve 100% fault coverage with 3-operand test.

7.3. Impact of design for testability

In this section we present the results of experiments to demonstrate the impact of improving the testability on the fault coverage and on the diagnosability of circuits.

In Table 7, it is shown how the fault coverage can be improved for the given test pattern set by inserting test points. Circuits from the following benchmark families were exercised: ISCAS'85 [34], ISCAS'89 [35], and ITC'99 [36] are listed in column 1.

Table 7. Improvement of fault coverage by DFT

Circuit characteristics				Improvement of fault coverage with DFT		
Circuit	Input	Output	Nodes	TP	FC, % before	FC, % after
c1908	33	25	866	5	99.48	99.88
c2670	233	140	1 313	65	88.65	98.67
c3540	50	22	1 648	59	95.54	100
c5315	178	123	2 712	16	98.89	100
c7552	207	108	3 552	51	94.09	98.27
s9234	247	250	3 637	135	92.19	98.6
s1320	700	790	5 228	72	98.19	99.97
s15850	611	684	6 075	182	94.20	98.93
s35932	1 763	2 048	19 547	17	88.5	99.99
bo5	35	70	1 332	153	77.52	97.37
b12	126	127	1 535	3	99.77	100
b14	277	299	11 858	93	92.76	99.41
b15	485	519	11 749	152	88.78	99.84

The main characteristics of circuits as the numbers of inputs, outputs, and nodes of the circuits are given in the columns 2, 3, and 4, respectively. The number of inserted test points TP is given in column 5 and the comparison of fault coverage FC before and after inserting test points are depicted in columns 6 and 7. The high number of needed test points is explained by the width of the circuit (number of inputs). The high number of inputs leads to the high number of faults, which can be made testable only by independent individual test points.

The goal of these experiments was not to achieve 100% fault coverage, rather to investigate the dependence of the increase in fault coverage on the number of test points. To generate a 100% test needs a time costly deterministic test pattern generation. To be able to carry out multiple steps of evaluation of the DFT results in the step-by-step test point insertion, we used a fast random test pattern generation. This was sufficient to demonstrate the efficiency of chosen test points in terms of improved fault coverage.

In Table 8 and Fig. 10, it is shown how the diagnosability of CUT can be improved by inserting test points. Diagnosability is measured as the average diagnostic resolution (the number of suspected indistinguishable faults in case a CUT has failed).

Table 8. Improvement of diagnosability by DFT

TP	c2670	c5315	c6288	c1908	c1355	c3540	AVRG
0	18.4	17.3	68.0	51.0	111.3	12.9	46.5
5	15.3	11.0	51.4	37.2	71.2	11.1	32.9
10	13.9	7.9	41.8	24.9	43.1	9.2	23.5
15	13.2	7.3	37.3	20.5	34.6	7.5	20.1
20	10.2	7.1	30.0	16.4	30.0	7.1	16.8

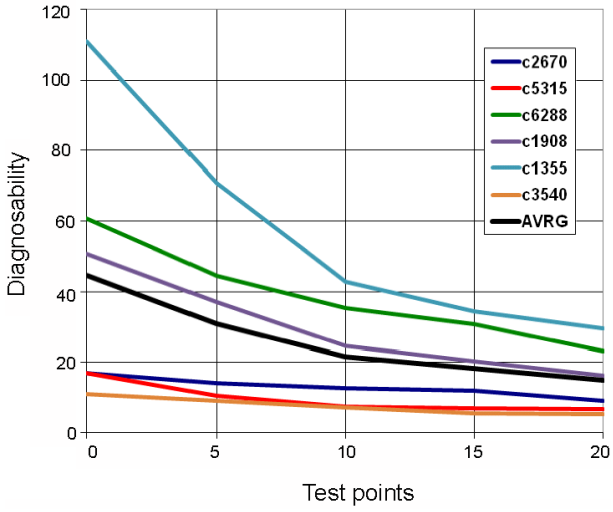

Fig. 10. Improvement of diagnosability by DFT.

