THESIS ON INFORMATICS AND SYSTEM ENGINEERING C66

Optimization of Built-In Self-Test in Digital Systems

HELENA KRUUS

TUT PRESS

TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Computer Engineering
Chair of Computer Engineering and Diagnostics

This dissertation was accepted for the defence of the degree of Doctor of Philosophy in Computer and Systems Engineering on July 7, 2011

Supervisor:	Prof. Raimund Ubar
Opponents:	Prof. Rimantas Sheinauskas, Kaunas University of Technology, LITHUANIA
	Prof. Leo Võhandu, Tallinn University of Technology, ESTONIA
Defence:	September 2, 2011

Declaration:
Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology has not been submitted before for any degree or examination.

/Helena Kruus/

Euroopa Liit
Euroopa Sotsiaalfond

Eesti tuleviku heaks

ISSN 104-4731
ISBN 978-9949-23-149-2 (publication)
ISBN 978-9949-23-150-8 (PDF)

INFORMAATIKA JA SÜSTEEMITEHNIKA C66

Sisseehitatud enesetestimise optimeerimine digitaalsüsteemides

HELENA KRUUS

TTÜ KIRJASTUS

To my family

Abstract

The developments in deep submicron and nanotechnologies allow new design and process technologies to emerge. These technologies provide higher integration densities, smaller interconnection delays and higher system performance, thus enabling new IC paradigms.

This thesis is dedicated to investigations to improve the efficiency and quality of Built-In Self-Test (BIST), which is a solution that allows to overcome many problems related to VLSI and SoC testing.

First, the fault modelling techniques are described and investigated and an overview is given about hierarchical defect modeling. Different defect modeling techniques are described, including Structurally Synthesized Binary Decision Diagrams, High-Level Binary Decision Diagrams and Boolean differential algebra. The proposed defect modelling methods are used for the BIST quality analysis.

Second, the BIST method that combines the use of pseudorandom test sequences and precomputed deterministic vectors is investigated and the optimization techniques for this approach are proposed.

Third, the hybrid BIST with reseeding and test set compaction method is presented and the techniques are propsed for contrained optimization of this approach. This approach allows to find the solution close to the shortest test under the given memory constraint. The method does not require the calculation of the whole solution space and therefore gives results much faster than the exhausive test set compaction method.

The new developed BIST optimization methods were implemented in the industrial designs of biosignal processors developed at the Centre for Integrated Electronic Systems and Biomedical Engineering CEBE. Testability analysis of the benchmark circuits family based on the biosignal processor designs was performed to investigate how different structural implementations would impact on the testability, and to find out which properties of the design will improve the testability.

Kokkuvõte

Submikron- ja nanotehnoloogiate arengusuunad võimaldavad uute disaini- ja tootmistehnoloogiate kasutamist. Need uued tehnoloogiad võimaldavad luua kompaktsemaid skeeme, saavutada väiksemaid süsteemisiseid viiteid, paremat süsteemide jõudlust ning soodustavad uute disainiparadigmade tekkimist ja kasutamist.

Väitekiri on pühendatud digitalsüsteemide isetestimise effektiivsuse ning kvaliteedi tõstmisega seotud uuringutele. Isetestimine on meetod, mis võimaldab lahendada paljusid ülisuurte integraalskeemide testimisega seotud probleeme.

Esiteks on uuritud vigade modelleerimise meetodeid ning on antud ülevaade hierarhilisest defektide modelleerimisest. On kirjeldatud erinevaid defektide modelleerimise meetodeid, sealhulgas struktuurselt sünteesitud binaarsed otsustusdiagrammid, kõrgetaseme binaarsed otsustusdiagrammid ning Boole'i differentsiaalalgebra. On uuritud isetestimise meetodi effektiivsust defektidele orienteeritud testimisel.

Teiseks on kirjeldatud isetestimise meetodit, mis võimaldab kombineerida pseudojuhulikke ning deterministlikke testsignaale ning on välja pakutud algoritmid selle meetodi optimeerimiseks.

Kolmandaks on kirjeldatud isetestimise meetodit kus deteministlikke testsignaale kasutatakse nihkeregistris uute pseudojuhulike jadade genereerimiseks. On välja töötatud algoritm, mis võimaldab kiiresti leida lühima vajalike determinstlike testide ja pseudojuhuslike jadade kombinatsiooni, mis vastaks etteantud mälupiirangutele. Välja töötatud meetod ei vaja kõikide võimalike lahenduste leidmist ning seetõttu annab sobiva tulemuse kordades kiiremini kui originaalkujul.

Välja töötatud optimeerimismeetodeid rakendati biosignaalide protsessorite enesetestimise automatiseerimiseks Eesti Teadustippkeskuses CEBE. On läbi viidud ka selle protsessori erinevate konfiguratsioonide testitavuse analüüs, selgitamaks millised digitaalskeemi omadused soodustavad digitaalsüsteemide testitavuse paranemist.

Acknowledgements

I would like to thank everyone who have supported and advised me during my PhD studies.

First of all, I would like to sincerely express my gratitude to my supervisor Prof. Raimund-Johannes Ubar for providing guidance, support and also tremendous motivation and encouragement in finishing this thesis.

I would also like to thank the whole Department of Computer Engineering team – Gert Jervan, Tarmo Robal, Elmet Orasson, Peeter Ellervee, Jaan Raik, Maksim Jenihhin and all other friends and colleagues.

Moreover, I would like to acknowledge the organizations that have supported my PhD studies: Tallinn University of Technology, National Graduate School in Information and Communication Technologies (IKTDK), Centre of Research Excellence in Dependable Embedded Systems (CREDES), Centre for Integrated Electronic Systems and Biomedical Engineering (CEBE) and Estonian IT Foundation (EITSA).

Finally, I would like to thank my family for the patience and support.

Helena Kruus,
Tallinn, September 2011

Selected publications

Journals

G. Jervan, E. Orasson, H. Kruus, R. Ubar. "Hybrid BIST Optimization Using Reseeding and Test Set Compaction". Microprocessors and Microsystems, 32(5-6), 254 - 262. 2011

R, Ubar, M. Aarna, H. Kruus, J. Raik. "High Quality Test Generation for Digital Systems". Romanian Journal of Information Science and Technology, 8(1), 73 - 84. 2005

Conferences

H.Kruus, R.Ubar, J.Raik. "Defect-Oriented BIST Quality Analysis". 12th International Biennial Baltic Electronic Conference BEC 2010, Tallinn, Estonia, October 4-6, 2010.

H. Kruus, G. Jervan, R. Ubar. "Using Tabu Search for Optimization of Memory-Constrained Hybrid BIST" International Biennial Baltic Electronic Conference BEC 2008, Tallinn, Estonia, October 6-8, 2008, pp.155 - 158.

G. Jervan, H. Kruus, E. Orasson, R. Ubar. "Hybrid BIST Optimization Using Reseeding and Test Set Compaction". 10th EUROMICRO Conference on Digital System Design DSD 2007, Lübeck, Germany, August 29-31, 2007, IEEE Computer Society Press, pp. 596 - 603.

G. Jervan, H. Kruus, E. Orasson, R. Ubar. "Optimization of Memory-Constrained Hybrid BIST for Testing Core-Based Systems". IEEE 2nd International Symposium on Industrial Embedded Systems SIES 2007, Lisbon, Portugal,
July 4-6, 2007. IEEE Computer Society Press, pp.71 - 77.

R. Ubar, G. Jervan, H. Kruus, E. Orasson, I. Aleksejev. "Optimization of the Store-and-Generate Based Built-In Self-Test". 10th Biennial Baltic Electronics Conference BEC 2006, Laulasmaa, Estonia, October 2-4, 2006. IEEE Computer Society Press, pp.199 - 202.

H. Kruus, E. Orasson, T. Robal, R. Ubar. "Investigating Defects in Digital Circuits by Boolean Differential Equations". The 4th International Conference "Distance Learning - Educational Sphere of XXI Century" DLESC'04, Minsk, November 10-13, 2004, pp. 432 - 435.

G. Jervan, Z. Peng, R. Ubar, H. Kruus. "A Hybrid BIST Architecture and its Optimization for SoC Testing". IEEE 2002 3rd International Symposium on Quality Electronic Design ISQED 2002, San Jose, California, USA, March 18-20, 2002. IEEE Computer Society Press, 2002, pp.273 - 279.

R. Ubar, H. Kruus, G. Jervan, Z. Peng. "Using Tabu Search Method for Optimizing the Cost of Hybrid BIST". 16th Conference on Design of Circuits and Integrated Systems DCIS 2001, Porto, Portugal, November 20-23, pp. 445 - 450.

List of abbreviations

ADC	Analog-to-Digital Converter
ATE	Automatic Test Equipment
ATPG	Automatic Test Pattern Generation
BDD	Binary Decision Diagram
BIST	Built-In Self-Test
CA	Cellular Automata
CMOS	Complementary Metal–Oxide–Semiconductor
CUT	Circuit Under Test
DD	Decision Diagram
DFT	Design For Testability
DMBA	Digital Multichannel Bioimpedance Analyzer
DSP	Dedicated Signal Processor
FC	Fault Coverage
FFR	Fan-out Free Region
FPGA	Field Programmable Array
FSM	Final State Machine
GA	Genetic Algorithms
HDL	Hardware Description Language
HLDD	High Level Decision Diagram
HTTF	Hard To Test Faults
HyBIST	Hybrid BIST
IC	Integrated Circuit
IDDQ	Quiescent supply current
ISCAS	International Symposium on Circuits and Systems
JTAG	Joint Test Action Group
LBIST	Logic BIST
LFSR	Linear Feedback Shift Register
LSSD	Level Scan Sensitive Design
MISR	Multiple Input Analyzer
MP-LFSR	Multi-Polynomial LFSR
NoC	Nework-on-Chip
ORA	Output Response Analyzer
PCB	Printed Circuit Board
PPB	Pseudorandom Pattern Block
PRPG	Pseudorandom Pattern Generator
RAM	Random Access Memory
ROBDD	Reduced Ordered BDD
ROM	Read-Only Memory
RPR	Random Pattern Resistant
RTL	Register Transfer Level

SA	Simulated Annealing
SAF	Stuck-At Fault
SoC	System-on-Chip
SSA	Single Stuck-At Fault
SSBDD	Structurally Synthesized BDD
TAM	Test Access Mechanism
TPG	Test Pattern Generator
TRA	Test Response Analyzer
TS	Tabu Search
TSP	Travelling Salesman Problem
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed IC
VLSI	Very Large Scale IC
WPS	Weighed Pseudorandom Sequence

Table of Contents

Chapter 1 Intoduction

1.1 Motivation

Continuous advances in deep submicron and nanotechnologies, as well as in design automation are enabling engineers to design more complex integrated circuits (IC) and driving them toward new design paradigms like System-on-Chip (SoC) and Network-on-Chip (NoC) [1][2]. SoC is usually designed by embedding predesigned complex functional blocks (cores) into one single chip. Such a design style allows to reuse previous designs and lead to shorter time to market and reduced cost.

However, increasing complexity of electronic systems has made testing and fault diagnosis one of the most complicated and time-consuming problems in system design and production [3]. The importance of testing, and design for testability is growing because the expenses of testing are becoming the major components of the design and manufacturing costs of new products. It is estimated that 70% of the design cycle for systems is spent on test and verification [4]. The more complex are systems getting the more probable will be the different kind of failures, and the more important will be the problems of fault modeling, fault detection, fault diagnosis and fault tolerance. Nanometer technologies are introducing new challenges making test quality and dependability of systems a very fast moving target [5]. Enhancing productivity and quality of test related solutions is thus a key competitive aspect, both in terms of time-to-market and end-product quality.

Simulators, fault simulators and test generators as software tools are used widely in electronic design. The quality of these tools rely on efficient simulation, especially in the case of simulation-based test generators (e.g. genetic algorithm-based ones). Traditional low-level (transistor or gate-level) test simulation methods are becoming quickly obsolete because of the rapidly increasing complexity of systems, and high-level approaches are becoming more attractive [6][7]. However, the trend towards higher level modeling moves us even more away from the real life of defects and, hence, from accuracy of testing. To handle adequately defects in deep-submicron technologies, new fault models and defect-oriented test methods should be used. On the other hand, to cope with the complexity of designs, higher-level approaches should be used. To get out from the deadlock, the two opposite trends – high-level modeling and defect-orientation – should be combined into hierarchical approaches [3].

The trend of growing complexities is the same all over the whole semiconductor industry, and as the result has made external test increasingly difficult [8]. Due to the requirements for the Automatic Test Equipment (ATE) speed and memory, the ATE-based test solution may not always be affordable in terms of cost and accuracy. Internal speed of SoC is constantly increasing and the technology used in external testers is always one step behind. Therefore, in order to apply at-speed tests and to keep the test costs under control, on-chip, Built-In Self-Test (BIST) solutions are becoming a mainstream technology for testing complex electronic systems. This trend evolves in accordance with predictions of International Technology Roadmap for Semiconductors [9], which pronounces the Built-In Self-test (BIST) being the main test technology of the future. Therefore, the ultimate goal of any SoC test solution is SoC BIST.

Traditional methods of BIST are based on pure pseudorandom testing which cannot quarantee the needed test quality. Therefore an intensive research has been going on to find solutions how to combine efficiently hybrid and mixed-mode approaches to BIST. Because of a lot of different criteria used in electronics production like design time, testing speed, test quality, restrictions on memory cost, hardware overhead or energy consumption, a lot of tradeoffs should be made, and therefore appropriate test strategies and test scheduling optimization methods are needed to come up with best solutions.

Scan paths are added to designs for implementing BIST, that can result in big number of inputs. Therefore, it is essential to apply optimization techniques in order to optimize the solution space. In this work, new methods for designing optimal BIST processes in digital systems have been developed. Different optimization methods have been implemented and evaluated with the goal to improve the quality of BIST. To adequately evaluate the quality of testing, new hierarchical fault modeling method is used which allows to improve the accuracy of handling physical defects, on one hand, and to cope with the complexity of fault management on the other hand.

1.2 Thesis contribution

The main contributions of this thesis are summarized below.

Fault modeling methods for hybrid BIST have been desribed and the efficiency of classical BIST for defect-oriented fault testing was analyzed.

Hybrid BIST approach for testing systems on chip which combines pseudorandom test sequnce and precomputed determinstic tests in order to achieve high fault coverage with optimal usage of time and memory has been desribed and optimization methods have been developed for selecting the optimal switching moment from pseudorandom test generation mode to the stored deterministic patterns mode.

Hybrid BIST appoach based on reseeding of the LFSR has been desribed. Different optimization techniques are proposed – local search based, test set compaction based on cumulative fault coverage and tabu search based optimization technique. The presented methods provide a possibility to find memory contrained test solution for every core in the multi core system and to use these solutions to construct an optimal test solution to the entire system.

The testability issues of an industrial design for bioimpedance measurements that was developed at the Department of Computer Engineering of Tallinn University of Technology have been analyzed and optimized cost calculation methods for hybrid BIST approach with reseeding have been developed. The experiments have proven the feasibility and efficiency of developed methods for both bioimpendace design and ISCAS'89 family benchmarks.

An overview of the optimization algorithms developed in this thesis is presented in *Fig. 1-1*.

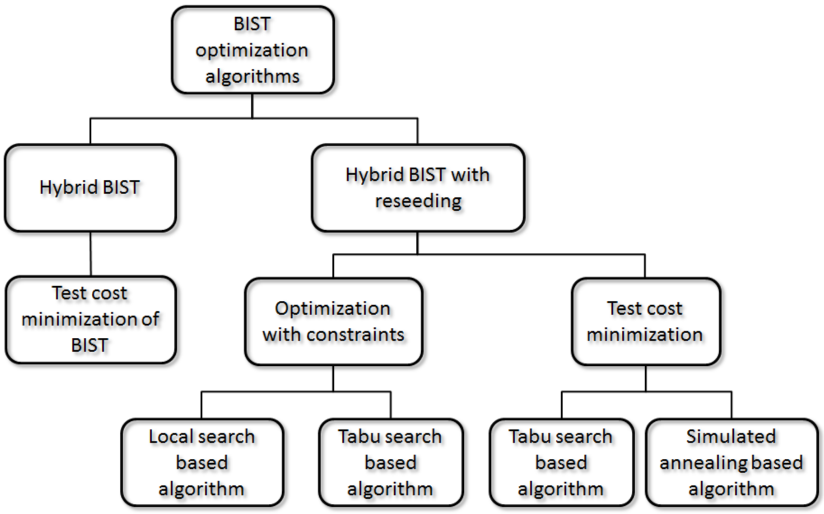

BIST optimization algorithms
Hybrid BIST
Hybrid BIST with reseeding
Test cost minimization of BIST
Optimization with constraints
Test cost minimization
Local search based algorithm
Tabu search based algorithm
Tabu search based algorithm
Simulated annealing based algorithm

Fig. 1-1. *Overview of the optimization algorithms*

1.3 Organization of the thesis

The thesis is organized into 8 chapters.

Chapter 2 gives the background information about testing, failures, test generation and the structure of the Built-In Self-Test (BIST). In Chapter 3, an overview of the state-of-the-art is given – issues of design for testability are discussed, application of Built-In Self-Test are described, short overview of some iterative optimization algorithms is given. Chapter 4 gives an overview of three most widely used iterative optimization techniques – simulated annealing, tabu search and genetic algorithms. Chapter 5 is dedicated to the issues of fault modeling with Decision Diagrams and hierarchical mapping of faults in digital systems for BIST quality analysis purposes. Chapter 6 explains the drawbacks of the classical BIST approach and descibes hybrid BIST. Also, a test cost calculation approach is presented, different cost calculation algorithms are shown and algorithms for optimization of test cost calculation are described and compared. In chapter 7, hybrid BIST with reseeding approach is discussed and algorithms for cost calculation optimization are presented. Chapter 8 decribes test cost minimization methods calculations for hybrid BIST with reseeding and analyzes the testability issues of the industrial design for bioimpedance measurements.

Chapter 2 Background

In this chapter, some background information is provided regarding testing of digital systems, failures and fault models that are used, also basics of test generation are presented and Built-In Self-Test approach is briefly described.

2.1 Testing

When a product is designed, fabricated and tested and it fails the test, then the cause of this failure must be searched for [10]. One of the following things might happen:

- the test was wrong
- the fabrication process was faulty
- the design was incorrect
- the specification had a problem

There are many things that might go wrong. The role of *testing* is to detect whether something went wrong and the role of *diagnosis* is to determine exactly what went wrong and where the process needs to be altered. Therefore, correctness and efectivness of testing is most important for quality products.

The benefits of testing are quality and economy. These two attributes are not independent and neither can be defined without the other. Quality means satisfying the user's needs at minimum cost. So, there are many testing issues that must be addressed during the design and development of a product, but the ultimate goal is to provide quality testing in a cost effective manner [11]. This goal has become more difficult to achieve as VLSI and PCB circuit component densities increase so much that the companies report testing costs to be more than a half of the total product cost [12]

The best way to ensure that the product is testable (with reasonable testing costs) is to consider *design for testability* (DFT) from the very beginning, during the design phase of the product life-cycle.

Testing typically consists of applying a set of test stimuli to the inputs of the circuit under test (CUT) while analyzing the output responses [3]. *Fig. 2-1* represents the testing process [11]. Circuits which produce the correct output responses for all input stimuli pass the test and are considered fault-free. Those circuits that fail to produce a correct response at any point during the test sequence are assumed to be faulty. Testing is performed at various stages in the lifecycle of the device.

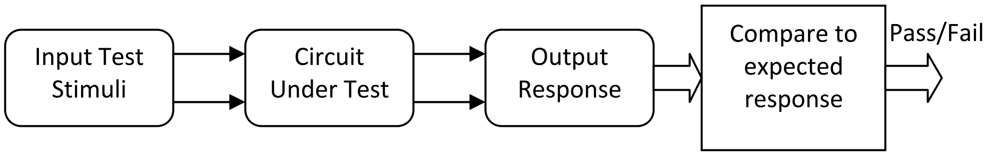


Fig. 2-1. *The flow of the testing process*

There are many possible criteria of classification of testing activities. In general, two different types of testing are applied to each chip:

1) ***Parametric Tests*** for digital circuits are concerned with the external behaviour of the circuits [13] and they include shorts test, opens test, maximum current test, leakage test, and threshold levels test. The specifications of the signal values on the input and output pins of the chip have a time-independent part (voltage and current levels) and a time-independent part (rise and fall times). A special case of parametric testing is I_{DDQ} test method. I_{DDQ} is the quiescent power supply current which is drawn when the chip is not switching. I_{DDQ} tests are based on the fact that defects like shorts and abnormal leakage can increase I_{DDQ} by orders of magntitude. It has been shown that the I_{DDQ} test method is effective in detecting faults of many models.

2) ***Functional Tests*** are aimed to determine the functional accuracy of the circuit under test [14]. To do that, test signals are applied to the inputs of the tested circuit and estimation of the appropriate responses is carried out. Functional tests cover a very high percentage of modelled faults in logic circuits and are mainly used during the manufacturing period, but some forms of these tests can be applied to the devices during the life-cycle to determine whether they work error-free. Realisation of the functional test usually means that the following tasks are solved: selection of input test signals, selections of the investigated parameters and test nodes, and verification of the observed output responses of the tested circuits.

When chips are manufactured, a certain percentage is expected to be faulty because of the manufacturing defects. The **process yield** is defined as the fraction of defect-free parts among all parts. In reality, it is hard to determine the exact value of process yield, as it is impossible to detect all faulty parts. So, the value of yield can be approximated as a ratio:

$$Yield = \frac{Number\ \ of\ \ not\ \ defective\ \ parts}{Total\ \ number\ \ of\ \ parts}$$

where the *number of not defective parts* is determined by counting the parts which pass the used test.

The **defect level** (also known as **reject rate**) is the ratio of faulty parts to all parts that pass the test. Values of defect level are usually given in terms of defects per million. In general, the defect level of 500 parts per million (PPM) chips may be considered to be acceptible. The goal of **zero defects** manufacturing is 3.4 PPM or less.

To measure the quality of the test, **fault coverage** (**FC**) is used. FC can be defined as the following ratio:

$$Fault\ \ coverage = \frac{Actual\ \ number\ \ of\ \ detected\ \ faults}{Total\ \ number\ \ of\ \ faults}$$

The faults are assumed to belong to a particular fault model. In practice, it may be impossible to obtain a fault coverage of 100% due to the fact that there exist faults that are undetectable, which means there is no test to discover those faults. In addition, fault modelling is not perfect - some actual faults may not correspond to modeled faults. So, in reality the fault coverage can be expressed as **fault detection efficiency** and it can be defined in the following way:

$$Fault\ detection\ efficiency = \frac{Number\ of\ detected\ faults}{Total\ \mathrm{n}umber\ of\ faults - number\ of\ undetectable\ faults}$$

To calculate the fault detection efficiency, all the undetectable faults in the circuit should be correctly identified, which is usually a difficult task. Fault covearge is linked to the *yield* and the *defect level* by following expression [15]:

$$Defect\ level = 1 - yield^{(1 - fault\ covrage)}$$

If we assume a manufacturing process where **yield** is 50% and a test with **fault coveage** of 80% then , according to the given formula, **defect level** value would be about 12,95% meaning that 12,95% of shipped parts would be defective. If, for example, defect level of 3,4 PPM is required then with given yield of 50%, the fault coverage should be 95%. Improving the fault coverage can be easier and less expensive than improving the process yield, so it is obvious that generating test stimuli with high fault coverage is very important.

2.2 Failures and fault models

There are many ways to describe incorrectness in electrical systems. The most common terms that are found in literature on testing are *defect, error* and *fault*. The definitions of these terms, according to [10] are the following:

Def 1.1 *A **defect** in an electronic system is the unintended difference between the implemented hardware and its intended design*

Defects can occur either during manufacture or during the use of devices and systems. There are many things that can cause a defect [14]: manufacturing process (missing contacts, parasitic transistors, oxide break-down etc), process fabrication marginality, material and age defects and so on.

Def 1.2 *Wrong output signal produced by a defective system is called an* ***error****.*

We can also say that an error is an "effect" caused by some defect.

Def 1.3 *Representation of a defect at the abstracted function level is called a* ***fault***

The difference between a defect and a fault is rather subtle. Defect means there is something wrong in the hardware, fault means there are imperfections in the functionality. In general, a physical defect in a chip can produce multiple faults and no single test type can detect all the defects.

Roughly, defects can be divided into two major groups [16]:

1. soft defects - defects which may cause speed faults; show up at high speed or produce some temperature; they need two or more test patterns for their activation and test observation; require tests to be applied at speed; examples of soft errors are high resistance bridges, x-coupling, tunneling break etc.

2. hard defects - defects manifested at all frequencies; a test can be applied at slow speed and they need only one-pattern test set; an example of a hard defect can be a bridge at a low resistance)

In order to alleviate the test generation complexity, the actual defects that may occur in a chip need to be modelled at higher levels of abstraction [13]**.** The process of fault modelling considerably reduces the burden of testing because it obviates the need for deriving tests for each possible defect. This is made possible by the fact that many physical defects map to a single fault at the higher level.

There are different levels of abstraction when describing a circuit - behavioural, functional, structural, switch-level and geometric (*Fig. 2-2*)

Behavioral description of a circuit is given by high level hardware description language (VHDL or Verilog, for example). It shows the data and control flow.

Functional description is given at register-transfer level (RTL). It refines the behavioural description. Operations identified at the behavioral level are elaborated upon in more detail. RTL decription can contain registers, modules (i.e. adders and multipliers) and interconnect structures (i.e. multiplexers and buses). This description is sometimes the product of behavioural synthesis which transforms a behavioural level decription into RTL circuit.

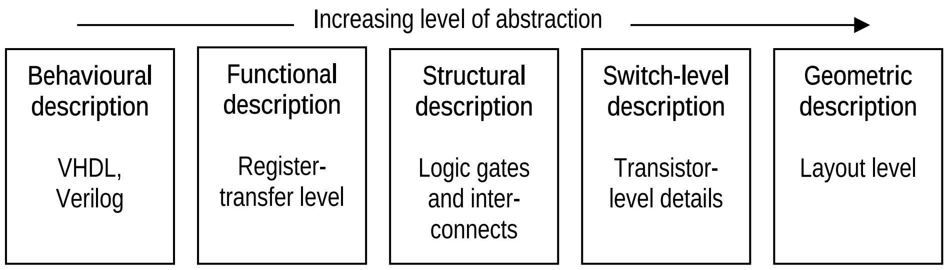


Fig. 2-2. *Levels of abstraction in circuits*

Structural description is given at the logic level. It consists of logic gates, such as AND, OR, NOT, NAND, NOR, XOR and interconnects between them.

Switch-level description establishes the transistor-level details of the circuit. In CMOS technology, each logic gate is described using an interconnection of a pMOS and nMOS network. These networks themselves consist of an interconnection of several transistors. These transistors are usually connected in series-parallel fashion, although non-series-parallel structures are also possible.

Geometric decription is given at the layout level. From this description, line widths, inter-line and inter-component distances and device geometries can be determined.

Modelling of the faults is closely related to the modelling of the circuit [10]. In the design hierarchy, the level refers to the degree of abstraction. Thus, the behavioral level (also referred to as high level) has fewer implementation details and fault models at this level may have no obvious correlation to manufacturing defects. High-level fault models play a greater role in simulation based design verification than in testing.

The first requirement of a good fault model is that it accurately reflects the behaviour of the actual defects that can occur during the fabrication and manufacturing process as well as behaviour of faults that can occur during operation of the system [11]. The second requirement of a good model and just as important as the first, is that it must be computationally efficient with respect to the fault simulation environment. There are many fault models described in [17]. Unfortunately, no single fault model can accurately reflect the behaviour of all the possible defects that can occur.

Behavioral fault models are defined at highest level of abstaction and they are based on the behavioural specification of the system. The type of faults that are included in a behavioural fault model depends on how easily is allows the detection of realistic faults at the lower levels of abstraction.

Functional fault models are defined at the functional block level. The purpose of these models is to make sure that the functions of the functional block are executed correctly. In addition, they should also make sure that unintended functions are not executed.

Structural fault models assume that the structure of the circuit is known. Faults under these fault models affect the interconnections in this structure. The most well-known fault model in this category is the *single stuck-at fault model.*

Switch-level fault models are defined at transistor level. The most prominent faults here are *stuck-open* and *stuck-on* fault models. The *stuck-open* fault is the case when a transistor is permanently non-conducting because of the fault. In case of *stuck-on*, the transistor is permanently conducting. These faults are specifically suited for CMOS techology. Because of using multiple transistors to construct

CMOS logical gates, the *stuck-at* fault model cannot accurately reflect the behaviour of these faults.

Geometric fault models assume the layout of the chip is known. To develop these fault models, the knowledge of line widths, inter-line and inter-component distances and device geometries is needed. Problems with the manufacturing process can be detected at this level. *Bridging fault model* is one example of geometric fault models, which can lead to accurate detection of realistic defects.

Stuck-at fault model

The s*tuck-at fault* (SAF) is a logical fault model that has been sucessfully used for decades [3] and according to [9] will remain to be the one of the fault models most utilized for testing of microelectronics for the next years.

The stuck-at fault model assumes that the elementary components are fault free and affects the state of logic signals on interconnects in a logic circuit, including primary inputs (PIs) and primary outputs (POs), internal gate inputs and outputs, fanout stems (sources) and fanout branches. The stuck-at fault transforms the correct value of the faulty signal line to appear to be stuck at constant logic value, either a logic 1 or logic 0 which are referred to as ***stuck-at-1*** (*SA1*) ***and stuck-at-0*** (*SA0*) respectively.

Generally speaking, there can be several stuck-at faults simultaneously presented in a circuit. A circuit with *n* lines can have *3n-1* possible stuck combinations [10]. This is because each line can be in one of three states: *SA1*, *SA0* or *fault-free*. All combinations except one having all lines in fault-free state are counted as faults. Obviously, even a moderate value of *n* will give an enourmously large number of multiple stuck-at faults. Therefore, its is common practice to model only one single stuck-at fault, so an *n*-line circuit can have at most *2n* single stuck-at faults. This number is further reduced by fault collapsing [17].

The stuck-at fault model is the most often used fault model in automatic test pattern generation (ATPG) systems. In ATPG systems, the following presumption is made - only one single and permanent fault is considered at the time. There are three properties that characterize a single stuck-at fault (or SSA) model:

- only one line is faulty
- the faulty line is permanently set either logical 1 or logical 0
- the fault can be assumed at an input or an output of the gate

The stuck-at fault model has been the industrial standard since 1959 and despite its death has been predicted there are a lot of reasons and properties making stuck-at fault model still being widely used in testing [18]:

- stuck-at model is simple enough and it is easily applicable

- as the fault behaviour can be determined logically, the simulations are straightforward and deterministic
- when using the stuck-at model the result is easy to measure - detection/not detection are easy
- the model is adaptable - it can be easily applied to transistors, gates, registers, systems etc.

Unfortunately, the stuck-at fault model has a major disadvantage - there are defects that cannot be covered using this model, mainly in CMOS technologies. Sometimes the **multiple stuck-at fault** model is used, meaning that there are more than one stuck-at faults simultaneously present in the circuit. The disadvantage of this approach is a huge number of possible combinations. If the circuit has *k* lines it can have *2k* single stuck-at faults, two for each line. However the number of multiple stuck-at faults is *3k-1* (similarly to single stuck-at fault case described above). Clearly, even for relatively small values of *k*, testing all multiple stuck-at faults is impossible. Also, this approach does not give the defect coverage that can be considered significant enough. A subset of the multiple stuck-at model, called the **unidirectional stuck-at fault model**, is also sometimes used. In this model, all the affected lines are assumed to be stuck at the same logic value, either 0 or 1.

Other fault models

The popularity of the stuck-at fault model depends on the fact that it can be applied to various semiconductor technologies, and the detection of all single stuck-at faults results in detection of majority of realistic physical defects (in many cases, up to 80-85% of the defects are detected). However the stuck-at fault model has its drawbacks (nether the less it is widely used), as many defects of the CMOS technologies cannot be covered [19][20][21]. Using the multiple stuct-at fault model does not increase the fault coverage significantly enough. There are number of other fault models that are used, such as *bridging faults, opens, delay faults, parametric faults*.

Bridging faults

Bridging faults (also known as *shorts*) cover all defects and failure mechanisms that cause unintentional elctrical connections across two or more circuit nodes. The causes for the bridging faults can be [14]:

- extra conducting material (e.g. photo-litographic printing error, conductive particle contamination etc causing horizontal shorts)
- missing insulating material (e.g. printing error, gate oxide defect causing pinhole, insulating particle contamination etc. causing vertical shorts)

Bridging fault models can be derived at various levels of abstraction. At the geometric level, such a fault model is the most accurate. However, bridging fault models can also be defined at the structural or switch levels. Bridging faults can be classified into two main groups: *inter-gate shorts* (shorts at the logic terminals of the gate) and *intra-gate shorts* (shorts at transistor nodes). For non-CMOS technologies, a bridging fault between two lines is assumed to result in **wired-AND** or **wired-OR** [22]. The wired-AND bridging fault means the signal net formed by the two lines will take on a logic 0 if either shorted line is sourcing a logic 0 (*0-dominant bridge*), while the wired-OR bridging fault means the signal net will take on a logic 1 if either of the two lines is sourcing a logic 1 (*1-dominant bridge*). However, shorts in CMOS circuit cannot be just mapped to either of these [13].

Bridging faults are sometimes also categorized as *feedback* and *non-feedback* faults. If one or more feedback paths are created in the circuit due to the fault, then it is called a feedback fault, otherwise a non-feedback fault. Non-feedback bridging fault coverage by SAF test is normally very high. Bridging faults can be detected by applying opposite logic values to the two wires [11]. I_{DDQ} techniques can be used to detect bridging faults.

Open circuit defects

Open circuit defects (also known as *opens*) can be interpreted as unintentional electrical discontinuities. Due to defects a node splits up into two or more distinct nodes. These type of defects can be caused by improper etching, masking error, electro-migration etc. There are two types of opens: *narrow* and *large* [20]. They can cause behaviour that may vary greatly and might be difficult to predict. The manifestation of an open defect depends not only on the size of the crack but also factors such as temperature, clock frequencies, location, and technological parameters, as well. Opens can be located at the gate level and also at the transistor level.

In literature, there exist different classifications for open defects. The basic general division is based on the charge transfer rate function with dependence on the break size and three groups can be recognized [18]:

- almost open or resistive open
- completely open (a stuck-open - the special case of a resistive open defect in which the resistance is very large)
- tunneling open

Some experiments with resistive open and completely open classes are reported in [23]. At the transistor level, classification of opens was done and six main open fault classes were specified in [24].

Delay faults

In order to logic circuit to operate fault-free, not only performing the logic function correctly is required, but also propagating the correct logic signal along paths within specified time limits are important.

The **delay faults** cause excessive delay along a path such that the total propagation delay falls outside the specified limit. Studies have shown that delay faults can be caused by resistive bridges with a resistance value above the critical resistance [25][26] which could be caused by the number of reasons such as weak transistors, subtle manufacturing, process defects etc.

The following models have been proposed for delay testing:

- **gate delay** and **transition delay** fault models, where each unit is designed with a pre-specified nominal delay; a delay fault occurs when the time-interval taken for a transition from the gate to output exceeds the given nominal delay
- **path delay** fault model consideres cumulative propagation delay along a signal path through the circuit - in other words, the sum of all gate delays along the path
- **transition faults** are faults of a gate characterised as *slow-to-rise* and *slow-to-fall* types; these fault times are used in time specification testing
- **line delay faults** are rising and falling delays on a given signal line
- **segment delay faults** mean delay through a chain of combinatorial gates with a specified length *L*.

Crosstalk

The use of nanometer technologies increases cross-coupling capacitance and inductance between interconnects, leading to sever cross-talk effects that may result in improper function of the chip [3]. Crosstalk effects can be separated into two categories:

- **crosstalk gliches** are pulses that are provoked by coupling effect among interconnect lines; the magnitude of the glich depends on the ratio of the coupling capacitance to the line-to-ground capacitance.
- **crosstalk delays** are signal delays that are provoked by the same coupling among interconnect lines, but it may be produced even if line drivers are balanced but have large loads.

As the crosstalk causes a delay in addition to normal gate and interconnect delay, it is difficult to estimate the true circuit delay, which may lead to severe signal delay problems.

Several design techniques, including physical design and analysis tools, are being developed to help design for margin and minimization of crosstalk problems; however, it may be impossible to anticipate in advance the full range of process variations and manufacturing defects that may significantly aggravate the cross-coupling defects.

2.3 Test generation

During the manufactoring process, defects may occur that may result in a faulty behaviour of the chip. The purpose of test generation is to generate a test vector for a fault in given circuit or declare it untestable. The practical version of test generation requires the generation of a set of test vectors which collectively detect all, or a maximal fraction of the testable faults in the given fault list [13].

The tests are generated by an automatic test pattern generator (ATPG) and applied to the circuitry under test, using the automatic test equpment (ATE). ATPG is the application of algorithmic based software to generate test patterns. The traditional goal of ATPG is to achieve high fault coverage by producing a small volume of test patterns.

Since test vectors are usually capable of detecting many faults in a circuit, *fault simulation* is typically used to evaluate the fault coverage obtained by that set of test vectors.

There exist several approaches to generating test patterns. Each test generation algorithm is evaluated by the following measures:

- test effectivness
- fault coverage
- test generation time
- length of the generated test

In the following, a description of some classes of test patterns are given.

Deterministic test patterns are developed targeting a specific fault or defect in a given circuit. In context of ***Built-In Self-Test*** applications, they are often referred to as "stored patterns". Deterministic testing is also known as *structural testing* and was introduced in 1960s [3]. There are number of algorithms developed, such as D-algorithm [27], which uses a logical value to represesnt both "good" and the "faulty" circuit values simultaneously and can generate a test for *stuck-at fault* as long as such exists. The next important algorithm developed was the PODEM algorithm (*Path Oriented DEcision Making* [28]) where path propagation contraints were used to limit ATPG search space and the notion of *backtrace* was introduced. FAN (fan-out oriented test generation algorithm [29]) and SOCRATES [30] were also very important developments, accelerating the ATPG process. Also, several more advanced algorithms exist, such as *dominator* ATPG approach TOPS

(TOpological Search [31]), *evaluation-frontier* approach EST (Equvalent STate hashing [32][33], Neural Network ATPG (NNATPG, [34][35]) and others.

Exhaustive test patterns: every possible combination of input test patterns is produced. When this test is applied, all detectable faults are covered for combinatorial circuits. As for a circuit with N inputs 2^N test patterns should be generated, this approach is obviously not practical for the large N.

Pseudo-exhaustive test patterns: this is an alternative for exhaustive test patterns [36]. The circuit is partitioned and every combinatorial logic subcircuit is exhaustively tested. In this case, the number of test patterns is much smaller as every K-input partition will receive 2^K patterns and $K<N$. The feasibility of this approach is obvious if we consider for example a parity generator network of the TI SN54/74LS630 wich has 23 inputs and 6 outputs where each output depends on only 10 of the inputs [36], so instead of 2^{23} patterns, only $6 \times 2^{10}=6144$ patterns are needed.

Pseudo-random test patterns: typically, these patterns are produced by Linear Feedback Shift Registers (LFSRs) or Cellular Automata (CA), LFSRs being by far most popular devices in use. The patterns have characteristics similar to those of random patterns but deterministic algorithm is used, and the sequences are repeatable. The advantage of pseudorandom testing is that very simple hardware and small design efforts are needed to implement it and the fact that these sequances can be repeated (as opposed to truly random patterns which cannot be). However, one strong disadvantage is the fact that pseudo-random test tends to be very long and due to the presence of random pattern resistant faults in many cases the high fault coverage is hard to achieve, meaning also long test application times and high cost of fault simulation. So, in many cases the pseudorandom test sequence is combined with deterministic vectors, targeting specific faults.

