THESIS ON INFORMATICS AND SYSTEM ENGINEERING C46

Fault Simulation of Digital Systems

SERGEI DEVADZE

TUT
PRESS

TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Computer Engineering

Dissertation was accepted for the defence of the degree of Doctor of Philosophy in Computer and Systems Engineering on July 27, 2009.

Supervisors: Prof. Raimund Ubar, D.Sc.
Prof. Peeter Ellervee, Ph.D.

Opponents: Prof. Heinrich Theodor Vierhaus,
Brandenburg University of Technology Cottbus, Germany

Prof. José Manuel Martins Ferreira,
University of Porto, Portugal

Defence: August 24, 2009

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 for any academic degree.

/Sergei Devadze/

ISSN 1406-4731
ISBN 978-9985-59-928-0

INFORMAATIKA JA SÜSTEEMITEHNIKA C46

Digitaalsüsteemide rikete simuleerimine

SERGEI DEVADZE

TTÜ Kirjastus

To my family

Abstract

The current thesis addresses issues in the field of digital testing. The presented work is focused on improving the efficiency of fault simulation methods that are widely used in the flow of designing tests for digital devices. Although the primary goal of fault simulation is assessment of quality of prepared test program, many test-related problems are strongly dependent on fault analysis. The tasks of test generation, fault diagnosis, optimization of built-in self test and test set compaction incorporate fault simulation as a part of process. Therefore the efficiency of fault analysis algorithm is an essential condition for solving the abovementioned tasks.

The main contribution of the research is the improvement of stuck-at fault simulation. The thesis presents several approaches for conducting fault analysis of a circuit represented by a special class of binary decision diagrams. The simulation is performed on macro-level but with gate-level accuracy. In particular, novel single-pattern and parallel-pattern simulation algorithms are introduced. Finally, the application of fault simulation for the hierarchical analysis of dependability is studied. The performed experiments confirm that the efficiency of the proposed methods overcomes the state-of-the-art approaches.

The thesis is based on the selected scientific papers published in journal and the proceedings of several international conferences.

Kokkuvõte

Antud väitekirja teematika on seotud digitaalsüsteemide projekteerimisega ja testimisega. Doktoritöö peaeesmärgiks on rikete simuleerimise meetodite parandamine.

Rikete simuleerimine ehk rikete analüüs on üks tähtsamaid ülesandeid digitaaltestimise valdkonnas, mille eesmärgiks on kindlaks teha, milliseid rikkeid on võimalik avastada etteantud testide abil. Kuna rikete analüüs kujutab endast sisuliselt protsessi, mis on aluseks paljude teiste testimisprobleemide lahendamisel (nt. testide kvaliteedi analüüsil, rikete diagnoosil, testide genereerimisel ja tihendamisel, süsteemide testkõlblikkuse hindamisel, isetestivate arhitektuuride projekteerimisel jne.), siis simulatsiooni kiirus on muutunud otsustavaks faktoriks loetletud ülesannete lahendamise ehk testide projekteerimise efektiivsuse tõstmisel.

Käesoleva doktoritöö tulemusena on välja töötatud efektiivsed meetodid ja algoritmid konstantrikete simuleerimiseks digitaalseadmetes. Erinevalt teistest meetoditest töötavad väljaarendatud simulaatorid kõrgemal abstraktsel tasandil kui loogikalülituste tase (tagades samal ajal loogikalülituste taseme täpsuse) ning kasutavad originaalset struktuurselt sünteesitud otsustusdiagrammide (OD) teooriat skeemide analüüsil.

Töö põhitulemused võib formuleerida järgmiselt. Esiteks on loodud OD teoorial põhinev deduktiivne algoritm rikete levimise analüüsiks skeemis ning selle algoritmi alusel ka vastav rikete simulaator. Teiseks on loodud uus ülikiire simulatsioonimeetod, mis võimaldab analüüsida rikkeid terve grupi testvektorite jaoks paralleelselt. Nimetatud meetodi uudsus seisneb erilise hargnemisanalüüsi meetodi väljatöötamises ning optimeeritud arvutusmudeli koostamises Boole'i diferentsiaalvõrrandite paralleelseks lahendamiseks. Eksperimendid näitasid, et võrreldes olemasolevate professionaalsete rikkesimulaatoritega, tõstab uus meetod tunduvalt rikete analüüsi kiirust. Rikete simulaatori rakendusena on väitekirjas välja töötatud originaalne hierarhiline meetod veakindluse hindamiseks, mis on üheks tsentraalseks ülesandeks usaldatavate süsteemide projekteerimisel.

Väitekirja aluseks on võetud neli teadusartiklit, mis on publitseeritud ühes ajakirjas ja kolme rahvusvahelise tippkonverentsi kogumikus.

Acknowledgements

I would like to thank everybody who helped me with advice and support during my Ph.D. studies.

First of all, I would like to sincerely thank my supervisor Prof. Raimund Ubar for guiding and consulting me through my studies and also encouraging me to finish this thesis. I am thankful to my other advisors, especially to Dr. Aleksander Sudnitsõn for helping me to make the first steps in my research activity and also Dr. Peeter Ellervee for giving valuable comments and remarks about this thesis.

Special thanks to Dr. Margus Kruus, the head of department of Computer Engineering for creating outstanding environment for productive work and study.

Furthermore, I want to thank Dr. Artur Jutman for his countless advises and interesting discussions. Also I would express my appreciation to my other colleagues, in particular Dr. Maksim Jenihhin, Dr. Jaan Raik and Uljana Reinsalu. The same holds for the group of young researches from the room IT231, Anton Tsertov, Igor Aleksejev, Sergei Kostin and Anton Tsepurov.

I am grateful to Dr. Dieter Wuttke from Technical University of Ilmenau for his hospitality and organizing fruitful summer projects that helped to seamlessly combine great vacation time with the fascinating work.

Moreover, I would like to acknowledge the organizations that have supported my Ph.D. studies: Tallinn University of Technology, Enterprise Estonia (project ELIKO), EU Regional Development Fund (project CEBE), National Graduate School in Information and Communication Technologies (IKTDK) and Estonian IT Foundation (EITSA).