7.4. Experimental data of the fault simulator

To carry out the experimental work of fault coverage analysis and evaluation of the efficiency of test points insertion, we updated our fault simulator [32].

Table 9 presents the characteristics of the fault simulator which was accommodated for the needs of FBIST synthesis framework. The experiments were carried out for the benchmark circuits of ISCAS'85 [34], and the full-scan versions of ISCAS'89 [35] and ITC'99 [36] (column 1) to be compared with two state-of-the-art commercial fault simulators C1 and C2 from major CAD vendors (columns 3, 4), and the developed new simulator (column 5). Fault simulation times in seconds were calculated for the sets of random 10 000 patterns. The experiments were run on a 1.5 GHz UltraSPARC IV+ workstation using SunOS 5.10 operating system.

8. CONCLUSIONS

We introduced a new approach to design self-testing processor architectures, which uses the inherent functionality of the given system for on-line test pattern generation. The approach does not need to store high volume test data in the processor memory. Instructions of the processor are used for launching the hardware components oriented test procedures whereas the test patterns applied to the selected CUT are generated on-line and observed by MISR. In general case, the best places for MISR are selected by profiling the test processes to determine the most frequently visited nodes in the tested subsystem, to get the maximum information when observing the test run.

Table 9. Speed characteristics of fault simulator

Circuit	No. of gates	Simulation time, s		
		C1	C2	New
c2670	883	2.2	24	0.4
c3540	1 270	7.4	43	0.9
c5315	2 079	5.6	57	0.8
c6288	2 384	27.8	284	7.4
c7552	2 632	8.1	88	1.2
s13207	3 214	5.6	70	2.0
s15850	3 873	12.1	111	2.7
s35932	12 204	23.6	390	5.7
s38417	9 849	31.4	310	7.0
s38584	13 503	23.2	320	6.4
b14	9 150	49.2	N/A	14.5
b15	8 877	39.1	N/A	26.6
b17	31 008	117.7	N/A	77.8
Average normalized run-time		4.7	43	1

The proposed FBIST combines clock cycle based response collection, as used in the hardwired ABIST, with software based flexibility to extend the restricted application area of ABIST from specific data-path architectures to a larger class of processor architectures. The clock cycle based test response observation by MISR allows to avoid fault masking compared to traditional SBST. The dedicated test data (e.g. instruction operands) generation coupled with CUT-oriented fault coverage analysis allows to achieve high fault coverage.

Another novelty is to combine test program generation with testability improvement of the CUT by inserting test points. This combination of test data generation with DFT improvement allows to explore different trade-offs between testing cost and quality.

For generating test data (operands) used by test instructions, a genetic algorithm was developed to achieve the needed high fault coverage. For example, in the case of having ALU for the division operation as a CUT, we needed only three division operations with a single inserted test point to achieve 100% stuck-at fault coverage of the responsible for this operation hardware.

We have created a framework for synthesis of self-testing processor architectures, supported by design for testability advisor and a very efficient fault simulator to carry out fast exploration of possible FBIST solutions.

ACKNOWLEDGEMENTS

This work was jointly supported by EU through European Regional Development Fund, and FP7-2013-ICT-11: 619871 project BASTION as well as by the institutional research funding IUT 19-1 of the Estonian Ministry of Education and Research, and ESF grants 8478 and 9429.

REFERENCES

1. Bernardi, P., Grosso, M., Sanchez, E., and Sonza Reorda, M. Software-based self-test of embedded microprocessors. In *Design and Test Technology for Dependable Systems-on-Chip* (Ubar, R., Raik, J., and Vierhaus, T., eds). Information Science Reference, Igi Global, Herschey, New York, 2011, 338–359.
2. Mak, T. M., Krstic, A., Cheng, K.-T., and Wang, Li.-C. New challenges in delay testing of nanometer, multi-gigahertz designs. *IEEE Des. Test Comput.*, 2004, **21**, 241–248.
3. Bushard, L., Chelstrom, N., Ferguson, S., and Keller, B. DFT of the cell processor and its impact on EDA test software. In *Proc. IEEE Asian Test Symposium*. Fukuoka, 2006, 369–374.
4. Wang, S. and Gupta, S. K. ATPG for heat dissipation minimization during scan testing. In *Proc. ACM IEEE*