In many cases, digital sytems are implemented as sequential circuits, which involve combinatorial logic and memory elements - the combinatorial part produces the result that is stored in memory elements (usually flip-flops). The testing of sequential circuits is much more complex than testing of combinatorial logic due to the fact that these circuits contain internal memory whose state is not known at the beginning of the test and a test for a fault in sequential logic contains three parts - initialization of the internal memory, combinational test and observation of the sate of affected elements. To compare, any fault in combinational circuit can be detected by a single vector.

There are several algorithms developed for generating test vectors for sequential circuits. One class on these algorithms employs *time-frame expansion method* - the most well-known of the implementations are ESSENTIAL [37], GENTEST [38][39], HITECH [40], SEST [41] and FASTEST [42]. Also, there are some approximate methods developed - SCIRSS system [43], STALLION [44] program and STEED [45]. In the class of *simulation-based methods* most well-known algorithms are CONTEST [46][47], CRIS [48], GATEST [49], GATTO[50] and

STARGATE [51][52]. Many of them are using Genetic Algorithms (GA), introduced by Holland [53] and described thouroughly by Goldberg in [54]. The simulation-based methods are applicable to any types of circuits, combinatorial or sequential.

Even though most practical circuits are sequential, they often incorporate the full-scan design for testability (DFT) feature, which enables tests to be generated using combinatorial generator.

2.4 Built-In Self-Test

The major argument for using Built-In Self-Test (BIST) is reduced dependence on expensive testers [56]. Nowadays, the testers are a major investment. As BIST approach can reduce or even eliminate this investment, it becomes more and more attractive as an alternative approach to test. To economically justify using of BIST, it is not even necessary to eliminate testers from the manufaturing flow completely. If the duration of a test can be reduced by generating stimuli and computing response on-chip then it becomes possible to achieve the same throughput with fewer, and possibly less expensive, testers. Also, when a new faster chip is released, the BIST circuits will benefit from that performance enhancement, making it possible to complete the test in less time.

BIST can substantially reduce the data management problem related to outside tester. When BIST is used to test a circuit it may be that the only input stimulus required is a reset that puts the circuit into test mode and forces a seed value in a pseudo-random pattern generators (PRG). Then, if a tester is controlling the self-test, a predetermined number of clocks are applied to the circuit and a response, called a signature is read out and compared to the expected signature.

One more advantage of BIST is that many thousands of pseudo-random vectors can be applied in BIST mode in the time that takes to load a scan path a few hundread times. As the test vectors come from PRG, there is no storage requirement for test vectors. It should also be mentioned that loading the scan chain(s) for every vector can be time-consuming, implying tester cost, in contrast to BIST where a seed value is loaded and then the PRG immediately starts generating and applying a series of test vectors on every clock. A further benefit of BIST is the abilty to run at speed which improves the likelyhood of detecting delay errors.

The basic architecture of BIST is shown on *Fig. 2-3*[11]. This BIST architecture includes two essential functions as well as two additional functions that are necessary to facilitate execution of the self-testing feature wile in the system. The two essential functions include the test pattern generator (TPG) and output response analyzer (ORA). While the TPG produces a sequence of patterns for testing the CUT, the ORA compacts the output responses of the CUT into some type of *Pass/Fail* indication. The other two functions needed for system-level BIST include the test controller (or BIST controller) and the input isolation circuitry. Aside from the normal system I/O pins, the incorporation of BIST may also require additional I/O pins for activating the BIST sequence (the *BIST Start* control signal), reporting the results of the BIST (the *Pass/Fail* indication), and an optional indication (*BIST Done*) that the BIST sequence is complete and that the BIST results are valid and ca be read to determine the fault-free/faulty status of CUT.

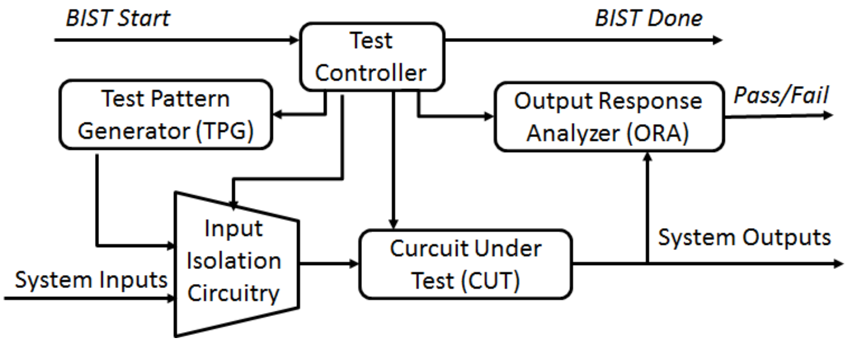
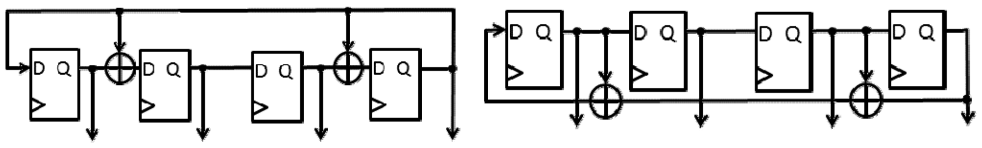


Fig. 2-3. *Basic BIST architecture*

The basic building block of BIST is linear feedback shift register LFSR - a simple n-stage counter that can generate 2^n unique input vectors but the high-order bit would not change until half the stimuli had been created and it would not change again until the counter returned to its starting value. LFSR can create pseudo-random sequences and it can be used to create signatures. When used to generate stimuli, the stimuli can be obtained serially, by either the high- or low-order stage of LFSR, or stimuli can be acquired from all stages in parallel.



a) internal feedback LFSR b) external feedback LFSR

Fig. 2-4. *LFSR implementations*

There are two basic types of LFSR implementations, the internal feedback LFSR and external feedback LFSR. Both are shown in *Fig 2-4*. Internal and external feedback LFSRs are dual to each other. Both implementations require the same amount of logic in terms of exclusive-OR gates and flip-flops. The internal feedback LFSR provides the implementation with the highest maximum operating frequency for use in high performance applications. The main advantage of external feedback LFSRs is the uniformity of the shift register; hence, there are some applications where external feedback is preferred.

As BIST enables a circuit to test itself, one of the main advantages of BIST is that is can be easily used for all levels of testing. Also, as BIST is incorporated into a VLSI device, it allows at-speed testing and reduces the need for external test equipment. In addition, time-to-market is significantly reduced. There are, however, some disadvatages to BIST solution - one of the most significant disadvantages is additional design time and effort that is needed. Due to additional circuitry, there is area overhead and performance penalies, that need to be considered.

Chapter 3 State-of-the-art of BIST

In this chapter an overview is given about the state-of-the-art in the related fields of the problem investigated in the thesis. First, a general overview about the broad field of design for testability is given, and the conclusion is made that BIST has become a mainstream. Different approaches to BIST are analyzed, and the drawbacks are highlighted. A general conclusion is made that the problem of BIST optimization is insufficiently investigated. It is also found that the traditional stuck-at-fault model is not adequate for using in evaluation of the BIST quality for deep-submicron technologies. The state-of-the-art of fault modelling techniques for digital systems is presented. Finally, an overview is given about the different methods which can be used for solving BIST optimization tasks.

3.1 Design for testability

The last step of chip manufacture is the operation referred to as "testing". The goal of testing is to recognize whether chip is working properly, i.e. no faults exist in the chip and the customer will obtain a good chip with proposed property. The designer of a circuit must consider both design and testability properties. Design for testability (DFT) methodologies help to detect possible faults, keep the test execution time and test developing time economical [14].

Design for testability can loosely be defined as changes to a given circuit design that help decrease the overall difficulty of testing [13]. The changes to the design typically involve addition or modification of circuitry such that one or more new modes of circuit operation are provided. Each new mode of operation is called a **test mode** in which circuit is configured only for testing. During normal use, the circuit is configured in the **normal mode** and has identical input-output logic behaviour as the original circuit design.

In the following, a basic overview of DFT will be given, including ad-hoc techniques, full and partial scan design techniques. The purpose is to provide the necessary background needed to understand advantages and disadvantages of Built-In Self-Test (BIST) compared to other DFT techniques [11].

There are two major concepts which are commonly used in assessing and enhancing the testability of a circuit under test: controllability and observability [13]. Controllability is a measure of how difficult it is to set a line to a value necessary to excite a fault. Observability is a measure of how difficult it is to propagate a faulty signal from a line to a primary output.

Integrated circuits are tested in the phase of prototyping new circuits and in the phase of serial production. The goal is to separate the good chips from the faulty ones. To enable testing, various structures can be inserted into the circuit to make the test application easier. While the additional DFT circuitry may increase design time, the resultant reduction in time-to-market is realized.

There are three main types of DFT approaches for digital circuits. These include:

- **ad-hoc techniques** (methods that are targeting difficult-to-test parts of the circuit under test)
- **scan design techniques** (also referred to as Level Scan Sensitive Design (LSSD), approaches that use scan architecture to conduct testing)
- **built-in self-test** (incorporates test pattern generation and output response analysis capabilities inside the chip)

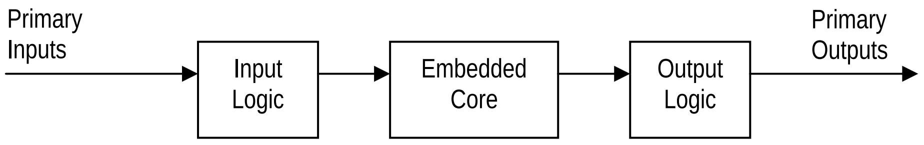
3.1.1 Ad-Hoc techniques.

The first DFT approach is referred to as *ad-hoc DFT.* The aim of this approach is to improve observability and controllability of the difficult-to-test portions of the design. Gate inputs and outputs, which are normally out of control or out of observation, are made accessible by inserting test points. This is usually done by incorporating multiplexers internal to the CUT to create one or more test modes of operation in which the primary inputs and outputs provide access to or from internal difficult-to test circuits via multiplexers. Since test points are inserted only where needed, control of the area overhead can be maintained and one can trade-off area overhead for fault coverage.

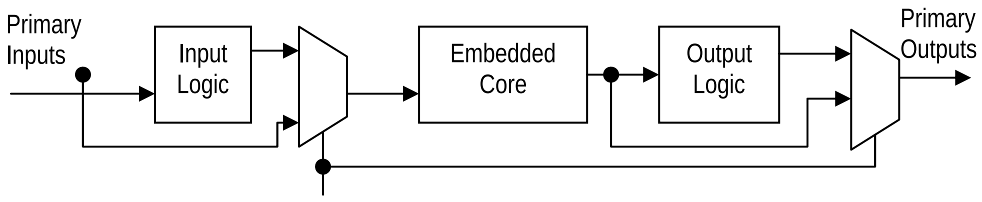
The down side of *ad-hoc* DFT is the number of additional I/O pins required for controlling the test modes, and the fact that we must determine the best place to insert test-points, developing test-vectors, running fault simulations, evaluating fault coverage, and repeating the entire process until the desired fault coverage is obtained.

Typical targets for test point insertion are feedback loops, large counters, embedded core logic, asynchronous logic, embedded clock generators, memory initialization inputs, intentional redundant logic [17].

An example of test point insertion is illustrated in *Fig. 3-1.* In the presented case, existing test vectors previously developed for the core logic can then be used to test the embedded core. While the example may not be very realistic, it does illustrate some system level considerations that must be addressed. As long as there are more primary inputs and outputs than inputs to and outputs from the embedded core logic, only one additional pin is needed for the *Test Mode* input. However, as there are multiplexers inserted between input logic and the embedded core, the multiplexer delay of the performance penalty is added. Also, multiplexers are added to the primary outputs, which adds the clock-to-output delay to the overall circuit. Therefore, in case of the time critical requirements, those primary outputs should be avoided.



(a) Before ad-hoc DFT



(b) After ad-hoc DFT

Fig. 3-1. *Ad-hoc DFT for embedded core logic*

3.1.2 Scan Design techniques.

Scan design is defined as the process of using scan architecture to conduct testing. The main idea of scan design is to obtain control and observability for flip-flops [10]. This is done by adding a test mode to the circuit such that when the circuit is in this mode, all flip-flops functionally form one of the shift registers. The inputs and outputs of these shift registers (also known as scan registers) are made into primary inputs and primary outputs. Thus, using the test mode, all flip-flops can be set to any desired states by shifting the contents of the scan register out. All flip-flops can be set or observed in a time (in term of clock periods) that equals the number of flip-flops in the longest scan register. In practice, however, a design can have any number of them.

As a result of adding the scan chain to the logic circuit, all flip-flops are easily controllable and easily observable [11]. Therefore, the problem of testing sequential logic circuit is reduced to simply that of testing combinatorial logic. The excellent controllability and observability of the flip-flops in the chain can also provide good controllability and observability of the embedded combinatorial logic of the CUT.

The area overhead and performance penalties associated with scan design are due to the multiplexers added to the inputs of each flip-flop. Typical area overhead values are on the order of a 2% to 10% increase of the total chip area [10].

Scan design based DFT does have drawbacks, including long test application time due to the serial application of the test vectors and retrieval of test results. Another problem at system-level testing is the difficulty of applying the tests at the system operating frequency. Also, using scan design-based approach makes the detection of transistor stuck-off faults and delay faults detection extremely difficult, as the result of the test patterns shifting through the scan chain between the application of each vector.

There are number of restrictions placed on circuits that are candidates for scan design implementation [10]. The circuit must use edge-triggered, D-type flip-flops that are clocked from the primary inputs. These restrictions are required in order to implement and correctly operate the scan chain shift register in both normal and scan modes of operation. Despite its drawbacks, scan design can be used at all levels of testing and can be applied hierarchically to chips, PCBs and systems.

3.1.3 Boundary Scan

The methods described are able to test an isolated part of integrated circuit (IC), or an isolated IC in IC tester. It assumes direct connections between IC tester and the tested integrated circuit [14].

The success of scan design led to the application of scan design-based DFT techniques for testing interconnect and solder joints on surface mount PCBs [11]. This became known as Boundary Scan, also referred to as JTAG (Joint Test Action Group).

Boundary Scan is in fact a family of test methodologies aimed at resolving a wide range of test problems: from chip level to system level, from logic cores to interconnects between cores, from digital circuits to analog or mixed-mode circuits, and from ordinary digital designs to very high-speed designs [3]. Standard 1149.1, usually referred to as *digital boundary-scan standard*, was approved by the IEEE in 1990. It defines general-purpose boundary-scan implementation for digital chips. The architecture based on Boundary Scan enables scanning in and out a shift register which controls circuit inputs and outputs, and it can be used for performing several other functions.

The board-level testing problems that Boundary Scan solves are significant. The Boundary Scan interface and circuitry can be used at all levels of testing and Boundary Scan can be hierarchically applied to the entire system. In addition to testing interconnections, the boundary scan interface provides access to other testing functions, including *Built-In Self-Test*.

The overhead associated with Boundary Scan can be significant. For example, for each I/O pin, two flip-flops and two multiplexers are added. Also, other parts of boundary scan architecture such as TAP controller, Instruction Register, Instruction Decoder etc. also have considerable impact on the area overhead.

3.2 Built-In Self-Test in digital systems

The rapid developments in the areas of deep-submicron electron technology are enabling engineers to design more and more complex integrated circuits, driving them towards design methodologies called System-on-Chip (SoC). SoC approach is very attractive from the designers' perspective. This technology enables designers to embed predesigned and preverified complex functional blocks, usually referred as cores, into a single die. Such a design style allows designers to reuse previous designs and will lead to a shorter time to market and a reduced cost. The cores can be very different by their nature (from analog to memories, including all types of logic) and can be represented in several different ways (RTL code, netlist or layout). Testing of SoC, on the other hand, shares all the problems related to testing modern deep submicron chips, and introduces also some additional challenges due to the protection of intellectual property as well as the increased complexity and higher density.

To test the individual cores of the system, the test pattern source and sink have to be available together with an appropriate test access mechanism (TAM) [56]. We can implement such a test architecture in several different ways. A widespread approach implements both source and sink off-chip and requires therefore the use of external Automatic Test Equipment (ATE). However, the complexity and speed of digital systems make external test difficult [58]. Since the internal speed of SoC is constantly increasing, the demands to the ATE memory size are increasing and the technology used in ATE is always one step behind. On the other hand, the ATE solution is becoming unacceptably expensive and inaccurate, and leading also to an unacceptable yield loss [9]. Therefore, in order to apply at-speed tests and to keep the test costs under control, on-chip test solutions are needed. Such a solution is usually referred to as Built-In Self-Test (BIST) [59][60]. Built-In Self-Test (BIST) has become a mainstream.

BIST is aimed at detecting faulty components in a system by incorporating test logic on-chip. In traditional BIST, test generation is mostly performed by ad hoc circuitry, typically Linear Feedback Shift Registers (LFSR) [59], cellular automata [60] or multi-functional registers like (Built-in Logic Block Observers) BILBO [3][13]. As it was described in *Section 2.4*, the classical way to implement the TPG for BIST is to use linear feedback shift registers (LFSR). But as the test patterns generated by the LFSR are pseudorandom by their nature and have linear dependencies [61], the LFSR-based approach because of random pattern resistant (RPR) faults [3][13] often does not guarantee a sufficiently high fault coverage (especially in the case of large and complex designs) and demands very long test application times in addition to high area overheads.

Several methods are used to improve fault coverage by inserting test points into the Circuit Under Test (CUT) [62], or using weighted pseudorandom sequences (WPS) [63]. In these approaches the hardware overhead may become large.

More efficient are mixed mode or hybrid BIST approaches [64]-[74] where deterministic data are combined with pseudorandom ones to improve detection of RPR faults, and compared to WPS less additional hardware is required. The pseudorandom and deterministic data are combined in different ways like using ROM compression [64], LFSR reseeding [63] either by bit-flipping [65] or bit-fixing [66], multi-polynomial scheme[67], embedding deterministic patterns[68]. However, in most of these approaches the architecture is extremely tailored to the CUT, and any change in the CUT requires resynthesis of the complete BIST hardware. Another drawback of traditional BIST is the use of special hardware for TPG on chip, which causes area overhead and performance degradation. Recently new methods have been proposed to reduce the hardware overhead, which exploit specific functional units such as arithmetic units or processor cores for on-chip test generation and test response evaluation [69][70]. This approach called functional BIST has the same disadvantages regarding the test quality as pure pseudorandom testing. Therefore a research is needed here how to combine hybrid and mixed-mode approaches with functional BIST. Because of a lot of different criteria used in electronics production like design time, testing speed, test quality, restrictions on memory cost, hardware overhead, energy consumption etc. a lot of tradeoffs should be made, and therefore appropriate test strategies and test scheduling optimization methods are needed to come up with best solutions [75].

The main concern of the hybrid BIST or reseeding BIST approaches has been to improve the fault coverage by mixing pseudorandom vectors with deterministic ones, while the issue of cost minimization or optimization according to the given criteria has not been addressed directly.

Very important is how the quality of BIST is evaluated. The traditional approach to characterize the quality of BIST is to use the measure of stuck-at-fault coverage (SAF). However, the traditional SAF model used in testing of digital circuits does not quarantee the quality of testing for deep-submicron technologies [76][77]. For adequately characterizing the today's BIST solutions and the test sequences generated by BIST, more advanced fault modeling methods are to be used.

3.3 Fault modeling

The reason why high SAF coverage can not quarantee high quality testing is that the model ignores the actual behavior of deep-submicron circuits, and does not adequately represent the majority of real IC defects and failure mechanisms which often do not manifest themselves as stuck-at faults. The types of faults that can be observed in a real gate depend not only on the logic function of the gate, but also on its physical design. These facts have been well known, but usually, they have been ignored in engineering practice, and the SAF model is used still as *de facto* standard. In earlier works on layout-based test techniques [76][77], the whole circuit having hundreds of gates was analyzed as a single block. Such an approach is computationally expensive and highly impractical as a method of generating tests for real VLSI designs.

Traditional fault simulators based on the single SAF model handle simple physical defects which force a single site to a fixed logic value of 0 or 1. For better modeling of arbitrary physical defects in the circuit components of nanometer technology, a conditional fault model has been proposed as one extension of the classical SAF model [78][79]. A conditional SAF model consists of a signal line with SAF (as a topological part of the model) and an activation condition (the functional part). Such a metric has been used for many years under different names like fault tuple model [80], pattern fault model [81], input pattern fault model [82], or functional fault model [83] which can represent any arbitrary change in the logic function of a circuit block, where a block is defined to be any combinational subcircuit described at any level of the design hierarchy. For complete exercising of blocks in combinational circuits on the gate level, a similar pattern oriented gate-exhaustive fault model was proposed in [84], which was extended to target bigger regions (collections of gates) by region-exhaustive fault model in [85]. Many researchers have focused on developing new fault models for particular types of failure mechanisms like signal line bridges [86]-[90], transistor stuck-opens [87][88], failures due to changes in circuit delays [93] etc.

A more complex defect, such as a resistive short or open, causes multiple effects around the defect site. For example, the behavior of a fanout gate may be affected by a defect which forces on the fanout branches of the gate intermediate voltages. As a result, multiple faulty logic values may appear on the fanout branches depending on the threshold voltages of the branches. A unified fault model for interconnect opens and bridges using constrained multiple line stuck-at faults is proposed in [94]. To deal with the ambiguities of the changing logic values on the branches, the Byzantine fault model was introduced [95][96] where a floating line with n branches may lead to *2n – 1* possible fault cases. Methods are proposed to reduce the number of *2n – 1* to a reasonable smaller subset, which however needs additional information about the layout, vias or buffers, threshold voltages of the transistors driven by the floating nodes, or about the occurrence probabilities of possible logic behaviors of physical defects [95].

To handle adequately defects in deep-submicron technologies, new defect-oriented fault models should be used. But, the defect-orientation is increasing the complexity. To get out from the deadlock, hierarchical approaches for diagnostic modeling have been proposed. One of the attractive ways to manage hierarchy in diagnostic modeling (test generation, fault simulation, fault location) in a uniform way on different levels of abstraction is to use decision diagrams (DD) [97]-[100].

Binary Decision Diagrams (BDD) were first introduced for logic simulation in [97], and for test generation in [98][99]. In 1986, Bryant proposed a new data structure called reduced ordered BDDs (ROBDDs) [100]. After this publication, the BDDs have become very popular. In [98][101], a special class of Structurally

Synthesized BDDs (SSBDD) were introduced. The most significant difference between the function-based BDDs [100] and SSBDDs [101] is in the method how they are generated. While BDDs are generated on the functional basis, the SSBDDs are generated directly from the structure of the circuit. This allows to establish between the faults of the circuit and the nodes of the SSBDD one-to-one mapping. The idea of representing the structure in DDs was generalized from logic level SSBDDs to High-Level DDs (HLDD) [101][102]. The similar feature of SSBDDs and HLDDs to model at the nodes the faults of digital systems at different levels of abstraction gives a good possibility to develop a uniform fault model for digital systems.

The advantage of hierarchical approaches compared to the plain gate-level modeling lies in the possibility of constructing test plans on higher levels, and modelling physical defects on more detailed lower levels. To handle physical defects in fault simulation, we need higher level logic fault models for the following reasons: to reduce the complexity of simulation (many physical defects may be modelled by the same logic fault), a single logic fault model may be applicable to many technologies, logic fault tests may be used for physical defects whose effect is not well understood. But the most important reason for logical modelling of physical defects is to get a possibility for moving from the lower physical level to the higher logic level which has less complexity. Furthermore, it would be possible to reduce even more the complexity of fault simulation by moving from the logic level to the higher register transfer levels.

3.4 Optimization algorithms

Combinatorial optimization problems can be encountered everywhere – and, among other things in methods for hardware testing. Some of the most popular and widely used iterative optimization techniques are *simulated annealing* [103][104], *genetic algorithms* [105][106] and *tabu search* [107][108][109].

Simulated annealing (SA) is a general adaptive heuristic and belings to the class on nondeterministic algorithms [110]. It has been applied to several combinatorial optimization problems from various fields of science and engineering [111]. The term *annealing* refers to heating a solid to a very high temperature (whereby the atoms gain enough energy to break the chemical bonds and become free), and then slowly cooling the molten material ina controlled manner until it crystallizes. A simple algorithm to simulate the evolution of a solid in a heat bath to its thermal equlibrium was proposed in [115]. Later, the correspondence between annealing and combinatorial optimization was established in [103] and [104]. It was observed that there is a correspondence between, on one hand, a solution to the optimization problem and a physical state of material, and between the cost of solution of the combinatorial optimization problem and free energy in the molten metal. As a result of this analogy, a solution method in the field of combinatorial optimization was introduced. The method is based on the simulation of the physical annealing process, and hence the name *simulated annealing*.

Genetic algorithm (GA) is a powerful domain-independent search technique that was inspired by the Darwinian theory [111]. It emulates the natural process of evolution to perform an efficient and systematic search of the solution space to progress toward the optimum. It is based on the theory of natural selection that assumes that individuals with certain characteristics are more able to survive, and hence pass their characteristics to their offsprings. By establishing a correspondence between, on one hand a solution to the optimization problem and the element of the population (represented by the chromosome) and between the cost of the solution and the fitness of an individual in the population, a solution method in the field of combinatorial optimization is introduced. The methods thus simulates the process of natural evolution based on Darwinian principles, and hence the name *genetic algorithm* [105][106]. When employing GAs to solve a combinatorial optimization problem, one has to find an efficient representation in form of the chromosome (encoded string). Associated with each chromosome is its fitness value. If the process of natural reproduction is simulated, combined with the biological principle of survival of the fittest, then, as each generation progresses, better and better individuals (solutions) with higher fitness values are expected to be produced.

Tabu search (TS) is based on selected concepts of artificial intelligence [111] and was introduced as a general iterative heuristic for solving combinatorial optimization problems [107][108][109]. Initial ideas of the technique were also proposed in Hansen's steepest ascent mildest descent heuristic [112]. Tabu search is a generalization of the local search[113]. At each step, the local neighbourhood of the current solution is explored and the best solution of that neighbourhood is selected as the new current solution. But unlike the local search that stops, when no better solution is found, tabu search continues the search from the best solution in the neighbourhood, even if it is worse than the current solution. The information about previously visited solutions is added to the *tabu list* – the moves to the solutions in that list are not allowed, thus the cycling around previously visited solutions is prevented. However, if a certain criteria are satisfied (so called *aspiration criteria*) the tabu status of the solution is overridden. An example of such aspiration criterion is the situation when the cost of the selected solution is better than the best seen so far.

3.5 Conclusions

1. The complexity and speed of digital systems make external test difficult. Since the internal speed of SoC is constantly increasing, the demands to the automated test equipment memory size are increasing and the technology used in ATE is always one step behind. Hence, in order to apply at-speed tests and to keep the test costs under control, Built-In Self-Test (BIST) solutions are needed.

2. The pure pseudorandom test approaches in the BIST solutions do not guarantee sufficiently high fault coverage and demand very long test application times. This has lead to different mixed-mode and hybrid BIST methods.

3. The main concern of the hybrid BIST and reseeding BIST approaches has been to improve the fault coverage by mixing pseudorandom vectors with deterministic ones, while the issue of BIST optimization has not been addressed directly. The problems of minimization of the cost of BIST processes according to given criteria (time, hardware cost, power consumption) at given constraints need still solutions.

4. The traditional approach to characterize the quality of BIST is to use the measure of stuck-at-fault coverage (SAF). However, the traditional SAF model used in testing of digital circuits does not quarantee the adequate quality evaluation for deep-submicron technologies.

5. To characterize adequately the today's BIST solutions and the test sequences generated by BIST, more advanced defect-oriented fault modeling methods are to be used.

6. The similar feature of SSBDDs and HLDDs to model at the nodes the faults of digital systems at different levels of abstraction gives a good possibility to develop a uniform fault model for digital systems for BIST quality evaluation purposes.

7. There are several iterative optimization algorithms available. Some of the most popular and well-thought-out ones are *simulated annealing*, *genetic algorithms* and *tabu search*. Simulated annealing mimics the thermodynamic process of annealing, genetic algorithms simulate biological processes according to Darwinian theory of evolution, and tabu search attempts to imitate intelligent search processes through the use of a memory component.

Chapter 4 Optimization algorithms

Combinatorial optimization problems can be encountered everywhere – and, among other things in methods for hardware testing. In this chapter, three algorithms belonging to the special class of combinatorial algorithms – *general iterative nondeterministic algorithms* are described: simulated annealing, genetic algorithms and tabu search. In this work, optimization algoritms will be used to optimize the calculation of test cost in different hybrid Built-In Self-Test approaches.

4.1 Simulated annealing

In the following, *simulated annealing* (SA) will be described. SA is one of the most well developed and widely used iterative techniques for solving optimization problems.

Simulated annealing is a general adaptive heuristic and belongs to the class of nondeterministic algorithms [111]. It has been applied to several combinatorial optimization problems from various fields of science and engineering. These problems include *travelling salesman problem* (TSP), *graph partitioning*, *quadratic assignent, matching, linear arrangement and scheduling*. In the area of engineering, simulated annealing has been applied to VLSI design (placement, routing, logic minimization, testing), image processing, code design, facilities layout, network topology design and so forth.

One typical feature of simulated annealing is that, besides accepting solutions with improved cost, it also, to a limited extent, accepts solutions with detoriated cost. It is this feature that gives the heuristic the hill climbing capability. Initially the probability of accepting solutions with larger cost is large, but as the search progresses, only smaller detoriations are accepted, and finally only good solutions are accepted. A strong feature of the SA heuristic is that it is both effective and robust. Regardless of the choice of the initial configuration it produces high-quality solutions. It is also relatively easy to implement.

Simulated annealing, like all other iterative techniques is very greedy with respect to runtime. The acceleration of simulated annealing has been an extensive area of research since the introduction of the algorithm.

The use of simulated annealing in the combinatorial optimization was originally heavily inspired by an analogy between the physical annealing process of solids and the problem of solving large combinatorial optimization problems [113].

Annealing is known as a thermal process for obtaining low-energy states of a solid in a heat bath. The process consists of the two following steps [103]:

- Increase of the temperature in the bath to a maximum value at which the solid melts.
- Decrease carefully the temperature of the heat bath until the particles arrange themselves in ground state of solid.

In 1953, a simple algorithm based on the Monte Carlo techniques [114] was introduced by Metropolis *et al* [115] for simulating an evolution of the solid in the heat bath to the thermal equlibrium. The sequence of states is generated as follows. Given a current state S_i of the solid with energy E_i, a subsequent state S_j with energy E_j is generated by applying a perturbation mechanism. This perturbation transforms the current state into a next state with slight distortion. For instance a new state can be constructed by randomly selecting a particle and displacing it by some random amount. If the energy associated with the new state is lower than the energy of the current state, that is $\Delta E = E_j - E_i \leq 0$, then the displacement is accepted, and the current state becomes the new state. However, if the energy of the new state is higher (the energy difference is greater than zero), then the state S_j is accepted with a certain probability which is given by

$$prob(accept) = e^{-(\frac{\Delta E}{K_B T})}$$

where K_B is the Bolzman constant and T denotes temperature. The acceptance rule described above is repeated a large number of times. The acceptance criterion is known as the *Metropolis step* and the procedure is known as the *Metropolis algorithm.*

When applied in simulated annealing, the Metropolis algorithm can be used to generate a sequence of solutions of combinatorial optimization problem.

Fig. 4-1 desribes the simulated annealing algorithm. The algoritm generates neighbours randomly. If the cost of the neighbour *j* is at the most the cost of the current solution *i*, then *j* is always accepted. If neighbour *j* has higher cost than *i*, then *j* is still accepted with a positive probability of

$$\exp = (\frac{f(i) - f(j)}{c})$$

where c is a control parameter that plays the role of the temperature. The probability of accepting a deterioration in cost depends on the value of the control parameter c: the higher the value of the control parameter, the higher the probability of accepting deterioration.

```
procedure SIMULATED_ANNEALING
begin
  i:=initial solution
  c:=initial value
  repeat
    for l:=0 to L do
    begin
      probabilistically generate neighbour j of i
      if $f(j) \le f(i)$ then accept j
      else accept j with probability
              $\exp = (\frac{f(i) - f(j)}{c})$
    end
    update L
    decrease c
  until stopcriterion
end;
```

Fig. 4-1. *The pseudocode of simulated annealing algorithm*

The value of the control parameter is decreased during the execution of the algorithm. In *Fig. 4-1* the value *L* specifies the number of iterations the the control parameter is kept constant before it is decreased. The values of c and *L* and the stop criterion are specified by the „*cooling schedule*“.

Initially, at large values of *c*, large deteriorations will be accepted; as *c* decreases, only smaller deteriorations will be accepted and, finally, as the value of *c* approaches 0, no deteriorations will be accepted at all. Note that there is no limitation on the size of deterioration with respect to its acceptance. In simulated annealing, arbitrarily large deteriorations are accepted with positive probability; for these deteriorations, probability is small, however. This feature means that simulated annealing, in contrast to iterative improvement, can escape from local minima while it still exibits the favourable features of iterative improvement, namely simplicity and general applicability. The speed of convergence of simulated annealing is determined by the cooling schedule.

As simulated annealing has a so-called „*hill-climbing*“ ability, it can be used for solving problems that have a non convex solution space – SA provides the means to escape local optima in such cases. As will be shown later in the thesis, cost calculation function for hybrid built-in self-test has number of local optimas and therefore, SA is a suitable approach for solving the problem.

4.2 Genetic algorithms

In the following, *genetic algorithm* is described. This search technique was inspired by evolution. To solve an optimization problem, a potential solution to a specific problem is encoded in a simple chromosome-like data structure and recombination operators are applied, thus emulating the natural process of evolution. Efficient and systematic search of the solution space is performed obtaining the new solutions from the combinations of the existing ones. The algorithm is based on the theory of natural selection that assumes that individuals with certain characteristics are more able to survive and hence pass their characteristics to their offspring.

The genetic algorithm is an adaptive learning heuristic. Similar to simulated annealing, it also belongs to the class of general nondeterministic algorithms. Several variations of the basic algorithm exist.

Genetic algoritms (GAs) operate on a population (or set) of individuals (or solutions) encoded in strings. These strings represent points in a search space. In each iteration, referred to as a generation, a new set of strings that represent solutions (called offsprings) is created crossing some of the strings of the current generation [106]. Occasionally new characteristics are injected to add diversity. GAs combine information exchange along with the survival of the fittest among individuals to conduct the search.

Since their appearance, GAs have been applied to solve several combinatorial optimization problems from various fields of science , engineering and business.

Genetic algorithms were invented by John Holland and his collegues [105] in the early 1970s. Holland incorporated features of natural evolution to propose robust, computationally simple and yet powerful technique for solving difficult optimization problems.

When employing GAs to solve a combinatorial optimization problem one has to find an efficient representation of the solution in the form of the chromosome (encoded string). Associated with each chromosome is its fitness value. If we simulate the process of natural reproduction, combined with the biological principle of survival of the fittest, then, as each generation progresses, better and better individuals (solutions) with higher fitness values are expected to be produced.

The structure that encodes how the organism is to be constituted is called a *chromosome*. One or more chromosomes may be associated with each member of the population. The complete set of chromosomes is called a *genotype* and the resulting organism is called a *phenotype*. Similarly, the representation of a solution to the optimization problem in the form of an encoded string is termed as a *chromosome*. In most combinatorial optimization problems a single chromosome is generally sufficient to represent a solution, that is the genotype and the chromosome are the same.

The symbols that make up chromosome are known as *genes*. The different values of a gene are called *alleles*.

The *fitness value* of an individual (genotype or a chromosome) is a *positive* number that is a measure of its goodness. When a chromosome represents a solution to the combinatorial optimization problem the fitness value indicates the cost of the solution. In the case of a minimization problem, solutions with lower cost correspond to individuals that are more fit.

Since the GAs work with a population of solutions, *an initial population constructor* is required to generate a certain predefined number of solutions. The quality of the final solution produced by GA depends on the size of the population and how the initial population is constructed. The initial solution generally comprises random solutions.

GAs work on chromosomes or pairs of chromosomes to produce a new solutions called offsprings. Common genetic operators are c*rossover* and *mutation*. They are derived by analogy from the biological process of evolution. crossover operator is applied to pairs of chromosomes. The two individuals selected for crossover are called *parents*. Mutation is another genetic oprator that is applied to a single chromosome. The resulting individuals produced when genetic oprators are applied on the parents are called *offsprings*.

The choice of parents for crossover from the set of individuals that form the population is probabilistic. To accomplish the selection, the method called the *roulette wheel method* can be used. When using this method, the wheel is constructed so that each member of the population is given a sector size proptional to its fitness as an individual. To select the parent the wheel is spun and whichever individual comes up is selected as a parent. So, the individuals with lower fitness also have a finite but lower probability to become a parent [106].

Crossover is the main genetic operator. It provides a mechanism for the offspring to inherit the characteristics of both parents. It operates on two parents (P_1 and P_2) to generate offsprings.

Mutation produces incremental random changes in the offspring by randomly changing allele values for some genes. In case of a binary chromosomes it corresponds to changing single bit positions. It is not applied to all members of the population, but is applied probabilistically only to some. Mutation has the effect of perturbing a certain chromosome in order to introduce new characteristics not present in any element of the parent population. For example in case of binary chromosomes, toggling some selected bit produces the desired effect.

The structure of a simple genetic algoritm is given in *Fig. 4-2*. During each generation of the genetic algorithm a set of offsprings are produced by application of the *crossover* operator. The crossover operator ensures that the offsprings generated have a mixture of parental properties. In order to introduce new allels into the chromosome, with a certain probability, *mutation* is also applied. Following this, from the entire pool comprising both the parents and their offsprings, a fixed number of individuals are chosen to form the population of the new generation. If the **M** best individuals are chosen from this pool, then the fitness of the best individual, will be the same or better than the fitness of the best individual in the previous generation. Similarly, the average fitness of the population will be the same or higher than the average fitness of the best individual increase in each generation.

```
procedure GENETIC_ALGORITHM
      M = population size              (# Of possible solutions at any instance)
      $N_g$ = Number of generations    (# Of possible iterations)
      $N_o$ = Number of offsprings     (To be generated by crossover)
      $P_\mu$ = Mutation probability   (Also called mutation rate $M_r$)
      P ←Ξ(M)                          (Construct initial population P)
                                       (Ξ is population constructor)
begin
  for j=1 to M                 (Evaluate fitness of all individuals)
    evaluate $f(P[j])$         (Evaluate fitness of P)
  end for

  for i = 1 to $N_g$
    for j = 1 to $N_o$
      (x,y) ←φ(P)              (Select two parents x and y from current population)
      offspring[j]←χ (x,y)     (Generate offsprings by crossover of parents x and y)
      evaluate f(offspring[j]) (Evaluate fitness of each offspring)
    end for

    for j=1 to $N_o$           (With probability $P_\mu$ apply mutation)
      mutated[j] ←$\mu$ (y)
      evaluate f(mutated[j])
    end for

  P ← Select(P, offsprings)    (Select best M solutions from parents and offsprings)
  end for
  return highest scoring configuration in P
end
```

Fig. 4-2. *Structure of a simple genetic algorithm*

4.3 Tabu search

In this section, an optimization method called *tabu search* will be described, which is based on the selected concepts of artificial intelligence.

Tabu search was introduced by Fred Glover [107][108][109] as a general iterative heuristic for solving combinatorial optimization problems. Hansen also proposed some initial idea for this technique in his steepest ascent mildest descent heuristic [112]

The concept of the tabu search is simple and elegant. In its essence, it's a form of local neighbourhood search - each solution has a associated set of neighbours. A neighboring solution can be reached by an operation called move. Normally, the neighbourhood relation is assumed to be symmetric.

Tabu search is a generalization of local search. At each step, the local neighbourhood of the current solution is explored and the best solution in that neighbourhood is selected as the new current solution. However, local search stops when no improved solution is found in the current neighbourhood, whereas tabu search continues the search from the best solution in the neighbourhood even if its is worse than the current solution. To prevent cycling, so-called tabu list is formed, containing information about the the most recently visited solutions. Moves to tabu solutions are not allowed. However, the tabu status of the solution can be overridden is some situations - for example if the cost of the selected solution is better than the best seen so far. This situation proves that the search is not cycling back but moving towards better solution. Such situations are reffered to as aspiration criteria.