Finally, I'd like to express my gratitude to my family and especially to my parents who were motivating and supporting me in all my undertakings. I am also indebted to my beloved wife Alla for her care and support throughout the time of my work and studies.

List of Publications

Fault simulation and fault analysis

- S. Devadze, R. Ubar, J. Raik, A. Jutman, "Parallel Exact Critical Path Tracing Fault Simulation with Reduced Memory Requirements", *Proc. of 4th IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era*, Cairo, Egypt, 2009.
- R. Ubar, S. Devadze, J. Raik, A. Jutman, "Parallel Fault Backtracing for Calculation of Fault Coverage", *Proc. of 13th Asia and South Pacific Design Automation Conference (ASPDAC'08)*, Seoul, Korea, 2008, pp. 667-672.
- R. Ubar, S. Devadze, M. Jenihhin, J. Raik, G. Jervan, P. Ellervee, "Hierarchical Calculation of Malicious Faults for Evaluating the Fault Tolerance", *Proc. of 4th IEEE International Symposium on Electronic Design, Test & Applications (DELTA'08)*, Hong Kong, China, 2008, pp. 222-227.
- R. Ubar, S. Devadze, J. Raik, A. Jutman, "Parallel Fault Backtracing for Calculation of Fault Coverage", *Proc. of 43rd International Conference on Microelectronics, Devices and Materials and the Workshop on Electronic Testing (MIDEM'07)*, Bled, Slovenia, September 12-14, 2007, pp. 165-170.
- R. Ubar, S. Devadze, J. Raik, A. Jutman, "Ultra Fast Parallel Fault Analysis on Structurally Synthesized BDDs", *Proc. of 12th IEEE European Test Symposium (ETS'2007)*, Freiburg, Germany, May 20-24, 2007, pp. 131-136.
- S. Devadze, R. Ubar, "Parallel Fault Analysis on Structurally Synthesized BDDs", *Proc. of 2nd IKTDK Conference*, Viinistu, Estonia, 2007, pp. 47-50.
- S. Devadze, J. Raik, A. Jutman, R. Ubar, "Fault Simulation with Parallel Critical Path Tracing for Combinational Circuits Using Structurally Synthesized BDDs", *Proc. of 7th IEEE Latin-American Test Workshop (LATW'2006)*, Buenos Aires, Argentina, 2006, pp.97-102.

- S. Devadze, “Efficient Fault Simulation Method on Structurally Synthesized BDDs”, *Proc. of 1st IKTDK Conference*, Estonia, 2006, pp. 123-126.
- J. Raik, R.Ubar, S.Devadze and A.Jutman, “Efficient Single-Pattern Fault Simulation on Structurally Synthesized BDDs”, *Proc. of 5th European Dependable Computing Conference (EDCC’2005)*, Budapest, Hungary, 2005, pp.332-344.

Board and system level test optimization

- S. Devadze, A.Jutman, I. Aleksejev, R. Ubar, “Fast Extended Test Access via JTAG and FPGAs”, *accepted for publishing in Proc. of 40th International Test Conference (ITC’2009)*.
- S. Devadze, A. Jutman, I. Aleksejev, R. Ubar, “Turning JTAG Inside Out for Fast Extended Test Access”, *Proc. of 10th Latin-American Test Workshop (LATW’2009)*, Rio de Janeiro, Brazil, 2009.
- S. Devadze, A. Jutman, A. Tsertov, M. Istenberg, R. Ubar, “Microprocessor-based System Test using Debug Interface”, *Proc. of 26th IEEE Norchip Conference (NORCHIP’2008)*, Estonia, 2008.
- A. Jutman, V. Rosin, S. Devadze, R. Ubar, “Trainer 1149.1: A Boundary-Scan Simulator”, *5th IEEE International Board Test Workshop (BTW'2006)*, Fort Collins, Colorado, Sept 13-15, 2006.

Finite State Machine decomposition

- S. Devadze, A. Sudnitson, “Software Environment for Synthesis of Testable FSM through Decomposition”, *Proc. of 26th International Conference on Microelectronics (MIEL’08)*, Nis, Serbia, 11-14 May 2008, vol. 2, pp. 433-436.
- S. Devadze, A. Sudnitson, “Synthesis of Testable FSM through Decomposition”, *Proc. of 3rd IKTDK Conference*, Voore, Estonia, 2008, pp. 101-104.
- A. Sudnitson, S. Devadze, “Web-Based Computer Aided Design Support of Finite State Machine Additive Decomposition for Low Power”, *Proc. of 5th IEEE East-West Design & Test International Symposium (EWDTS’07)*, Yerevan, Armenia, 2007, pp. 494-498.
- A. Sudnitson, S. Devadze, “Computer Aided Design Support of FSM Multiplicative Decomposition” *Proc. of IEEE East-West Design&Test International Workshop (EWDTW'06)*, Sochi, Russia. 2006, pp. 241-246.
- S. Devadze, A. Sudnitson, “FSM Decomposition Software for Education and Research”, *Proc. of IEEE EUROCON 2005 International Conference on 'Computer as a Tool',* Belgrade, Serbia and Montenegro, 2005, pp. 839-842.
- S. Devadze, “Web-Based System for Finite State Machines Decomposition”, M.Sc. thesis, Tallinn University of Technology, 2004.

- S. Devadze, E. Fomina, M. Kruus, A. Sudnitson, "Web-Based System for Sequential Machines Decomposition", *Proc. of IEEE EUROCON 2003 International Conference on 'Computer as a Tool'*, Slovenia, 2003, vol. 1, pp. 57-61.

HW-SW co-design

- U. Reinsalu, S. Devadze, A. Jutman, A. Chertov, P. Ellervee, "Hardware/Software co-design in practice: MEMOCODE'08 contents experience", *Proc. of 3rd IKTDK Conference*, Voore, Estonia, 2008, pp. 55-58.