Design Automation Conference. Anaheim, 1997, 614–619.
5. Chen, L., Ravi, S., Raghunathan, A., and Dey, S. A scalable software-based self-test methodology for programmable processors. In *Proc. IEEE/ACM Design Automation Conference*. Anaheim, 2003, 548–553.
6. Wang, L.-T., Wu, C.-W., and Wen, X. *VLSI Test Principles and Architectures*. Morgan Kaufmann, San Francisco, 2006.
7. Gizopoulos, D. *Advances in Electronic Testing: Challenges and Methodologies*. Springer, 2006.
8. Hetherington, G., Fryars, T., Tamarapalli, N., Kassab, M., Hassan, A., and Rajski, J. Logic BIST for large industrial designs. In *Proc. IEEE International Test Conference*. Atlantic City, 1999, 358–367.
9. Hortensius, P. D., McLeod, R. D., and Podaima, B. W. Cellular automata circuits for BIST. *IBM J. Res. Dev.*, 1990, **34**, 389–405.
10. Eichelberger, E. B. and Lindbloom, E. Random pattern coverage enhancement and diagnosis for LSSD logic self-test. *IBM J. Res. Dev.*, 1983, **27**, 265–272.
11. Tamarapalli, N. and Rajski, J. Constructive multi-phase test point insertion for scan-based BIST. In *Proc. IEEE International Test Conference*. Washington, DC, 1996, 649–658.
12. Touba, N. A. and McCluskey, E. J. Test point insertion based on path tracing. In *Proc. 14th IEEE VLSI Test Symposium*. Princeton, 1996, 2–8.
13. Yang, J.-S., Nadeau-Dostie, B., and Touba, N. Reducing test point area for BIST through greater use of functional flip-flops to drive control points. In *Proc. International Symposium on Defect and Fault Tolerance in VLSI Systems*. Chicago, 2009, 20–28.
14. Koenemann, B. LFSR-coded test patterns for scan designs. In *Proc. European Test Conference*. Munich, 1991, 237–242.
15. Zhao, Z., Pouya, B., and Touba, N. A. BETSY: synthesizing circuits for a specified BIST environment. In *Proc. IEEE International Test Conference*. Washington, 1998, 144–153.
16. Agrawal, V. K. and Cerny, E. Store and generate built-in test approach. In *Proc. Fault-Tolerant Computing Symposium*. Portland, Maine, 1981, 35–40.
17. Wunderlich, H.-J. and Kiefer, G. Bit flipping BIST. In *Proc. IEEE ACM International Conference on Computer-Aided Design*. San Jose, CA, 1996, 337–343.
18. Touba, N. A. and McCluskey, E. J. Bit-fixing in pseudorandom sequences for scan BIST. *IEEE Trans. CAD Integr. Circ. Syst.*, 2001, **20**, 545–555.
19. Hellebrand, S., Tarnick, S., Rajski, J., and Courtois, B. Generation of vector patterns through reseeding of multiple-polynomial linear feedback shift registers. In *Proc. IEEE International Test Conference*. Baltimore, MD, 1992, 120–129.
20. Touba, N. A. and McCluskey, E. J. Transformed pseudorandom patterns for BIST. In *Proc. VLSI Test Symposium*. Princeton, NJ, 1995, 410–416.
21. Dorsch, R. and Wunderlich, H.-J. Accumulator based deterministic BIST. In *Proc. IEEE International Test Conference*. Washington, DC, 1998, 412–421.
22. Rajski, J. and Tyszer, J. *Arithmetic BIST in Embedded Systems*. Prentice-Hall, NJ, 1998, 268.
23. Hellebrand, S., Wunderlich, H.-J., and Hertwig, A. Mixed-mode BIST using embedded processors. *J. Electron. Test. (JETTA)*, 1998, **12**, 127–138.
24. Voyiatzis, I., Gizopoulos, D., and Paschalis, A. Accumulator-based test generation for robust sequential fault testing in DSP cores in near-optimal time. *IEEE Trans. VLSI Syst.*, 2005, **13**, 1079–1086.
25. Knuth, D. E. *The Art of Computer Programming*. Vol. 2. Addison-Wesley, Reading, Massachusetts, 1981.
26. Rajski, J. and Tyszer, J. Accumulator-based compaction of test responses. *IEEE Trans. Comput.*, 1993, **42**, 643–650.
27. Gizopoulos, D., Psarakis, M., Hatzimihail, M., and Maniatakos, M. Systematic software-based self-test for pipelined processors. *IEEE Trans. VLSI Syst.*, 2008, **16**, 1441–1453.
28. Chen, L. and Dey, S. DEFUSE: A deterministic functional self-test methodology for processors. In *Proc. VLSI Test Symposium*. Montreal, 2000, 255–262.
29. Jayaraman, K., Vedula, V. M., and Abraham, J. A. Native mode functional self-test generation for systems-on-chip. In *Proc. International Symposium on Quality Electronic Design*. San Jose, CA, 2002, 280–285.
30. Apostolakis, A., Psarakis, M., Gizopoulos, D., and Paschalis, A. A functional self-test approach for peripheral cores in processor-based SoCs. In *Proc. International On-Line Testing Symposium*. Crete, 2007, 271–276.
31. Koal, T., Kothe, R., and Vierhaus, H. T. SoC self test based on a test-processor. In *Design and Test Technology for Dependable Systems-on-Chip* (Ubar, R., Raik, J., and Vierhaus, T., eds). IGI Global, Herschey, New York, 2011, 360–376.
32. Ubar, R., Devadze, S., Raik, J., and Jutman, A. Parallel X-fault simulation with critical path tracing technique. In *Proc. IEEE Conference on Design, Automation & Test in Europe*. Dresden, 2010, 879–884.
33. Java Genetic Algorithms Package. http://jgap.sourceforge.net (accessed 18.02.2014).
34. Brglez, F. and Fujiwara, H. A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. In *Proc. IEEE International Symposium on Circuits and Systems*. Kyoto, 1985, 785–794.
35. Brglez, F., Bryan, D., and Kominski, K. Combinational profiles of sequential benchmark circuits. In *Proc. International Symposium on Circuits and Systems*. Portland, OR, 1989, 1929–1934.
36. Corno, F., Reorda, M. S., and Squillero, G. RT-level ITC'99 benchmarks and first ATPG results. *IEEE Des. Test Comput.*, 2000, **17**, 44–53.