An algorithmic decription of a simple implementation of the tabu search is given in *Fig. 4-3* [111]. The procedure starts from an initial feasible solution S (current solution) in the search space Ω. A neighbourhood N(S) is defined for each S. A sample of neighbour solutions $V^* \subset$ N(S) is generated. An extreme case is to generate entire neighbourhood that is to take V^* = N(S). Since this is generally impractical (computationally expensive), a small sample of neighbours is generated called *trial solutions*. From these trial solutions the best solutions, say $S^* \in V^*$, is chosen for consideration as the next solution. The move S^* is considered even if S^* is worse than S, that is *Cost(S*) > Cost(S).* A move from S to S^* is made provided certain conditions are satisfied.

Selecting the best move in V^* is based on the assumption that good moves are more likely to reach optimal or near-optimal solutions. As mentioned before, the best candidate solution $S^* \in V^*$ *may* or *may not* improve the current solution, but is still considered. it is this feature that enables *escaping* from local optima. However, even with this strategy, it is possible to reach a local optimum, ascend (in case of the minimization problem) since moves with *Cost (S*) > Cost (S)* are accepted, and then in a later iteration return back to the same local optimum. That is, there is a possibility of cycling by returning back to previously visited solutions. This may cause the search to go though the same subset of solutions forever.

```
algorithm TABU_SEARCH
Ω      : Set of feasible solutions
S      : Current solution
S*     : Best admissable solution
Cost   : Objective function
N(S)   : Neighbourhood of  S ∈Ω
V*     : Sample of neighbourhood solutions
T      : Tabu list
AL     : Aspiration level

Begin
Start with an initial feasible solution  S∈Ω
Initialize tabu lists and aspiration level
For fixed number of iteartions Do
      Generate neighbour solutions V* ⊂N(S);
      Find best S*∈ V*
      If move S to S* is not in T Then
              Accept move and update best solution;
              Update tabu list and aspiration level;
              Increment iteration number;
      Else
              If  Cost(S*) < AL Then
                      Accept move and update best solution;
                      Update tabu list and aspiration level;
                      Increment iteration number;
              EndIf
      EndIf
EndFor
End.
```

Fig. 4-3. *Algorithmic description of tabu search*

To prevent returning to the previously visited solutions, a *tabu list* is maintained. The list contains *attributes* of some most recent moves. The size of the *tabu list* is the number of iterations for which a move containing that attribute is forbidden after it has been made. One can visualize the tabu list as a window on accepted moves as can be seen on *Fig. 4-4.*

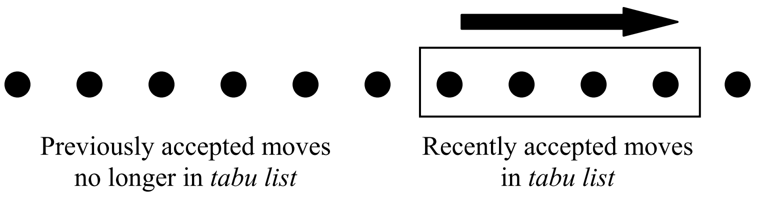


Fig. 4-4 *The tabu list*

Tabu list prevents cycling back to the previously visited solutions. However, since only the *attributes* of moves (not complete solutions) are stored in tabu list, these tabu moves may also prevent the considerations of some solutions which were not visited earlier. In order to relax the actions of the tabu lists, *aspiration criteria* are introduced. Then, the solutions that are the result of moves having attributes found in the tabu list are also considered *if* they satisfy the aspiration criteria. The aspiration criterion must make sure that the reversal of a recently made move leads the search to an unvisited solution, generally a better one.

There are several aspiration criteria that have been suggested in literature, the customary one, also the simplest and most commonly used, overrides the tabu status is the reversal of the move in the tabu list produces a solution better than the best obatained so far during the search. This is also known as the *best solution* aspiration criterion.

Coming back to the algorithmic description given on *Fig. 4-3*, initially the current solution is the best solution. Copies of the current solution are perturbed with moves to get a new set of solutions. The best among these is selected and if it is not tabu then it becomes the current solution. If the move is tabu, its aspiration criterion is checked. If it passes the aspiration criterion then it becomes the current solution. If the move to the next solution is accepted, then the move or some of its attributes are stored in the tabu list. Otherwise moves are regenerated to get another set of new solutions. If the current solution is better than the best seen so far, then the best solution is updated. Whenever the move is accepted, the iteration number is incremented. The procedure continues for fixed number of iterations or if some stop criterion is satisfied.

4.4 Conclusions

1. In many optimization problems it is not possible to use optimal enumerative and deterministic techniques in order to find the solution. Instead, approximation algorithms, also known as heuristic methods, can be used. When properly exploited, it is usually possible to develop reasonable heuristic which will quickly find an *acceptable* solution. Examples of such algorithms are *simulated annealing, genetic algorithms* and *tabu search*.

2. Simulated annealing is a general-purpose optimization technique for combinatorial optimization problems. Theoretical studies have shown that the algorithm can find global optimum provided a set of conditions on the annealing schedule are satisfied. For many problems, simulated annealing has produced excellent results but requires massive computing resources.

3. Genetic algorithms emulate the natural process of evolution. Unlike other search heuristics, they conduct the search by operating on a set of solutions called a population. They work with chromosomal representations (encode strings) of solutions. The basic idea is to combine solutions called parents to produce new solutions called offsprings, with the objective that the offsprings will inherit some parental characteristics.

4. Tabu search is different from other techniques in several aspects. One is the use of memory. In addition, reasonably sized subset of neighbours is explored and the best move among these is chosen. Also, in tabu search, "best" refers to change in evaluation which depends not only on the objective/cost function but also on search history, region being searched, and so forth.

5. When solving a minimization or optimization problem where the cost curve is nonconvex (that is, it has multiple optima), it is necessary that the algorithms that is applied, has a "*hill-climbing*" ability in order to escape local optima. The algorithms presented in this section have such ability and thus can be used for solving this type of problems. In this research, ***simulated annealing*** and ***tabu search*** based approaches are used to optimize the cost of hybrid built-in self-test implementations, described in this work.

Chapter 5 Fault modeling for BIST analysis

In this chapter, an overview is given about hierarchical defect modeling. Different defect modeling techniques are described, including Structurally Synthesized Binary Decision Diagrams, High-Level Binary Decision Diagrams and Boolean algebra. Also, experimental data on defect-oriented testing has been presented. The goal of this research was to investigate different fault models to be used in quality analysis of the BIST structures.

5.1 Introduction

One of the key problems in testing today's complex digital systems is: how to improve the testing quality at increasing complexities of systems? Two main trends can be observed when searching solutions for the problem: defect-orientation, and high-level modeling. Unfortunately, these trends are contradictory. Low-level defect modeling methods cannot be used for complex digital systems because of their complexity. On the other hand, high-level methods used for managing the complexity, loose in the accuracy of handling defects. To get out from the deadlock, these two opposite trends – high-level modeling and defect-orientation – should be combined into hierarchical approaches.

It has been shown that high SAF coverage cannot guarantee high quality of testing, for example, for CMOS integrated circuits [116]. The reason is that the SAF model ignores the actual behaviour of CMOS circuits, and does not adequately represent the majority of real IC defects and failure mechanisms. These facts are well known but usually, they have been ignored in engineering practice. In earlier works on layout-based test techniques [117], a whole circuit having hundreds of gates was analyzed as a single block. Such an approach is computationally expensive and highly impractical as a method of generating tests for real VLSI designs.

To handle physical defects in fault simulation, we still need logic fault models for the following reasons: to reduce the complexity of simulation (many physical defects may be modelled by the same logic fault), a single logic fault model may be applicable to many technologies, logic fault tests may be used for physical defect whose effect is not well understood. But the most important reason for logical modelling of physical defects is to get a possibility for moving from the lower physical level to the higher logic level which has less complexity.

In this chapter, an approach is presented to model physical defects by generic Boolean differential equations with the goal to map them from the physical level to the logic level. A new fault model is defined on that basis called *functional fault* *model*. It is shown how the functional fault model can be retreated as a uniform interface for mapping faults from a given arbitrary level of abstraction to the next higher level in test generation processes. This mapping faults from level to level is demonstrated on the basis of the model of decision diagrams.

5.2 Fault modeling with Structurally Synthesized Binary Decision Diagrams

In [116][117], an extension of the traditional model of Binary Decision Diagrams (BDD) was intrrodused called Structurally Synthesized BDDs (SSBDD). The name of this extension came from the fact that the model was synthesized not from the Boolean function as in the case of BDDs, but directly from the structure of the logic gate-level circuit by superposition of elementary BDDs of the gates. This method of synthesis created the possibility to map directly the faults of the circuit to the model of SSBDD.

Let us have a tree-like gate level combinational circuit with *n* inputs. For such a circuit we can create by a superposition of elementary BDDs of the gates a SSBDD with *n* nodes [116]. Between the paths in the tree and the nodes in the graph, there exists a one-to-one mapping. Every combinational circuit can be regarded as a network of modules, where each module represents a fan-out-free region (FFR) of maximum size. The SSBDD model for a given circuit can be regarded as a set of SSBDDs, where each of them represents such a FFR. This way of modeling the circuit by BDDs allows to keep the complexity of the model (the total number of nodes in all graphs) linear to the number of gates in the circuit.

Definition 5.1. SSBDD model for a given combinational circuit is a set of SSBDDs covering all FFRs of maximum size and a set of 1-node SSBDDs covering all primary inputs which have fan-out branches.

As a side effect of the synthesis of the SSBDD model, we have got a strict relationship between the nodes in the SSBDDs and the signal paths in the modules (FFRs) of the circuit.

SSBDDs reflect two types of mapping between the graph model and the related logic circuit:

(1) the nodes in SSBDDs represent signal paths, and

(2) certain groups of the nodes in SSBDDs represent certain subcircuits of the whole circuit.

Example 5.1. In *Fig.5-1*, we have a combinational circuit with a FFR-module and a SSBDD which represents the function and the structure of the module. To each of all 7 signal paths in the circuit, a node in the SSBDD corresponds. For example, to the path $L(7_1) = (7_1,a,d,e,y)$ from the input of the module 7_1 through internal nodes *a, d*, and *e* in the module up to the output *y*, the node 7_1 in the SSBDD corresponds.

On the other hand, for example, the group of nodes 6 and 7_3 in the SSBDD represent a subcircuit of two gates c and y in the circuit.

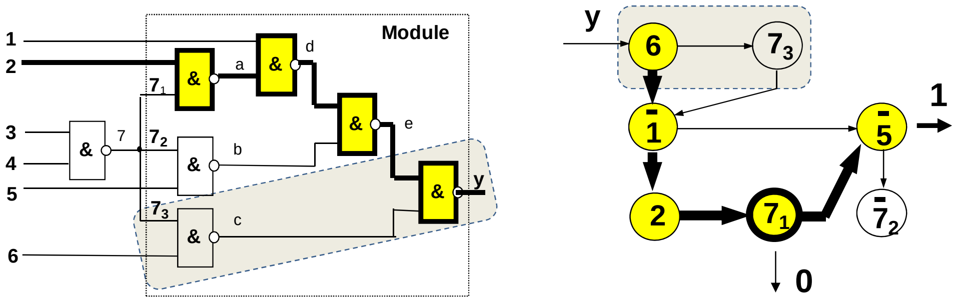


Fig. 5-1. *Combinational circuit and SSBDD*

Direct relation of nodes to signal paths and groups of nodes to subcircuits allowes to handle with SSBDDs easily such problems like fault modeling and fault diagnosis. Since the node 7_1 in the graph represents the path $L(7_1) = (7_1, a, d, e, y)$ in the circuit, it is easy to understand that the stuck-at-fault $7_1 \equiv \alpha$, where $\alpha \in \{0,1\}$, in the graph represent a subset of faults in the circuit $7_1 \equiv \alpha$, $a \equiv \neg\, \alpha$, $d \equiv \alpha$, $e \equiv \neg\, \alpha$, $y \equiv \alpha$. In such a way, the set of all faults in the original circuit can be pruned by using the SSBDD model.

In the following we will consider how the faults can be represented at higher levels of abstractions by High-Level Decision Diagrams (HLDD) to cope with the complexity problem.

5.3 Fault modeling with High-Level Binary Decision Diagrams

High-level approaches to diagnostic analysis of digital systems lay on different languages and models. Most frequent examples are state transition diagrams for finite state machines (FSM), abstract execution graphs, register transfer level (RTL) flowcharts, system graphs, instruction set architecture (ISA) descriptions, hardware description languages (HDL, VHDL, Verilog, System C), Petri nets for system level description. All these models need dedicated for the given language manipulation algorithms, which makes it difficult to create a uniform high-level approach to diagnostic analysis of digital systems. Existing high-level modeling methods which are efficient for simulation, lack the capability of analytical reasoning that is needed for formalizing test generation and fault diagnosis problems.

Promising opportunities for multi-level and hierarchical diagnostic modeling of digital systems provide decision diagrams (DD) because of their uniform cover of different levels of abstraction, and because of their capability for uniform graph-based fault analysis and diagnostic reasoning. High-Level Decision Diagrams (HLDD) for representing digital systems at higher levels of abstraction were introduced in [116].

The goal of using HLDDs was to generalize the logic level methods and algorithms of fault simulation, test generation and fault diagnosis from logic level to higher RTL and functional levels. For this purpose, the class of variables was extended from Boolean ones to the Boolean Vector, and integer variables, and the class of Boolean functions was extended to the data manipulation operations typically used in high-level descriptions of digital systems.

In *Fig.5-2* an example of a RTL data-path and its HLDD is presented. The variables R_1 and R_2 represent registers, *IN* denotes the input bus, the integer variables y_1, y_2 , y_3, y_4 represent control signals, M_1, M_2, M_3 are multiplexers, and the functions R_1+R_2 and R_1*R_2 represent the adder and multiplier, correspondingly. Each node in the DD represents a subcircuit of the system (e.g. the nodes y_1, y_2, y_3, y_4 represent multiplexers and decoders). The whole DD describes the behaviour of the input logic of the register R_2. To test a node in the DD means to test the corresponding to the node component or subcircuit.

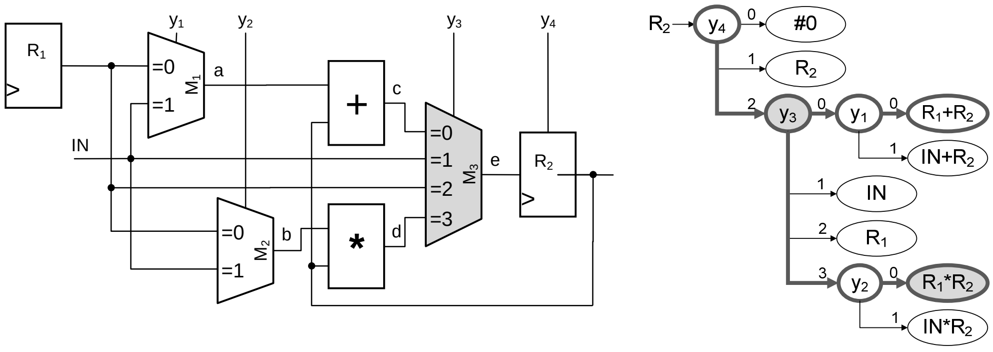


Fig. 5-2. *Representing a register transfer level data path by a HLDD*

Depending on the class of the system (or its representation level), we may have various HLDDs where the nodes have different interpretations and relationships to the system structure. In the RTL descriptions, we usually partition the system into control and data paths. In this case, the nonterminal nodes in the HLDDs correspond to the control path, and they are labeled by state or output variables of the control part, serving as addresses or instruction words. On the other hand, the terminal nodes in the HLDDs correspond to the data path, and they are labeled by the data words or functions of data words, which correspond to buses, registers, or data manipulation blocks. When using HLDDs for describing complex digital systems, we have to represent the system by a suitable set of interconnected components (combinational or sequential subcircuits). Thereafter, we have to describe the components by their functions which can be represented by HLDDs. The methods for synthesis of HLDDs for representing digital systems were described in [118].

HLDDs allow to represent formally and in a uniform way different high-level faults that traditionally are represented informally and in different languages. Let us define the formal fault model on HLDDs in the following way by showing the interrelations with the informal high-level fault model introduced for RTL circuits or microprocessors in [119][120][121].

Definition 5.2. Fault model for internal (nonterminal) nodes of HLDDs. In the case of register-transfer level (RTL) addressing schemes or for microprocessor adressing mechanisms, for a given source address any of the following may happen (multiplexer behavior faults) :

- no source is selected;
- a wrong source is selected;
- more than one source is selected and the multiplexer output is either a wired-AND or a wired-OR function of the sources, depending on the technology.

All these faults can be related to the faults of internal nodes of the HLDDs which leads to the exhausted testing of the nonterminal nodes of HLDDs for correct behavior of all edges of the nodes.

Definition 5.3. Fault model for terminal nodes of HLDDs. In the case of data-transfer along the buses between the registers and functional units of the RTL circuits or in microprocessors, the following may happen:

- one or more lines can be stuck at 0 or 1;
- one or more lines may form a wired-OR or wired-AND function due to shorts or spurious coupling;
- data manipulation faults.

All these faults can be related to the faults of terminal nodes of the HLDDs, which leads either to the exhausted testing of the functions at the nodes, or to generating the test for the node function or data transfer at the lover hierarchical level based on the structural model

The disadvantage of the referenced RTL or microprocessor fault models are in that they are defined in a very dedicated way and cannot be extended to cover the general digital systems test problem. The fault model of HLDDs allows a formal approach and is general for all digital systems described by the HLDD model.

Example 5.2. Consider a generic RTL statement as a pseudoinstruction in the following form [119]:

$$K\colon (T,C)\, R_d \leftarrow f(R_{S1}, R_{S2}, \ldots, R_{Sn}),\ \rightarrow N. \qquad (5\text{-}1)$$

Here K is the RTL statement label, T is the timing, and C is the logic condition to execute this statement, R_d is the destination register, R_{Si} is the i-th source register, f is an operation on source registers, $\leftarrow$ represents data transfer, and $\rightarrow$ N represents a jump to statement N.

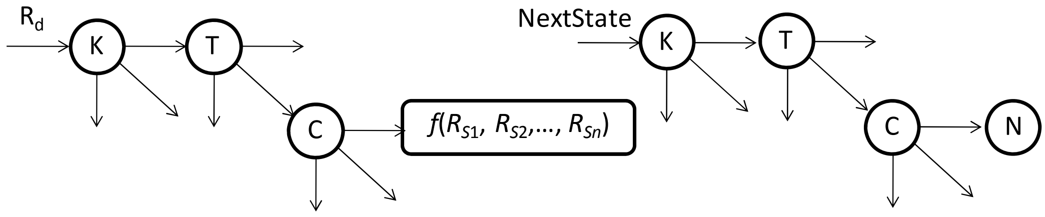


Fig. 5-3. *Generic partial HLDD for the generic statement (5-1)*

The statement *(5-1)* can be represented by the partial HLDD model, depicted in *Fig. 5-3*. The model consists of two graphs for calculating the content of the destination register R_d , and the *NextState*, respectively. Here, the labels K and N are interpreted as the *State* and *NextState* variables, respectively. Between the variables of the statement *(5-1)* and the nodes of the HLDD, there exists one-to-one mapping. There is also one-to-one mapping between the nodes $K(N)$, T, and C of the HLDD model and the corresponding registers of the RTL circuit. The terminal node $f(R)$ in the graph R_d represents a functional unit for data manipulations.

Based on the above notation, nine categories of functional faults can be identified as follows:

F1: label faults denoted by (K/K'), which means that the label K will be changed to K' due to the low-level faults;
F2: timing faults (T/T');
F3: logic condition faults (C/C');
F4: register decoding faults (R_i/R_i');
F5: function decoding faults (f/f');
F6: control faults ($\rightarrow N/\rightarrow N$');
F7: data storage faults ((R_i)/(R_i)'), which means that the content of the register R is changed from (R) to (R)' due to the low-level faults;
F8: data transfer faults ($\leftarrow/\leftarrow$'), which means that the fault occurs in the transfer path between the sources and the destination;
F9: data manipulation (function execution) faults ((f)/(f)', which means the operation execution fault – the operation f is executed, but the result of the operation is wrong.

In the HLDD fault model, the RTL faults F1 – F5 are represented by the faults of the nonterminal nodes of the HLDDs, whereas the RTL faults F6 – F9 are represented by the faults of the shown terminal nodes (the HLDDs for the decoding functions of f and R are not shown in *Fig. 5-3*).

5.4 Defect modeling with Boolean Differential Algebra

New failure mechanisms in today's deep-submicron electronic devices cannot be modeled by traditional stuck-at faults (SAF) which in case of DDs are directly associated with the nodes of the graphs. As the result, new advanced fault models are continuously being developed to improve the confidence of test quality measures and to increase the accuracy of fault diagnosis. The types of faults that can be observed in a real gate depend not only on the logic function of the gate, but also on its physical design. Good possibilities to combine logical and physical level fault modeling provide pattern fault model [122] or conditional fault models [78][79]. A similar pattern related fault modeling approach called functional fault model was presented in [123] for the module level fault diagnosis in combinational circuits.

Consider a parametric model of a component (e.g. a complex gate) in a combinational circuit with a correct function $y = f_y(x_1, x_2, \ldots x_n)$, and including a Boolean fault variable Δ to represent an arbitrary physical defect (Δ = 0 when the defect is missing, and Δ = 1 when the defect is present) as a generic function

$$y^* = f_y^*(x_1, x_2, \ldots, x_n, \Delta) = \overline{\Delta} f_y \vee \Delta f_y^{\Delta} \tag{5-2}$$

where f_y^{Δ} represents the faulty function of the component because of the defect Δ. The solution $W_y(\Delta)$ of the Boolean differential equation

$$\frac{\partial f_y^*}{\partial \Delta} = 1 \tag{5-3}$$

describes a condition which activates the defect *Δ* to produce an error on the output *y* of the component. The parametric modeling of a given defect Δ by the condition $W_y(\Delta) = 1$ allows to use it either for defect-oriented fault simulation (to check whether the condition $W_y(\Delta) = 1$ is fulfilled), or for defect-oriented test generation under the constraint $W_y(\Delta) = 1$ when a test pattern is searched for detecting the defect Δ .

If the components of the circuit represent standard library (complex) gates, the described analysis for finding conditions should be made once for all library components, and the sets of calculated conditions will be included into the library of components in the form of fault tables. The defect characterization may be computationally expensive, but it is performed only once for each library cell. The defect lists W^F_y of library components embedded in the circuit can be extended by additional physical defect lists W^S_y for the interconnect structure in the neighboring of the component to take into account also different defects (bridging faults, crosstalks etc.) outside the components. For these defects additional characterization should be carried out by a similar way as for the library cells.

Example 5.3. Consider a subcircuit (module or complex gate) of two simple gates with outputs *y* and *c* in the circuit in *Fig.5-1* (shown by the area in grey colour). The Boolean function of the module is

$$y = \bar{e} \vee x_6 x_{73} .$$

Consider a defect inside the module in the form of a bridging fault of the wired-AND type between the nodes *e* and x_{73} . The faulty function of the module with the defect can be presented as

$$y^{\Delta} = \overline{e x_{73}} \vee e x_{73} x_6 = \bar{e} \vee \overline{x_{73}} \vee x_6 .$$

Using the defect variable *Δ* for the short, we can create a generic differential equation for this defect and solve it as follows:

$$y^* = \bar{\Delta}(\bar{e} \vee x_6 x_{73}) \vee \Delta(\bar{e} \vee \overline{x_{73}} \vee x_6) = \bar{e} \vee x_6 x_{73} \vee \Delta(x_6 \vee \overline{x_{73}})$$

$$\frac{\partial y^*}{\partial \Delta} = \frac{\partial(\bar{e} \vee x_6 x_{73} \vee \Delta(x_6 \vee \overline{x_{73}}))}{\partial \Delta} = \overline{\bar{e} \vee x_6 x_{73}} \, \frac{\partial(\Delta(x_6 \vee \overline{x_{73}}))}{\partial \Delta} =$$

$$= \overline{\bar{e} \vee x_6 x_{73}} (x_6 \vee \overline{x_{73}}) = e\overline{x_{73}} = 1$$

As it results from the single solution of this differential equation, the bridging fault between the nodes *e* and x_{73} inside the module can be activated by the condition $W_y(\Delta) = e\overline{x_{73}} = 1$ which will be satisfied by the input signals of the module $e = 1$, and $x_{73} = 0$.

The example illustrated how the arbitrary physical defects can be mapped from the low physical (e.g. transistor circuit) level to higher logic level. The precondition for such a mapping is the possibility of representing the circuit with the defect by a faulty Boolean function.

5.5 Hierarchical mapping of faults in digital systems

The method of defining faults by logic conditions $W_y(\Delta)$ allows us to unify the diagnostic modelling of components of a circuit (or system) without going into structural details of components, or into the diagnostic simulation of interconnection network of components. In both cases, a condition $W_y(\Delta)$ describes how a lower level fault Δ should be activated to a given higher-level node in a circuit (or system). The conditions $W_y(\Delta)$ can be used both in fault simulation and in test generation.

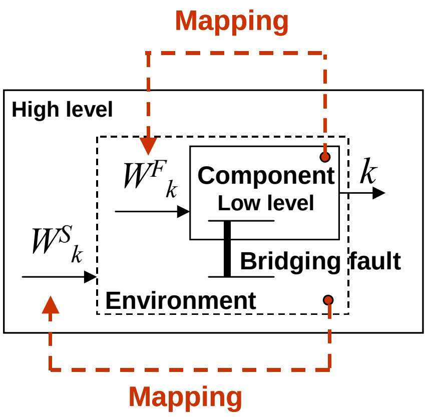


Fig. 5-4. *Mapping faults from lower level to higher level*

Consider a node k in a circuit (*Fig.5-4*) as the output of a module M_k, and which is represented by a function variable y_k. Let us associate with the node k a set of faults $R_\mathrm{k} = R^F{}_k \cup R^S{}_k$ where $R^F{}_k$ is the subset of faults in the module M_k, and $R^S{}_k$ is a subset of structural faults (defects) in the "neighbourhood" of M_k in the higher level environment. Denote by $W_k(\Delta)$ the condition when the fault $\Delta \in R_\mathrm{k}$ will change the value of y_k. Denote by $W^F{}_k$ the set of conditions $W_k(\Delta)$ which activate the defects $\Delta \in R^F{}_k$ and by $W^S{}_k$ the set of conditions $W_k(\Delta)$ which activate the defects $\Delta \in R^S{}_k$.

By using the sets of conditions $W^F{}_k$ and $W^S{}_k$ we can set up a mapping of faults from lower level to higher level for test generation purposes, and also in opposite direction, from a higher level to a lower level for fault simulation or fault diagnosis purposes.

In test generation, to map a lower level fault $\Delta \in R_\mathrm{k}$ to the higher level variable y_k, a solution of the equation $W^\Delta = 1$ is needed. In other words, if the condition $W^\Delta = 1$ is fulfilled then the presence of the defect $\Delta \in R_\mathrm{k}$ will change the value of the variable y_k.

In fault simulation (or in fault diagnosis) an erroneous value of y_k (denoted by a Boolean differential $dy_k = 1$) can be formally explained by implication

$$dy_k \rightarrow \Delta_1 W^{\Delta 1} \vee \Delta_2 W^{\Delta 2} \vee \ldots \vee \Delta_n W^{\Delta n} \qquad (5\text{-}4)$$

where for $j = 1,2,\ldots n$: $\Delta_j \in R_\mathrm{k}$. To the higher level event $dy_k = 1$, we set into correspondence a lower level event Δ_j if the condition $W^{\Delta j} = 1$ is fulfilled.

For hierarchical testing purposes we should construct for each module M_k of the circuit a list of faults R_k with logical conditions W^Δ for each fault $\Delta \in R_\mathrm{k}$. The set of conditions $W^F{}_\mathrm{k}$ for the functional faults $\Delta \in R^F{}_\mathrm{k}$ of the module can be found by low level test generation for the defects in the module. The set of conditions $W^S{}_\mathrm{k}$ for the structural faults $\Delta \in R^S_k$ in the environment of the module can be found by Boolean differential analysis of generic fault-free/faulty functions as explained above.

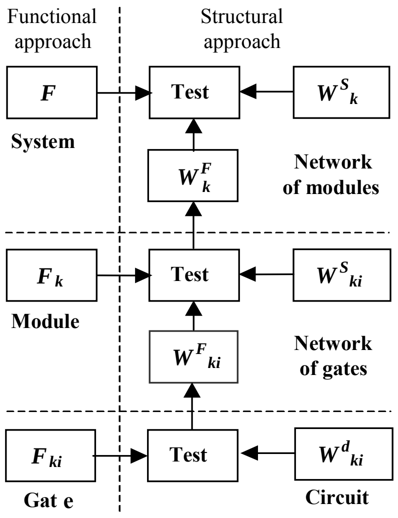


Fig. 5-5. *Hierarchical approach to diagnostic modeling of digital systems*

In *Fig. 5-5*, a hierarchical testing concept based on parametric fault modeling and functional fault model for a 3-level system is illustrated. In the functional approach, only the information about the functional behaviour is used. In the structural approach, tests are targeted to detect the faults in the networked components and in the network interconnections.

Let Y be the system variable representing an observable point of the system, y_M be an output variable of a logic level module, and y_G be the output of a complex gate with a defect Δ. Then, the condition of detecting the defect Δ on Y is

$$W = \partial Y/\partial y_M \wedge \partial y_M/\partial y_G \wedge W^{\Delta} = 1,$$

where $\partial Y/\partial y_M$ means the fault propagation condition calculated by high-level modeling, $\partial y_M/\partial y_G$ is the fault propagation condition (Boolean derivative) calculated by gate-level modeling, and W^{Δ} is the functional fault condition for a given gate calculated from the differential equation (5-3) by the gate preanalysis.

Example 5.4. Consider the following two examples of fault mappings between different levels of abstractions for digital systems. For the mappings, Decision Diagrams provide a suitable uniform environment.

(1) The nodes of SSBDDs represent signal lines in a digital circuit, where physical defects Δ may cause erroneous signals if the conditions $W^{\Delta}= 1$ are satisfied. Hence, to model a SAF fault of the node m at the conditions $W^{\Delta}= 1$ is equivalent to mapping the physical defect Δ to the logic level by means of SSBDD. The

conditions are calculated at the lower transistor level by using Boolean differential algebra or simulation tools, whereas the modeling of faults (e.g. during fault simulation or test generation) is carried out at the logic level with SSBDDs.

(2) The nodes of HLDDs represent modules in a digital system, where defects Δ inside the module may cause erroneous signals on the outputs of the module, if the input conditions (patterns or pattern sequences) $W^{\Delta}= 1$ on the inputs of the modules are satisfied. Hence, to model the behaviour of a node in a HLDD at the conditions $W^{\Delta}= 1$ is equivalent to mapping the module level defect Δ to the system level by means of HLDD. The conditions are calculated at the module level by test generation, for example, with SSBDDs, whereas the modeling of faults (e.g. during high-level fault simulation or test generation) is carried out at the system level with HLDDs.

The main concept of the fault model used in HLDDs is to test exhaustively each node. For non-terminal nodes which model the control variables such a concept is meaningful because of the low number of possible values for these variables. The situation is different with terminal nodes which model the units of data paths. In this case, hierarchical approach is advisable. Assume the terminal node $R_1 * R_2$ in a graph in *Fig. 5-2* labeled by multiplication expression. The high-level test pattern (control word) for activating the working mode $R_2 = R_1 * R_2$ is fault simulated using the HLDD. The set of local test patterns $\{R_1^t, R_2^t\}$ to be applied to the inputs R_1 and R_2 of the multiplier, are fault simulated at the lower gate-level. The set of patterns $\{R_1^t, R_2^t\}$ can be regarded as a set of conditions (as the functional fault model) when fault modeling the terminal node $R_1 * R_2$. On the other hand, the set of local test patterns $\{R_1^t, R_2^t\}$ can be regarded as interface between two levels in hierarchical fault modeling: the conditions are generated at lower level and used in the higher level.

A novel functional fault model was introduced, which is a general concept for mapping faults in digital systems between different levels of abstraction. The conditional SAF model can be regarded as a special case of fault mapping from physical to logic level.

5.6 Experimental data

The defect-oriented conception of hierarchical diagnostic modeling of digital circuits was compared with the traditional SAF model oriented approach for the ISCAS'85 benchmark family. Only bridging faults between lines in transistor circuits were considered. The experiments showed the advantage of the defect-oriented conception. The investigations have been carried out in cooperation with TU Warsaw and IISAS Bratislava targeted to defect-oriented test pattern generation [124].

Table 5-1. *Comparison of SAF and defect-oriented test generation*

Circuit	Defect coverage, %			
	For 100% SAF test		Defect-oriented test generation	
	OR model	AND model	OR model	AND model
c17	92.59	100.0	**100.0**	100.0
c432	99.38	99.33	**100.0**	**100.0**
c499	92.80	100.0	92.80	100.0
c880	95.95	100.0	95.95	100.0
c1355	93.42	100.0	93.42	100.0
c1908	92.91	99.94	92.91	**100.0**
c3540	94.21	99.68	**94.38**	**99.74**
c5315	94.71	100.0	94.71	100.0
c6288	92.59	100.0	92.59	100.0

The results are depicted in *Table 5-1*. In the left part of the table, comparison with traditional SAF test of 100% fault coverage is given. We see that, especially for the OR-shorts the defect coverage of the 100% SAF tests is rather low, and the quality of these tests cannot be trusted.

In the right part of the table, the results of defect-oriented test generation are given. In some cases, we see that defect-orientation can help to increase the fault coverage. The cases where 100% coverage was not achieved show that the not detected defects are with high probability redundant. This conclusion about the redundancy of defects could not be made based on the SAF-oriented test results in the left part of the table, since for these tests the detection of bridging fault defects were not the target.

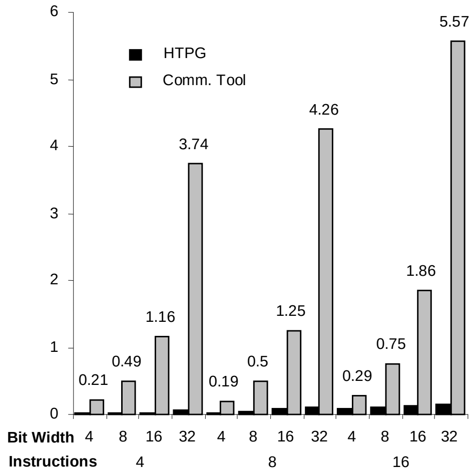


Fig. 5-6. *Comparison of plain gate-level and hierarchical test generation*

In *Fig. 5-6* an example is depicted to illustrate the impact of fault mapping from gate level to system level on the efficiency of test generation. Test generation experiments were carried out for a benchmark family of RISC processors which vary in the the instruction set (processors with 4, 8 and 16 instructions) and in the bitwidth (4, 8, 16 and 32-bit processors) [124]. Test generation for data manipulation modules represented by the terminal nodes of HLDDs was carried out on the logic level with SSBDDs. The locally generated test vectors were used as conditions for testing the terminal nodes in the HLDDs. During test generation for HLDDs, no logic level details were considered, all the operatios were carried out exclusively on the hig-level. The experimental results (test generation times in s) are shown in *Fig.5-6* for 12 different processors, differing in the complexity (bit width, size of the instruction set), where the high-level ATPG is compared to a commercial gate-level ATPG tool. Here we see, the higher is the complexity of the digital system, the larger is the advantage of the high-level ATPG compared to the low-level ATPG, in other words, the larger is the impact of the hierarchical fault mapping concept.

In addition, the experiments were carried out to determine the efficiency on pseudorandom test sequence generated by Built-In Self-Test giving high percent of SAF fault coverage for defect-oriented testing.

The experiments were carried out in two stages:

(1) the experiments to find best pseudorandom test sequences for the ISCAS'85 benchmarks – the LFSR configuration that would result in the shortest sequence giving high fault coverage (100% was reached in the most cases)

(2) defect-oriented fault simulation was carried out

In *Fig. 5-7*, an example of experimental data for one of the ISCAS'85 benchmarks is shown.

It can be seen that pseudorandom test sequence has different efficiencies in detecting stuck-at faults and different classes of defects or combinations of them.

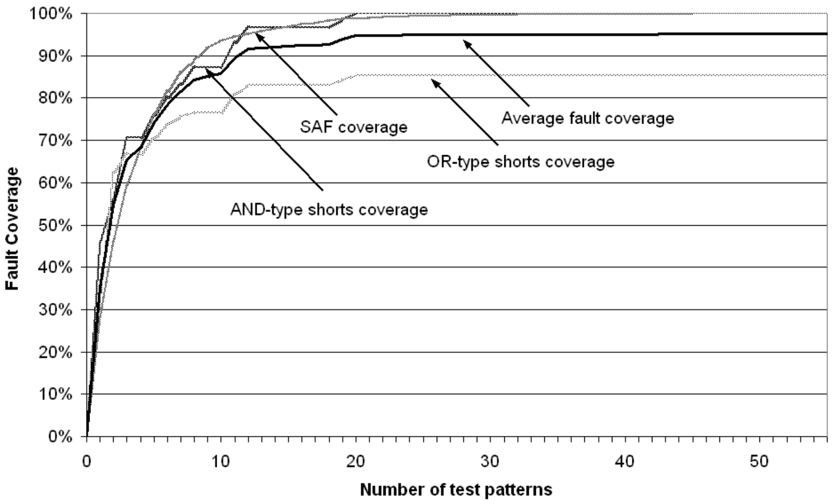


Fig. 5-7. *Fault coverage by pseudorandom test sequence in case of SAF and defect-oriented fault simulation for* ***c6288***

Table 5-2. *Test qualities at different simulated fault classes*

Circuit	Test length	SAF cover %	Defect coverage, %					
			Counted defects			Probabil. defects		
			AND	OR	Total	AND	OR	Total
c432	100	95,5	95,0	96,6	**95,8**	96,1	97,8	**97,0**
	200	98,9	98,3	98,8	**98,5**	98,5	99,0	**98,8**
	300	100	100	100	**100**	100	100	**100**
c499	750	98,9	99,0	92,1	**95,6**	99,5	85,5	**92,5**
	1000	99, 4	99,4	92,4	**95,9**	99,7	85,6	**92,7**
	1270	100	100	92,8	**96,4**	100	85,8	**92,9**
c880	500	98,2	99,3	95,7	**97,5**	99,4	92,2	**95,8**
	820	99,5	99,6	96,0	**97,8**	99,7	92,4	**96,1**
	3355	100	100	92,5	**96,3**	100	96,2	**98,1**
c1355	200	90,9	93,9	88,5	**91,2**	95,5	85,0	**90,3**
	750	97,9	98,4	92,1	**95,3**	98,9	87,1	**93,0**
	1315	100	100	93,4	**96,7**	100	87,8	**93,9**
c1908	750	96,5	98,7	92,0	**95,3**	99,2	85,5	**92,4**
	1200	98,9	99,2	92,3	**95,8**	99,6	85,7	**92,6**
	2075	100	100	93,4	**96,7**	100	92,9	**96,5**
c3540	1500	91,3	90,3	82,6	**86,5**	90,3	85,7	**88,0**
	2500	92,5	91,1	83,7	**87,4**	90,9	86,6	**88,7**
	10000	98,1	97,1	91,9	**94,5**	97,0	88,5	**92,7**
c5315	750	99,6	100	94,7	**97,4**	100	91,1	**95,6**
	1110	99,9	100	94,7	**97,4**	100	91,1	**95,6**
	4100	100	100	94,7	**97,4**	100	91,1	**95,6**
c6288	20	98,9	100	92,6	**96,3**	100	85,3	**92,7**
	40	99,9	100	92,6	**96,3**	100	85,3	**92,7**
	55	100	100	92,6	**96,3**	100	85,3	**92,7**

Table 5-2 presents the experimental results for resynthesised ISCAS'85 benchmarks. As it can be seen from the results, the pseudorandom test sequence is quite efficient in covering AND-type shorts but in case of OR-type shorts the coverage tends to be quite low. Also, it is obvious that similarly to the results shown in *Table 5-1*, even though pseudorandom sequence giving 100% stuck-at fault is giving quite high defect coverage, in many cases it is still not enough and many defects remain untested. In addition, typically for such built-in self-test approach, the test sequences tend to be very long leading to long test application times.