Laboratory environment for education and research of design and test

- R. Ubar, A. Jutman, S. Devadze, H.-D. Wuttke, "Bringing Research Issues into Lab Scenarios on the Example of SOC Testing", *ICEE Proceedings*, University of Coimbra, Portugal, 2007.
- R. Ubar, A. Jutman, M. Kruus, E. Orasson, S. Devadze, H.-D. Wuttke, "Learning Digital Test and Diagnostics via Internet", *International Journal of Online Engineering, 3(1)*, 2007, pp. 1–9.
- W. Pleskacz, A. Jutman, R, Ubar, S. Devadze, "DefSim – the defective IC", *In University Booth section of Design Automation and Test in Europe (DATE 2007)*, France, 2007.
- R. Ubar, A. Jutman, M. Kruus, E. Orasson, S. Devadze, H.-D. Wuttke, "Learning Digital Test and Diagnostics via Internet", *International Journal of Computing and Information Sciences, 7(4)*, 2006.
- S.Devadze, "Web-Based Training System for Teaching Digital Design and Test", *Proc. of 7th International Student Conference on Electrical Engineering (POSTER2003)*, Prague, Czech Republic, May, 2003.
- S.Devadze, R.Gorjachev, A.Jutman, E.Orasson, V.Rosin, R.Ubar, "E-Learning Tools for Digital Test", *Proc. of 3rd International Conference 'Distance learning – educational sphere of XXI century'*, Minsk, Republic of Belarus, 2003, pp. 336-342.
- S. Devadze, A. Jutman, A. Sudnitson, R. Ubar, and H.-D. Wuttke, "Teaching Digital RT-Level Self-Test Using a Java Applet", *Proc. of 20th IEEE NORCHIP Conference 2002*, Denmark, 2002, pp. 322-328.
- S. Devadze, A. Jutman, A. Sudnitson, R. Ubar, and H.-D. Wuttke, "Java Technology Based Training System for Teaching Digital Design and Test", *Proc. of 8th Biennial International Baltic Electronics Conference (BEC'2002)*, Tallinn, Estonia, 2002, pp. 283-286.
- S. Devadze, A. Jutman, M. Kruus, A. Sudnitson, and R. Ubar, "Web Based Tools for Synthesis and Testing of Digital Devices", *Proc. of International Conference on Computer Systems and Technologies (CompSys'2002)*, Sofia, Bulgaria, 2002, vol.1, pp. 91-96.

- S. Devadze, A. Jutman, A. Sudnitson, and R. Ubar, "Web-Based Training System for Teaching Basics of RT-level Digital Design, Test, and Design for Test", *Proc. of 9th International Conference Mixed Design of Integrated Circuits and Systems (MIXDES'2002)*, Wroclav, Poland, 2002, pp.699-704.
- S. Devadze, M. Kruus, and A. Sudnitson, "Web-Based Software Implementation of Finite State Machine Decomposition for Design and Education", *Proc. of International Conference on Computer Systems and Technologies (CompSys'2001)*, Bulgaria, 2001, vol.4, pp. 1-7.

List of Abbreviations

ASIC	Application Specific Integrated Circuit
ATPG	Automatic Test Pattern Generation
BDD	Binary Decision Diagram
BIST	Built-In Self Test
CAD	Computer-Aided Design
CPT	Critical Path Tracing
CPU	Central Processing Unit
CUT	Circuit Under Test
DD	Decision Diagram
DFT	Design-For-Testability
DUT	Device Under Test
FFR	Fanout-Free Region
FPGA	Field Programmable Gate Array
FSM	Finite-State Machine
HDL	Hardware-Description Language
HLDD	High-Level Decision Diagram
IC	Integrated Circuit
I/O	Input Output
PI	Primary Input
PO	Primary Output
PPSFP	Parallel-Pattern Single Fault Propagation
PRPG	Pseudo-Random Pattern Generator

RTL	Register-Transfer Level
SA	Stuck-At
SAF	Stuck-At Fault
SSA	Single Stuck-at model
SSBDD	Structurally Synthesized Binary Decision Diagram
SoC	System-on-Chip
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLSI	Very-Large Scale Integration

Contents

Chapter 1

INTRODUCTION

This introductory chapter gives an overview of the area addressed by current thesis. At first, the motivation for the work is given followed by the formulation of the problem and the outline of main contributions. The last part of the chapter describes the organization of the thesis.

1.1 Motivation

It is not an overstatement to designate the microelectronics as the one of the most rapidly developing industries in the world. The only half of the century has been passed since the first integrated circuit had been invented in sixties and already billions of ICs stuffed with millions of transistors are produced nowadays. The trend in the digital device development that has been held in accordance with the famous Moore' law [1], [2] for past forty years gives the clear signals: the tendency is not going to stop since the microelectronic market constantly demands new devices with richer functionality, smaller dimensions and better performance.

The success of digital electronics has made a deep impact on the society. The world of digital devices become tightly tied with the everyday life and made people be much more dependent on the correct functioning of surrounding electronics. The last statement brings the problems of reliability of digital devices in a front place, causing dependability to be even more vital than the added value of novel functionality: having a new feature in a mobile phone is certainly very desirable but only if this innovation will not lead to the failures in the basic functionality.

The one of the solutions for the reliability issues is to perform comprehensive testing of the microelectronic product before the shipping it to end customer.

However the development of high quality tests had never been a simple task in digital world, is now turned into a real challenge because of the drastically grown complexity of integrated circuits. According to International Technology Roadmap for Semiconductors [3] increasing integration of microelectronic devices remains the key driver for enhancing the technology of manufacturing test.

The well-known example illustrates that the straightforward test of functionality of a 32-bit adder by exhaustive verification of the correct operation for any combination of the operands will require at least 2^{64} test steps to be executed. Indeed, even using a high-end test equipment, this operation will turn into several hundreds years of testing. However the development of advanced methods had allowed much more sophisticated devices such as microprocessors, ASICs, Systems-on-Chip, etc to be thoroughly tested. Nevertheless the permanent advances in the field of microelectronics demands continuous development of the test technology in order to cope with the increasing complexity of digital devices.