Digitaalsüsteemide funktsionaalse isetestimise kvaliteedi analüüsi keskkond

Raimund Ubar, Sergei Kostin, Helena Kruus, Margit Aarna ja Sergei Devadze

Digitaalsüsteemide usaldatavusest on saanud üks tähtsamaid inseneriprobleeme. Süsteemide usaldatavuse suurendamise perspektiivseks võimaluseks on projekteerida isetestivaid süsteeme, millega saab testimisseansse reaalajas ja õigel töökiirusel läbi viia ning sellega tagada testimise kõrge kvaliteet. Artiklis on kirjeldatud uut isetestimise metodoloogiat, mis põhineb süsteemi enda ressursside kasutamisel, millega välditakse vajadust viia süsteemi täiendavaid spetsiaalseid testimisvahendeid, nagu see traditsiooniliselt toimub. Saavutatavateks eelisteks on testimiseks vajalike lisavahendite minimeerimine ja nende negatiivse mõju välistamine süsteemi töökiirusele. Reaalajas töökiirusel testimine võimaldab kasutusel olevate meetoditega võrreldes paremini avastada võimalikke dünaamilisi rikkeid, näiteks suurenenud signaaliviiteid, mis kokkuvõttes tõstab testimise kvaliteeti ja tulemuste usaldatavust. On välja töötatud meetodid testandmete ja -punktide minimeerimiseks. On kirjeldatud tarkvarakeskkonda, milles sisalduvad tööriistad võimaldavad emuleerida projekteeritavat isetestivat süsteemi ja analüüsida testimiskvaliteeti.