The fact that the pseudorandom test sequence often fails to reach satisfactory defect coverage can be explained by the fact that in these cases we are dealing with either redundant or random pattern resistant (so-called *"hard-to-test"*) defects. This issue could be addressed by generating test patterns targeting specifically these defects.

Table 5-3 presents the correlation between fault coverage and test length for some benchmark circuits. As it can be seen, 100% coverage of stuck-at faults can be reached with much shorter pseudorandom test sequence than the highest achievable defect coverage. This result clearly shows that there is the need to consider extended classes of defects, and not only the simple SAF class when determining the length of the pseudorandom test of acceptable quality.

Table 5-3. *Comparison of test lengths for SAF coverage and total (SAF and defects) coverage*

Circuit	Fault coverage. %	Test length		Test length correlation
		SAF	Total (SAF & Defects)	
c432	100.00	297	297	1.00
c880	97.78	535	656	1.23
	100.00	3352	-	N/A
c1355	97.31	617	1315	2.40
	100.00	1315	-	N/A
c1908	96.26	716	1096	1.53
	100.00	2054	-	N/A
c5315	97.35	148	465	3.14
	100.00	4026	-	N/A
c6288	96.29	14	20	1.43
	100.00	51	-	N/A

5.7 Conclusions

1. An approach is presented to map faults between the levels of abstraction in digital systems to improve the efficiency of fault simulation and test generation.

2. For modelling physical defects, generic Boolean differential equations were introduced which allow to map the physical faults from lower physical level to higher logic level.

3. It was shown that the Decision Diagrams provide an efficient tool for uniform fault mapping in digital systems from lower levels to higher levels of abstraction.

4. A new *functional fault model* was developed as a uniform basis for modelling arbitrary physical defects. It was shown how the functional fault model can be regarded as a uniform interface for mapping faults from lower levels to higher levels.

5. The conditional SAF model can be regarded as a special case of the functional fault model to facilitate fault mapping from physical to logic level.

6. Experimental data demonstrate that the higher is the complexity of the digital system, the larger is the advantage of the hierarchical ATPG compared to the plain low-level ATPG, in other words, the larger is the impact of the hierarchical fault mapping concept.

7. Experimental data also showed that BIST test that has 100% stuck-at fault covarage, can reach quite high defect coverage but in many cases the test quality is not satisfactory, thus extended fault model should be used. This fact explains the need of evaluation of the BIST quality using defect-oriented functional fault model instead of traditional SAF model.

Chapter 6 Test cost minimization of Hybrid BIST

Classical Built-In Self-Test solutions are in large part based on using LFSRs for generating pseudorandom test sequences and also test response compaction. In this chapter, an approach known as hybrid BIST is described, which combines pseudorandom test patterns with stored precomputed deterministic test patterns. Methods are described for finding optimal balance between pseudorandom and stored patterns. In order to speed up the calculation process, a method based on *tabu search* has been applied to find the global cost minimum.

6.1 Introduction

To test the individual cores of the system the test pattern source and sink have to be available together with an appropriate test access mechanism (TAM [56] as depicted in *Fig. 6-1*.

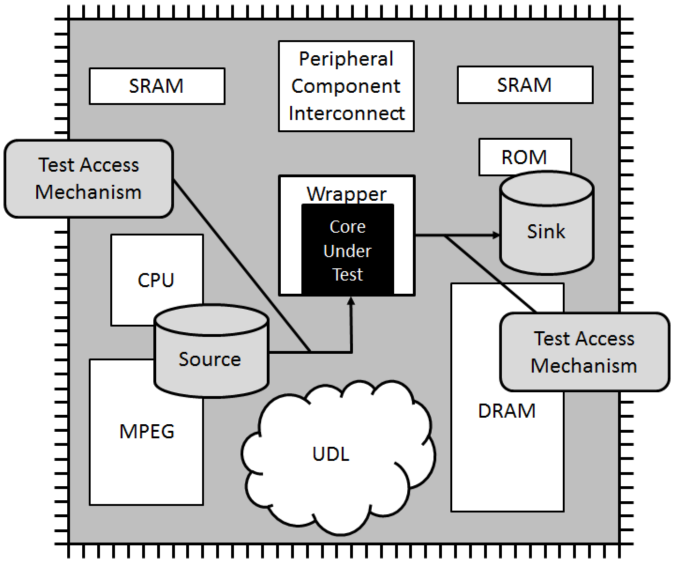


Fig. 6-1. *Testing a System-on-chip*

The traditional approach implements both source and sink off-chip and therefore requires the use of external Automatic Test Equipment (ATE). But, as the requirements for the ATE speed and memory size are continuously increasing, the ATE solution can be unacceptably expensive and inaccurate. Therefore, in order to apply at-speed tests and to keep the test costs under control, on-chip solutions are becoming more and more popular. Such a solution is usually referred to as Built-In Self-Test (BIST). The classical BIST has been described in *Chapter 2.*

The main idea behind BIST approach is to eliminate or reduce the need for an external tester by integrating active test infra-structure on a chip. The test patterns are not any more generated externally as it is done with Automatic Test Equipment (ATE), but internally, using special BIST circuitry.

6.2 Drawbacks of the classical BIST approach

The classical way to implement TPG for logic BIST (LBIST) is to use linear feedback shift registers (LFSR). But as the test patterns generated by LFSR are pseudorandom by their nature [61] and have linear dependencies, the LFSR based approach often does not guarantee a sufficiently high fault coverage (especially in the case of large and complex designs) and demands very long test application times in addition to high area overheads.

The pseudorandom test sequence can be more than ten times longer than the deterministic sequence with the similar efficiency [13]. The reason of this problem is the presence of so-called random pattern resistant faults [125] in the circuitry. The random-pattern resistant faults are those that are detected by only few patterns, sometimes just one. Obviously, if this pattern is not generated by the LFSR during the test, the fault will remain undetected.

The typical situation where the described situation occurs is when the circuit includes large input AND (NAND) logic functions or large input OR (NOR) logic functions [11]. A large AND (NAND) function will produce a logic 1 (logic 0) infrequently due to the equal likelihood of logic 1s and logic 0s in each bit of the pseudo-random test patterns. Similarly a large OR (NOR) function will produce a logic 0 infrequently. Faults that would require many logic 1s generated at the output of a large AND function or logic 0s at the output of a large OR function are not easily detected. *Fig. 6-2* illustrates the situation when the stuck-at-1 fault needs to be detected on the output of OR-gate. To detect this fault, we would need a test pattern consisting of all 0s to activate the fault. The more inputs the gate has the lower is the probability of discovering such fault.

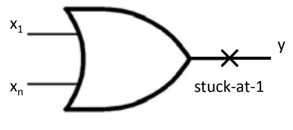


Fig. 6-2. *PR-resistant fault - OR gate*

In general, pseudorandom test patterns can seldom achieve 100% fault coverage. *Fig. 6-3* shows the fault coverage of pseudorandom tests as a function of the test length for ISCAS'85 benchmark [126]. This figure illustrates an inherent property of pseudorandom test: the first few test vectors can detect a large number of faults while later test vectors detect few new faults if any. Moreover, many faults will never be detected with pseudorandom tests alone.

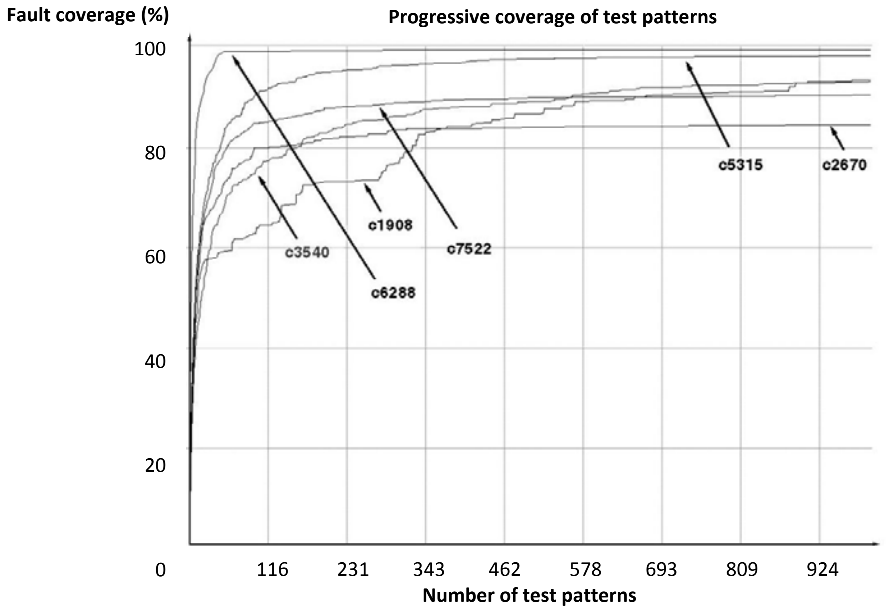


Fig. 6-3. *Pseudorandom tests form some ISACS'85 circuits*

Therefore, several questions have to be answered while developing a LFSR-based self-test solution:

- What is the fault coverage achievable with pseudorandom patterns, compared to that of deterministic test methods?
- Will the required fault coverage be achieved by the number of pseudorandom patterns that can be generated in some acceptable interval of time?
- What are the characteristics of LFSR that produce the test sequence with acceptable fault coverage?

Such an analysis shows that in most cases the pseudorandom test leads to either unacceptably long test sequences or fault coverage figures that are not acceptable and much below those achievable by deterministic test sequences.

One solution to this problem is to complement pseudorandom test patterns with deterministic test patterns, applied from an on-chip memory, in special situations, from an ATE, using different techniques. These approaches are usually referred to as mixed-mode or hybrid BIST approaches [127].

6.3 BIST impovement techniques

Different methods have been proposed for improving the classical BIST schemes. For example, the fault coverage can be increased by modifying the CUT by either inserting the test points [62] or by redesigning the CUT itself [128]. The drawback of those techniques is that generally they add additional logic to the circuitry that can degrade the performance.

Some other approaches are using *weighted pseudorandom sequences* for BIST fault coverage improvement. In these approaches additional logic is needed to weight the probability of each bit in the test sequence. An example of practical application for weighted pseudo-random test patterns is a CUT that incorporates a global reset or preset to the flip-flops. Frequent resetting of flip-flops by pseudorandom test pattern will clear the test data propagated into the flip-flops and prevent internal faults from being detected. In a study of pseudorandom test patterns applied to the 1989 International Symposium on Circuits and Systems (ISCAS'89) sequential benchmark circuits [129], it was found that single stuck-at level-gate fault coverage was as low as 11% to 15% for circuits that had global resets or presets to the flip-flops due to this fault-detection blocking effect of pseudorandom patterns [130]. When the reset/preset signal is controlled by other means, which can include the use of weighted pseudorandom test patterns, the fault coverage climbs greater than 90% for these circuits. A solution to this problem is to take the equal likelihood of 0s and 1s in pseudorandom sequences of an LFSR and use additional logic to create weighted pseudorandom patterns [131]. For example, more frequent logic 1s can be generated by a logical NAND of two or more bits of LFSR. Given that the probability of a given bit in LFSR being a logic 0 is approximately 0.5 (denoted $p_0 \approx 0.5$) NANDing two bits of the LFSR will produce a bit that has $p_0 \approx 0{,}25$; NANDing three bits will result in $p_0 \approx 0{,}125$ and so on. *Fig. 6-4* shows an example of weighted LFSR based TPG where each of the normal LFSR outputs has $p_0 \approx 0.5$ for a given test pattern while the weighted output has $p_0 \approx 0{,}125$.

The probabilities (or weights) can be controlled for a higher coverage test patterns generation and/or shorter length BIST sequences for a given CUT [132]. Of course the weights and the number of CUT inputs that need to be "weighted" has to be determined on a case-by-case basis since this is a function of the CUT.

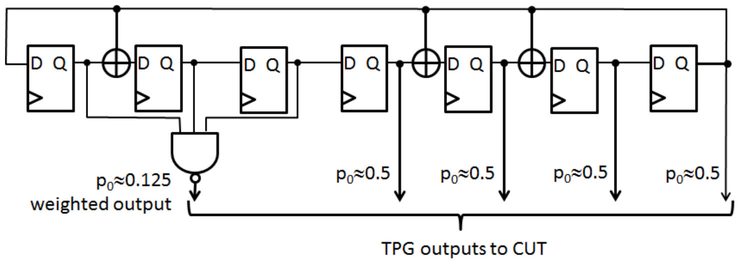


Fig. 6-4. *Example of weighted LFSR implementation*

The weight logic can be placed either at the input of the scan chain [133] or in the individual scan cells themselves [63]. The disadvantage of the WPS approach is that the weight sets have to be stored on chip and additional control logic is required to switch between weights. Therefore, the silicon area overhead may become too large, as the weighting requires extra hardware in terms of the NAND/NOR gates as well as additional LFSR bits since multiple bits are logically combined to produce a single test pattern bit. The increased area overhead along with the increased design time are a limitation of this TPG approach. However, the weighted-LFSR is a powerful technique for increasing fault coverage in circuits with pseudorandom pattern resistant circuits.

A third alternative for improving the fault-coverage is to use a mixed or hybrid approach [72][73][127][134] -[136]. Such approaches use pseudorandom patterns to cover easy-to-detect faults and subsequently, deterministic patterns to target the remaining hard-to-detect faults.

There are number of ways for generating patterns on-chip. Three of them are described in the following

ROM compression

The simplest approach for generating deterministic patterns on-chip is to store them in a *read-only memory* (ROM). The problem is that quite often the solutions require a big size of ROM to store all the necessary patterns. There have been several ROM compression techniques proposed that allow reducing the size of ROM [64][67][137]-[139]

LFSR reseeding

Instead of storing the test patterns in ROM, there are many techniques developed for storing LFSR seeds that can be used to generate pseudorandom test patterns [63]. The LFSR that is used for generating the pseudorandom patterns is also used for generating the deterministic patterns by reseeding it with the computed seeds. The seeds can be computed with linear algebra as described in [63]. Because the seeds are smaller than the test patterns themselves, they require less ROM storage. One problem is that for LFSR with a fixed characteristics (feedback) polynomial, it may not always be possible to find a seed that will efficiently generate the required deterministic test patterns. A solution to that problem was proposed in [67], where a multiple-polynomial LFSR (MP-LFSR) is used. *Fig. 6-5* shows such an MP-LFSR.

An MP-LFSR is an LFSR with a reconfigurable feedback network. A polynomial identifier is stored with each seed to select the characteristic polynomial that will be used for that seed. Techniques for further reductions in storage can be achieved by using variable-length seeds [140], a special ATPG algorithm [141], folding counters [142] and seed encoding [143].

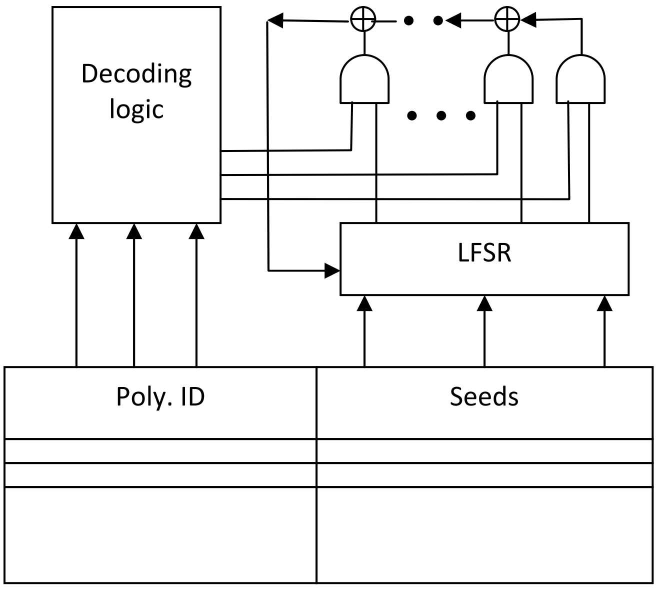


Fig. 6-5. *Reseeding with multiple-polynomial LFSR*

Embedding deterministic patterns

One more approach for mixed-mode BIST is to embed the deterministic patterns in the pseudorandom sequence. Many of the pseudorandom patterns generated during pseudorandom testing do not detect any new faults, so some of those "useless" patterns can be transformed into deterministic patterns that detect PR-resistant faults [67]. This can be done by adding *mapping logic* between the scan chains and the CUT or in a less intrusive way by adding the mapping logic at the inputs to the scan chains to either perform a bit fixing [66]or bit-flipping [144]. *Fig. 6-6* illustrates this approach as shown in [67].

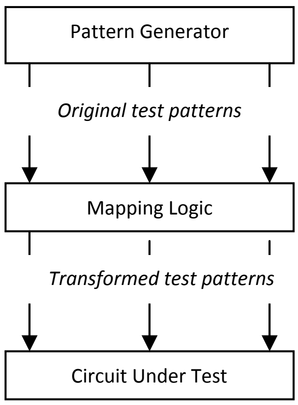


Fig. 6-6. *Using the mapping logic*

The main strength of the described approaches lays in the possibility to have a trade-off between test data storage and test application time by varying the ratio of pseudorandom and deterministic test patterns. In general, such a hybrid approach reduces the memory requirements compared to the pure deterministic testing, while providing higher fault coverage and reduced test times compared to the stand-alone BIST solution.

6.4 Hybrid BIST - basic principles

Usually, in self-test approaches some type of pseudorandom test patterns is used. But as test patterns generated by LFSR are pseudorandom by their nature, the generated test sequences are usually very long and not sufficient to detect all the faults. To avoid the test quality loss due to random pattern resistant faults and in order to speed up the testing process, deterministic patterns targeting the random resistant and difficult to test faults have to be applied.

In the previous sections, some methods based on this concept were described. These methods successfully increased the quality of the test by targeting random pattern resistant faults. However, these described approaches were able to find a solution in situations when test process was constrained by limitations such as memory or test time, which is often the case in realistic situations.

The hybrid BIST approach used in this work has been described in [127] and it is based on the intelligent combination of pseudorandom and deterministic test sequences that would provide a high-quality test solution.

This hybrid BIST approach is illustrated in *Fig. 6-7*. It starts with on-line generation of pseudorandom test sequence with a length of *L*. On the next stage, a stored test approach with length *S* takes place [127]. For the stored approach, precomputed test patterns are applied to the core under test to reach the desirable coverage level. The precomputed deterministic set of test patterns is stored in the memory. For off-line generation of the deterministic set, arbitrary software test generators based on deterministic, random or genetic algorithms may be used [145].

Definition: *A hybrid BIST set* $TH_k = \{PR_k, DT_k\}$ for a core C_k is a sequence of tests, constructed from a subset of the complete pseudorandom test sequence $PR_k \subseteq PR^F{}_k$, and a subset of the complete deterministic test sequence $DT_k \subseteq DT^F{}_k$. The test sequences PR_k and DT_k complement each other to achieve the maximum achievable fault coverage, and define the hybrid test set TH_k.

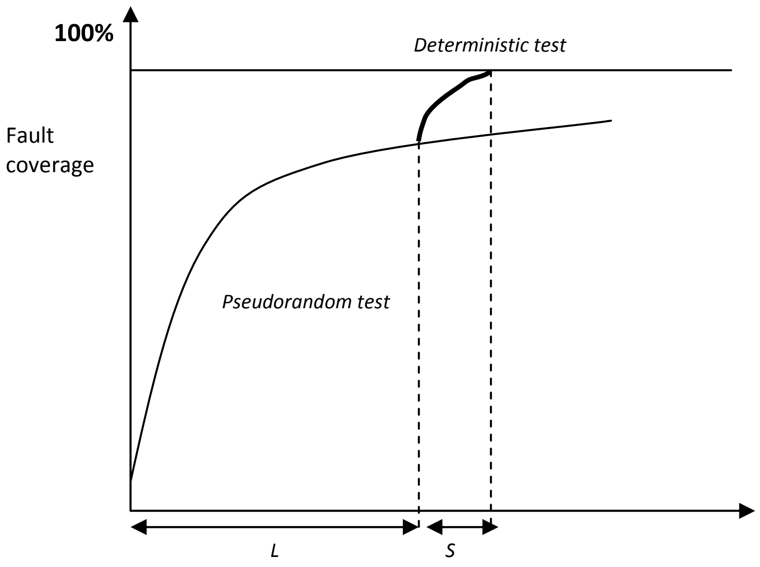


Fig. 6-7. *Hybrid BIST fault coverage*

In general, a shorter pseudorandom test implies a larger deterministic set. This requires additional memory space, but at the same time, shortens the overall test process, since deterministic test vectors are more effective in covering faults than the pseudorandom ones. A longer pseudorandom test, on the other hand will lead to longer test application time with reduced memory requirements [127].

6.5 Hybrid BIST - cost calculation and optimization

The important parameter of hybrid BIST is the length L of the pseudorandom test sequence. It determines the behaviour of the whole process. In case a shorter pseudorandom test set is used, the larger deterministic set will be needed. Larger deterministic test set requires additional memory for these patterns to be stored. This approach, however, shortens the overall test. On the other hand, a longer pseudorandom test will lead to longer test application time with reduced memory requirements. Therefore it is crucial to determine the optimal test length of pseudorandom test in order to minimize the total testing cost.

The total cost of BIST solution consisting of pseudorandom test patterns and stored test patterns is graphically shown on *Fig. 6-8*. The horizontal axis denotes the fault coverage achieved by the pseudorandom test before switching from the pseudorandom to the stored test. Zero fault coverage is the case when only stored test patterns are used and therefore the cost of stored test is the biggest at this point. The figure illustrates the situation where 100% fault coverage is achievable with pseudorandom test alone, although this can demand a very long pseudorandom test sequence (in particular, in the case of large and complex designs, 100% fault coverage might not be achievable at all).

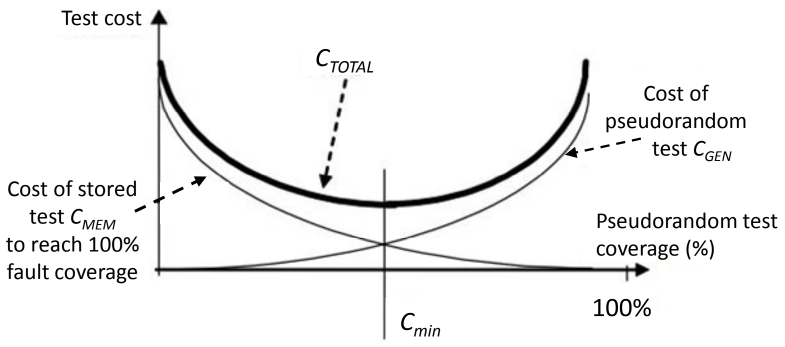


Fig. 6-8. *Cost calculation for hybrid BIST*

Therefore, the total cost of the hybrid BIST C_{TOTAL} can be defined as follows:

$$C_{TOTAL} = C_{GEN} + C_{MEM} \approx \alpha L + \beta S$$

where C_{GEN} is the cost related to the time for generating L pseudorandom test patterns (number of clock cycles), C_{MEM} is the memory cost for storing S precomputed test patterns needed to improve the results of the pseudorandom test sequence [126]. α and β are constants to map the test length and memory space to the costs of the two parts of the test solutions to be mixed.

Fig. 6-8 illustrates how the cost of pseudorandom test C_{GEN} is increasing when striving to higher fault coverage. In general, it can be very expensive (time-consuming) to achieve high fault coverage with pseudorandom patterns only. The C_{MEM} curve on the other hand, describes the cost we have to pay for storing additional pre-computed tests at the given fault coverage reached by pseudorandom testing to achieve the required fault coverage level. The total cost of C_{TOTAL} is the sum of the mentioned costs αL and βS. The weights α and β reflect the correlation between the cost and pseudorandom test time (number of clock cycles used) or between the cost and the memory size needed for storing the precomputed test sequence. For simplicity it is assumed that $\alpha = 1$ and $\beta = B$ where B is the number of bytes of the input vector to be applied to the core under test CUT. Hence, in the following the number of clocks used for pseudorandom test generation and the number of bytes in the memory needed for storing precomputed test patterns are used are the cost units. The total cost C_{TOTAL} is illustrated in *Fig. 6-8* where the minimum point is marked as C_{min}.

In many situations, 100% fault coverage is not achievable with only pseudorandom vectors. Therefore, an assumption has to be included into total cost calculation. This situation is illustrated in *Fig. 6-9* where the horizontal axis indicates the number on pseudorandom patterns applied, instead of the fault coverage level. The curve of the total cost C_{TOTAL} is still the sum of two cost curves $C_{GEN}+C_{MEM}$ with the new assumption that the maximum fault coverage is achievable only by either the hybrid BIST or pure deterministic test. *Fig. 6-9* illustrates the calculation of the *Cost* curve under these more realistic assumptions.

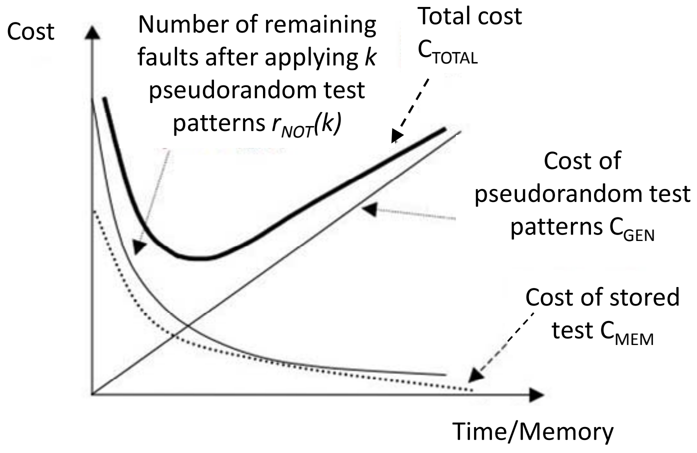


Fig. 6-9. *Cost calculation of hybrid BIST under realistic assumptions*

The main purpose of the approaches described in this chapter is to develop a fast method for finding the length *L* of the pseudorandom test sequence when the total cost C_{TOTAL} has the minimal value C_{min}.

Creating the curve C_{MEM} is not difficult. For that purpose, a simulation of the behaviour of LFSR, a pseudorandom test pattern generator (PRG) is needed. The fault simulation should be carried out for the complete test sequence generated by LFSR. As a result of such a simulation, for each clock cycle the list of faults can be found, which were covered at this clock cycle. By removing these faults from the complete fault list, it is possible to know the number of faults remaining to be tested.

Table 6-1 represents a fragment of the results of BIST simulation for ISCAS'85 circuit `c880`. In this table:

- *k* denotes the number of clock cycles
- *$r_{DET}(k)$* is the number of new faults detected (covered) by the test pattern generated at the clock signal *k*
- *$r_{NOT}(k)$* is the number of remaining faults after applying the sequence of patterns generated by the k clock signals
- *FC(k)* is the fault coverage reached by the sequence of patterns generated by the k clock signals

Table 6-1. *BIST analysis data*

k	$r_{DET}(k)$	$r_{NOT}(k)$	FC(k)	t(k)
0	155	839	15.59%	104
1	76	763	23.24%	104
2	65	698	29.78%	100
3	90	608	38.83%	101
4	39	564	43.26%	99
5	104	421	57.65%	99
10	66	355	64.28%	95
15	66	355	64.28%	92
20	44	311	68.71%	87
28	42	269	72.94%	81
50	51	218	78.07%	74
70	57	161	83.80%	58
100	16	145	85.41%	52
148	13	132	86.72%	46
200	18	114	88.53%	41
322	13	101	89.53%	35
411	31	70	92.96%	26
707	24	46	95.37%	17
954	18	28	97.18%	12
1535	4	24	97,58%	11
1560	8	16	98.39%	7
2153	11	5	99.50%	3
3449	2	3	99.70%	2
4519	2	1	99.89%	1
4520	1	0	100.00%	0

In the list of BIST simulation results not all clock cycles are presented. We are only interested in the clock numbers at which at least one new fault will be covered, and thus the total fault coverage for the pseudorandom test sequence up to this clock number increases. Let us call such clock numbers and the corresponding pseudorandom test patterns resultative clocks and resultative patterns. The rows in *Table 6-1* represent the resultative clocks, but not all (only some resultative points are given for illustrative purpose), for the circuit `c880`.

If we decide to switch from the online pseudorandom test generation mode to the deterministic stored pattern mode after the clock number *k,* the *L* = *k*.

More difficult is to find the values of *βS*, the cost for storing additional deterministic patterns in order to reach the given fault coverage level (100% in the ideal case). Let *t(k)* be the number of test patterns needed to cover *$r_{NOT}(k)$* not yet detected faults (these patterns should be precomputed and used as stored test patterns). The calculation of the data in the column *t(k)* of *Table 6-1* is the most expensive procedure.

6.6 Target architecture for Hybrid BIST

In the following, two possible solutions are presented for implementing the hybrid BIST - hardware based and software based.

A hardware based hybrid BIST architecture is depicted in *Fig. 6-10* where the pseudorandom pattern generator (PRPG) and the Multiple Input Signature Analyzer (MISR) are implemented inside the circuit under test (CUT). The deterministic test patterns are pre-computed off-line and stored inside the system.

To avoid the hardware overhead caused by the PRPG and MISR, and the performance degradation due to excessively large LFSRs, a software based hybrid BIST can be used where pseudorandom test patterns are produced by the test software. However, the cost calculation and optimization algorithms are general, and can be applied as well to the hardware based as to the software based hybrid BIST solutions.

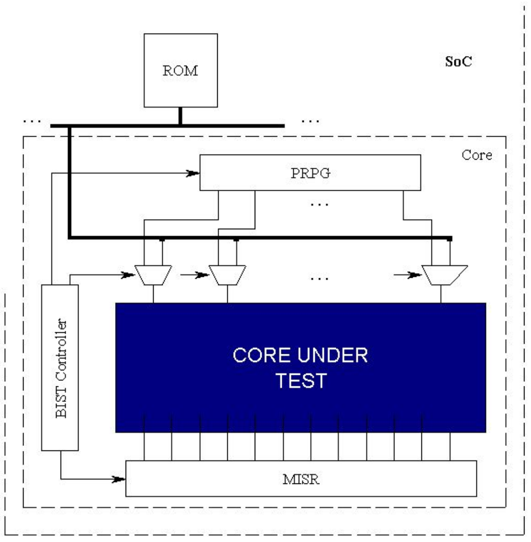


Fig. 6-10. *Hardware based hybrid BIST architecture*

In case of the software based solution, the test program, together with test data (LFSR polynomials, initial states, pseudorandom test length, signatures), is kept in ROM. The deterministic test vectors are generated during the development process and are stored in the same place. For transporting the test patterns, we assume that some form of TAM is available.

In test mode, the test program is executed in the processor core. The test program proceeds in two successive stages. In the first stage the pseudorandom test pattern generator which emulates LFSR, is executed. In the second stage the test program will apply precomputed deterministic test vectors to the core under test.

The pseudorandom TPG software is the same for all cores in the system and is stored as one single copy. All characteristics of the LFSR needed for the emulation are specific to each core and are stored in ROM. They will be loaded upon request. Such an approach is very effective in the case of multiple cores, because for each additional core, only the BIST characteristics for this core have to be stored. The general concept of the software based pseudorandom TPG is depicted in *Fig. 6-11.*

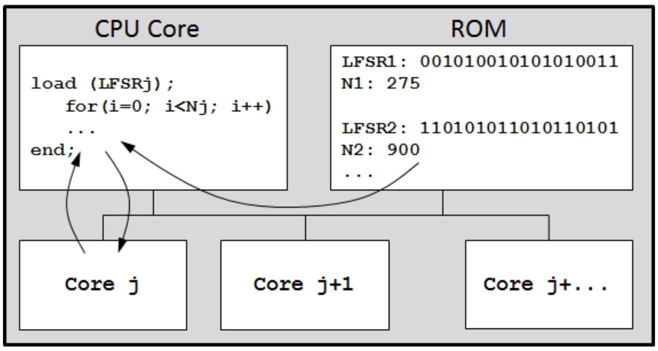


Fig. 6-11. *LFSR emulation*

Although it is assumed that the best possible pseudorandom sequence is used, not always all parts of the system are testable by a pure pseudorandom sequence. It can also take a very long test application time to reach a good fault coverage level. In case of the hybrid BIST, we can dramatically reduce the length of the initial pseudorandom sequence by complementing it with deterministic stored test patterns, and achieve 100% fault coverage. The method proposed in the paper helps to find tradeoffs between the length of the best pseudorandom test sequence and the number of stored deterministic patterns.

6.7 Cost calculation algorithms

In the following, different cost calculation algorithms are presented for calculating and optimization of hybrid BIST cost.

In *Table 6-1* the test data for `c880` was presented, where the column *t(k)* represents the number of patterns needed to cover the fault that have not been covered yet. As it has been mentioned, calculation of *t(k)* is quite difficult. In the following, two approaches are described to calculate the values of *t(k)*. These algorithms are desribed in [126][146].

The ways to find *t(k)*

- ATPG based approach
- fault table based approach

Let us have the following notations:

- *i* - the current number of the entry in the table of BIST analysis data;
- *k(i)* - the number of the clock cycle of the resultative clock at entry *i*
- $R_{DET}(i)$ - the set of new faults detected (covered) by the pseudorandom test pattern which is generated at the efficient clock signal number *k(i)*
- $R_{NOT}(i)$ - the set of not yet covered faults after applying the pseudorandom test pattern number *k(i)*
- *T(i)* -the set of test patterns needed and found by the ATPG to cover the faults in $R_{NOT}(i)$
- *N* - the number of all resultative patterns in the sequence created by the pseudorandom test.
- *FT* - the fault table for a given set of test patterns *T* and for the given set of faults *R*: the table defines the subsets $R(t_j) \in R$ of detected faults for each pattern $t_j \in T$

6.7.1 ATPG based algorithm for cost calculation

This ATPG based algorithm generates a new deterministic test set for the not yet detected faults at every resultative clock cycle. In this way we have the complete test set (consisting of pseudorandom and deterministic test vectors) for every resultative clock, which can reach maximum achievable fault coverage. The number of deterministic test vectors at all resultative clocks are then used to create the curve $C_{MEM}(\beta S)$. This algorithm is straightforward but very time consuming because of repetitive use of ATPG.

ATPG based approach algorithm of generation *t(k)*:

```
1. Let $k := N$;
2. Generate for $R_{NOT}(k)$ a test $T(k)$, $T := T(k)$, $t(k) := |T|$;
3. For all $k = N-1, N-2, .. 1$:
       Generate for the faults $R_{NOT}(k)$ not covered by $T$ a test set $T(k)$,
       $T := T+T(k)$, $t(k) := |T|$;
4. END.
```

Since usage of ATPG is very time consuming procedure, another algorithm was proposed, based on iterative transformations of fault tables. This algorithm allows a dramatic reduction of computation time for the hybrid BIST cost calculation.

6.7.2 Fault table based algorithm for cost calculation

This fault table based starts by generating a test set *T* for all detectable faults. Based on the fault simulation results a fault table *FT* will be created. By applying k pseudorandom patterns, we can remove from the original fault table all faults, which were covered by the pseudorandom vectors and by using static test compaction reduce the original deterministic test set. Those modifications should be performed iteratively for all possible breakpoints to calculate the curve $C_{MEM}(\beta S)$ and to use this information to find the optimal C_{TOTAL}.

Fault table based approach algorithm of generation *t(k)*

```
1.  Calculate the whole test $T = \{t_j\}$ for the whole set of faults $R$ by any ATPG to reach
    as high fault coverage as possible;
2.  Create for $T$ and $R$ the fault table $FT = \{R(t_j)\}$;
3.  Take $k = 0, T_k = T, R_k = R, FT_k = FT$;
4.  Take $k = k+1$;
5.  calculate by fault simulation $R_{DET}(k)$;
6.  Update the fault table: $\forall j, t_j \in T_k$: $R(t_j)-R_{DET}(k)$;
7.  Remove from the test set $T_k$ all the test patterns $t_j \in T_k$ where $R(t_j) = \varnothing$;
8.  If $T(k) = \varnothing$ go to END;
9.  Optimize the test set $T_k$ by any test compaction algorithm; $t(k) = |T_k|$; go to 4;
10. END.
```

In the case of very large circuits both of these algorithms may lead to very expensive and time consuming experiments. It would be desirable to find the global optimum of the total cost curve by as few selected values of *k* as possible.

In the following, an approach based on a *tabu search* is shown that allows to speed up the calculations.

6.7.3 Tabu search based algorithm for cost optimization

For reducing the number of total calculations in ATPG based and fault table based algorithms for finding the minimum value, the method of *tabu search* [107][108][109] as a general iterative heuristic for solving combinatorial optimization problems can be used. The main ideas of this approach are presented in *Chapter 4***.** The description of the algorithm is given in *Fig. 6-12.*

The procedure of *tabu search* for hybrid BIST cost optimization starts from an initial feasible solution *SO* (current solution) in the search space *Ω*. In the presented approach, the fast estimation method proposed in [127] is used to find the initial solution. The estimation method is based on number of not yet covered faults $R_{NOT}(i)$ and can be obtained from the pseudorandom test simulation results

(*Table 6-1*). A neighbourhood *N(SO)* is defined for each *SO*. Based on experimental results its was concluded that the most efficient step size for defining the neighbourhood *N(SO)* was 3% of efficient clocks. Larger step size, even if it can give a considerable speedup, will decrease the accuracy of the final result. A sample of neighbour solutions $V^* \subset N(SO)$ is generated. An extreme case is to generate the entire neighbourhood, that is to take $V^* = N(SO)$. Since it is generally impractical (computationally expensive), a small sample of neighbours ($V^* \subset N(SO)$) is generated, and called *trial* solutions ($|V^*| = n << |N(SO)|$). In case of ISCAS'85 benchmark circuits the best results were obtained when the size of the sample neighbourhood was 4. Increase of the size V^* had no effect to the improvement of the results. From these trial solutions the best solution $SO^* \in V^*$ is chosen for the consideration as the next solution. The move to SO* is considered even if *SO** is worse than *SO*, that is, $Cost(SO^*) > Cost(SO)$. A move from *SO* to *SO** is made provided certain conditions are satisfied. The best candidate solution $SO^* \in V^*$ *may* or *may not* improve the current solution but is still considered. It is this feature that enables *escaping* from local optima.

```
Start with initial solution $SO \subset \Omega$
BestSolution:=SO;
Initialize Tabu list T:=$\varnothing$
While number of empty iterations <E
      Or there is no return to previously visited solution
         Do
            Generate the sample of neighbour solutions $V^* \subset N(SO)$;
            Find best $Cost(SO^* \subset V^*)$;
M:          If move to solution SO* is not in the T Then
                  $SO^{trial}$ :=SO*;
                  Update Tabu list;
            Else
                  Find the next best $Cost(SO^* \subset V^*)$;
                  Go to M;
            End If;
            If $Cost(SO^{trial}) < Cost(BestSolution)$ Then
                BestSolution:=SOtrial;
            Else
                Increment number of empty iterations E;
            End If;
End While;
END.
```

Fig. 6-12. *Tabu search based algorithm*

One of the parameters of the algorithm is the size of the tabu list. A *tabu list T* is maintained to prevent returning to previously visited solutions. The list contains information that to some extent forbids the search from returning to a previously visited solutions. Generally the *tabu list* size is small. The size can be determined by experimental runs, watching the occurrence of cycling when the size is too small, and the deterioration of solution quality when the size is too large [111]. Results have shown that the best average size of the *tabu list* for the ISCAS'85 benchmark family was 3. The size of the *tabu list* can be determined by experimental runs, watching the occurrence of cycling when the size is too small and the deterioration of the solution quality when the size is too large.