The current thesis is focused on improving of fault simulation technique that is the one of the major issues in the area of digital test. Many test-oriented tasks solved during the digital device design flow are heavily relying on fault analysis. For instance test generation, test quality assessment, fault diagnosis, test set compaction, optimization of built-in self test controller and others problems typically incorporate fault simulation as an intermediate step. Certainly this gives a very clear motivation for the attempts to raise the efficiency of fault simulator: the more accelerated fault analysis is – the more comprehensively the aforementioned tasks could be performed.

Although the successfulness of the designed device is influenced by very many different factors, but the availability of the efficient fault analysis tool could make a significant impact on reliability of the final product.

1.2 Problem formulation

Testing of microelectronic device is a special procedure that aims to check whether the device is working correctly or not. Typically test is conducted after device fabrication in order to ensure that no defects have been appeared in the device during this process.

In general, testing procedure consists of set of test stimuli (also referred as *test patterns*) that are being applied to primary inputs of the device under test (DUT). At the same time the output responses are recorded and compared with the expected ones. If the output response mismatches with the reference, then it is said that a failure has occurred. The reason for the failure could be the presence of defect inside the manufactured device.

However the real defects typically are not considered directly during the preparation of test but rather their behavior is simulated by fault models. The

subtask of fault simulation (also referred as fault analysis) has a goal of determining the effectiveness of test patterns in terms of the detectability of faults. For each of test patterns in the test set fault simulator is capable to determine which faults could be detected by applying the given test stimuli.

Obviously the fault simulation can require a lot of CPU and memory resources. In contrast with logic (fault-free) simulation that is done in a one pass and has linear time complexity to the number of gates in the circuit under simulation, the fault analysis requires many copies of the same circuit (that imitate presence of different faults) to be simulated. Thus the straightforward approach to fault simulation is unfeasible in case of large circuits (or large test set).

In this thesis the problem of stuck-at fault simulation of combinational (or scan-path) circuits is addressed. In particular, the presented work attempts to improve the efficiency of the fault analysis methods in terms of time and memory required for the fault simulation of circuit.

1.3 Thesis contribution

The main contributions of the current thesis are outlined below.

- The thesis introduces several techniques to perform fault simulation on Structurally Synthesized Binary Decision Diagrams (SSBDD). The usage of SSBDDs gives an opportunity to represent a gate-level design on a slightly higher abstraction level that immediately results in a higher speed of circuit evaluation.
- An efficient single-pattern fault simulation method is proposed [I]. The algorithm is essentially based on the introduced technique of deductive fault list propagation through SSBDD graphs. A reconvergency analysis carried out prior to the simulation determines the most efficient way for simulating each part of the circuit. Besides this, fault-free simulation on SSBDD is used for reducing the list of potential faults.
- The thesis introduces an efficient parallel-pattern fault analysis method. Two novelties are proposed here: the exact parallel critical path tracing algorithm on SSBDD model [II],[V] and the extension of the results of exact critical path tracing beyond the fanout-free regions [II], [III]. The latter uses a special calculation model to determine the detectability of fanouts.
- The approach of construction of optimized calculation model was proposed [III], [VII]. The usage of optimized calculation model lessens the number of unnecessarily repeated computations thus results in a higher analysis speed of parallel-pattern fault simulator.

- The problem of the memory requirements for fault simulation was studied. A novel approach for reducing amount of memory for simulation is presented together with the results of experiments [III].
- An approach for hierarchical dependability analysis is proposed [IV]. The method uses high-level Decision Diagrams for representation of circuit on register-transfer level and SSBDDs for lower-level description in order to determine the list of malicious faults.

1.4 Thesis structure

The presented thesis is organized in a form of overview of the research results that have been published in four scientific papers. The thesis has the following structure.

Chapter 2 forms a background on the discussed topic and makes a review of the state-of-the-art in the corresponded area. Chapter 3 presents an overview of the research results based on the selected publications presented in the last part of the thesis. The conclusions and the perspectives for future work are drawn in Chapter 4. In the last part, the selected papers that lay in the basis of current thesis are presented.

Chapter 2

BACKGROUND

This chapter presents background information on the topics related to current research. The chapter begins with the brief introduction to the digital test concept followed by the review of the fault modeling technique. The notion of stuck-at faults is described since the fault simulation methods proposed in the thesis are intended to work with stuck-at fault model. Next, the classical fault simulation methods are considered and the review of state-of-the-art in this area is given. At the end of the chapter the various applications of fault simulators in the flow of digital design are analyzed.

2.1 Introduction to digital test

The ultimate goal of digital test is to ensure that the device under test (DUT) is functioning according to its specification. In contrast to the verification that checks the correctness of the model of a circuit, testing is performed after the device is physically manufactured. Test program is typically developed during the design cycle with the assumption that the design itself was already verified and is correct.

During the test procedure a special test stimuli are applied to the primary inputs of DUT and the responses of the device outputs are analyzed (Figure 2.1). Because the fabrication process is not perfect, unintendent defects could incidentally appear in the DUT. As the result of defectiveness, the actually recorded responses could differ from the expected ones. In the latter case, it is said that the defect has manifested itself by a failure.

Generally speaking, testing helps to discriminate good devices from the faulty ones. In addition, a diagnosis of the failing device can be performed in order to identify the location and type of the defect.

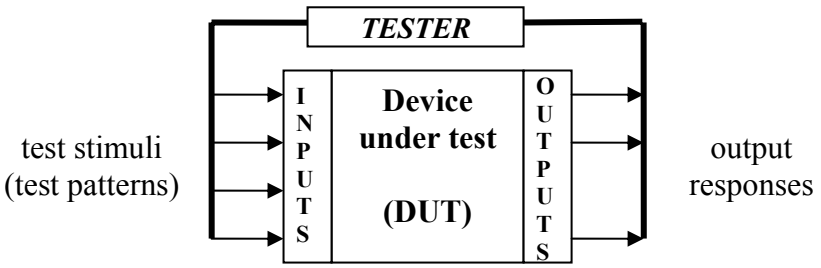


Figure 2.1: The concept of testing

2.2 Fault Modeling

By a term *defect* an unacceptable physical deviation of digital circuit from the normal case is assumed [5], [6]. As it was mentioned before, the presence of defect in a circuit could manifest itself by a *failure* (i.e. erroneous behavior of device). However the wide diversity of physical defects that could lead to malfunction of digital devices makes it almost impossible to exactly classify and evaluate them. The list of some sorts of defects in digital circuits could include [6],[4]:

- defects due to imperfection of manufacturing process (e.g. photolithographic errors, missing contact windows, parasitic transistors, incorrect spacing, misalignment, etc)
- material defects (e.g. insufficient purity of surface, contamination)
- age defects (e.g. electromigration)
- package-related defects

In view of the fact that working with large variety of physical defects is impractical, the *fault models* were introduced to offer simplified mathematical description of erroneous behavior. Hence, the term *fault* refers to the representation of a defect using a kind of abstraction.