Let's have the following additional notations:

- E - number of allowed empty iterations (i.e. iterations that do not result in finding a new best solution) defined for each circuit
- SO^{trial} - solution generated from current solution as a result of the move.

For finding a good initial feasible solution in order to make *tabu search* more productive, a fast estimation method for a local optimal *L* proposed in [127]is used. For this estimation, the number of not yet covered faults in $R_{NOT}(i)$ can be used. The value of $|R_{NOT}(i)|$ can be acquired directly from the PRG simulation results and be available for every significant time moment (*Table 6-1*). Based on the value of $|R_{NOT}(i)|$ it is possible to estimate the expected number of test patterns needed for covering the faults in $|R_{NOT}(i)|$. The starting point for the *tabu search* procedure can be found by giving rough estimation of the total cost based on the value of $|R_{NOT}(i)|$. Based on the statistical analysis of the costs calculated for ISCAS'85 benchmark circuits, in [127] the following approximation was proposed: one remaining fault results in 0,45 test patterns needed to cover it. In this way, a simplified cost prediction function was derived:

$$C'_{TOTAL}(k)=C_{GEN}(k) + 0,45\beta R_{NOT}(k)$$

The value k^*, where $C'_{TOTAL}(k^*)=min(C'_{TOTAL}(k))$ was used as the initial solution for *tabu search*.

6.8 Experimental results

6.8.1 Tabu search

Experiments were carried out on ISCAS'85 benchmarks in order to demonstrate the advantage of *tabu search* compared to the known methods. Turbo Tester toolset [147] was used for deterministic test pattern generation, fault simulation, and test set compaction.

Investigations were carried out to find the best initial solution, the step defining *N(S)*, the size of *V** and the size of *tabu list* for using the *tabu* strategy in a most efficient way.

For finding the best initial solution the cost prediction proposed in [126] was used. For finding the *tabu list* size, experiments were carried out with different sizes of the list. Results showed that the best average size for the ISCAS'85 benchmark family was 3. Smaller list size would cause cycling around local minimum, larger size would result in deterioration of the solution quality (see *Fig. 6-13a* and *Fig. 6-13b*).

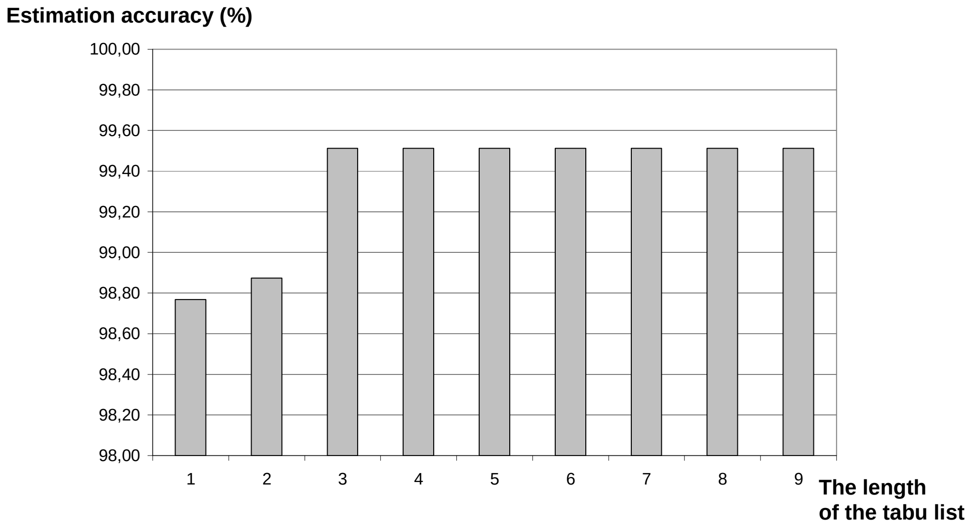


Fig. 6-13(a) *Dependency of solution esimation accuracy from the Tabu list size*

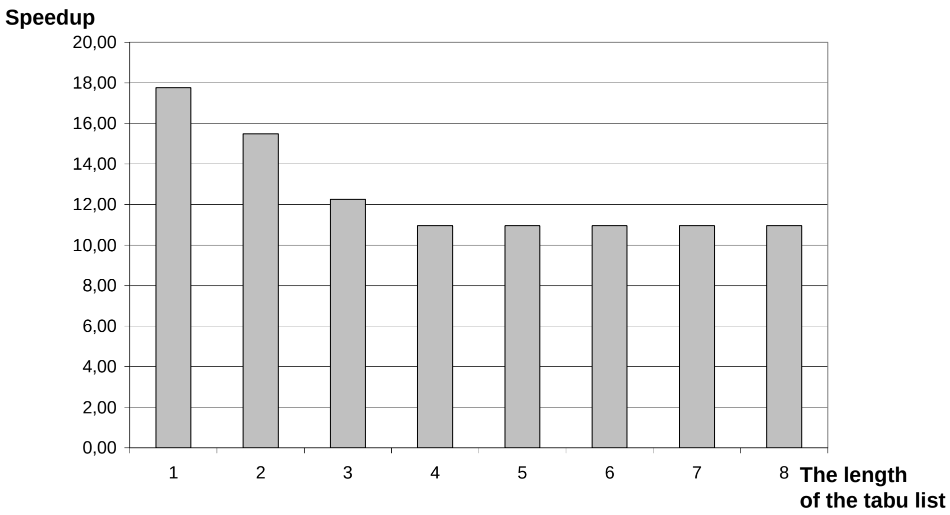


Fig. 6-13(b*)* *Dependancy of the solution estimation accuracy from the Tabu list size*

The size of the sample of neighbourhood solutions V* giving the best results for all circuits, was 4. Smaller size would make the process of finding the minimum very long, resulting in very small speedup. Larger size of *V** did not improve the results.

The efficiency of the search depends significantly of the step size defining the neighbourhood *N(S)*. Based on the experimental results, the charts of dependency of overall estimation accuracy and the overall speedup from step size were composed. Analyzing results depicted in *Fig. 6-14a* and *Fig. 6-14b* led to the conclusion that the most admissible step size can be counted as 3% of the resultative clocks where the average estimation accuracy is the highest. Though the larger step size would give us the increase of the speedup, it was found inadmissible because of the rapid decrease in the cost estimation accuracy.

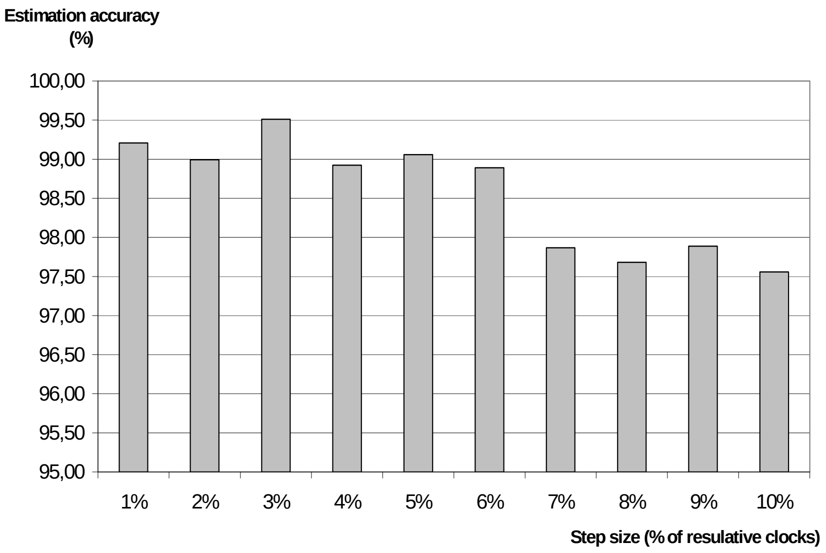


Fig. 6-14(a) *Dependency of estimation accuracy from the neighbourhood step size*

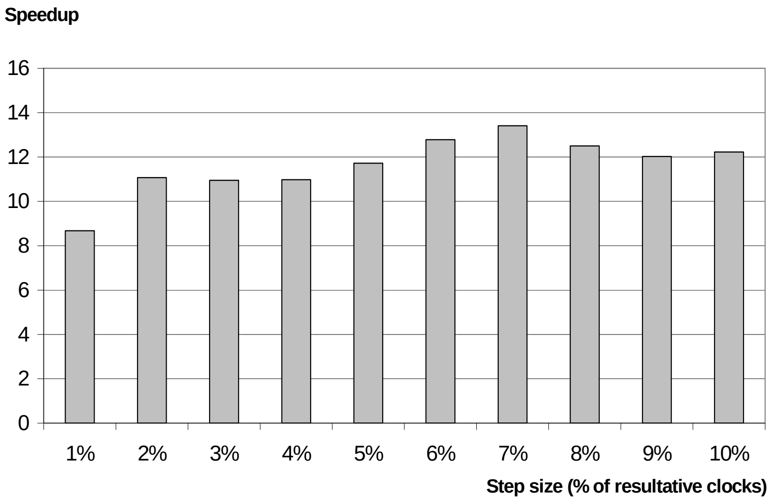


Fig. 6-14(b) *Dependency of average speedup from the neighbourhood size*

Investigations were carried out to find out the criteria to stop iterations. Analysing the experiments with a fixed number of iterations to find out which number of empty iterations *E* (iterations not giving new best cost) will most probably end up in getting no better solution, the value was found *E=7*. The other criteria to stop the iterations is the return of the search to a previously visited solution.

The results of optimization of the hybrid BIST for the ISCAS'85 benchmark circuits obtained using the parameters described above are depicted in *Table 6-2* The number of cost calculations is given in row 6, the number of total iterations in row 8.

Table 6-2(a) *Experimental results - cost optimization using Tabu search*

	c432	c499	c880	c1355	c1908
Simulated clocks	780	2036	5589	1552	5803
Resultative clocks	81	114	114	109	183
Actual total cost	165	398	366	374	487
Estimated total cost	165	398	367	376	487
Estimation accuracy, %	100.00	100.00	99.73	99.47	100.00
Number of calculations	11	19	15	18	28
Speedup	7.36	6.00	7.60	6.06	6.54
Iterations made	7	14	10	13	17

Table 6-2(b) *Experimental results - cost optimization using Tabu search*

	c2670	c3540	c5315	c6288	c7552
Simulated clocks	6581	8734	2318	210	18704
Resultative clocks	118	265	252	53	279
Actual total cost	2397	771	1072	63	3203
Estimated total cost	2420	771	1103	63	3213
Estimation accuracy, %	99.05	100.00	97.19	100.00	99.69
Number of calculations	9	16	12	15	8
Speedup	13.11	16.56	21.00	3.53	34.87
Iterations made	6	10	9	12	5

6.8.2 Comparing Tabu search to ATPG and FT based approaches

Experiments were carried out on the ISCAS'85 benchmark circuits for comparing the previously described algorithms based on ATPG and fault table, and for investigating the efficiency of *tabu search* based method for optimizing hybrid BIST. Again, Turbo Tester toolset was used.

The results of the experiments are presented in *Table 6-3* and *Table 6-4* and illustrated by a diagram in *Fig. 6-15.*

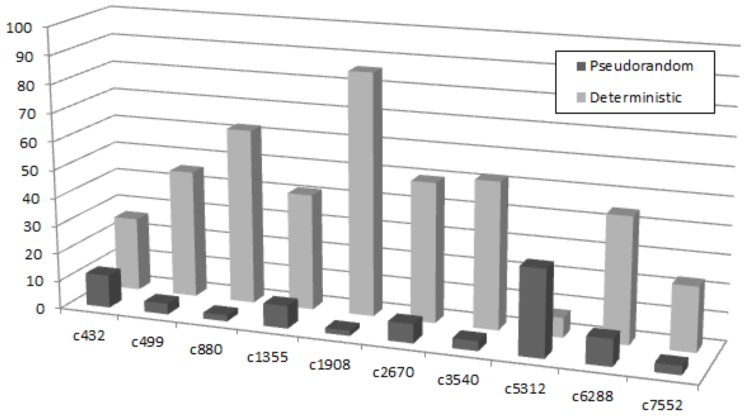


Fig. 6-15. *Percentage of test patterns in the optimized test sets compared to the original test sets.*

For calculating the total cost of hybrid BIST the previously described formula was used: $C_{TOTAL}=\alpha L+\beta S$. For simplicity, it is assumed that $\alpha = 1$ and $\beta =$ B where B is the number of bytes of the input test vector applied to the CUT. To carry out some experimental work for demonstrating the feasibility and efficiency of the algorithm, the number of clocks is used as a cost for pseudorandom test generator and the number of bytes is used as a cost for storing precomputed deterministic test patterns.

In the columns of the *Table 6-3* the following data is depicted: ISCAS'85 benchmark circuit name, L - length of the pseudorandom test sequence, FC - fault coverage, S - number of test patterns generated by deterministic ATPG to be stored in BIST, C_T - total cost of BIST.

In the *Table 6-4* the following data is shown: T_G - the time (sec) needed for ATPG to generate the deterministic test set, T_A - the time (sec) needed for carrying out manipulation on fault tables (subtracting faults, and compacting test set), N - number of efficient patterns in the pseudorandom sequence, T_1 and T_2 - the time (sec) to find the optimal cost by using *tabu search*. T_S - the number of calculations in *tabu search*, *Acc* - accuracy of the *tabu search* solution in percentage compared to the exact solution found from the full cost curve. The total testing time for ATPG and FT based algorithms and for *tabu search* was calculated as follows:

$$T_1 = N * T_G,$$
$$T_2 = T_G + N * T_A,$$
$$T_3 = T_2 * (T_S/N) + \Delta,$$

respectively, where Δ is the time needed to perform the *tabu search* calculations (was below 0.1 sec in given experiments).

Table 6-3. *Experimental results - pseudorandom, stored and hybrid test*

Circuit	*Pseudorandom test*		*Stored test*		*Hybrid test*		
	L	*FC*	*S*	*FC*	*L*	*S*	C_T
c432	780	93.0	80	93.0	91	21	196
c499	2036	99.3	132	99.3	78	60	438
c880	5589	100.0	77	100.0	121	48	505
c1355	1522	99.5	126	99.5	122	52	433
c1908	5803	99.5	143	99.5	105	123	720
c2670	6581	84.9	155	99.5	444	77	2754
c3540	8734	95.5	211	95.5	297	110	1067
c5315	2318	98.9	171	98.9	711	12	987
c6288	210	99.3	45	99.3	20	20	100
c7552	18704	93.7	267	97.1	583	61	2169

In fact, the values for T_G and T_A differ for the different values of i = 1,2,..,N. However the differences were in the range of few percents which allowed us to neglect this impact and to use the average values of T_G and T_A.

In *Fig. 6-15* the amount of pseudorandom and deterministic test patterns in the optimal BIST solution is compared to the sizes of pseudorandom and deterministic test sets when only either of the sets is used. In the typical cases less than half of the deterministic vectors and only a small fraction of pseudorandom vectors are needed, however the maximum achievable fault coverage is guaranteed and achieved.

Table 6-4. *Experimental results - calculation cost*

Circuit	Calculation cost							
	T_G	T_A	*N*	T_1	T_2	T_3	T_S	*Acc*
c432	20.1	0.01	81	1632	21	2.85	11	100.0
c499	0.7	0.02	114	174	3	0.50	19	100.0
c880	0.2	0.02	114	17	2	0.26	15	99.7
c1355	1.2	0.03	109	133	5	0.83	18	99.5
c1908	11.7	0.07	183	2132	25	3.83	28	100.0
c2670	1.9	0.09	118	230	13	0.99	9	99.1
c3540	85.3	0.14	265	22601	122	7.37	16	100.0
c5315	10.3	0.11	252	2593	38	1.81	12	97.2
c6288	3.8	0.04	53	200	6	1.70	15	100.0
c7552	53.8	0.27	279	15004	129	3.70	8	99.7

Fig. 6-16 compares the costs of different approaches using for Hybrid BIST cost calculation the equation $C_{TOTAL}=\alpha L+\beta S$ with parameters α = 1 and β = B where B is the number of bytes of the input vector to be applied on the CUT. As

pseudorandom test is usually the most expensive method, it has been selected as reference with given value 100%. The other methods give considerable reduction in terms of cost, and as it can be seen, hybrid BIST approach has significant advantage compared to the pure pseudorandom or stored test approach in most cases.

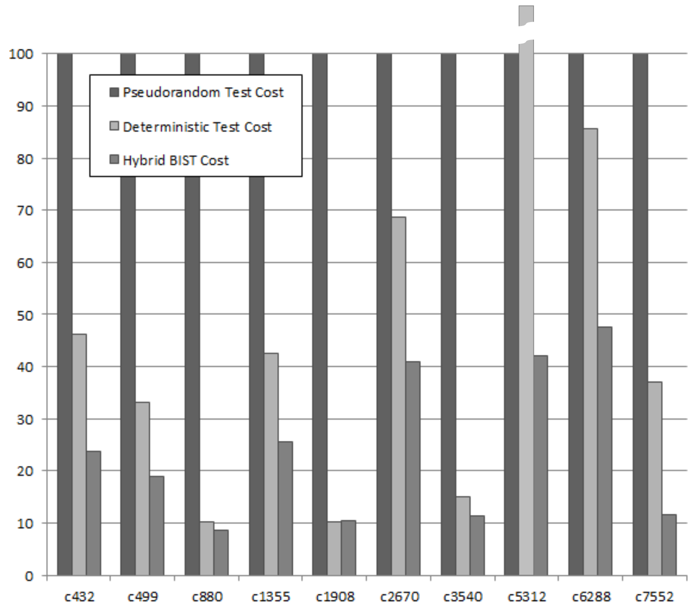


Fig. 6-16. *Cost comparison of different methods. Cost of pseudorandom test is taken as 100%*

6.9 Conclusions

1. In this chapter, hybrid BIST approach for testing systems-on-chips has been described. It supports the combination of pseudorandom test patterns with deterministic test patterns in a cost-effective way. The self-test architecture can be implemented either in classical way, by using LFSRs, or in software to reduce the area overhead and to take advantage of the SoC architecture.

2. For selecting the optimal switching point from pseudorandom test mode to stored test mode, two approaches - ATPG and fault table based algorithms were described that allow calculating the complete cost curve of different hybrid BIST solutions.

3. The *tabu search* approach based algorithm for finding a global minimum in a search space containing many local minimums was presented and described. The proposed solution was developed to reduce the number of calculations in search for the optimal solution for hybrid BIST. The speedup of using *tabu search* for ISCAS'85 benchmark family varies from 3,5 to 34,9 (10,5 in average) in comparison to fault table manipulations based algorithm, whereas the calculated accuracy of the solution (the minimum cost compared to the exact minimum) was not less than 97,2% for the whole family of ISCAS'85 benchmarks.

4. The experimental results demonstrate the feasibility of the method and algorithms described, and the efficiency of the fault table based cost calculation method combined with *tabu search* for finding optimized cost-effective solutions for hybrid BIST.

Chapter 7 Constraints based optimization of Hybrid BIST with reseeding

In this chapter, hybrid BIST algorithms that combine stored precomputed and pseudorandom test vectors in order to perform SoC testing under given memory contraints, such that the test time is minimized, but miaximum achievable fault coverage is still guaranteed. The methods provide possibility to find a memory contrained test solution for every individual core in the system. The experiments have been conducted on different ISCAS benchmarks and the results are compared with the techniques developed earlier. The results show the feasibility and advantage of the new proposed approaches.

7.1 Basic principle of Hybrid BIST with reseeding

To illustrate the reseeding BIST method let us depict all possible *N* test patterns as a line (*Fig. 7-1* and *Fig. 7-2*). When using hybrid BIST methodology many faults will be covered by pseudorandom patterns and the remaining hard to test faults (HTTF) are covered by *M* deterministic patterns. The method could be improved if we could devise such a pseudorandom sequence which would also cover HTTFs. The solution is to use many smaller pseudorandom sequences, starting with some specific test pattern (for example, a test pattern targeting HTTF) as a seed. Consequently, the full test set will be constructed as a collection of pseudorandom pattern blocks (PPB), represented in *Fig. 7-2* as separate intervals, in such a way that all HTTF will be covered by these test patterns. Each block has its own seed. The main problem of this approach is how to calculate the number and size of PPBs ad how these tests should be spread all over test patterns, i.e. which seeds for PPBs should be used. The limiting factor is that in general case, we don't know which faults are HTTF and which test patterns are needed for detecting them.

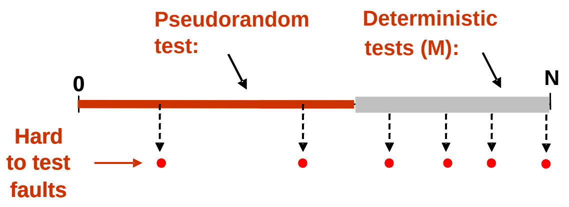


Fig. 7-1. *Hybrid BIST with M deterministic test vectors*

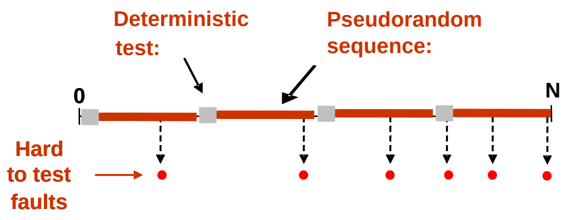


Fig. 7-2. *Reseeding with R seeds (R<M)*

Such a test set can be found using the following algorithm. Let *DT* be the deterministic test set for a given CUT, and *R* the set of all possible faults in the CUT. Let us call all the faults in $R_H \subset$ R, which are covered only by a single test pattern *DT*, HTTFs. With $DT_H \subset DT$ we denote the subset of test patterns which cover HTTFs R_H. Obviously $|DT_H| = |R_H|$. The first seed $Ti \in DT_H$ for the first pseudorandom pattern block B_i, $i = 1$, will be selected from the DT_H. Let $b_i = |B_i|$ be the length of the block B_i. The algorithm now removes all the faults covered by B_i from *R*, and keeps in *DT* only these patterns that are needed for covering the faults in the updated *R*. A new $R_H \subset R$, and a new $DT_H \subset D$ are calculated, and the next seed $T_i \in DT_H$, $i = 2$ from the updated DT_H will be chosen for starting the next block B_i, $i = 2$, of pseudorandom patterns. This procedure should be continued the set of faults *R* is empty. Let the number of iterations be *k*. Then the length of the full test is calculated as

$$L = \sum_{i=1}^{k} b_i$$

and the amount of memory *M* needed for storing the seeds is determined by *k* test patterns that are needed to generate these *k* blocks. The characteristics of the solutions *L* and *M* are heavily depending on the lengths of the blocks. These blocks can be with equal length or variable length. In order to illustrate the situation the simulations have been carried out for a range of different block lengths for the ISCAS benchmarks to see how the values of *L* and *M* are changing with the length of *b*. As an example in *Fig. 7-3* the curves of $L_1(b)$, $L_2(b)$ and $M(b)$ for ISCAS circuit `c1908` have been depicted. $L_1(b)$ is the length of the final test set under given memory constraint when the length of the blocks is equal, and $L_2(b)$ is the test length when the length of the block is variable. The variable length is a result of a simple optimization procedure where all blocks are fault simulated in order to remove useless patterns from the end of the block. Useless patterns are those that do not detect any new faults.

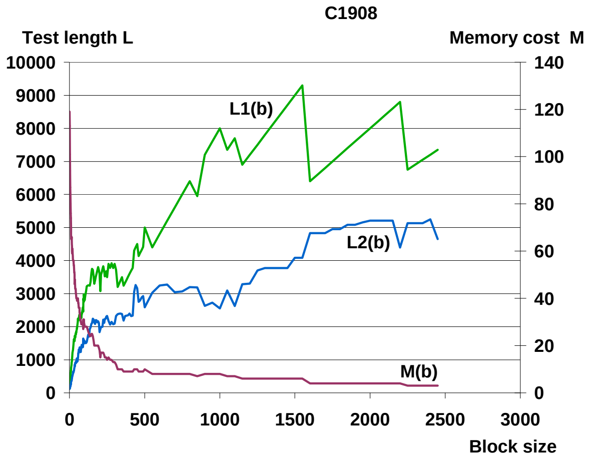


Fig. 7-3. *Comparison of equal length reseeding versus variable length reseeding for the circuit* ***c1908***

In *Fig. 7-4* a possible structure of the assumed BIST architecture is shown [148]. ROM contains the seeds. Each pattern P_i in the ROM serves as an initial state of the LFSR for test pattern generation. BIST controller counts the number of L_i pseudorandom patterns that are generated starting from P_i. After finishing the cycle, ROM controller is forwarding the next pattern P_{i+1} from the ROM to the core under test. The important task is to find lengths of the blocks so that $L = min$ at the given constraint $M \leq M_{max}$.

There are two important issues to consider when constructing the reseeding based solutions: the number of seeds and the number of pseudorandom patterns generated from each seed. Both have significant impact on the final solution in terms of fault coverage, test length and test memory requirements. In the following sections some approaches will be described to perform and optimize hybrid BIST with reseeding to obtain the test that satisfies the given memory constraint.

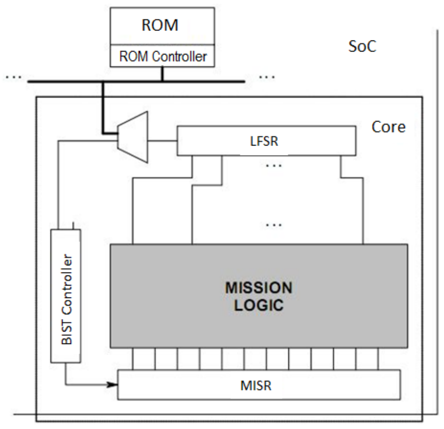


Fig. 7-4. *Test Architecture for hybrid BIST with reseeding*

7.2 Hybrid BIST length reduction using reseeding and test set compaction

In the following, a method is described for finding the length of pseudorandom seeds, such that the test memory (number of seeds) is constrained, the test time is minimized, and the maximum achievable fault coverage guaranteed. The pseudocode of the algoritm is shown in *Fig. 7-5*.

Algorithm using reseeding and test set compaction

```
- Generate deterministic test patterns;
- for ( each deterministic pattern )
       - Form a block by generating L pseudorandom patterns, using
       deterministic pattern as a seed;
       - Fault simulate the block;
- block_length = 1;
- loop ( until L has been reached )
       - for each ( block )
              - calculate coverage summary (block_length);
              // Find coverage summary for a block with block_length
              // number of vectors starting from the beginning
              - minimize the block length;
              // finds a point within a block from where the
              // fault coverage is not any more increasing. Removes
              // the useless pseudorandom vectors
       - order blocks based on length
       // order the tests in increasing length (shortest first)
       - minimize the number of blocks
       // finds minimal number of blocks needed to obtain
       // the maximum fault coverage: remaining_blocks
       - initialize total_coverage_summary;
       - for each ( remaining_block ); // calculates final test length
              - optimize block length;
              // remove all unnecessary pseudorandom patterns
              // that do not contribute to the final fault coverage
              // taking into consideration ALL blocks that have been
              // applied earlier (using total coverage summary)
              - total_coverage_summary += coverage( block );
              // add new faults from a block into total coverage summary
       - set block_length (block_length += block_length/stepping_const);
       // next iteration step
- done;
```

Fig. 7-5. *The pseudocode of the algorithm using reseeding and test set compaction*

Let *DT* be the deterministic test set $DT = \{DT_i\}$ for a given *CUT* and *R* the set of possible faults in the *CUT*. Let us denote by $R(DTi) \subset R$ the subset of faults detected by a test pattern $DT_i \in DT$. We assume that *DT* obtains the maximum achievable fault coverage, hence $R(DT) = R$.

Algorithm starts by generating pseudorandom sequences PR_i with a given length *L* where DT_i is used as a seed for the pseudorandom sequence. Let us denote the set of these hybrid sequences as $PR = \{PR_i\}$. For all $PR_i \in PR$ and for each test pattern $t_k \in PR_i$ cumulative fault coverage can be calculated as:

$$FC(PR_{i,k}) = \frac{\left| \bigcup_{j=\overline{1,k};\, t_j \in PR_i} R(t_j) \right|}{|R|} \tag{7-1}$$

Thereafter the pseudorandom sequences can be minimized. From each $PR_i \in PR$ all pseudorandom test patterns $t_j, t_{j+1}, ..., t_L$ where $FC(PR_{i,k}) = FC(PR_{i,k-1})$, for all $k = j, j+1, ..., L$ will be removed, because these patterns will not contribute for the increase in fault coverage. These first steps are illustrated on *Fig. 7-6*. Let us denote L_i the new reduced length of the pseudorandom sequence PR_i, with $FC(PR_i)$ the fault coverage of the sequence PR_i calculated using the *Equation (7-1)* of the last pattern in PR_i and with $R(PR_i) \subset R$ the subset of faults detected by PR_i.

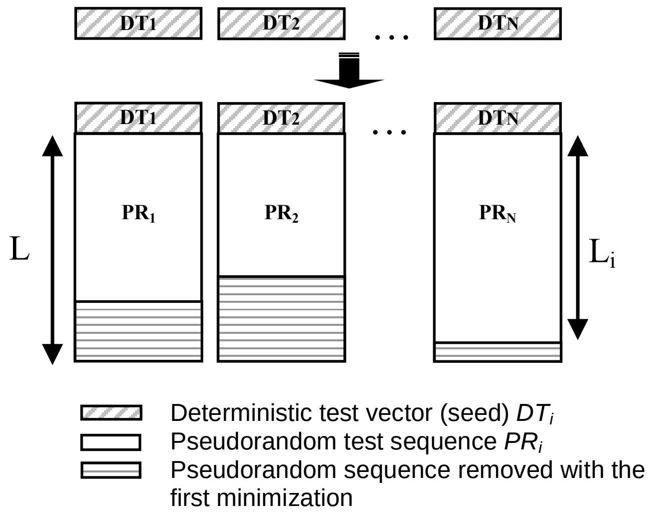

DT1
DT2
DTN
DT1
DT2
DTN
PR1
PR2
PRN
L
Li
Deterministic test vector (seed) DTi
Pseudorandom test sequence PRi
Pseudorandom sequence removed with the first minimization

Fig. 7-6. *First minimization of the pseudorandom sequences.*

Thereafter, all hybrid test sequences PR_i will be ordered in increasing order, so that $L_i \geq L_{i-1}$ for every $i=1,2, ..., N$ where $N = |DT|$.

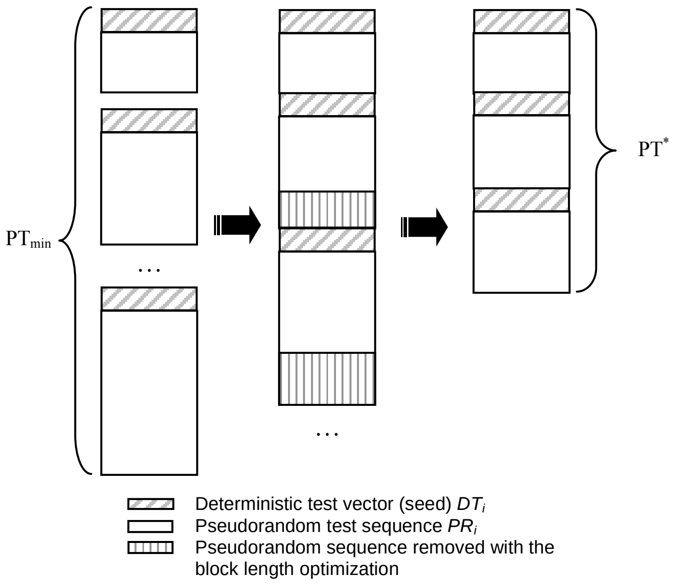
Consider now a composite hybrid test sequence PT, composed of all sequences $PR_i \in PR$ in the order how they were ranked. Since all initially generated deterministic test patterns $t \in DT$ are included in PT, we have

$$FC(PT) = FC(DT),$$
$$R(PT) = FC(DT) = R.$$

To minimize the length of the multi-seed hybrid test sequence we will use here the test pattern optimization algorithm. In this algorithm the minimal subset T_{min} of the given test set T will be found based on the information of fault subsets $R(t)$ detected by the test patterns $t_i \in PR$, so that the fault coverage remains the same $FC(T_{min}) = FC(T)$. To do this we interpret the sequences $PR_i \in PR$ as test patterns $t \in T$, and the fault sets $R(PR_i)$, respectively as fault sets $R(t)$. As the result of optimization we will find a minimal subset $PR_{min} \subset PR$, so that $FC(PR_{min}) = FC(PR)$. This step is rather fast as we do not optimize at the level of individual patterns but only at the level of complete sequences (blocks).

Now a new composite hybrid sequence PT_{min} will be created from the ordered set of subsequences $PT_{min} = (PR_1, PR_2, \ldots, PR_m)$ where $m < N$.

The next step will be minimization of the total length of the sequence PT_{min}. As the result, a reduced final multi-seed hybrid sequence PT * will be created.



PT_min
PT*
...
▨ Deterministic test vector (seed) DT_i
▭ Pseudorandom test sequence PR_i
▥ Pseudorandom sequence removed with the block length optimization

Fig. 7-7. *Calculation of the final hybrid sequence*

To do that we calculate again the cumulative fault coverages for all the test patterns $t_k \in PT_{min}$ similarly to *Equation (7-1)* for all subsequences $PR_i \in PR_{min}$ in the order how they were ranked and put into PT_{min}. After calculating the fault coverage of a current subsequence PR_i in PT * we remove from PR_i all the test patterns $t_j, t_{j+1}, \ldots, t_{L,i}$ where $FC(PR_{i,k}) = FC(PR_{i,k-1})$, for all $k = j, j+1, \ldots, L_i$. This procedure is illustrated in *Fig. 7-7.* As the optimization procedure takes into account only the cumulative fault coverage of earlier blocks and does not analyze individual patterns in the current block, then also this step is rather fast.

Since the described reduction of the whole multi-seed hybrid sequence will not reduce the fault coverage, we have

$$FC(PT^*) = FC(PT) = FC(DT).$$

As a result of this algorithm we can find the length of a hybrid sequence for any arbitrary memory constraint. As a by-product we can also find the length of the longest hybrid block, that remained at the end of the optimization sequence.

Such an optimization is very necessary when developing a solution for testing core-based systems, such as SoCs or NoCs. The memory constraints can be seen as limitations of the on-chip memory or ATE, where the deterministic test set will be stored, and are therefore of great practical importance.

7.3 Optimization methods

7.3.1 Local search based algorithm for BIST length minimization

In this section, a local search based algorithm for test legth minimization is described. The algorithm consists of two main parts – in the first part a solution satisfying the memory contraint is found, in the second part the search is performed in the neighbourhood of the solution to find an optimal solution, keeping in mind the given memory constraint. The pseudocode of the algorithm is presented in *Fig 7-8.*

Local search based algorithm

```
- blockSize = initialBlockSize;
- currentMemory = seedCount (blockSize);

- while (currentMemory > maxMemory)
        - blockSize' = blockSize + ΔblockSize;
        - currentMemory' = seedCount (blockSize);
        - Δmemory = currentMemory' - currentMemory;

        - speed = Δmemory/ΔblockSize;
        - ΔblockSize = ROUNDUP ((currentMemory'-maxMemory)/speed);

        - blockSize = blockSize';
        - currentMemory = currentMemory';

- do
        - step = ROUNDUP (step/2);
        - chooseBest (HyBISTLength(blockSize+step),
                      HyBISTLength(blockSize-step));
- until step =1;
```

Fig. 7-8. *Local search based algorithm for test length minimization*

The key notations of the algorithm are presented in *Fig. 7-9* where both block length and memory curves are presented.

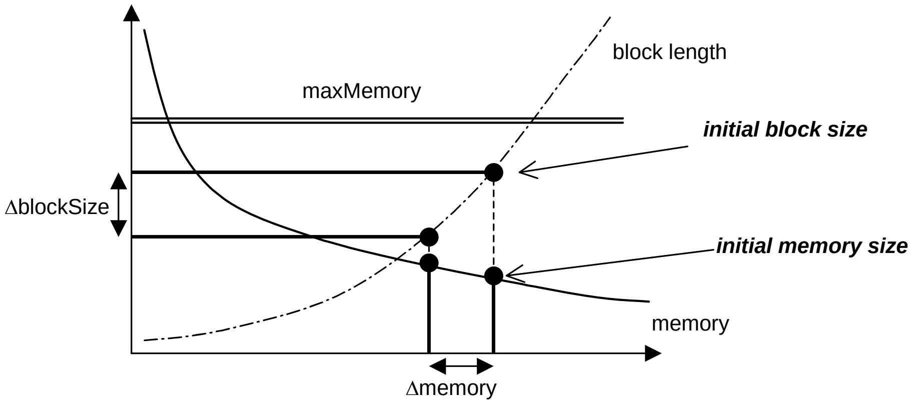


Fig. 7-9. *Illustration of the key terms used in the local search based algorithm*

The algorithm is initialized with a given block size *initialBlockSize* and a step length Δ*initBlockSize* that determines the next block size to be investigated. For the both block sizes the numbers of seeds that are needed to perform the test are calculated. Seeds are chosen from the precomputed set of deterministic test vectors *DT* in accordance to the method described in *Section 7.1*. First steps of the algorithm are illustrated in *Fig. 7-10*.

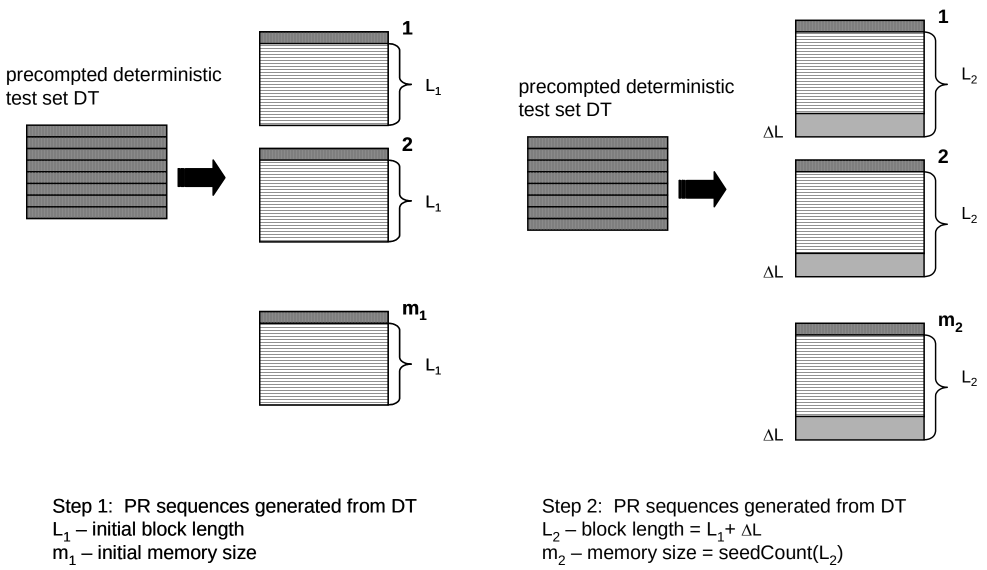

precompted deterministic test set DT
1
L_1
2
L_1
m_1
L_1
precompted deterministic test set DT
1
L_2
ΔL
2
L_2
ΔL
m_2
L_2
ΔL
Step 1: PR sequences generated from DT
L_1 – initial block length
m_1 – initial memory size
Step 2: PR sequences generated from DT
L_2 – block length = L_1+ ΔL
m_2 – memory size = seedCount(L_2)

Fig. 7-10. *Local search based algorithm - first steps*

Based on this data, so-called speed is calculated by which it is expected to reach the given memory constraint *maxMemory*. The speed determines the next block size calculated via Δ*blockSize* and again the corresponding number of seeds that are needed is calculated. After that, the data is analyzed to determine what is the next expected block size that would satisfy the memory constraint *maxMemory*. This process is repeated until the memory constraint is not satisfied.