Although most of fault models neither provide direct correspondence between faults and defects nor exact description of erroneous behavior of circuit, they are very useful for generating and evaluating quality of tests. A good fault model needs to reflect the presence of defects inside circuit precisely enough and be efficient for usage with computational algorithms.

Depending on their nature the fault models have been categorized by several levels of abstraction [4]. Defect-oriented fault models [10] are targeted to describe

the behavior of the defects of specific types, such as open and shorts between transistor lines. As result, defect-oriented fault models usually provide better conformity with the reality but demand sophisticated algorithms for processing. On the other hand, logical-level fault models that deal with the description of circuit at the level of logical signals are easier in processing. They are also independent on physical implementation of design. These facts have been made this class of models be commonly used in VLSI testing.

Unfortunately there is no single model that is sufficient enough for modeling all the possible sorts of physical defects. In spite of this fact, many fault models were proposed so far [5]: stuck-at model, bridging fault models (dominant, wired-AND/-OR, dominant-AND/-OR), delay fault models (gate-delay, path-delay), transistor-level fault models (stuck-opens, stuck-shorts [6]), models dealing with crosstalk faults [7], parametric fault models, etc. Indeed, there exist attempts to create generalized concepts (e.g. faults tuples [8]) that aim to incorporate several types of faults in a single model.

In general every fault model falls into one of two classes: the class of *multiple-fault models* and the class of *single-fault models*. When using single-fault model it is assumed that the only sole fault could exist in a circuit at a time. On the contrary, multiple fault models permit combinations of different faults to occur simultaneously. Obviously, the multiple-fault assumption increases the number of possible combinations of faults exponentially in comparison with single-fault model. For instance, if a fault model permits n different types of faults occur at m different fault sites, then for single-fault assumption the total number of faults in the model is $(n \times m)$. In case of multiple-fault model is considered, there exist $((n + 1)^m - 1)$ possible combinations of faults. Because the latter amount of faults is too large even for small values of n and m, the single-fault assumption is usually considered in practice. Fortunately, the experiments have shown that 100% coverage of single faults detects the most of multiple faults as well [4], [9].

2.2.1 Stuck-at faults

The stuck-at fault model is the one most commonly used in digital testing. According to current prognosis made by [3] the stuck-at faults will remain to be the one of the fault models most utilized for the testing of microelectronics for next years.

The presence of stuck-at fault in a digital circuit permanently fixes the value of corresponded signal line to logic one (stuck-at 1, *SA1*) or logic zero (stuck-at 0, *SA0*). Although stuck-at faults can be straightforwardly interpreted as a short between signal net and ground (or power) line, many other defects manifest themselves as *SA0* or *SA1* [10].

In general, there could be $3^n - 1$ various combinations of stuck-at faults in a circuit with n lines (each net could either be affected by presence of *SA0* or *SA1* or

do not contain any fault; the sole combination of totally fault-free circuit is excluded). However the single stuck-at model (*SSA*) that is commonly used in practice reduces this number to *2n* faults. As the current work is focused on SSA-based fault simulation issues, the only single stuck-at model is considered from this point.

Even in case of single-fault assumption not all the faults need to be considered. For instance, two different faults could affect circuit in the exactly same way, i.e. be indiscriminate. Certainly the processing of both such faults is redundant, thus one of them could be dropped out of the list of faults to consider. The technique of reduction of the complete list of faults without losing the quality of defect coverage is called *fault collapsing*

The algorithms of fault simulation are usually very sensitive to the total number of faults need to be handled: the less faults has been included into source fault list, the less time is required for their evaluation. Hence the possible reduction of fault list is very important, because it offers a kind of "pre-optimization" prior to execution of an algorithm itself. Some of well-known techniques of SSA collapsing are discussed below.

2.2.2 Fault equivalence for SSA

The reduction of fault list is possible by applying *equivalent fault collapsing* for SSA [5],[4],[11]. Assume we have single n-input AND gate to test. The presence of SA0 fault on any of gate I/O fixes the output of gate to logic zero (see Figure 2.2). This leads us to the conclusion that all SA0 faults for AND gate are *equivalent* (i.e. indistinguishable), and considering only one of them is enough. Strictly speaking, two faults are equivalent if their presence changes the output function of circuit in the exactly same way.

As for SA1 there are no equivalent faults for inputs and outputs of AND gate. As result, due to equivalent fault collapsing, the number of non-equivalent faults for n-input AND gate decreases to *(n + 2)* out of *(2n + 2)*.

It could be shown [5],[11] that similar relations exist between faults for other types of gates, e.g. OR, NAND, etc. For inverter and buffer gates, each fault on input has the equivalent fault on gate output, i.e. the total number of collapsed fault 2 out of 4 uncollapsed.