The second part of the algorithm is illustrated in *Fig. 7-11*. In the second part, the search of the optimal solution is performed considering the memory constraint *maxMemory*. The length of the hybrid BIST is analyzed using the function *chooseBest* which determines the next possible solution (the one having the sorter Hybrid BIST length) within the neighbourhood of the current solution determined by the value of *step*. At every iteration, *step* becomes smaller thus approaching the optimal solution or the solution close to optimal.

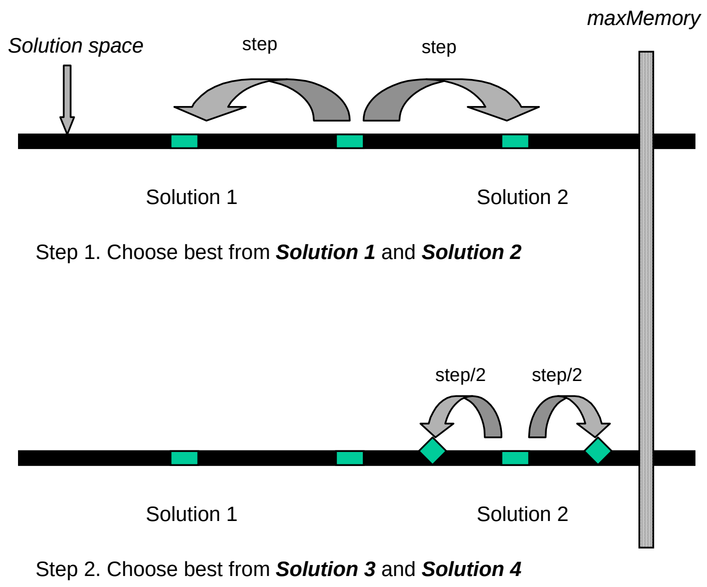

maxMemory
Solution space
step
step
Solution 1
Solution 2
Step 1. Choose best from **Solution 1** and **Solution 2**
step/2
step/2
Solution 1
Solution 2
Step 2. Choose best from **Solution 3** and **Solution 4**

Fig. 7-11. *Local search based approach - the second part of the algorithm*

As an example several search processes for the ISCAS'85 circuit c880 with different initial block sizes and different first step lengths are shown on *Fig. 7-12* to illustrate how optimum is approached when applying the described algorithm.

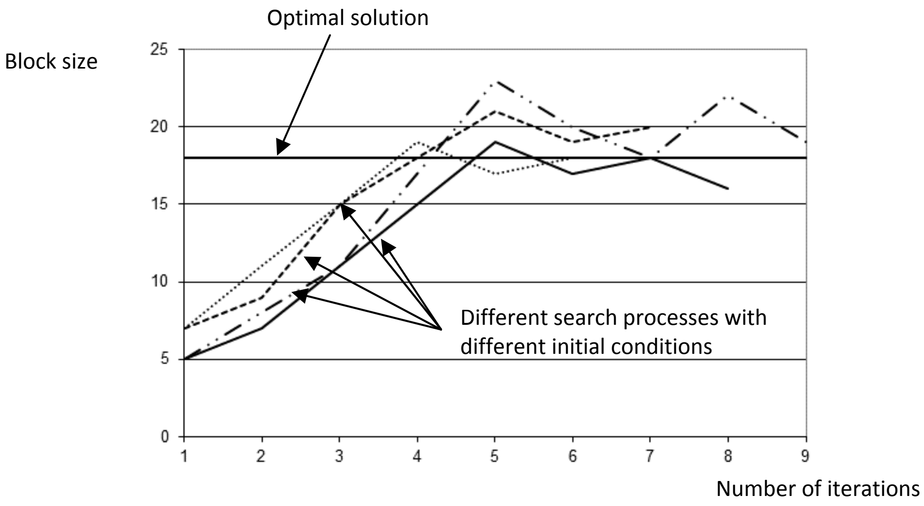


Fig. 7-12. *Search process of looking for the optimal solution*

7.3.2 Tabu search based algorithm for optimization of memory-constrained hybrid BIST

The method described in the previous section gives us the exact best solution that satisfies the predefined memory constraint. However, this method is quite time-consuming and computationally heavy as the entire solution space needs to be calculated. Therefore, a new heuristic is needed for finding the optimal solution and in the following, a new method is proposed, that allows to find the solution for the predined memory contraint by calculating only a subset of the solution space.

As it can be seen from the simulations and experiments presented earlier, the solution space is not linear, containing number of local optima. Experimental data for benchmark circuit `c7552` is shown on *Fig. 7-13.*

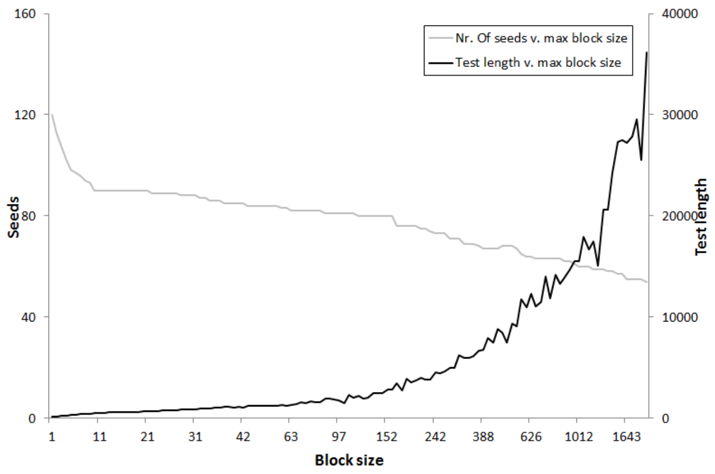


Fig. 7-13. *Relationships between number of seeds, maximal block size and total length for ISCAS benchmark* ***c7552****.*

The proposed method takes into consideration all the key issues of applying re-seeding based approaches and is based on *tabu search* which allows escaping from the local optima in the solution space that may satisfy the given memory constraint but not being best possible solution or close to that.

The pseudocode of the algorithm is presented in *Fig. 7-14.*

Tabu Search based algorithm

```
-   Generate set of deterministic test patterns DT such that best possible
    coverage is achieved;
-   Choose block length L of an initial solution;
-   Declare initial solution the best solution
-   Calculate the initial solution $S_I$, apply test block compaction
   //generate pseudorandom blocks of length
   //L for all deterministic patterns
   //remove test patterns that do not
   //contribute to overall fault coverage
- Initialize Tabu List
- until stopping criteria met
  - Calculate the current solution S, apply test block compaction
  - Update Tabu List
  - Create the neighbourhood N(S) of the
    current solution
  - Calculate the solutions in N(S), apply test block compaction
  - Find best S' solution from N(S)
    //find the shortest test set
    //where the number of deterministic test
    //patterns satisfies given
    //memory constraint M
  - loop (until solution S* is found)
    - if S' is not in Tabu List then
         - trial solution S*=S'
         - Update Tabu List
      else
         - Find the next best S'
  - if the current solution is best seen so far
      Update best solution
      //if the test set is shorter
      //than the previous best solution
  //next iteration step
- done
```

Fig. 7-14. *The pseudocode of the tabu search based algorithm*

The main steps of the algorithm are illustrated in *Fig. 7-15.* The algorithm starts from the initial solution defined by the longest pseudorandom block length *L* of the test set. Based on the statistical analysis of performed experiments, the initial solution was determined by:

*L=0,3*number of deterministic test vectors*

Next, initial solution is calculated using the method described in *Section 7.2* resulting in a set of precomputed deterministic test vectors from the set *DT* and the corresponding pseudorandom test sequences. Then, a test set compaction is applied, to remove the test patterns which do not contribute to fault coverage. After that, *tabu list* is initialized and the search loop is started. Inside the loop, the neighbourhood of the current solution is calculated.

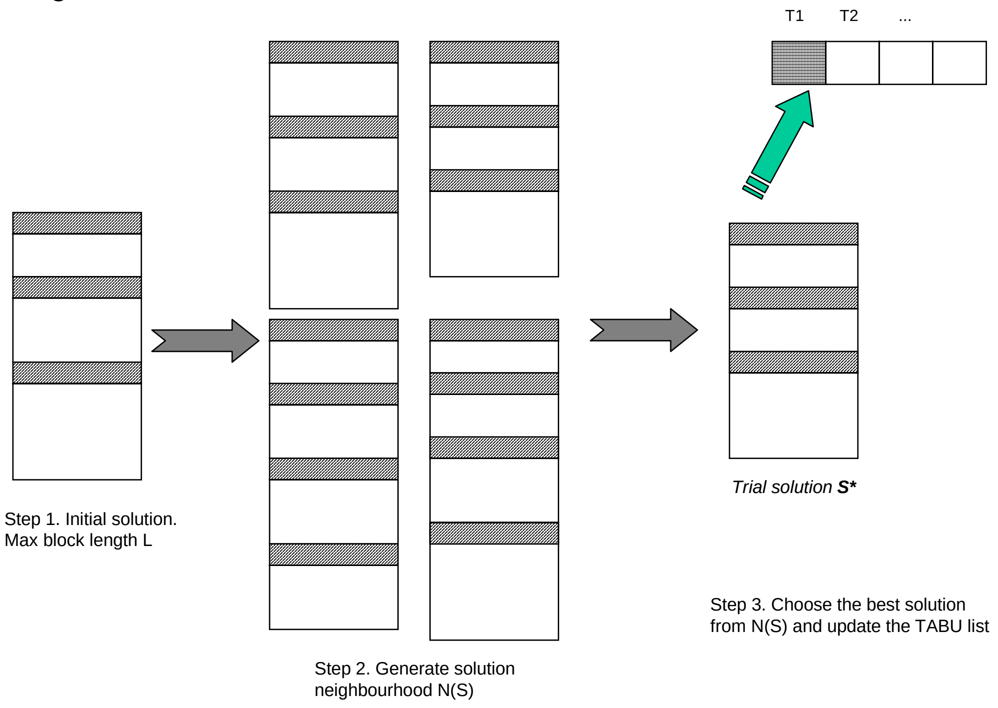

T1
T2
...
Trial solution **S***
Step 1. Initial solution. Max block length L
Step 2. Generate solution neighbourhood N(S)
Step 3. Choose the best solution from N(S) and update the TABU list

Fig. 7-15. *Tabu search based algorithm*

From the generated neighbourhood, the best feasible solution is chosen for the consideration as the next solution. If that solution happens to be in the *tabu list* the next best solution is chosen, thus giving the search process a chance to escape from the local optima and eventually reach a solution close to the best one. The search loop continues until *stopping criteria* are met - either the search returns to the point that was already passed or the number of empty iterations (i.e. iterations that do not result in finding a new best solution) exceeds the given constraint *E.*

7.4 Experimental results

7.4.1 Local search based algorithm for test length minimization

Experiments were performed on ISCAS'85 benchmark circuits. The results are depicted in *Table 7-1.* Test length *L* and the number of seeds *k* for different memory constrains M_{max} were calculated. For each circuit, experiments with three different memory constraints were carried out. The algorithm was executed several times for different starting points and lengths of iteration steps, and for each run of the algorithm, the solution and the number of itearations to reach it were found. The minimum and maximum number of iterations are depicted in columns 3 and 4. Column 5 shows lower bound of number of iterations for exhaustive search.For each experiment the exact minimum was reached (the exact minimum was determined based on the complete curves as shown on *Fig. 7-3*). In the presented experiments, the fixed length of pseudorandom pattern blocks (PPBs) was investigated. The algorithm can also be used for variable length PPBs.

Table 7-1. *Characteristics of the benchmark circuits*

Circuit	Constraint M_{max}	Min no. of iterations	Max no. of iterations	Exhaustive search	Test length *L*	No. of seeds *k*
1	2	3	4	5	6	7
c880	20	6	8	18	360	20
	15	7	13	30	450	15
	12	13	12	50	600	12
c1908	60	6	8	14	840	60
	40	9	11	45	1776	40
	35	11	14	51	1785	35
c2670	70	9	11	16	1120	70
	67	7	15	34	2278	67
	65	15	18	67	4288	64
c3540	60	5	10	16	896	56
	40	10	12	36	1224	34
	30	10	13	55	1650	30
c7552	90	7	8	17	1513	89
	75	9	13	42	3150	75
	70	11	13	58	4060	70

In *Table 7-2* the length of the test created by this *"store-and-generate"* approach is compared with the length of pure pseudorandom and pure deterministic test approaches. In columns 3 and 4 the lengths of the test with fixed and variable lengths, correspondingly, are given.

Table 7-2. *Comparison of "store-and-generate" with other test sequences*

Circuit	Pseudorandom test length	"Store and generate" test length		Deterministic test length	No. of seeds
1	2	3	4	5	6
c880	2694	600	208	56	12
c1908	4420	1785	1358	119	35
c2670	22682	4288	535	155	65
c3540	9631	1650	1095	211	30
c7552	24337	4060	1344	254	70

Experiments showed that the method converges well to the exact minimum whereas the number of iterations does not exceed 18. Compared to exhaustive search the speedup achieved by this method was from 1,5 up to 5,5.

7.4.2 Hybrid BIST length reduction using reseeding and test set compaction

Experiments were carried out on the ISCAS'85 and ISCAS'89 benchmark circuits for investigating the efficiency of the method for reducing the test length based on the proposed algorithm of calculating the hybrid test sets for each possible memory constraint. All ISCAS'89 circuits have been redesigned to include full scan path. For some tasks (like ATPG and fault simulation) tools from the Turbo Tester toolset [147] were used.

The results are presented in *Table 7-3* and *Table 7-4* where the length of the final solution under different memory constraints (different number of seeds) is depicted. Also, the described method has been compared to the hybrid BIST method proposed in [149] and to the "*store-and-generate*" approach with local search based algorithm [150], described in *Section 7.3.1* The method proposed in [149] was originally developed for multi-core systems but it can equally well be used also for individual cores. These techniques have one serious shortcoming. The deterministic and pseudorandom test sets are calculated in isolation and the sequences were applied sequentially. Therefore, it might happen that the test sequences could be substantially shortened if the tests are applied in different order. This issue was addressed in the approach where test set compaction was implemented.

Table 7-3. *Comparison of test length, ISCAS'85*

Circuit	Memory Constraint (bits)	Nr. of seeds	Test length		
			Reseeding and test set compaction	Hybrid BIST [149]	Store and generate [150]
c499	820	20	313	492	940
	1230	30	236	326	540
	1640	40	174	193	400
	2050	50	140	149	350
c1908	990	30	777	1318	1680
	1320	40	589	869	1240
	1650	50	444	735	1050
	1914	58	337	667	870
c2670	11650	50	238	598	6400
	12582	54	211	342	2538
	13048	56	171	311	1568
	13980	60	148	290	1080
c5315	2670	15	663	753	1290
	6230	35	273	451	560
	7476	42	173	299	504
	8188	46	134	268	368
c7552	19665	95	379	900	1140
	21114	102	333	500	714
	23805	115	210	334	460
	25668	124	165	192	372

Table 7-4. *Comparison of test lengths, ISCAS'89*

Circuit	Memory Constraint (test patterns)	Test Length (scan cycles)	
		Reseeding and test set compaction	Hybrid BIST [149]
s3330	79	1272	1347
	70	2121	1636
	67	2775	1861
s4863	29	87	154
	24	165	211
	19	255	298
s6669	10	102	211
	7	121	304
	4	143	626

In all the presented solutions the fault coverage was guaranteed at the same level as it was obtained by the deterministic generator.

As it can be seen from the results the method with reseeding and test set compaction can always find a test set that is shorter than the test set found using methods from [149] and [150]. The main explanation lies in the fact that the new method handles the deterministic and pseudorandom sequences together and the test sets are optimized using a fast optimization method, based on cumulative fault coverage figures. The implementation of the store and generate method [150] is not optimizing the length of the individual hybrid blocks (the reseeding blocks are generated in isolation) and therefore also the results are worse than result of hybrid BIST [149].

In order to illustrate the importance of the test set compaction, the lengths of reseeding blocks for the same arbitrary solution are compared in *Table 7-5*. As it can be seen the length of reseeding blocks is substantially longer when no compaction is used and with proposed method we can reduce the size of the hybrid block. This means that many pseudorandom test patterns that are not contributing to the overall fault coverage and are only prolonging the test sequences have successfully been removed.

Table 7-5. *Comparison of block sizes*

Circuit	Reseeding and test set compaction	Store and generate [150]
	Max block size	
c432	11	28
	7	14
	4	8
	3	5
	2	4
c499	23	47
	10	18
	6	10
	4	7
	3	4
c1908	30	56
	21	31
	13	21
	10	15
	9	13
c1355	10	31
	6	12
	4	7
	3	6

Circuit	Reseeding and test set compaction	Store and generate [150]
	Max block size	
c2570	15	128
	8	47
	7	28
	4	18
	2	13
c3540	6	15
	5	10
	3	7
	2	4
c5315	10	16
	5	12
	4	8
	3	5
c7552	10	12
	6	7
	3	4
	2	3

In *Fig. 7-16* and *Fig. 7-17,* more detailed results of some circuits are depicted. In *Fig. 7-16* the relationships between the memory constraint (nr. of seeds), the test length and the length of the shortest block are illustrated. As it can be seen from these charts the reduction of the number of seeds will increase the length of the blocks and consequently, also the test length will be increased.

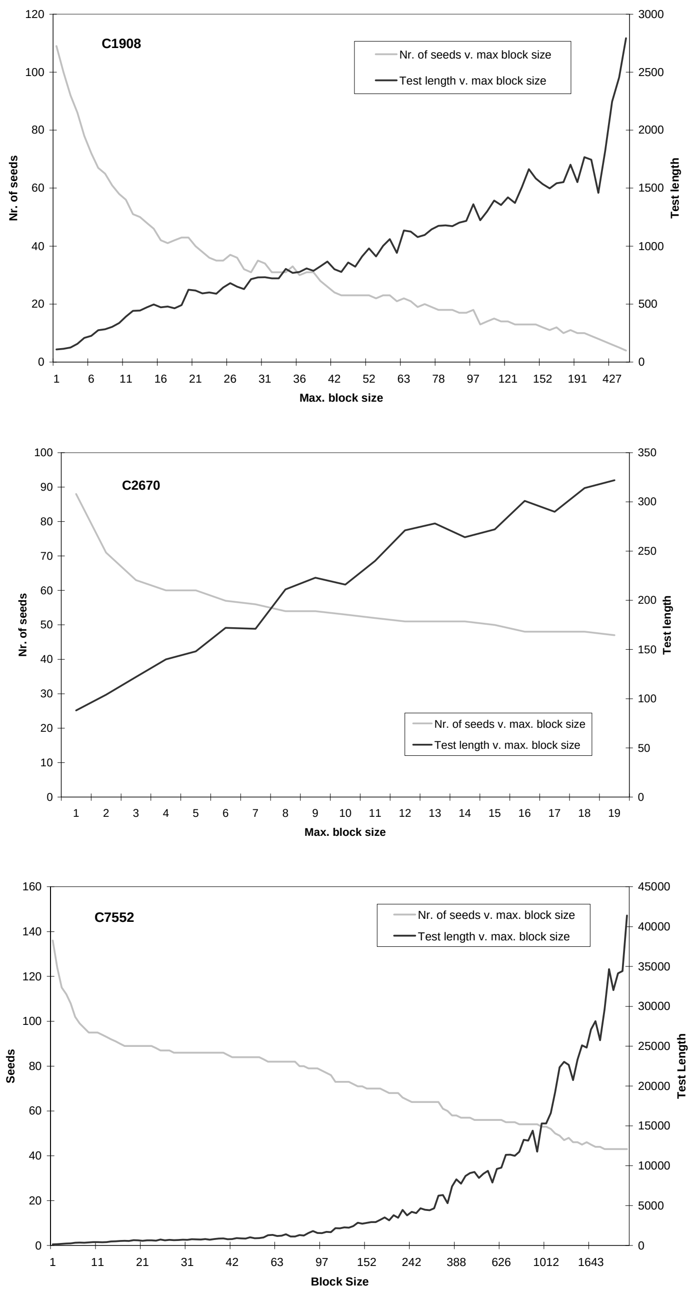
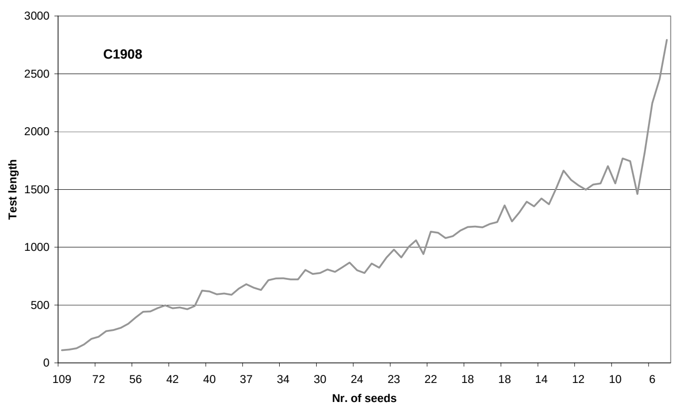
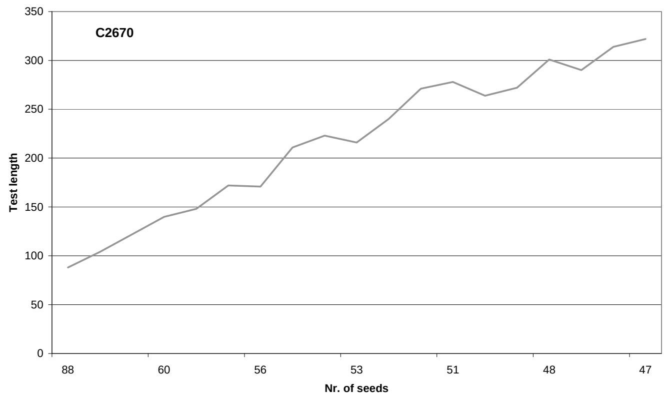


Fig. 7-16. *Relationships between number of seeds, maximal block size and total test length*





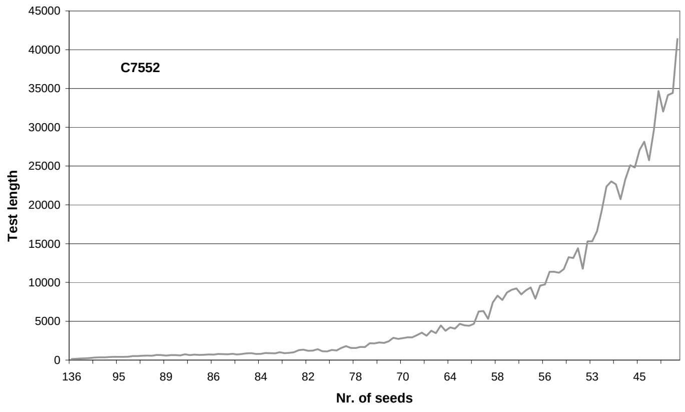


Fig. 7-17. *Relationship between the test length and number of seeds.*

In *Fig. 7-17* this relationship is illustrated in more straightforward manner, showing how the test length is increasing if the number of seeds is reducing.

The CPU times have not been included in these experimental results, as with the described method the entire solution space is being calculated, while the method described in [149] and store and generate approach [150] can find a single solution for a predefined memory constraint. This requires an optimization heuristic that would help to avoid calculation of the complete curves and thereafter the comparison of CPU times would be also possible.

In *Fig. 7-18* the total cost of the hybrid BIST solution for different number of seeds are depicted. The cost calculation is performed according to the formula:

$$C_{TOTAL} = C_{PR} + C_{DET} + C_{MEM} \approx \alpha_{PR}L + \alpha_{DET}S + \beta S$$

where C_{PR} is the cost of pseudorandom sequence, C_{DET} is the cost of applying deterministic patterns and C_{MEM} is related to the cost of storing deterministic test patterns in the memory; L is the length of the pseudorandom test, S is the number of deterministic vectors used. Parameters α_{PR} and β_{PR} can be used to align the application times of different test sequences. Constant β can be used to map the number of test patterns in the deterministic test sequence into the memory cost, measured in bits.

In the experiments described above, the parameters α_{PR}, α_{DET} and β had the same value. In *Fig. 7-18* the situation where the tester (or on-chip memory) has different value is illustrated. The curve *Total Cost* has been calculated so that $\alpha_{PR} = \alpha_{DET} = \beta = 1$. For the *Total Cost 2* $\alpha_{PR} = \alpha_{DET} = 1$, $\beta = 2$, and for *Total Cost 3* $\alpha_{PR} = \alpha_{DET} = 1$, $\beta = 6$. The values $\beta = 2$ and $\beta = 6$ mean that it is respectively two times and six times more expensive to store one deterministic pattern than to generate one pseudorandom pattern. These curves can be used for finding total cost minimums and to deduce from this information also the optimal number of seeds, which according to the presented algorithm also determines the optimal block sizes. Here it can be seen that the memory component influences the total cost of hybrid BIST solution significantly.

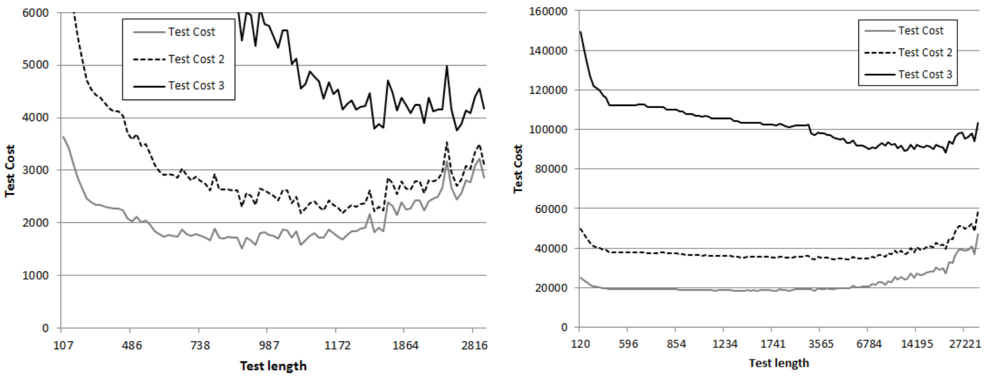


Fig. 7-18. *Comparison of the total test cost using different memory cost parameters.*

7.4.3 Tabu search based algorithm for optimization of memory-constrained hybrid BIST

Experiments were carried out on ISCAS'85 and ISCAS'89 benchmarks. ISCAS'89 circuits were redesigned in order to include full scan path. The results of these experiments are shown in *Table 7-6* and *Table 7-7*. In these tables, the test lengths under different memory contraints found by the Tabu Search based method and test lengths that are found by the reseeding with test set compaction method where the full curve is calculated, are depicted.

Table 7-6. *Comparison of test lengths and number of calculations for ISCAS'85 benchmarks*

Circuit	Memory Constraint (bits)	Tabu Search based optimization		Reseeding and test set compaction	
		Test Length	Nr. of calculations	Test Length	Nr. of calculations
c499	2050	140	15	140	144
	1640	179	15	174	
	1230	236	14	236	
	820	335	11	313	
c1908	1980	392	12	337	159
	1650	444	11	444	
	1320	600	8	589	
	990	778	13	777	
c2670	13980	183	17	148	181
	12815	201	15	191	
	11650	252	11	238	
	10485	366	13	361	
c5315	8010	157	10	141	144
	6230	273	11	273	
	4450	433	9	433	
	2670	691	22	663	
c7552	23805	210	34	210	167
	21735	333	31	333	
	19665	408	20	379	
	17595	790	9	790	
Average		**370**	**15**	**358**	**159**

Table 7-7. *Comparison of test lengths and number of calculations for ISCAS'89 benchmarks*

Circuit	Memory Constraint (bits)	Tabu Search based optimization		Reseeding and test set compaction	
		Test Length	Nr. of calculations	Test Length	Nr. of calculations
s3330	4000	637	24	637	120
	3800	730	22	694	
	3600	869	20	861	
s4863	2205	79	15	64	144
	1470	99	17	87	
	980	202	13	202	
s6669	2158	56	5	43	95
	1162	76	7	75	
	415	123	16	114	
Average		**319**	**15**	**309**	**120**

As it can be seen from the presented results, the length of the test found by the *tabu search* based method is not always exact minimum but nevertheless its length is quite close to the shortest test under the given memory constraint found by exhaustive method. However, the number of calculations needed to find the solutions significantly smaller when compared to the results of the reseeding and compaction approach.

It can be seen that the *tabu search* based method allows to find the close to optimum solution for ISCAS’85 benchmarks in average 10 times faster than the exhaustive search, whereas the calculated average test length is only 3,12% worse than the average exact optimum. For ISCAS’89, the method is in average 7,75 times faster and, the calculated average test length is 3,39% worse.

7.5 Conclusions

1. In this chapter, several approaches to optimization of the hybrid BIST with reseeding have been described. The purpose was to develop methods for finding an optimal balance between the number of precomputed deterministic test vectors and pseudorandom test sequences taking into consideration the constrained memory size (number of LFSR seeds) and guaranteeing maximum achievable fault coverage.

2. The BIST synthesis method is based on the test set compaction using cumulative fault coverage of the hybrid test sequences. The method provides a possibility to find a memory constrained test solution and to use it for constructing an optimal test solution for the entire system.

3. Two methods were developed for optimization of the BIST sequence, the *local search* method and *tabu search* method.

4. In the local search method, two different scenarios were considered, where the length of the pseudorandom blocks was either constant or variable. In the latter case the total length of the test was considerably smaller. However, in the case of the variable block length the memory cost would increase, since for each seed the length of the block should be stored as well.

5. The second described method incorporates the ideas of the test set compaction approach with an approach known as *tabu search* which is targeting optimization of the problems with multiple local optima. The experiments showed that the approach allows to find the solution quite close to the shortest test under the given memory constraint. This method does not require the calculation of the whole solution space and therefore gives results much faster than the exhaustive search method.

Chapter 8 BIST cost minimization and testability analysis for industrial designs

In this chapter, the methods of minimization of cost for hybrid BIST with reseeding are developed and experimental results are presented for the biosignal processors and also ISCAS'89 family benchmarks. Experiments are carried out with the industrial design developed at the Centre for Integrated Electronic Systems and Biomedical Engineering CEBE for acquiring and measuring bioimpedance signals. The testability of these designs is thoroughly analyzed. Different testability characteristics are taken into consideration and the impact of alterations made to the original design are investigated.

8.1 The role of the cost components of the Hybrid BIST

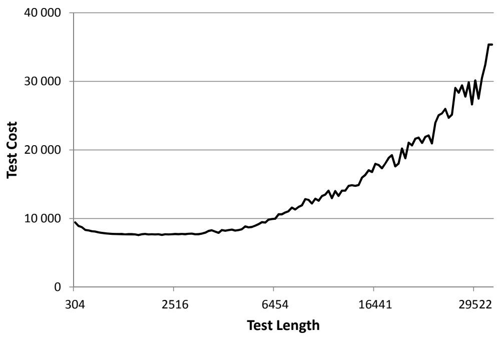
The function to calculate the cost of the hybrid BIST solution was thoroughly described in *Chapter 6*. Let us recall that the total cost of the hybrid BIST C_{TOTAL} can be defined as follows:

$$C_{TOTAL} = C_{GEN} + C_{MEM} \approx \alpha L + \beta S$$

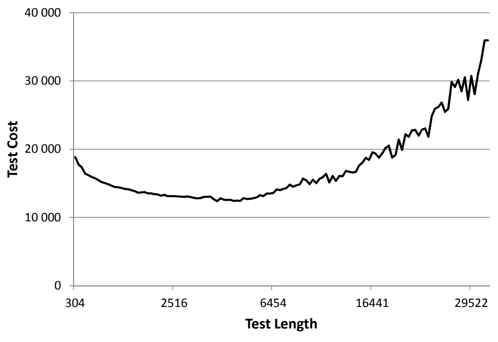
where C_{GEN} is the cost related to the time for generating L pseudorandom test patterns (number of clock cycles), C_{MEM} is the memory cost for storing S precomputed test patterns needed to improve the results of the pseudorandom test sequence [126]. α and β are the constants to map the test length and memory space to the costs of the two parts of the test solutions to be mixed.

It is obvious that the correspondence values of α and β have a great influence on the total cost of the applied test. When $\alpha >> \beta$ it means the time used by the test has more impact on the total cost of the test than the memory needed to store the seeds that are used in the test, that is – the more time the test application takes, the more expensive it becomes. On the other hand $\alpha << \beta$ means that the memory component of the total cost is more significant – the more seeds need to be stored, the more expensive will be the test.

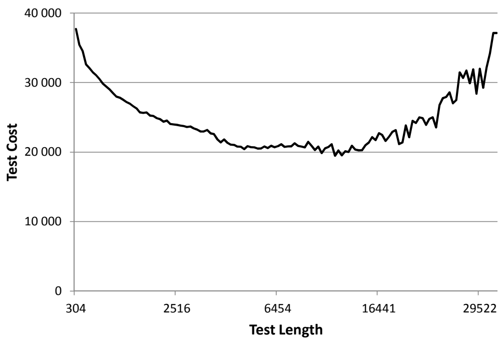
Fig. 8-1 represents the situation where the memory component β (which represents the cost of storing one test pattern in the memory) has different values for the ISCAS'89 circuit `s13207`.



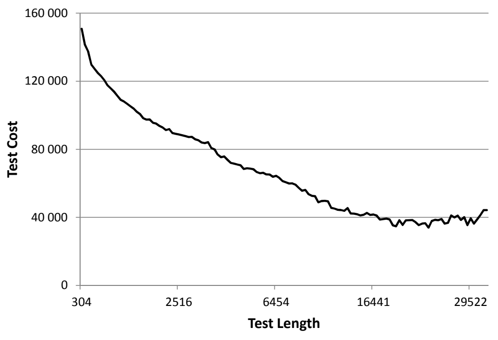
a) β = 25



b) β=75



c) β=100



d) β=500

Fig. 8-1. *The cost curve for ISCAS'89 benchmark* ***s13207*** *for different values of memory component*

Experiments showed that for the ISCAS'89 benchmarks the cost function becomes nonconvex in case when the value of time component α is 1 and the value of memory component β is between 25 and 750, that is when the memory component of the total cost is 25 to 750 times more "expensive". When memory component β is less than 25 the cost function tends to be linearly dependent from the time component – the longer time is needed for applying the test, the higher the value of the cost function is, that is – the more expensive the test becomes. When memory component β is more than 750 the cost function tends to be linearly dependent from the memory component – the more expensive it is to store one precomputed test pattern, the higher value of the cost function is.

In the area where the cost function is nonconvex, suitable optimization algorithms are needed to perform fast calculations in order to find the solution (the combination of seeds and the corresponding pseudorandom test blocks) with minimal cost taking into consideration the time and memory component.

8.2 Tabu Search based algorithm for minimizing the cost of reseeding

As the as the test cost function is nonconvex in certain range, in order to find the minimum value and reduce the number of calculations needed, the *tabu search* based method for the hybrid BIST with reseeding (described in *Chapter 7*) was implemented, as this approach allows escaping local optima while performing the search.

The algorithm is described in *Fig. 8-2*.

The algorithm starts from generating the set of deterministic patterns *DT*. Then, the pseudorandom blocks of length L_0 (that denotes the block length of the *initial solution*) are generated so that the patterns in *DT* are used as seeds. Thereafter, all the created blocks PR_i are minimized – the patterns that do not contribute to the fault coverage of the block are removed. After the minimization is performed, the blocks are reordered in increasing order and test pattern optimization algorithm is applied, that takes into account cumulative fault coverage of the blocks and removes the patterns that do not contribute to overall fault coverage (see details in *Section 7.2*). The remaining test set that consists of precomputed patterns (seeds) and the compressed pseudorandom sequences, is the *initial solution* S_0. The *cost* of the solution is calculated, according to the cost calculation function shown in *Section 8.1*.

Initial solution S_0 is the starting point of the algorithm. The values are set for current solution $S^{CURR}=S_0$, best solution $S^{BEST}=S^{CURR}$. Also, the *tabu list* is initialized. After that, the loop starts to find optimal solution.

First, the neighbourhood N of the current solution is generated (neighbouring solutions are defined by the length of the pseudorandom block). For neighbours, the solutions are calculated (by generating the test set consisting of deterministic seeds and corresponding pseudorandom test pattern blocks). Then, the *costs* of all solutions in the generated neighbourhood are calculated. The solution S^{TRIAL} with *best cost* in the neighbourhood is chosen and compared with the *tabu list*. If the solution S^{TRIAL} is not in *tabu list*, it is chosen as the next *current solution* S^{CURR}. If the solution is *tabu* then the next best solution from the neighbourhood is chosen. The chosen current solution S^{CURR} is compared to best solution seen so far S^{BEST} and if the cost of S^{CURR} is better than the cost of S^{BEST} then S^{CURR} becomes the *best solution* S^{BEST}. The loop continues until the stop criterion is met – the search return to an already visited solution or there in no improvement in S^{BEST} for defined number of steps.

Tabu Search based algorithm

```
-   Generate set of deterministic test patterns DT
-   Choose the block length $L_0$ of the initial solution
-   Generate the initial solution $S_0$ based $L_0$
    //generate pseudorandom blocks with length $L_0$ for all
    //patterns from the set DT
    //apply test set compaction removing the patterns that
    //do not contribute to the overall FC
-   Calculate Cost ($S_0$)
-   Declare current solution $S^{CURR}=S_0$
-   Declare best solution $S^{BEST} = S^{CURR}$
-   Initialize tabu list TL
-   repeat (until stopping criterion met)
        –   Generate neighbourhood N of $S^{CURR}$
        –   Calculate the solutions in N, apply test set compaction
            //generate pseudorandom blocks of length $L^{N1}$, $L^{N2}$ etc.
            //apply test block compaction based on FC
            //rearrange the blocks
            //minimize the number of blocks
            //apply test set compaction
        –   Calculate the Cost of solutions in N
        –   Find best Cost($S^{TRIAL} \in N$)
        –   loop (until best solution $S^{TRIAL} \in N$ is found)
                –   if $S^{TRIAL}$ is not in TL, then
                        –   Declare $S^{CURR}=S^{TRIAL}$
                        –   Update TL
                –   else
                        –   find the next best $S^{TRIAL}$
        –   if Cost ($S^{CURR}$)< Cost($S^{BEST}$)
                –   Declare $S^{BEST}=S^{CURR}$
-   done
```

Fig. 8-2. *Tabu search based algorithm for reducing number of calculations*

The resulting S^{BEST} is the solution found by the algorithm.

In the presented approach, the following algorithm parameters were chosen:

initial solution S_0: 0,3*length of the deterministic test
size of the neighbourhood: Size (N) = 10
tabu list TL length: Length (TL) = 7
stopping criterion: 10 unresulative iterations

The experimental results of using the method are described and discussed in *Section 8.6*

8.3 Simulated annealing based algorithm for minimizing the cost of reseeding

As *simulated annealing* possesses the "*hill climbing*" property needed for escaping local optima of nonconvex solution space, an optimization algorithm for reducing the number of calculations in order to find minimal cost of hybrid BIST with reseeding was also implemented using an approach of *simulated annealing*. The description of the algorithm is shown in *Fig. 8-3*.

Simulated annealing based algorithm

```
-   Generate set of deterministic test patterns DT
-   Choose the block length $L_0$ of the initial solution
-   Generate the initial solution $S_0$ based $L_0$
    //generate pseudorandom blocks with length $L_0$ for all
    //patterns from the set DT
    //apply test set compaction removing the patterns that
    //do not contribute to the overall FC
-   Calculate Cost ($S_0$)
-   Declare current solution $S^{CURR}=S_0$
-   Declare new solution $S^{NEW}=S_0$
-   Declare best solution $S^{BEST} = S^{CURR}$
-   Set initial temperature T
-   repeat
        –   repeat
            –   Generate neighbour $S^{NEW}$ of the current solution $S^{CURR}$
                //generate pseudorandom blocks of length $L^N$
                //apply test block compaction based on FC
                //rearrange the blocks
                //minimize the number of blocks
                //apply test set compaction
            –   Calculate Cost ($S^{NEW}$)
            –   Calculate $\Delta$Cost = Cost ($S^{CURR}$)-Cost($S^{NEW}$)
            –   if ($\Delta$Cost<0) then
                –   Declare $S^{CURR}=S^{NEW}$
                –   if Cost($S^{CURR}$)<Cost($S^{BEST}$) then
                    –   Declare $S^{BEST}=S^{CURR}$
            –   else
                –   if (random_number < -$\Delta$Cost/T)
                    –   $S^{CURR}=S^{NEW}$
            –   reduce the value of M
        –   until M=0
        –   reduce the value of T
-   until max number of empty Markov's chains achieved
-   $S^{BEST}$ is the solution
```

Fig. 8-3. *Simulated annealing based algorithm for reducing the number of calculations*

The algorithm starts from generating the set of deterministic patterns DT. Then, the pseudorandom blocks of length L_0 (that denotes the block length of the *initial solution*) are generated so that patterns in DT are used as seeds. Thereafter, all the created blocks PR_i are minimized – the patterns that do not contribute to the fault coverage of the block are removed. After minimization is performed, the blocks are reordered in increasing order and test pattern optimization algorithm is applied, that takes into account cumulative fault coverage of the blocks and removes the patterns that do not contribute to overall fault coverage (see details in *Section 7.2*). The remaining test set that consists of precomputed patterns (seeds) and the compressed pseudorandom sequences, is the *initial solution* S_0. The *cost* of the solution is calculated, according to the *cost function.*

Solution S_0 becomes the starting point of the algorithm. It is also declared to be the *current solution* S^{CURR} and the *best solution* S_{BEST} (solution with the best cost seen so far). The important parameter of *simulated annealing – the initial temperature* T is set. After that, the calculations are performed in a loop – the heart of the algorithm, the *Metropolis procedure* which simulates annealing of metal at given temperature. In this loop, a local neighbour is chosen from the neighbourhood of S^{CURR}. For the chosen neighbour (determined by the block length L_N) the solution S^{NEW} is generated (by generating the test set consisting of deterministic seeds and corresponding pseudorandom test pattern blocks). The *cost* of the generated solution is calculated. Now, if the *Cost*(S^{NEW}) is better than the Cost(S^{CURR}), the new solution is accepted, thus S^{CURR}=S^{NEW}. In case *Cost*(S^{NEW}) is worse than Cost (S^{CURR}), the solution S^{NEW} might still be accepted on *probabilistic basis* (here, the "*hill climbing*" property of *simulated annealing* is realized). The random number is generated in range 0 to 1. In case this generated number is smaller than $e = \frac{-\Delta Cost}{T}$ where $\Delta Cost$ is the difference of costs and T is the current temperature, the *uphill solution* is accepted. The loop is performed M times (the value of M represents the amount of time for which annealing must be applied at given temperature T). After that, the value of temperature T is reduced (based on cooling rate defined by α_{SA}) and the value of M that defines the duration of the one annealing loop is also reduced. The loop is repeated with decreasing values of T and M until the given number of unresultative Markov's chains is achieved.

In the current approach, based on empirical studies, the following parameters for *simulated annealing* were chosen:

initial solution: 0,3*length of the deterministic test
initial temperature: T = 100
constant to determine cooling schedule: α_{SA} = 0,9
initial value of length of Metropolis procedure: M = 15
constant to reduce value of M in a controlled manner: β_{SA} = 1,1

The experimental results of using the method are described and discussed in *Section 8.6*

8.4 Benchmark circuits family based on biosignal processor designs

In biomedical engineering, bioimpedance is a term used to describe the response of a living organism to an externally applied electric current. Measurement of electrical bioimpedance enables to characterize tissues and organs, to get diagnostic images, etc.[152]. Multi-channel data-acquisition devices are used often in biomedicine to measure the properties of organs and tissues. The main reason is the fact that the useful information is hidden under background signals generated by the normal body activity [153][154]. An example would be respiration generated noise when measuring heart activities. Electrical bioimpedance

$$\dot{Z}=R+jX$$

is determined by measuring of voltage response V to the excitation current I flow through the tissue or organ, and calculated as

$$\dot{Z}=V/I$$

The impedance of tissues and organs is measured between the electrodes having different locations. Multisite and multifrequency bioimpedance information has a great diagnostic value [156][157].

In the following, the DSP (digital signal processing)-based solution for a multi-frequency measurement unit prototype has been described.

The basic architecture of the digital multichannel bioimpedance analyzer (DMBA) is shown in *Fig. 8-4* [153]. The parameters of the response receiving part (multiplexer and signal analyzer) are defined by the use of single analog-to-digital converter (ADC) for multichannel measurements. For instance, practical measurement on body surface (thorax EBI measurements) with 8 excitation sources and also 8 response signals require operating in the frequency range of interest between 30 kHz and 100 kHz. The task can be accomplished using single ADC with at least 10MHz multisampling rate. The resolution must be between 18 to 20 to represent low (0.01% range) impedance changes adequately [158].

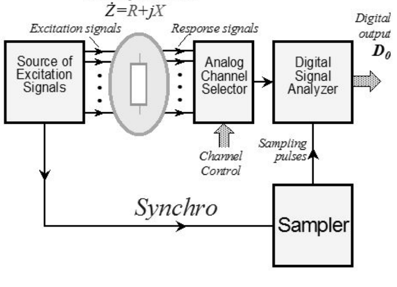


Fig. 8-4. *A simplified block diagram of DMBA*

The heart of the electronic test arrangement (prototype) is the Field Programmable Gate Array (FPGA) Spartan^{TM}-3 from Xilinx. The FPGA handles input channel selection, sampling pulse generation, preamplifier gain control, compensating voltage code generation, reading samples from ADCs (analog-to-digital converters). The functional block diagram of the FPGA unit is shown on *Fig. 8-5.*

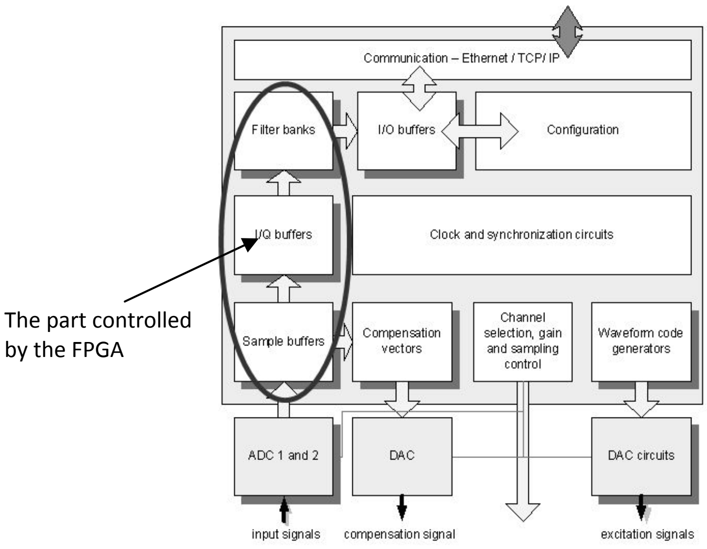


Fig. 8-5. *Functional block of the FPGA unit with I/O connections and peripheral components*

There has been a number of different modifications developed in order to compare different architectural solutions of this functionality.

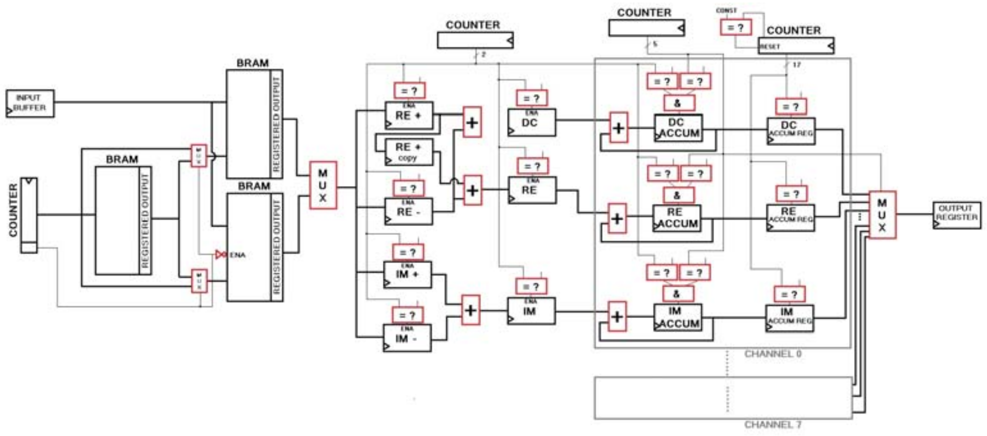


Fig. 8-6. *Schematic of the industrial design*

Fig. 8-6 shows an example block diagram of the design – reconfigurable multi-channel multi-frequency application specific signal processor (DSP), which was designed for acquiring and measuring bioimpedance signals.

The acquired data is first sampled, then sorted and finally processed. The processing includes calculating the values that are needed to calculate the bioimpedance of the tissue. The sampling order is controlled by a programmable decoder, implemented in block RAM. The sampled data is then placed into one of two on-chip memories, which are working in parallel – when one is collecting the incoming data, the other is sending the previously collected values for processing. Data is accumulated and accumulation registering is performed. After that, data from registered accumulators are multiplexed to a single output register.

The general structure of the 8-channel signal processor for bioimpedance measurements is shown in *Fig. 8-7*. During the design process, alterations were made to both preprocessor part of the design and the integrator part of the design, resulting in eight different configurations performing the same function: `8a,8b,8be,8bk,8bs,8c,8d` and `8de`. The goal of the research in this thesis was to investigate how different structural implementations would impact on the testability of the design, and to find out which properties of the design will cause worse testability.

Fig. 8-8 shows which successive changes were introduced into the designs. Design *8a* was the original version with eight data channels, other designs are different alterations.

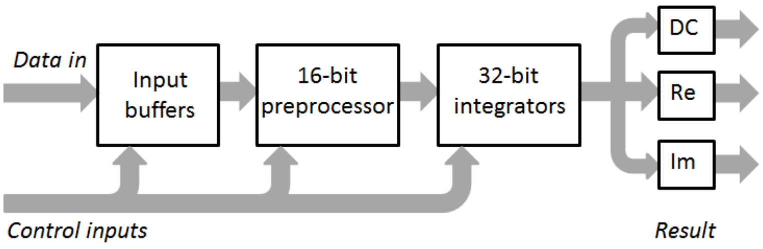


Fig. 8-7. *General structure of the biosignal processor*

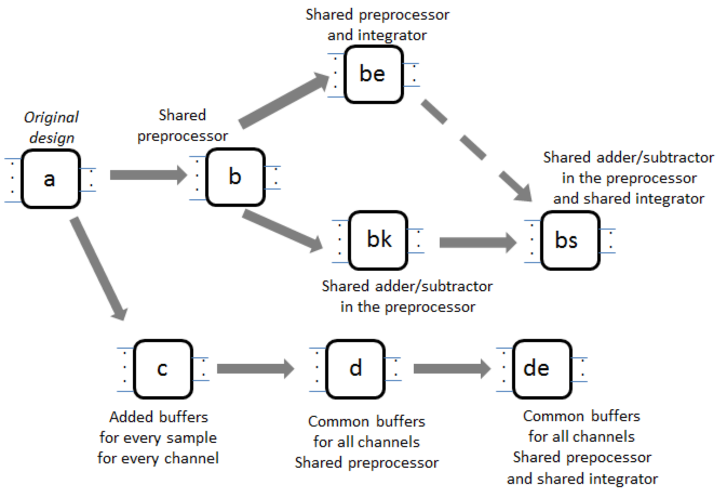

Shared preprocessor and integrator
be
Original design
a
Shared preprocessor
b
Shared adder/subtractor in the preprocessor and shared integrator
bk
bs
Shared adder/subtractor in the preprocessor
c
d
de
Added buffers for every sample for every channel
Common buffers for all channels Shared preprocessor
Common buffers for all channels Shared prepocessor and shared integrator

Fig. 8-8. *Overview of the benchmark designs*

8.5 Testability analysis of the benchmark family

Testability analysis of different configurations of biosignal processor design was performed by using deterministic and pseudorandom test pattern generators, fault simulator and by using the algorithms for hybrid BIST optimization developed in this thesis. Several testability characteristics were analyzed: the test length achieved and the needed time for deterministic test pattern generation, the time needed for fault simulation and for the pseudorandom test generation, the hybrid BIST length and the calculated optimal test cost of hybrid BIST. The results are presented in *Table 8-1.*

Table 8-1. *Testability characteristics of 8-channel signal processors*

Design	Deter-ministic test length	Deter-ministic test generation time, s	Fault simulation time, s	Random test generation time, s	HyBIST length	HyBIST optimal cost
8a	1364	47	13,7	1408	23 038	197 823
8b	1201	34	11,8	1130	18 540	138 324
8be	995	114	27,9	2784	14 202	104 474
8bk	1288	35	11,3	1129	17 497	144 876
8bs	1186	296	69,0	7095	14 086	113 038
8c	1320	75	15,5	1583	35 641	224 121
8d	1394	62	16,6	1647	32 610	209 384
8de	1096	112	33,4	3344	33 968	162 557

The most significant changes in testability characteristics because of the design modifications are highlighted in *Table 8-2*.

Table 8-2. *Changes in testability characteristics because of the design modifications*

Design	Deter-ministic test length	Deter-ministic test generation time, s	Fault simulation time, s	Random test generation time, s	HyBIST length	HyBIST optimal cost
8a → 8b	↓	↓	↓	↓	↓	↓
8b → 8be	↓	↑	↑	↑	↓	↓
8b → 8bk						
8bk → 8bs	↓	↑	↑	↑	↓	↓
8a → 8c		↑	↑	↑	↑	↑
8c → 8d		↓			↓	↓
8d → 8de	↓	↑	↑	↑		↓

The different design implementations are characterized by different levels of sharing of resources such as input buffers, preprocessing units, adders and subtractors in preprocessors, and integrators. Sharing of the resources was accompanied by introducing additional multiplexers and control circuits which in their turn increased the number of reconvergent fan-out branches in the topology of the circuits. A rough estimation of the number of convergent control signals is given in *Table 8-3*.

Table 8-3. *Modifications in the different benchmark designs*

Design	Number of reconvergent control signals			Modification made in the designs
	Pre-processor	Integrator	Total	
8a	32	64	96	Initial design
8b	32	64	96	Shared preprocessor
8bk	64	64	128	Shared preprocessor and adder/subtractor
8c	32	64	96	Initial design with input buffers
8d	32	64	96	Shared preprocessor
8be	32	512	544	Shared preprocessor and integrator
8de	32	512	544	Shared preprocessor and integrator
8bs	64	1536	1600	Maximum sharing of resources

The transition from 8a to 8b was the replacement of eight different channels in preprocessing part of the circuit by one common channel, thus the redundancy was removed, resulting in shared preprocessor. This change resulted in improvement of all the testability characteristics under consideration. The best improvements were seen in reduction of test generation time (generation of deterministic test 1,4 times and generation of pseudorandom test 1,25 times faster). Fault simulation time was 1,16 times faster. Also, the optimal cost and the test length of the hybrid BIST (with reseeding) significantly improved – one of the reasons is smaller number of inputs in 8b which results in the less cost of the memory component of the hybrid BIST.

The transition from 8b to 8be was the replacement of eight channels of integrators with one channel. Multiplexers were added to the inputs of adders in the integrator. As it can be seen from the presented results, deterministic test length improved - it was 1,2 times shorter which can be explained by the reduction of the circuit because of sharing a single channel instead of using eight different channels. On the other hand, the time needed for deterministic test generation was 3,35 times higher because of increasing of the number of reconvergent control signals in the circuit from 94 to 544, which causes higher number of backtracks during search for consistent solutions. Also, fault simulation time became 2,36 times slower, and the time needed for pseudorandom test generation was 2,46 times higher. This is explained by the use of exact critical path tracing algorithm [159] used for fault simulation which is highly sensitive to the number of reconvergent control signals. Since pseudorandom test generation uses the same fault simulator, the test generation time consequently as well increases as well. The cost of the hybrid BIST was improved due to the smaller number of deterministic vectors needed.

The transition from 8b to 8bk was using one adder/subtractor and additional multiplexers in the preprocessor part. The increase of the reconvergent control signals (from 96 to 128) did not significantly influence the testability of the circuit.

The transition from 8bk to 8bs combined the preprocessor part of the design 8bk and the integrator part of the design 8be. As it can be seen from the *Table 8-3*, the number of reconvergent control signals increased drastically (128 for 8bk and 1600 for 8bs). *Table 8-2* shows worsening of the testability characteristics regarding test generation and fault simulation: the time of deterministic test generation became 8,45 times longer, the time of fault simulation 6,1 times longer, and the time of random test generation became 6,28 times longer. On the other hand, because of the reduction in circuit size, the length of deterministic test set became slightly shorter (1,08 times). The length of optimal hybrid BIST was 1,24 times shorter and optimal cost was 1,28 times smaller due to the smaller number of seeds needed to achieve the best possible fault coverage.

The transition from 8a to 8c resulted in adding additional buffer registers to the inputs of input buffers. The eight channels of data remained. In *Table 8-2* it can be seen that test generation and fault simulation time related characteristics have become worse: generation time of deterministic test became 1,59 times longer, generation time of random test became 1,13 times longer and fault simulation time became 1,13 times longer. This worsening of indicators can be explained by the increase of the number of reconvergencies because of adding control signals for addressing the buffer registers. The test length did not change because of the circuit size remained the same. The length of hybrid BIST sequence test became 1,54 times longer, and the cost of Hybrid BIST was bigger for `8c` due to the bigger number of inputs (buffer registers).

In the transition from `8c` to `8d` eight channels of the preprocessor were removed and replaced with a single channel. Multiplexers were added to the inputs of the preprocessor. The characteristics that changed most significantly were deterministic test generation time (became shorter) and the length of the optimal hybrid BIST became slightly shorter, similarly as in the case of "from `8a` to `8b`"

In transition from `8d` to `8de`, the eight channels of integrator were replaced by a single channel and also some multiplexers were added. This change resulted in a bigger number of reconvergent control signals (*Table 8-3*) and, consequently, also in longer times for test generation and fault simulation: test generation time for deterministic test was 1,81 times longer and test generation time for random test was 2,03 times longer. Fault simulation time was 2,01 times longer. The length of deterministic test set was 1,27 times shorter and the cost of the optimal hybrid BIST test was 1,28 times smaller (due to the smaller number of seeds needed). This case affected the testability characteristics in the similar way as in the case "from `8b` to `8bk`"

8.6 Experimental results of BIST optimization

Experiments were performed on ISCAS'89 benchmarks and on the different configurations of the biosignal processor design to investigate the efficiency of the described optimization algorithms. The circuits were redesigned to include full scan path.

The results for ISCAS'89 benchmarks are presented in *Table 8-4*, where the number of calculations needed to find the minimum value of the cost function is shown for all three cases – exhaustive calculation, *tabu search* based algorithm and *simulated annealing* based algorithm. In all solutions the highest fault coverage was guaranteed.

For test cost was defined by the formula, which takes into account the deterministic part of the test sequence:

$$C_{TOTAL} = C_{GEN} + C_{MEM} \approx \alpha L + \beta S$$

where L is the total length of the pseudorandom sequences of the test and S is the number of precomputed deterministic test patterns, stored in the memory.

The values of parameters were chosen as follows:

α = *1* (the cost of applying one test pattern)
β = *3*B* (the cost of storing one deterministic test in memory; B is the number of bits of the input vector, 3 is the cost of storing one bit in the memory)

Table 8-4. *Comparison of Tabu search and Simulated annealing based approaches for ISCAS'89*

Circuit	s3271	s3330	s5378	s13207	s15850	s3982	Average
Test Length	314	950	1896	1733	606	3982	
No. of seeds	24	83	55	195	208	484	
Length of the PR block	21	29	52	20	7	22	
Real best cost	938	4270	3821	7778	3518	17500	
Best cost found by SA	938	4270	3821	7778	3518	17500	
Best cost found by TS	950	4300	3821	7778	3558	17500	
No. of calculations	3000	1500	2500	2000	2000	500	1917
No. of calculations by SA	81	49	53	71	52	141	75
No. of calculations by TS	24	25	20	35	34	106	41

For calculating the test cost of the biosignal processor design, the values of the constants were chosen as follows:

α = *1* (the cost of applying one test pattern)
β: number of bytes needed to store one deterministic test in memory

Table 8-5(a) *Comparison of TS and SA based approaches for biosignal processor*

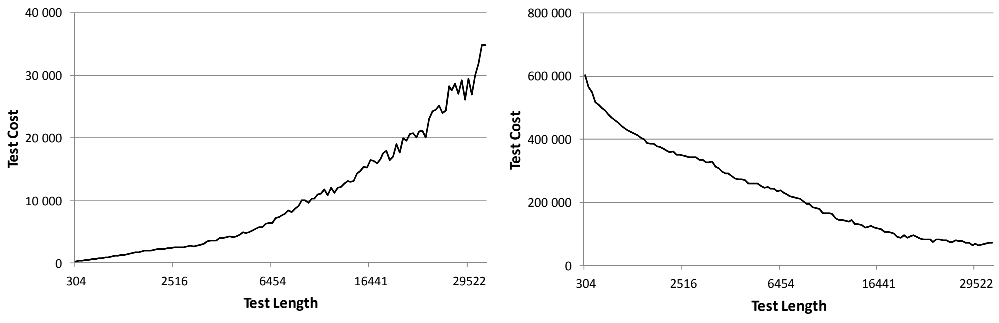
Design	8a	8b	8be	8bk	Average
No. of primary inputs	2528	1744	1744	1744	
No. of nods	31070	23785	26263	23656	
Test Length	28038	18540	14202	17497	
Fault coverage, %	98,69	98,31	98,19	98,25	
No. of seeds	539	552	416	587	
Length of the PR block	174	116	89	97	
Real best cost	197823	138324	104474	144876	
Best cost found by SA	197823	138324	104474	144876	
Best cost found by TS	200382	142037	106725	148805	
No. of calculations	1500	1500	1500	1500	1500
No. of calculations by SA	103	65	73	51	73
No. of calculations by TS	54	31	29	21	34

Table 8-5(b) *Comparison of TS and SA based approaches for biosignal processor*

Design	8bs	8c	8d	8de	Average
No. of primary inputs	1744	3043	2707	2707	
No. of nods	26868	34314	33577	35600	
Test Length	14086	35641	32610	33968	
Fault coverage, %	97,85	97,97	98,83	98,71	
No. of seeds	456	496	523	424	
Length of the PR block	66	231	210	121	
Real best cost	113038	224121	209384	162557	
Best cost found by SA	113698	224121	210359	162557	
Best cost found by TS	120848	224123	212270	163423	
No. of calculations	1500	1500	1500	1500	1500
No. of calculations by SA	51	134	45	62	73
No. of calculations by TS	32	43	37	42	39

As it can be seen from the presented results, *simulated annealing* based approach can in almost all cases find the exact solution with the smallest value of the cost function. The *tabu search* based approach needs less calculations but does not always guarantee the best possible solution. However, if the solution found by *tabu search* based approach is not really the best possible, it is still quite close to minimal solution. For both ISCAS'89 benchmarks and the biosignal processor designs, *simulated annealing* based method found the solution in average about 20 times faster and the *tabu search* based method found the solution in average about 40 times faster.

Additional experiments were carried out in order to determine if the described approaches are also applicable in cases where the cost function loses its parabolic nature. Examples of such cases are presented in *Fig. 8-9*.



a) *β=1* b) *β=2000*

Fig. 8-9. *The cost curve for ISCAS'89 benchmark* ***s13207*** *for extreme values of memory component*

Table 8-6. *The search results for extreme values of β for benchmark s**13207***

	β=1	**β=2000**
No. of seeds	304	19
Length of the PR block	1	1565
Real best cost	304	64040
Best cost found by SA	379	73179
Best cost found by TS	379	76084
No. of calculations	1500	1500
No. of calculations by SA	38	511
No. of calculations by TS	40	175

Experiments showed that even in these extreme cases, the algorithms are applicable and capable of finding the solution that is close to optimal. *Table 8-6* shows the results for extreme case for circuit `s13207` (also illustrated on *Fig. 8-9*) – in one case cost of storing one deterministic test pattern in memory $\beta = 1$, and in the other case the cost of storing one deterministic test pattern in memory $\beta = 2000$. As it can be seen, in case of both approaches the search time is similar in case of "*cheap*" memory and tends to be quite long in the case of "*expensive*" memory.

8.7 Conclusions

1. Two algorithms for cost minimization of hybrid BIST with reseeding based on *tabu search* and *simulated annealing* were presented, described and investigated.

2. The impact of constants α and β in the cost function of the hybrid BIST with reseeding was investigated, where α represents the cost of applying one test vector thus allowing to measure the time component of the total cost and β represents the cost of storing one precomputed test pattern (seed for LFSR) thus allowing to measure the memory component of the total cost.

3. Experimental research was carried out both on the ISCAS'89 and on the industrial circuits which represented a family of biosignal processors. In all processors the same functions were implemented, however by different circuit architectures and sharing of resources.

4. Experiments showed that the cost minimization based on *simulated annealing* and *tabu search* are suitable for solving the problem, giving the minimal or near minimal solution within reasonable time cost – number of times faster than in case where it is needed to calculate the whole cost curve

5. Experiments also showed that *simulated annealing* based solution tends to be more exact but takes more time than *tabu search* based solution; *tabu search* based solution will give less exact results but works faster than *simulated annealing* based approach.

6. In cases, where the cost function tends to lose its parabolic nature – that is, when the value of memory component β is either very small or very big compared to the time component α, the presented algorithms are also capable of finding the solutions with a close to minimal cost.

7. It has been shown that the BIST quality considerably depends on the testability of circuits. By experimental research a correlation was established between the structural properties of circuits and their testability characteristics. It was shown that sharing of resources, which leads to the increase of number of reconvergencies, on one hand, will increase the time needed for test generation and fault analysis, but on the other, hand will reduce the test length.

Summary

In this thesis, several important issues in the area of built-in self-test of digital systems were presented:

- defect-oriented analysis of pseudorandom built-in self-test efficiency was investigated
- optimization techniques for different approaches of hybrid built-in self-test were developed
- throughout the research, two optimization techniques *tabu search* and *simulated annealing* were taken as basis of algorithms and compared with each other.
- the testability issues and relations to BIST of the industrial circuits developed for bioimpedance measurements were analyzed

Research on different fault models with the goal of adequate representing of physical defects for improving the quality of BIST analysis was carried out. A novel defect-oriented functional fault model called also conditional SAF model was introduced to support high quality BIST analysis.

As the classical built-in self-test approach has a number of drawbacks, there are many improvement techniques proposed. In this work, the emphasis was made on hybrid BIST approach – the combination of precomputed deterministic tests stored in memory and pseudorandom sequence, and on the "*store-and-generate*" BIST approach based on reseeding the LFSR.

In order to evaluate the test, a *cost function* was introduced that takes into consideration, on one hand, the cost related to applying the test patterns, and on the other hand, the cost related to storing the deterministic test patterns in on-chip memory. This cost function allows to find an optimal balance between pseudorandom and deterministic test sets, and to perform the hybrid self-test with minimum cost of both, time and memory, and without losing in test quality

For the Hybrid BIST with reseeding, the methods were developed which allow to minimize the test length at the given constraints for memory cost, and the methods which allow to optimize the BIST regarding both, time and memory.

The algorithms developed for solving the optimization tasks are based on two methods: *tabu search* and *simulated annealing*. Experiments showed that the proposed optimization techniques allow to find the optimal balance between the deterministic data and pseudorandom sequences for both the hybrid

BIST where pseudorandom sequence is combined with deterministic patterns and for “*store-and-generate*” hybrid BIST with reseeding. The results obtained by *simulated annealing* were more exact, but needed more calculation time, *tabu search* allowed to obtain results faster, but the solution found was less exact. However, the difference was still minor. Therefore, if the solution needs to be found fast, *tabu search* can be used, but if the exact solution is more important, then *simulated annealing* should be used.

A special testability related experimental research was carried out for a family of industrial designs - for different modifications of the biosignal processor system. The developed cost optimization techniques have proven to be feasible and efficient for creating an optimal "*store-and-generate*" type BIST with minimum cost for these systems. However, the testability analysis of the whole family showed that the quality and cost of the BIST may considerably depend on the structure of the biosignal processor.

References

[1] L. Benini, G.De Micheli. “Networks on Chip: a New SoC Paradigm”. IEEE Computer, Vol.35, No.1, pp.70-78, 2002.

[2] A.Jantsch, H.Tenhunen. “Networks on Chip. Kluwer Academic Publishers”. 2003.

[3] L.-T.Wang, Ch.-W.Wu, X.Wen. “VLSI Test Principles and Architectures”. Elsevier, 2006.

[4] R.Klein, T.Piekarz. “Accelerating Functional Simulation for Processor Based Designs”. Mentor Graphics Corporation. White paper, 2005.

[5] K.Roy, T.M.Mak, K.-T.T.Cheng. “Test consideration for nanometer-scale CMOS circuits”. IEEE Design and Test of Computers, vol.23, no 2, pp.128-136, 2006.

[6] A.Fin, F.Fummi. “Genetic Algorithms: the Philosopher’s Stone or an Effective Solution for High-Level TPG?”. In Proc. of IEEE HLDVT, pp. 163–168. 2003.

[7] F. Xin, M. Ciesielski, I. Harris. “Design validation of behavioral VHDL descriptions for arbitrary fault models”. In Proc. of IEEE ETS, pp. 156–161. 2005.

[8] E. J. Marinissen, Y. Zorian. “Challenges in Testing Core-Based Ics”. IEEE Communic. Mag, pp. 104-109, June 1999.

[9] International Technology Roadmap for Semiconductors http://www.itrs.net/

[10] M. Bushnell, V. Agrawal, “Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits”, Boston, Kluwer Academic Publishers, 2000.

[11] Charles E. Stroud “A Designer's Guide to Build-In-Self-Test”, Kluwer Academic Publishers, 2002.

[12] I.Dear, C.Dislis, A.Amber, J.Dick “Economic Effects in Design and Test”, IEEE Design and Test of Computers, Vol. 8, No. 4, pp 64-77, Dec 1991.

[13] N. K. Jha, S.Gupta. “Testing of Digital Systems” Cambridge University Press, 2003.

[14] O.Novák, E. Gramatova, R.Ubar. “Handbook Of Testing Electronic Systems”. Chech Technical University Publishing House, 2005.

[15] T.Williams, N. Brown. “Defect Level as a Function of Fault Coverage” IÈEE Trans. Comput., C-30(12), pp. 987-988, 1981.

[16] S. Chakravarty. “Defect Based Testing” Invited talk at the 4th International Workshop on IEEE Design and Diagnostics of Electronic Circuits and Systems Inc, 2001.

[17] M. Abramovici, M.Bauer, A. Friedman “Digital Systems Testing and Testible Desig”", Piscataway, New Jersey. IEEE Press, 1994.

[18] P.Maxwell, R.Aitken. “Defect-Oriented Testing”. IEEE European Test Workshop. Tutorial, 2003.

[19] M.Blyzniuk, T.Cibakova, E.Gramatova, W.Kuzmicz, M.Lobur, W.A.Pleskacz, J.Raik, R.Ubar. “Defect Oriented Fault Coverage of 100% Stuck-at fault Test Sets”. Proc. of the 7th International Conference on Mixed Design of Integrated Circuits and Systems. Gdynia (Poland), June 15-17, 2000, pp.511-516.

[20] S.Chakravarty, P.J.Thadikara. “Introduction to IDDQ Testing”. Kluwer Academic Publisher, 1997.

[21] M.Sachev. “Defect Oriented Testing for CMOS Analog and Digital Circuits” Kluwer Academic Publisher, 1998.

[22] K.L.Kodandapani, D.K.Pradhan. “Undetectability of bridging faults and validity of stuck-at fault test sets”. IEEE Trans. on Computers, C-29 (1), 1980, pp. 55–59.

[23] J.C.-M Tseng, Ch.-W. Tseng, E.J.McCluskey. “Testing for Resistive Opens and Stuck Opens”. Proceedings of International Test Conference, 2001, pp 1049-1057.

[24] Ch.F.Hawkins, J.M.Soden, A.Righter, F.J.Ferguson. “Defect Classes -an Overdue Paradigm for CMOS IC Testing”. Production Testing. Proceedings of International Test Conference ITC, 1994, pp 513-425.

[25] S.Chakravarty, A.Jain. “Fault Models for Speed Failures Caused by Bridges and Opens", Proceedings of VLSI Test Symposium, 2002, pp. 1-3.

[26] J.Yi, J.P.Hayes. “A fault Model for Function and Delay Testing”. Proceedings of IEEE European Test Workshop, 2001, pp. 77-78.

[27] J.Roth. “Diagnosis of automata failures: A calculus and a method”. IBM J. Res develop., 10(4) 278-291, 1966.

[28] P.Goel. "An implicit enumeration algorithm to generate tests for combinatorial logic circuits”. IEEE Trans. Comput. C-30(3) 215-222, 1981.

[29] H.Fujiwara, T.Shimono. “On the acceleration of test generation algorithms”. IEEE Trans. Comput. C-32(12), 1137-1144.

[30] M.H.Schulz, E. Trischler, T.M. Serfert. “SOCRATES: A highly efficient automatic test pattern generation system”. IEEE Trans. Computer –Aided Design, CAD 7(1), 126-137, 1988.

[31] T.Kirkland, M.R.Mercer. “A Topological Search Algorithm for ATPG”. Proc of the 24th Design Automation Conference. June-July 1987, pp. 502-508.

[32] M.L.Bushnell, J.Giraldi. “A Functional Decomposition method for Redundancy Identification and Test Generation”. Journal of Electronic Testing: Theory and Applications. vol 10, no3, pp. 175-195, June 1997.

[33] J. Giraldi, M.L. Bushnell. “EST: The New Frontier in Automatic Test Pattern Generation”. Proc. of the 27th Design Automation Conference, June 1990, pp 667-672.

[34] S.T. Chakradhar “Neural Network Models and Optimization Methods for Digital Testing”. PhD thesis, Computer Scence Department, Rutgers University, New Brunswick, New Jersey, Oct 1990.

[35] S.T.Chakradhar, V.D. Agrawal, M.L.Bushnell. “Neural Models and Algorithms for digital Testing”. Boston: Kluwer Academic Publishers 1991.

[36] E.McCluskey. “Verification Testing - A Pseudoexhaustive Test Technique” IEEE Trans. on Computers, Vol.33, No.6 pp541-546, June 1984.

[37] M.H. Schulz, E. Auth. “ESSENTIAL: An Efficient Self-Learning Test Pattern Generation Algorithm for Sequential Circuits”. Proc. of the International Test Conf., Aug. 1989, pp. 28–37.

[38] R. Bencivenga, T. J. Chakraborty, S. Davidson. “The Architecture of the Gentest Sequential Test Generator”. Proc. of the Custom Integrated Circuits Conf., May 1991, pp. 17.1.1–17.1.4.

[39] W.-T. Cheng, T. J. Chakraborty. “Gentest: An Automatic Test Generation System for Sequential Circuits”. Computer, vol. 22, no. 4, pp. 43–49, Apr. 1989.

[40] T. M. Niermann, J. H. Patel. “HITEC: A Test Generation Package for Sequential Circuits”. Proc. of the European Design Automation Conf., Feb. 1991, pp. 214–218.

[41] X.Chen, M.L.Bushnell. “Efficient Branch and Bound Search with Application to Computer-Aided Design”. Boston: Kluwer Academic Publishers, 1996.

[42] T.P.Kelsey, K.K.Saluja, S.Y.Lee. “An Efficient Algorithm for Sequential Circuit Test Generation”. IEEE Trans. on Computers, vol. 42, no. 11, pp. 1361–1371, Nov. 1993.

[43] F. J. Hill, B.Huey. “SCIRTSS: A Search System for Sequential Circuit Test Sequences”. IEEE Trans. on Computers, vol. C-26, no. 5, pp. 490–502, May 1977.

[44] H.-K. T. Ma, S. Devadas, A. R. Newton, A. Sangiovanni-Vincentelli. “Test Generation for Sequential Circuits”. IEEE Trans. on Computer-Aided Design, vol. 7, no. 10 pp. 1081–1093, Oct. 1988.

[45] A.Ghosh, S.Devadas, A.R.Newton. “Sequential Logic Testing and Verification”. Boston: Kluwer Academic Publishers, 1992.

[46] V. D. Agrawal, K.-T. Cheng, P. Agrawal. “A Directed Search Method for Test Generation using a Concurrent Simulator”. IEEE Trans. on Computer-Aided Design, vol. 8, no. 2, pp. 131–138, Feb. 1989.

[47] K.-T. Cheng, V.D.Agrawal. “Unified Methods for VLSI Simulation and Test Generation”. Boston: Kluwer Academic Publishers, 1989.

[48] D.G.Saab, Y.G.Saab, J. A. Abraham. “Automatic Test Vector Cultivationfor Sequential VLSI Circuits Using Genetic Algorithms”. IEEE Trans. on Computer-Aided Design, vol. 15, no. 10, pp. 1278–1285, Oct. 1996.

[49] E. M. Rudnick, J. H. Patel, G. S. Greenstein, T. M. Niermann. “A Genetic Algorithm Framework for Test Generation”. IEEE Trans. on Computer-Aided Design, vol. 16, no. 9, pp. 1034–1044, Sept. 1997.

[50] F. Corno, P. Prinetto, M. Rebaudengo, M.S.Reorda. “A Genetic Algorithm for Automatic Test Pattern Generation for Large Synchronous Sequential Circuits”. IEEE Trans. on Computer-Aided Design, vol.15, no.8, pp. 991–1000, Aug. 1996.

[51] M.S.Hsiao, E.M.Rudnick, J.H.Patel. “Sequential Circuit Test Generation using Dynamic State Traversal”. in Proc. of the European Design and Test Conf.,1997, pp. 22-28.

[52] M.S.Hsiao,E.M.Rudnick, J.H.Patel. “Dynamic State Traversal for Sequential Circuit Test Generation”. ACM Trans. on Design Automation of Electronic Systems (TODAES), vol. 5, no. 3, July 2000.

[53] J. H. Holland. “Adaptation in Natural and Artificial Systems”. Ann Arbor, Michigan: University of Michigan Press, 1975.

[54] D. E. Goldberg. “Genetic Algorithms in Search, Optimization, and Machine Learning”. Reading, Massachusetts: Addison-Wesley, 1989.

[55] C.Stover. “ATE: Automatic Test Equipment”. New York: McGraw-Hill, 1984.

[56] A.Miczo. “Digital Logic Testing and Simulation”. John Wiley & Sons, inc., Publications, 2003, USA.

[57] Y. Zorian, E. J. Marinissen, S. Dey, “Testing Embedded Core-Based System Chips”. IEEE International Test Conference (ITC), pp. 130-143, Washington, DC, October 1998. IEEE Computer Society Press.

[58] E.J.Marinissen, Y.Zorian. "Challenges in Testing Core-Based ICs". IEEE Communic. Mag, pp. 104-109, June 1999.

[59] C.Fagot, O.Gascuel, P.Girard, C.Landrault. "Calculating Efficient LFSR Seeds for BIST". Eur.Test Symp, Munich, 1999.

[60] P.D.Hortensius, et al. "Cellular automata based pseudorandom generators for BIST". IEEE Trans. CAD, (8), 8, 1990.

[61] S. W. Golomb. "Shift Register Sequences", Aegan Park Press, Laguna Hills, 1982.

[62] N.A.Touba, E.J.McCluskey. "Test Point Insertion Based on Path Tracing". Proc. VLSI Test Symposium, 1996, pp.2-8.

[63] B.Koenemann. "LFSR-Coded Test Patterns for Scan Designs". Proc. European Test Conference, March 1991, pp.237-242.

[64] V.K.Agrawal, E.Cerny. "Store and Generate Built-In Test Approach". Fault-Tolerant Computing Symp. 1981, pp.35-40.