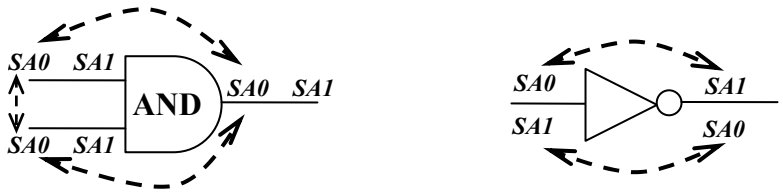


Figure 2.2: Stuck-at fault equivalence

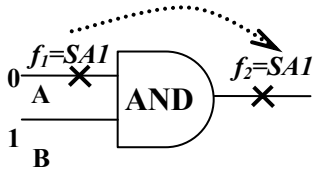


Figure 2.3: Example of fault dominance

2.2.3 Fault dominance for SSA

Let us consider two stuck-at faults f_1 and f_2 for 2-input AND gate in Figure 2.3. In order to detect f_1 it is needed to apply the following stimulus for the gate inputs: $A = 0$ and $B = 1$ (this is the sole test vector for detection of f_1). However, it is easily seen, that the same test vector detects the fault f_2 as well. As result the following conclusion is drawn: the detection of fault f_1 also detects the fault f_2 (it is said that fault f_2 *dominates* fault f_1). However this statement is not reflexive because the detection of f_2 will not necessarily indicate that fault f_1 is also detected (e.g. in case of test pattern $A=0$ and $B=0$).

Similar to equivalent fault collapsing, the *dominant fault collapsing* helps to reduce fault list further. For example, by using both fault collapsing techniques for n-input AND gate, only $(n + 1)$ faults need to be considered.

2.2.4 Single stuck-at fault collapsing for arbitrary circuit

It has been proven that in case of fanout-free circuit the only fault sites on primary inputs need to be considered to test the circuit for all single stuck-at faults [11]. The example of such reduction is illustrated in Figure 2.4. Here the faults that can be removed are marked by a grayed background. The equivalence fault collapsing is illustrated by dashed lines, while the elimination of faults due to fault dominance is marked by dotted lines.

The *checkpoint theorem* [11] states that for an arbitrary circuit, the only faults at primary inputs and fanout branches need to be detected in order to achieve 100%

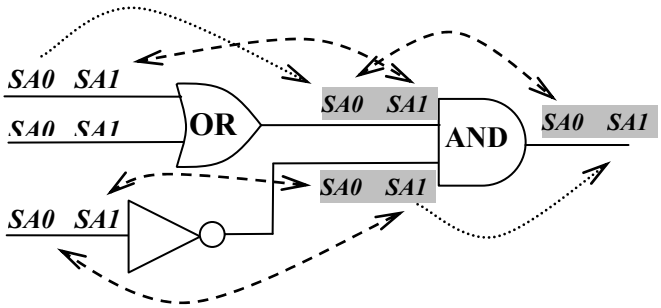


Figure 2.4: Fault collapsing in fanout-free circuit

fault coverage of the circuit. In Figure 2.5, a sample circuit with two fanout points is illustrated and the fault sites mandatory for consideration are marked.

The proof of this theorem could be illustrated in the following way. If a circuit has fanouts, the fanout points split the circuit into several fanout-free regions (*FFR*). According to the previous statement, for each of the FFRs we need to test stuck-at faults at the inputs of the region. As the input of a FFR is either fanout branch (e.g. *FFR 2* in Figure 2.5) or primary input (e.g. *FFR 1*) it could be stated that considering stuck-at faults in those fault sites is enough for complete single stuck-at testing of arbitrary circuit. The proof of both theorems is given in [11].

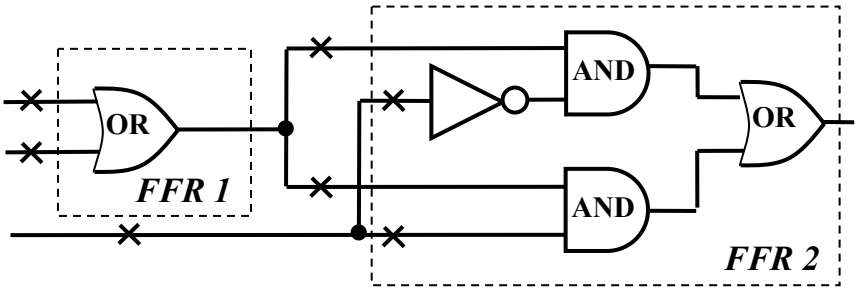


Figure 2.5: Example of fault collapsing in the circuit with reconvergent fanout

2.2.5 Contributions of current thesis

Instead of performing explicit stuck-at fault collapsing on gate-level netlist, the fault simulation algorithms proposed in the thesis take advantage of usage of SSBDD graphs for circuit representation (see Section 3.1). Besides other features, SSBDD model provides automatic fault collapsing thus eliminates the procedure of checking whether a fault belongs to the collapsed list or not.

2.3 Fault simulation

In contrast with logic (fault-free, true-valued) simulation, the task of fault simulator is to evaluate the behavior of circuit in case of the presence of faults inside. In particular, fault simulator has to determine whether the output response of a circuit is changing due to the influence of a fault or not. A fault which effect propagates to primary outputs under current input stimulus is referred as detected by the current test pattern.

Fault simulator typically works with a specific fault model. The input data of fault simulator is a set of test patterns together with the model of a circuit. In general case, the result of the execution of fault simulator is a *fault table* that shows what of the modeled faults are detectable by each of the given test patterns. In

addition, *fault coverage* (i.e. number of detected faults with respect to the total number of faults) is calculated.

The sections below present the brief description of the traditional fault simulation approaches and the review of the state-of-the-art.

2.3.1 Serial fault simulation

Serial fault simulation is the most simple and straightforward way to obtain fault table for a set of patterns. For every test pattern serial fault simulator evaluates fault-free version of circuit at first. Next, the circuit is modified to imitate the presence of a fault (such modification is called *fault injection*). The simulation process is conducted for the fault injected version of the circuit and the outputs responses of both faulty and fault-free copies are compared. The result of the comparison determines whether the fault is detectable by the given test pattern or not. After that, the injected fault is removed and the next fault is inserted. The whole procedure is repeated until all faults in the fault list have been simulated for the given test pattern.

Obviously, serial fault simulation algorithm is very simple in implementation: the only capability to inject faults need to be added to any logic simulator for converting it to fault simulator. However, for m faults in circuit the pure serial implementation of fault simulation is at least $(m + 1)$ times slower than just a true-valued simulation.

However there exist general optimizations that can be applied to overcome the inherited slowness of serial fault simulation. For instance, the *fault equivalence* and *fault collapsing* techniques that were discussed in the Sections 2.2.2 and 2.2.4 decrease the number of total faults thus results in the immediate speed gain.

Another optimization called *fault dropping* could be used for the situation when only overall fault coverage of the given test set is requested. For the simulation with fault dropping, the fault is excluded from the list of faults immediately after its detection. Because most of the faults are likely discovered by the first test patterns, the list of non-detected faults will shrunk very quickly and the simulation continues on small set of active faults. This property allows drastically speed-up fault simulation, however the capability to obtain fault table is lost in this case.

It is also possible to slightly decrease the algorithmic complexity of serial fault simulator by putting gates of a circuit in levelized order. In the ordered netlist, all the elements driving values to the inputs of a specific gate are placed prior to this gate. Consequently, the injection of fault can only influence the gates located after the fault insertion point while the I/O values of preceding gates remain unaffected. For this reason, fault simulator has to evaluate only part of the faulty copy of the circuit instead of processing the whole gate list.

Usage of compiled-code simulation instead of evaluation of gate-level model is another well-known technique for increasing the simulation speed. In the latter case, the simulator produces a special program (or machine code directly) that is executed on host computer. The purpose of this program is to use native set of CPU instructions for emulation of the behavior of the circuit. Although direct execution is faster than evaluation of circuit model, this approach lacks the flexibility. The compiled program requires complete execution even if the states of the most of nets did not changed, thus making this approach inefficient for the circuits with a small part of signals changing at a time. In addition, this method has portability issues (the simulator needs to work in combination with external compiler or be able to synthesize machine code for different platforms).

In spite of the speed inefficiency of serial fault simulators their simplicity allows to easily adapt them for usage with any kind of fault model. This is the main benefit of serial fault simulation in comparison with more complicated methods (the vast majority of fault simulators are not very flexible in handling different types of faults). However this approach lacks the efficiency on carrying out simulation on stuck-at fault model.

2.3.2 Parallel fault simulation

The fundamental idea of parallel fault simulation is to fully utilize the width of processor data word in order to reduce fault simulation time. For example, if host computer has 32-bit architecture then a logic operation on 32 binary variables can be performed simultaneously by execution of just one CPU instruction (e.g. AND, XOR, etc). The two types of parallel fault simulation are distinguished: *parallel fault simulation* (simulates many faults in parallel) and *parallel pattern simulation* (processes many patterns in parallel).

The only small overhead of parallel simulation is introduced by the demand in conversion (*packing*) of several integer values into bits of a single data word (*packet*). However the additional CPU resources needed for packing and unpacking data are rather insignificant and can be neglected because of overall speed gain.

All the optimization techniques that were discussed in the previous section are also applicable for parallel fault simulator. However the effectiveness of fault dropping is less in case of analysis of many faults in parallel.

2.3.2.1 Parallel fault simulation

An approach that utilizes the width of processor word for processing of multiple faults was firstly proposed in [13]. Assuming that each signal line can have either 0 or 1 value, w signals can be processed simultaneously on a w-bit CPU. The injection of fault to specific circuit line is made by altering of certain bit of w-bit data word associated with the signal. As result, w different copies of the same circuit can be processed simultaneously.

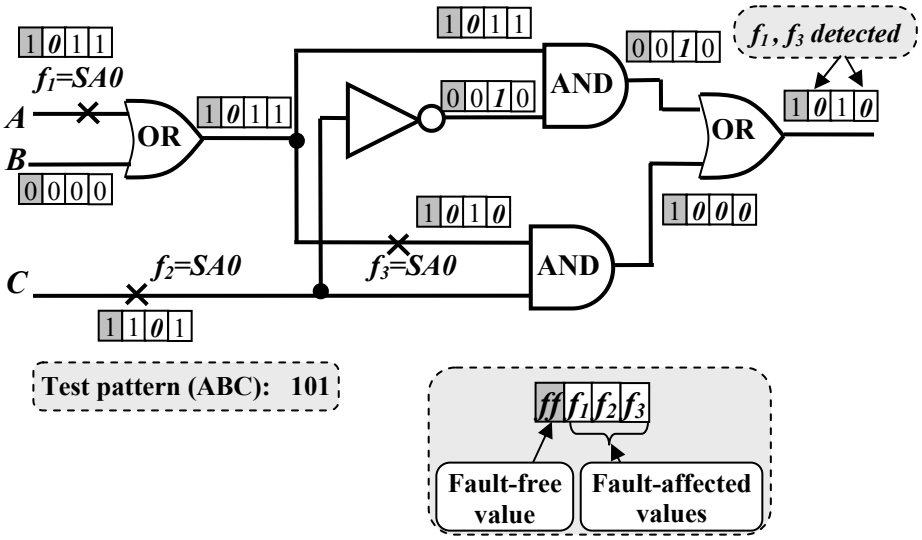


Figure 2.6: Parallel fault simulation example

The parallel fault simulation is illustrated in Figure 2.6 is performed for one test pattern (*ABC=101*) and three faults by using 4-bit packets: the first bit is used for fault-free simulation and the rest bits reflect the simulation results for injected faults f_1,f_2 and f_3 respectively. The faults f_1 and f_3 are detected because the value of corresponded bit in the packet associated with the primary output differs from fault-free value.

Comparing to the serial fault simulator this approach will increase the fault simulation speed in approximately $(w - 1)$ times.

As was mentioned before, the fault dropping could be significantly less effective when using parallel fault simulation. While serial fault simulator is capable to exclude fault just at time of its detection, parallel fault simulator does not terminate the simulation of a packet until all the faults that belong to this packet become detected. For example, if a packet contains one hard-to-test fault, the whole packet is kept in simulation run even if the remaining faults in the packet have been already detected.

2.3.2.2 Parallel-pattern fault simulation

On the contrary to parallel fault simulation *parallel-pattern fault simulation* technique takes advantage of bitwise parallelism for evaluating many test patterns simultaneously. The parallel-pattern simulation (also called *Parallel Pattern Single Fault Propagation* or *PPSFP*) was introduced by Waicukauski et al. in 1985 [14].