[65] H.-J. Wunderlich, G.Kiefer. "Bit flipping BIST". Proc. ICCAD, Nov. 1996, pp.337-343.

[66] N.A.Touba,E.J.McCluskey. "Bit-fixing in pseudo-random sequences for scan BIST". IEEE Trans. CAD, (20), 4, 2001.

[67] S. Hellebrand, S. Tarnick, J. Rajski, B. Courtois. "Generation of Vector Patterns through Reseeding of Multiple-Polynomial Linear Feedback Shift Registers". IEEE International Test Conference, pp. 120-129, 1992.

[68] N.A.Touba, E. McCluskey. "Transformed Paseudo-Random Patterns for BIST". VLSI Test Symp, 1995, pp.410-416.

[69] S.Hellebrand, H.-J.Wunderlich, A.Hertwig. "Mixed-Mode BIST using Embedded Processors". JETTA 12, pp. 127-138, 1998.

[70] J.Rajski, J.Tyszer. "Arithmetic Built-In Self-Test For Embedded Systems". Prentice-Hall, New Jersey, 1998.

[71] M. Chatterjee, D. K. Pradhan. "A novel pattern generator for near-perfect fault-coverage". VLSI Test Symposium, pp. 417-425, 1995.

[72] M.Sugihara, H.Date, H.Yasuura. "Analysis and Minimization of Test Time in a Combined BIST and External Test Approach". Design, Automation & Test In Europe Conference DATE 2000, pp. 134-140, Paris, France, March 2000.

[73] N.A Touba, E. J. McCluskey. "Synthesis of mapping logic for generating transformed pseudo-random patterns for BIST". IEEE Int. Test Conference ITC'95, pp. 674-682, 1995.

[74] N. Zacharia, J. Rajski, J. Tyzer. "Decompression of Test Data Using Variable-Length Seed LFSRs". 13th VLSI Test Symposium, pp. 426-433, 1995.

[75] Z.Peng, Z.He, P.Eles. "Challenges and Solutions for Thermal-Aware SoC Testing". Invited paper. MIDEM Workshop on Electronic Testing, Bled, Slovenia, Sept. 12-14, 2007, pp.11-17.

[76] P.Nigh,W.Maly. "Layout - Driven Test Generation". Proc. ICCAD, 1989, pp. 154-157.

[77] M.Jacomet, W.Guggenbuhl. "Layout-Dependent Fault Analysis and Test Synthesis for CMOS Circuits". IEEE Trans. on CAD, 1993, 12, 888-899.

[78] U.Mahlstedt, J.Alt, I.Hollenbeck. "Deterministic Test Generation for Non-Classical Faults on Gate Level". 4th ATS, 1995, pp. 244-251.

[79] S.Holst, H.-J.Wunderlich. "Adaptive Debug and Diagnosis Without Fault Dictionaries". 13th ETS, 2008, pp.199-204.

[80] K.N.Dwarakanath, R.D.Blanton. "Universal Fault Simulation using fault tuples". DAC, Los Angeles, June 2000, pp.786-789.

[81] K.B.Keller. "Hierarchical Pattern Faults for Describing Logic Circuit Failure Mechanisms". US Patent 5546408, Aug. 13, 1994.

[82] R.D.Blanton, J.P.Hayes. "On the Properties of the Input Pattern Fault Model". ACM Trans. Des. Automat. Electron. Syst., Vol. 8, No. 1, pp. 108-124, Jan. 2003.

[83] R.Ubar. "Fault Diagnosis in Com. Circuits by Solving Bool. Diff. Equations". Automatics & Telemechanics, No.11, 1979, Moscow, pp.170-183 (in Russian). Transl. in: "Detection of Suspected Faults in Comb. Circuits by Solving Bool. Diff. Equations", Automation and Remote Control, Vol.40, No 11, part 2, Nov. 1980, Plenum Publishing Corporation, USA, pp. 1693-1703.

[84] Y.Cho, S.Mitra, E.J.McCluskey. "Gate Exhaustive Testing". International Test Conference, 2005.

[85] A.Jas, S.Natarajan, S.Patil. "The Region-Exhaustive Fault Model". 16th Asian Test Symp. Beijing, China, Oct. 2007, pp. 13-18.

[86] J.M.Acken, S.D.Millman. "Accurate Modeling and Simulating of Bridge Faults". Custom Integrated Circuits Conf., San Diego, CA, May 1991, pp. 17.4.1-17.4.4.

[87] P.Maxwell, R.Aiken. "Biased Voting: A Method for Simulating CMOS Bridging Faults in the Presence of Variable Gate Logic Thresholds". Proc. ITC, 1993, pp. 63-72.

[88] L.Zhuo, X.Lu, W.Qiu, W.Shi, D.M.H.Walker. ”A Circuit Level Fault Model for Resistive Opens and Bridges”. Proc. VLSI Test Symp., Napa, CA, Apr./May 2003, pp. 379-384.

[89] P.Engelke, I.Polian, M.Renovell, B.Becker. “Simulating resistive bridging and stuck-at faults”. IEEE Trans. on CAD of IC and Systems, Vol. 25, No. 10, pp. 2181-2192, Oct. 2006.

[90] A.Rousset, A.Bosio, P.Girard, C.Landrault, S.Pravossoudovitch, A.Virazel. “Fast Bridging Fault Diagnosis Using Logic Information”. 16th ATS. Beijing, China, Oct. 2007, pp.33-38.

[91] S.K.Jain, V.D.Agrawal. “Modeling and Test Generation Algorithms for MOS Circuits”. IEEE Trans. Comput., Vol. C-34, No. 5, pp. 426-433, May 1985.

[92] H.K.Lee, D.S.Ha. “SOPRANO: An Efficent Automatic Test Pattern Generator for Stuck-Open Faults in CMOS Combinational Circuits”. DAC, Orlando, FL, June 1990, pp. 660-666.

[93] A.Kristic, K.T.Cheng. “Delay Fault Testing for VLSI Circuits”. Dordrecht, The Netherlands, Kluwer Acad. Publishers, Oct. 1998.

[94] G.Chen, S.Reddy, I.Pomeranz, J.Rajski, P.Engelke, B.Becker. “A Unified Fault Model and Test Generation Procedure for Interconnect Opens and Bridges”. 10th ETS, Tallinn, May 2005.

[95] D.Lavo, T.Larrabee, B.Chess. “Beyond the Byzantine Generals: Unexpected Behavior and Bridging Fault Diagnosis”. Proc. ITC, 1996, pp. 611-619.

[96] S.Huang. “Speeding up the Byzantine Fault Diagnosis Using Symbolic Simulation”. Proc. VTS, 2002, pp.193-198.

[97] C.Y.Lee. “Representation of Switching Circuits by Binary Decision Programs. The Bell System Technical Journal”. pp.985-999. July 1959.

[98] R. Ubar, R. “Test Generation for Digital Circuits with Alternative Graphs”. Proceedings of Tallinn Technical University, No 409, pp.75-81, 1976 (in Russian).

[99] S.B. Akers. “Functional Testing with Binary Decision Diagrams”. Journal of Design Automation and Fault-Tolerant Computing (Vol.2), pp.311-331, Oct. 1978.

[100] R.E. Bryant. “Graph-based algorithms for Boolean function manipulation”. IEEE Trans. on Computers (Vol.C-35), No 8, pp.667-690, 1986.

[101] R.Ubar. “Test Synthesis with Alternative Graphs”. IEEE Design&Test of Computers, Spring, pp.48-57, 1996.

[102] R. Ubar, A. Moraviec, J.Raik. “Cycle-based Simulation with Decision Diagrams”. IEEE Proc. of Design Automation and Test in Europe – DATE, pp.454-458, 1999.

[103] S.Kirkpatrick, C.Gelatt, M.Vecchi. “Optimization by Simulated Annealing”. Science, 220 (4598): 498-516. 1983.

[104] V. Černy. “Thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm”. Journal of Optimization Theory and Applications, 45(1):41-51, 1985.

[105] J.H. Holland. “Adaptation in Natural and Arificial Systems”. University of Michigan Press, Ann Arbor. 1975.

[106] D.E. Goldberg. “Genetic Algorithms in Search, Optimization and Machine Learning”. Addison-Wesley, 1989.

[107] F.Glover, M.Laguna. “Tabu Search”. Kluwer, MA 1997.

[108] F.Glover. “Tabu search – Part I”. ORSA Journal on Computing, 1(3): 190-206, 1989.

[109] F.Glover. “Tabu search – Part II”. ORSA Journal on Computing, 2(1): 4-32, 1990.

[110] S.Nahar, S.Sahni, E.Shragowitz. “Experiments with simulated annealing and combinatorial optimization”. International Journal of Computer-Aided VLSI Design. 1989.

[111] S.M. Sait, H. Youssef. “Iterative Computer Algorithms with Application in Engineering”. IEEE Computer Society, 1999.

[112] P.Hansen. “The steepest ascent mildest descent heuristic for combinatorial programming”. Congress on Numerical Methods in Combinatorial Optimization, 1986.

[113] Teofilo F.Gonzales. “Handbook on Approximation and Metaheuristics”. Chapman & Hall/CRC, 2007.

[114] K.Binder. “Monte Carlo Methods in Statistical Physics”. Springer, Berlin 1978.

[115] M. Metropolis, A.Rosenbluth, M. Rosenbluth, A. Teller, E. Teller. “Equation of state calculations by fast computing machines”. Journal of Chemical Physics, 21, 1953, 1087-1092.

[116] R. Ubar. “Test Synthesis with Alternative Graphs”. IEEE Design&Test of Computers, Spring, pp.48-57, 1996.

[117] R.Ubar. “Multi-Valued Simulation of Digital Circuits with Structurally Synthesized Binary Decision Diagrams”. OPA (Overseas Publishers

Assotiation) N.V. Gordon and Breach Publishers, Multiple Valued Logic, Vol.4 pp. 141-157, 1998.

[118] R.Ubar, J.Raik, A.Karputkin, M.Tombak. "Synthesis of High-Level Decision Diagrams for Functional Test Pattern Generation". 16th Int. Conference MIXDES 2009. June 25-27, 2009 Lodz, pp.519-524.

[119] A.K.Gupta, J.R.Armstrong. "Functional Fault modelling and Simulation for VLSI Devices". 22nd Design Automation Conference, 1985, pp.720-726.

[120] S.M.Thatte, J.A.Abraham. "Test Generation for Microprocessors". IEEE Trans. On Computers, Vol. C-29, No. 6, pp.429-441, June 1980.

[121] D.Brahme, J.A. Abraham. "Functional Testing of Microprocessors". IEEE Trans. On Computers, Vol. C-33, No.6, pp.475-485, June 1984.

[122] R.D. Blanton, J.P. Hayes. "On the Properties of the Input Pattern Fault Model". ACM Trans. Design Automation of Electronic. Systems, (8)1, 108-124, 2003.

[123] R.Ubar, M.Aarna, H.Kruus, J.Raik. "How to Generate High Quality Tests for Digital Systems". IEEE International Semiconductor Conference, CAS'2004, Sinaia, Romania, Oct. 4-6, 2004, pp.459-462.

[124] T.Cibáková, M.Fischerová, E.Gramatová, W.Kuzmicz, W.Pleskacz, J.Raik, R.Ubar. "Hierarchical Test Generation for Combinational Circuits with Real Defects Coverage". Pergamon Press. Journal of Microelectronics Reliability, Vol. 42, 2002, pp.1141-1149.

[125] M. Bershteyn. "Calculation of Multiple Sets of Weights for Weighted Pseudorandom Testing". Proc. IEEE International Test Conference. 1993, pp. 1031-1040.

[126] G. Jervan, Z. Peng, R. Ubar, H. Kruus. "A Hybrid BIST Architecture and its Optimization for SoC Testing". IEEE 2002 3rd International Symposium on Quality Electronic Design. March 18-20, 2002 San Jose, California, USA. IEEE Computer Society Press, 2002, 273 - 279.

[127] G. Jervan, Z. Peng, R. Ubar. "Test Cost Minimization for Hybrid BIST". IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems, pp. 283-291, 2000.

[128] Z. Zhao, B. Pouya, N. A. Touba. "BETSY: Synthesizing Circuits for a Specified BIST". International Test Conference, pp. 144-153, 1998.

[129] F. Brglez, D. Bryan, K.Kominski. "Combinatorial Profiles of Sequential Benchmark Circuits". Proc. IEE International Test Conference, 1990, pp.1929-1934.

[130] A. Russ, C.Stroud. "Non-Intrusive Built-In Self-Test for FPGA and MCM Applications". Proc IEEE Automatic Test Conf. 1995, pp. 480-485.

[131] P.Bardell, W.McAnney, J.Savir. “Built-In Self-Test for VLSI: Psudorandom Sequences”. Somerset, New Jersey. John Wiley & Sons 1987.

[132] F. Muradali, V. Agarwal, B.Nadeau-Dostie. “A New Procedure for Weighted Random Built-In Self-Test”. Proc IEEE International Test Conference. 1990, pp. 660-669.

[133] M. F. AlShaibi, C. Kime. “MFBIST: A BIST Method for Random Pattern Resistant Circuits”. International Test Conference, pp. 176-185, 1996.

[134] G. Kiefer, H. Vranken, E. J. Marinissen, H.-J. Wunderlich. “Application of Deterministic Logic BIST on Industrial Circuits”. International Test Conference, pp. 105-114, 2000.

[135] H. Hashempour, F. J. Meyer, F. Lombardi. “Analysis and Measurement of Fault Coverage in a Combined ATE and BIST Environment”. IEEE Transactions on Instrumentation and Measurement, Vol. 53, Issue 2, pp. 300-307, 2004.

[136] P. Fišer. “Pseudo-Random Pattern Generator Design for Column Matching BIST”, Euromicro Conference on Digital Systems Design, pp. 657-663, 2007.

[137] M.E. Aboulhamid, E.Cerny. “A class of test generators for built-in testing”, IEEE Trans. Comput, C-32(10), 957 -959, 1983.

[138] R. Dandapani, J. Patel, J. Abraham. “Design of test pattern generators for built-in test”. In Proc Int. Test Conf. October 1984, pp. 315-319.

[139] G. Edirisooriya, J.P. Robinson. “Design of low cost ROM based test generators”. In Proc. VLSI Test Symposium. April 1992, pp. 61-66.

[140] J.Rajski, J.Tyszer, N.Zacharia. “Test data decompression for multiple scan designs with boundary scan”, IEEE Trans. Comput. 47(11), 1188-1200, 1998.

[141] S.Hellebrand, B.Reeb, S.Tarnick, H.-J. Wunderlich. “Pattern generation for deterministic BIST scheme”. In Proc. Int. Conf on Compput.-Aided Design, November 1995, pp. 88-94.

[142] H.-G.Liang, S.Hellebrand, H.-J. Wunderlich.”Two-dimentional test data compression for scan based deterministic BIST”. International Test Conference, September 2001, pp. 894-902.

[143] A.Al-Yamini, S.Mitra, E.J.McCluskey. “Optimized reseeding by seed ordering and encoding”. IEEE Trans. Computer-Aided Design. 24(2), pp. 264-270, 2005.

[144] G.Kiefer, H.-J. Wunderlich. “Deterministic BIST with multiple scan chains”. Proc International Test Conference, October 1998, pp. 1057-1064.

[145] G. Jervan, A. Markus, P. Paomets, J. Raik, R. Ubar. “A CAD system for Teaching Digital Test”. European Workshop on Microelectronics Education, pp. 287-290, 1998.

[146] R. Ubar, G. Jervan, Z. Peng, E. Orasson, R. Raidma. “Fast Test Cost Calculation for Hybrid BIST in Digital Systems”. Euromicro Symposium on Digital Systems Design, pp. 318-325, 2001.

[147] Turbo Tester Reference Manual. Version 3.2002.10. Tallinn University of Technology, www.pld.ttu.ee/tt

[148] V. K. Agarwal, E. Cerny. “Store and Generate Built-In Testing Approach”. Proc. of Fault Tolerant Computing Conference, Portland, 1981, pp.35-40.

[149] G. Jervan, P. Eles, Z. Peng, R. Ubar, M. Jenihhin. “Test Time Minimization for Hybrid BIST of Core-Based Systems”. Journal of Computer Science and Technology, 21(6), pp. 907-912, 2006.

[150] R. Ubar, G. Jervan, H. Kruus, E. Orasson, I. Aleksejev. “Optimization of the Store-and-Generate Based Built-In Self-Test”. Baltic Electronic Conference, pp. 199-202, 2006.

[151] F.Glover, E.Taillard, D. De Werra. “Auser’s guide to Tabu Search”. Annals of Operations Reserach, 41:3-28, 1993.

[152] E. T. McAdams, J. Jossinet, “Tissue impedance: a historical review”. Physiological Measurements,Vol. 16, pp. A1-A13.

[153] V. Pesonen, M. Gorev, P. Annus, M. Min, P. Ellervee. “Reconfigurable Data Acquisition Unit for Bioimpedance Measurements”. The 12th Biennial Baltic Electronics Conference BEC'2010, Tallinn, Estonia, pp.257-260, Oct. 2010.

[154] V. Pesonen, M. Gorev, P. Annus, M. Min, P. Ellervee. “Reprogrammable Data Acquisition Unit to Reduce Aliasing Effect in Bioimpedance Measurements”. The 7th Annual FPGAworld Conference, Copenhagen, Denmark, 6 pp., Sept. 2010.

[155] P. Ellervee, P. Annus, M. Min. “High Speed Data Preprocessing for Bioimpedance Measurements: Architectural Exploration”. The 27th NORCHIP Conference, Trondheim, Norway, pp.1-4, Nov. 2009.

[156] S. Grimnes, O. G. Martinsen. “Bioimpedance and Bioelectricity Basics”. Academic Press, San Diego, 2000.

[157] T. Dudykevych, E. Gersing, F. Thiel and G. Hellige. “Impedance Analyzer Module for EIT and Spectroscopy Using Undersampling”. Physiological Measurement, No. 22, Institute of Physics Publ. Ltd, UK, pp. 19-24, 2001.

[158] M.Min, T.Parve, P.Annus, T.Paavle. “A method of synchronous sampling in Multifrequency Impendace Maesurments”. IMTC 2006, Sorrento, Italy, Apr. 2006.

[159] R.Ubar, S.Devadze, J.Raik, A.Jutman. “Parallel X-Fault Simulation with Critical Path Tracing Technique”. IEEE Conf. Design, Automation & Test in Europe - DATE-2010, Dresden, Germany, March 8-12, 2010, pp. 1-6.

Curriculum Vitae

Personal data

Name	Helena Kruus
Date of birth	23.06.1979
Place of birth	Estonia
Citizenship	Estonian

Contact information

Address	Raja 15, Tallinn 12618, ESTONIA
Phone	+372 620 2260
E-mail	helena.kruus@ati.ttu.ee

Education

2003 - ...	Ph.D. student in Computer Engineering, Tallinn University of Technology
2001 - 2003	M.Sc. in Computer Engineering, TUT
1997 - 2001	B.Sc. in Computer Engineering, TUT
1986 - 1997	Secondary Education from Tõnismäe Reaalkool, Tallinn

Career

2004 - ...	Tallinn University of Technology Faculty of Information Technology Department of Computer Engineering Chair of Computer Engineering and Diagnostics Researcher
2002 - 2004	Tallinn University of Technology Faculty of Information Technology Department of Computer Engineering Chair of Systems Programming Teaching assistant
2000 - 2002	Tallinn University of Technology Faculty of Information Technology Department of Computer Engineering Chair of Systems Programming Lecturer

Scientific work

Publications

H.Kruus, R.Ubar, J.Raik. "Defect-Oriented BIST Quality Analysis." 12th International Biennial Baltic Electronic Conference BEC 2010, Tallinn (Estonia), October 4-6, 2010.

G.Jervan, E.Orasson, H.Kruus, R.Ubar. "Hybrid BIST Optimization Using Reseeding and Test Set Compaction." Microprocessors and Microsystems, 32(5-6), pp. 254 – 262, 2008.

H.Kruus, G.Jervan, R.Ubar. "Using Tabu Search for Optimization of Memory-Constrained Hybrid BIST". IKTDK Annual Conference, Voore (Estonia), 2008.

R.Ubar, J.Raik, H. Kruus, H.Lensen, T.Evartson. "Diagnostic Modelling of Digital Systems with Binary and High-Level Decision Diagrams." Bonilla, L.L.; Moscoso, M.; Platero, G.; Vega, J.M. (Toim.) Progress in Industrial Mathematics at ECMI 2006 (902 - 907). Springer-Verlag, 2008

H.Kruus, G.Jervan, R.Ubar. "Using Tabu Search for Optimization of Memory-Constrained Hybrid BIST" International Biennial Baltic Electronic Conference, pp. 155 – 158, 2008.

G.Jervan, H.Kruus, E.Orasson, R.Ubar. "Hybrid BIST Optimization Using Reseeding and Test Set Compaction." 10th EUROMICRO Conference on Digital System Design DSD 2007, Lübeck (Germany), IEEE Computer Society Press, 2007, pp. 596 – 603, 29-31 August 2007.

G.Jervan, H.Kruus, E.Orasson, R.Ubar. "Optimization of Memory-Constrained Hybrid BIST for Testing Core-Based Systems". IKTDK Annual Conference, Viinistu (Estonia), 2007.

G. Jervan, H.Kruus, E.Orasson, R.Ubar. "Optimization of Memory-Constrained Hybrid BIST for Testing Core-Based Systems." IEEE 2nd International Symposium on Industrial Embedded Systems SIES 2007, Lisbon (Portugal), IEEE Computer Soc, 2007, pp. 71 – 77, 4-6 July 2007.

R.Ubar, G.Jervan, H.Kruus, E.Orasson, I.Aleksejev. "Optimization of the Store-and-Generate Based Built-In Self-Test." 10th Biennial Baltic Electronics Conference BEC 2006, Laulasmaa (Estonia), IEEE Computer Society Press, 2006, pp. 199 - 202. October 2-4, 2006.

R.Ubar,G.Jervan,H.Kruus, E.Orasson, I.Aleksejev. "Optimization of the Store-and-generate Based Built-in Self-Test". IKTDK Annual Conference, Jäneda (Estonia), 2006.

R.Ubar, M.Aarna, H.Kruus, J.Raik. "High Quality Test Generation for Digital Systems." Romanian Journal of Information Science and Technology, 8(1), pp. 73 – 84, 2005.

V.Vislogubov, A.Jutman, H.Kruus, E.Orasson, J.Raik, R.Ubar. "Diagnostic software with WEB interface for teaching purposes." Proceedings of the 9th Biennial Baltic Electronics Conference BEC 2004, Tallinn (Estonia) pp. 255 – 258, October 3-6, 2004.

R.Ubar, M.Aarna, H.Kruus, J.Raik. "How to Generate High Quality Tests for Digital Systems." IEEE International Semiconductor Conference CAS2004 Sinaia (Romania) IEEE-Inst Electrical Electronics Engineers Inc, 2004, October 04-06, 2004.

T. Robal, H.Kruus. "e-Bibliothecula - A Virtual Library Service." 13th International Conference on Information Systems Development. Advances in Theory, Practice and Education, Vilnius (Lithuania); (Toim.) O. Vacilecas, A. Caplinskas, W. Wojtkowski, W.G. Wojtkowski, J. Zupancic, S. Wrycza. Vilnius: Technika, pp.139 – 148, 2004.

H.Kruus, E.Orasson, T.Robal, R.Ubar. "Investigating Defects in Digital Circuits by Boolean Differential Equations." The 4th International Conference "Distance Learning - Educational Sphere of XXI Century" DLESC'04 Minsk (Belarus), pp. 432 – 435, November 10-13, 2004.

G.Jervan, Z.Peng, R.Ubar, H.Kruus. "A Hybrid BIST Architecture and its Optimization for SoC Testing." IEEE 2002 3rd International Symposium on Quality Electronic Design ISQED'02, San Jose, California (USA), IEEE Computer Society Press, 2002, pp. 273 – 279, March 18-20, 2002.

R.Ubar, H.Kruus, G.Jervan, Z.Peng. "Using Tabu Search Method for Optimizing the Cost of Hybrid BIST." 16th Conference on Design of Circuits and Integrated Systems DCIS 2001, Porto (Portugal), pp. 445 – 450, November 20-23, 2001.

Defended theses

Master of Science in Computer Engineering, TUT
"Iterative Nondeterministic Optimization Algorithms. Special Course"
supervisor: Prof. R.Ubar

Bachelor of Science in Computer Engineering, TUT
"Using Tabu Search for Optimizing BIST in Digital Systems"
supervisor: Prof. R.Ubar

Awards

“Tiger University” of Estonian Information Technology Foundation grant for ITC students, 2006

AS Eesti Energia grant, Development fund of TUT, 2005

Nominated as a candidate for the Gerald W. Gordon Award, a joint International Test Conference (ITC) and IEEE Test Technology Technical Council (TTTC) award that recognizes outstanding student volunteer contributions to the IEEE Computer Society, 2005

“Tiger University” of Estonian Information Technology Foundation grant for ITC students, 2004

“Tiger University” of Estonian Information Technology Foundation grant for ITC students, 2003

AS Merko Ehitus grant, Development fund of TUT, 2002

Main areas of scientific work

Design and test of digital systems, optimization of built-in self-test, iterative optimization algorithms

Elulookirjeldus

Isikuandmed

Nimi	Helena Kruus
Sünniaeg	23.06.1979
Sünnikoht	Eesti
Kodakondsus	Eesti

Kontaktandmed

Adress	Raja 15, Tallinn 12618, EESTI
Telefon	+372 620 2260
E-post	helena.kruus@ati.ttu.ee

Hariduskäik

2003 - ...	Dokturantuur, Info- ja kommunikatsioonitehnoloogia Tallinna Tehnikaülikool
2001 - 2003	Tehnikateaduste magister, Tallinna Tehnikaülikool
1997 - 2001	Tehnikateaduste bakalaureus Tallinna Tehnikaülikool
1986 - 1997	Keskharidus, Tõnismäe Reaalkool, Tallinn

Teenistuskäik

2004 - ...	Tallinna Tehnikaülikool Infotehnoloogia teaduskond Arvutitehnika instituut Arvutitehnika ja -diagnostika õppetool Teadur
2002 - 2004	Tallinna Tehnikaülikool Infotehnoloogia teaduskond Arvutitehnika instituut Süsteemitarkvara õppetool Assistent
2000 - 2002	Tallinna Tehnikaülikool Infotehnoloogia teaduskond Arvutitehnika instituut Süsteemitarkvara õppetool tunnitasuline õppejõud

Teadustegevus

Publikatsioonid

H.Kruus, R.Ubar, J.Raik. "Defect-Oriented BIST Quality Analysis." 12th International Biennial Baltic Electronic Conference BEC 2010, Tallinn (Estonia), October 4-6, 2010.

G.Jervan, E.Orasson, H.Kruus, R.Ubar. "Hybrid BIST Optimization Using Reseeding and Test Set Compaction." Microprocessors and Microsystems, 32(5-6), pp. 254 – 262, 2008.

H.Kruus, G.Jervan, R.Ubar. "Using Tabu Search for Optimization of Memory-Constrained Hybrid BIST". IKTDK Annual Conference, Voore (Estonia), 2008.

R.Ubar, J.Raik, H. Kruus, H.Lensen, T.Evartson. "Diagnostic Modelling of Digital Systems with Binary and High-Level Decision Diagrams." Bonilla, L.L.; Moscoso, M.; Platero, G.; Vega, J.M. (Toim.) Progress in Industrial Mathematics at ECMI 2006 (902 - 907). Springer-Verlag, 2008

H.Kruus, G.Jervan, R.Ubar. "Using Tabu Search for Optimization of Memory-Constrained Hybrid BIST" International Biennial Baltic Electronic Conference, pp. 155 – 158, 2008.

G.Jervan, H.Kruus, E.Orasson, R.Ubar. "Hybrid BIST Optimization Using Reseeding and Test Set Compaction." 10th EUROMICRO Conference on Digital System Design DSD 2007, Lübeck (Germany), IEEE Computer Society Press, 2007, pp. 596 – 603, 29-31 August 2007.

G.Jervan, H.Kruus, E.Orasson, R.Ubar. "Optimization of Memory-Constrained Hybrid BIST for Testing Core-Based Systems". IKTDK Annual Conference, Viinistu (Estonia), 2007.

G. Jervan, H.Kruus, E.Orasson, R.Ubar. "Optimization of Memory-Constrained Hybrid BIST for Testing Core-Based Systems." IEEE 2nd International Symposium on Industrial Embedded Systems SIES 2007, Lisbon (Portugal), IEEE Computer Soc, 2007, pp. 71 – 77, 4-6 July 2007.

R.Ubar, G.Jervan, H.Kruus, E.Orasson, I.Aleksejev. "Optimization of the Store-and-Generate Based Built-In Self-Test." 10th Biennial Baltic Electronics Conference BEC 2006, Laulasmaa (Estonia), IEEE Computer Society Press, 2006, pp. 199 - 202. October 2-4, 2006.

R.Ubar,G.Jervan,H.Kruus, E.Orasson, I.Aleksejev. "Optimization of the Store-and-generate Based Built-in Self-Test". IKTDK Annual Conference, Jäneda (Estonia), 2006.

R.Ubar, M.Aarna, H.Kruus, J.Raik. "High Quality Test Generation for Digital Systems." Romanian Journal of Information Science and Technology, 8(1), pp. 73 – 84, 2005.

V.Vislogubov, A.Jutman, H.Kruus, E.Orasson, J.Raik, R.Ubar. "Diagnostic software with WEB interface for teaching purposes." Proceedings of the 9th Biennial Baltic Electronics Conference BEC 2004, Tallinn (Estonia) pp. 255 – 258, October 3-6, 2004.

R.Ubar, M.Aarna, H.Kruus, J.Raik. "How to Generate High Quality Tests for Digital Systems." IEEE International Semiconductor Conference CAS2004 Sinaia (Romania) IEEE-Inst Electrical Electronics Engineers Inc, 2004, October 04-06, 2004.

T. Robal, H.Kruus. "e-Bibliothecula - A Virtual Library Service." 13th International Conference on Information Systems Development. Advances in Theory, Practice and Education, Vilnius (Lithuania); (Toim.) O. Vacilecas, A. Caplinskas, W. Wojtkowski, W.G. Wojtkowski, J. Zupancic, S. Wrycza. Vilnius: Technika, pp.139 – 148, 2004.

H.Kruus, E.Orasson, T.Robal, R.Ubar. "Investigating Defects in Digital Circuits by Boolean Differential Equations." The 4th International Conference "Distance Learning - Educational Sphere of XXI Century" DLESC'04 Minsk (Belarus), pp. 432 – 435, November 10-13, 2004.

G.Jervan, Z.Peng, R.Ubar, H.Kruus. "A Hybrid BIST Architecture and its Optimization for SoC Testing." IEEE 2002 3rd International Symposium on Quality Electronic Design ISQED'02, San Jose, California (USA), IEEE Computer Society Press, 2002, pp. 273 – 279, March 18-20, 2002.

R.Ubar, H.Kruus, G.Jervan, Z.Peng. "Using Tabu Search Method for Optimizing the Cost of Hybrid BIST." 16th Conference on Design of Circuits and Integrated Systems DCIS 2001, Porto (Portugal), pp. 445 – 450, November 20-23, 2001.

Kaitstud lõputööd

2003 Magistritöö "Iteratiivsed mittedeterministlikud optimeerimisalgoritmid. Erikursus". TTÜ ATI. juh.R.Ubar

2001 Bakalaureusetöö "Tabu otsingu algoritmi rakendamine digitaalsüsteemide isetestimise optimeerimiseks". TTÜ ATI. juh.R.Ubar

Teaduspreemiad ja -tunnustused

EITSA “Tiigriülikooli” stipendium, 2006

AS Eesti Energia stipendium, TTÜ Arengufond, 2005

Gerald W. Gordon Award nominent (IEEE Computer Society), 2005

EITSA “Tiigriülikooli” stipendium, 2004

EITSA “Tiigriülikooli” stipendium, 2003

AS Merko Ehitus stipendium, TTÜ Arengufond, 2002

Teadustöö põhisuunad

Digitaalsüsteemide disain ja test, digitaalsüsteemide enesetestimise optimeerimine, iteratiivsed optimeerimisalgoritmid

DISSERTATIONS DEFENDED AT TALLINN UNIVERSITY OF TECHNOLOGY ON *INFORMATICS AND SYSTEM ENGINEERING*

1. **Lea Elmik**. Informational Modelling of a Communication Office. 1992.
2. **Kalle Tammemäe**. Control Intensive Digital System Synthesis. 1997.
3. **Eerik Lossmann**. Complex Signal Classification Algorithms, Based on the Third-Order Statistical Models. 1999.
4. **Kaido Kikkas**. Using the Internet in Rehabilitation of People with Mobility Impairments – Case Studies and Views from Estonia. 1999.
5. **Nazmun Nahar**. Global Electronic Commerce Process: Business-to-Business. 1999.
6. **Jevgeni Riipulk**. Microwave Radiometry for Medical Applications. 2000.
7. **Alar Kuusik**. Compact Smart Home Systems: Design and Verification of Cost Effective Hardware Solutions. 2001.
8. **Jaan Raik**. Hierarchical Test Generation for Digital Circuits Represented by Decision Diagrams. 2001.
9. **Andri Riid**. Transparent Fuzzy Systems: Model and Control. 2002.
10. **Marina Brik**. Investigation and Development of Test Generation Methods for Control Part of Digital Systems. 2002.
11. **Raul Land**. Synchronous Approximation and Processing of Sampled Data Signals. 2002.
12. **Ants Ronk**. An Extended Block-Adaptive Fourier Analyser for Analysis and Reproduction of Periodic Components of Band-Limited Discrete-Time Signals. 2002.
13. **Toivo Paavle**. System Level Modeling of the Phase Locked Loops: Behavioral Analysis and Parameterization. 2003.
14. **Irina Astrova**. On Integration of Object-Oriented Applications with Relational Databases. 2003.
15. **Kuldar Taveter**. A Multi-Perspective Methodology for Agent-Oriented Business Modelling and Simulation. 2004.
16. **Taivo Kangilaski**. Eesti Energia käiduhaldussüsteem. 2004.
17. **Artur Jutman**. Selected Issues of Modeling, Verification and Testing of Digital Systems. 2004.
18. **Ander Tenno**. Simulation and Estimation of Electro-Chemical Processes in Maintenance-Free Batteries with Fixed Electrolyte. 2004.

19. **Oleg Korolkov**. Formation of Diffusion Welded Al Contacts to Semiconductor Silicon. 2004.

20. **Risto Vaarandi**. Tools and Techniques for Event Log Analysis. 2005.

21. **Marko Koort**. Transmitter Power Control in Wireless Communication Systems. 2005.

22. **Raul Savimaa**. Modelling Emergent Behaviour of Organizations. Time-Aware, UML and Agent Based Approach. 2005.

23. **Raido Kurel**. Investigation of Electrical Characteristics of SiC Based Complementary JBS Structures. 2005.

24. **Rainer Taniloo**. Ökonoomsete negatiivse diferentsiaaltakistusega astmete ja elementide disainimine ja optimeerimine. 2005.

25. **Pauli Lallo.** Adaptive Secure Data Transmission Method for OSI Level I. 2005.

26. **Deniss Kumlander**. Some Practical Algorithms to Solve the Maximum Clique Problem. 2005.

27. **Tarmo Veskioja**. Stable Marriage Problem and College Admission. 2005.

28. **Elena Fomina**. Low Power Finite State Machine Synthesis. 2005.

29. **Eero Ivask**. Digital Test in WEB-Based Environment 2006.

30. **Виктор Войтович**. Разработка технологий выращивания из жидкой фазы эпитаксиальных структур арсенида галлия с высоковольтным p-n переходом и изготовления диодов на их основе. 2006.

31. **Tanel Alumäe**. Methods for Estonian Large Vocabulary Speech Recognition. 2006.

32. **Erki Eessaar**. Relational and Object-Relational Database Management Systems as Platforms for Managing Softwareengineering Artefacts. 2006.

33. **Rauno Gordon**. Modelling of Cardiac Dynamics and Intracardiac Bio-impedance. 2007.

34. **Madis Listak**. A Task-Oriented Design of a Biologically Inspired Underwater Robot. 2007.

35. **Elmet Orasson**. Hybrid Built-in Self-Test. Methods and Tools for Analysis and Optimization of BIST. 2007.

36. **Eduard Petlenkov**. Neural Networks Based Identification and Control of Nonlinear Systems: ANARX Model Based Approach. 2007.

37. **Toomas Kirt**. Concept Formation in Exploratory Data Analysis: Case Studies of Linguistic and Banking Data. 2007.

38. **Juhan-Peep Ernits**. Two State Space Reduction Techniques for Explicit State Model Checking. 2007.

39. **Innar Liiv**. Pattern Discovery Using Seriation and Matrix Reordering: A Unified View, Extensions and an Application to Inventory Management. 2008.

40. **Andrei Pokatilov**. Development of National Standard for Voltage Unit Based on Solid-State References. 2008.

41. **Karin Lindroos**. Mapping Social Structures by Formal Non-Linear Information Processing Methods: Case Studies of Estonian Islands Environments. 2008.

42. **Maksim Jenihhin**. Simulation-Based Hardware Verification with High-Level Decision Diagrams. 2008.

43. **Ando Saabas**. Logics for Low-Level Code and Proof-Preserving Program Transformations. 2008.

44. **Ilja Tšahhirov**. Security Protocols Analysis in the Computational Model – Dependency Flow Graphs-Based Approach. 2008.

45. **Toomas Ruuben**. Wideband Digital Beamforming in Sonar Systems. 2009.

46. **Sergei Devadze**. Fault Simulation of Digital Systems. 2009.

47. **Andrei Krivošei**. Model Based Method for Adaptive Decomposition of the Thoracic Bio-Impedance Variations into Cardiac and Respiratory Components.

48. **Vineeth Govind**. DfT-Based External test and Diagnosis of Mesh-like Networks on Chips. 2009.

49. **Andres Kull**. Model-Based Testing of Reactive Systems. 2009.

50. **Ants Torim**. Formal Concepts in the Theory of Monotone Systems. 2009.

51. **Erika Matsak**. Discovering Logical Constructs from Estonian Children Language. 2009.

52. **Paul Annus**. Multichannel Bioimpedance Spectroscopy: Instrumentation Methods and Design Principles. 2009.

53. **Maris Tõnso**. Computer Algebra Tools for Modelling, Analysis and Synthesis for Nonlinear Control Systems. 2010.

54. **Aivo Jürgenson**. Efficient Semantics of Parallel and Serial Models of Attack Trees. 2010.

55. **Erkki Joasoon**. The Tactile Feedback Device for Multi-Touch User Interfaces. 2010.

56. **Jürgo-Sören Preden**. Enhancing Situation – Awareness Cognition and Reasoning of Ad-Hoc Network Agents. 2010.

57. **Pavel Grigorenko**. Higher-Order Attribute Semantics of Flat Languages. 2010.

58. **Anna Rannaste**. Hierarcical Test Pattern Generation and Untestability Identification Techniques for Synchronous Sequential Circuits. 2010.

59. **Sergei Strik**. Battery Charging and Full-Featured Battery Charger Integrated Circuit for Portable Applications. 2011.

60. **Rain Ottis**. A Systematic Approach to Offensive Volunteer Cyber Militia. 2011.

61. **Natalja Sleptšuk**. Investigation of the Intermediate Layer in the Metal-Silicon Carbide Contact Obtained by Diffusion Welding. 2011.

62. **Martin Jaanus**. The Interactive Learning Environment for Mobile Laboratories. 2011.

63. **Argo Kasemaa**. Analog Front End Components for Bio-Impedance Measurement: Current Source Design and Implementation. 2011.

64. **Kenneth Geers**. Strategic Cyber Security: Evaluating Nation-State Cyber Attack Mitigation Strategies. 2011.

65. **Riina Maigre**. Composition of Web Services on Large Service Models. 2011.