In PPSFP a sequence of test patterns is packed into w-sized data word where each bit corresponds to a separate pattern. The packet is simulated on fault-free

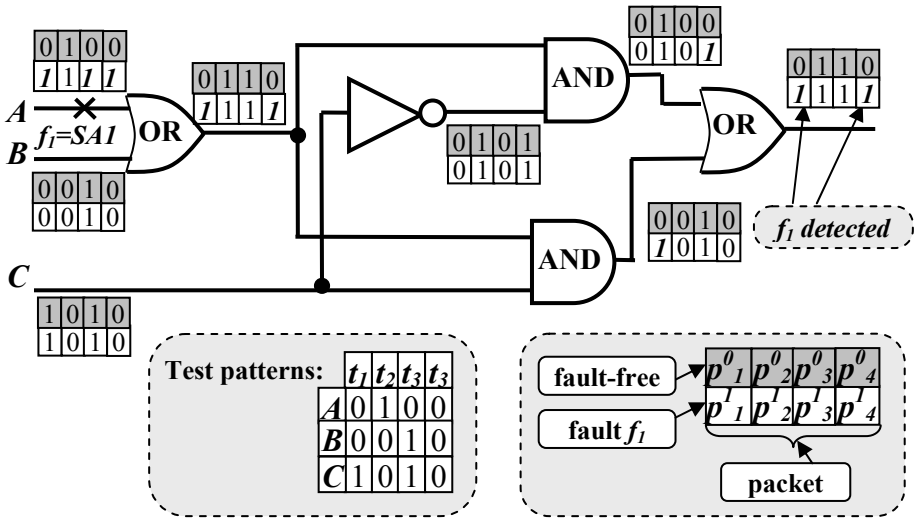


Figure 2.7: Parallel-pattern single fault propagation

circuit to obtain the values of primary outputs for the first *w* test patterns. Then for each fault in the fault list the following actions are repeated:

1. fault is injected into circuit;
2. parallel-pattern simulation is performed on faulty circuit;
3. output responses are compared with the results of true-valued simulation;
4. the fault is removed and the next fault is taken for consideration;

The whole procedure is executed until all test patterns become simulated.

The example depicted in Figure 2.7 conducts PPSFP simulation for 4 test patterns and a single fault. In the first run patterns are simulated in parallel for fault-free circuit, while the second run simulates the same patterns after injection of fault f_1.

Thanks to its parallelism, PPSFP method is almost *w* times more effective than serial fault simulation. Moreover, unlike parallel fault simulator, PPSFP does not have the effect of degradation of speed gain offered by fault dropping. The drawback of PPSFP approach is the limitation in use only with combinational circuits. For sequential design the state of the circuit should be computed before applying the next test pattern. However this condition is not held for test patterns that are simultaneously processed.

2.3.3 Concurrent fault simulation

Concurrent fault simulator [15] is essentially based on the idea of event-driven logic simulation. The simulator exploits the hypothesis that typical fault effect

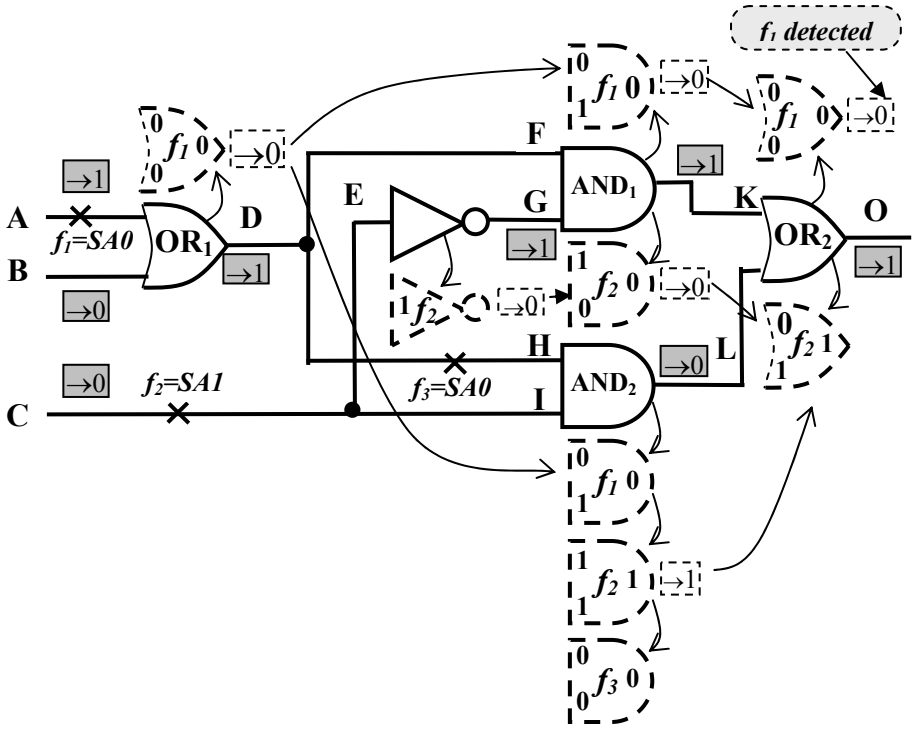


Figure 2.8: Concurrent fault simulation

results in differences for a small part of circuit. Consequently the only affected region need to be analyzed for fault detection.

In concurrent simulation each gate in a circuit has a set of associated *bad gates* (i.e. virtual copy of a gate in case of a presence of fault *f*). Besides the fault id, a bad gate also contains faulty signal values on its I/O. The bad gates in the sample circuit in Figure 2.8 are drawn by dashed lines.

Initially concurrent fault simulator creates bad gates for the faults with fault site in the same gate (such bad gate is called *fault origin gate*). In the example these are the bad gates created for faults f_1 (OR_1), f_2 (INV, AND_2) and f_3 (AND_2). With the fault effect propagation other bad gates could replicate from the original gates (these gates are called *fault effect gates*).

Concurrent simulation is performed pattern by pattern. A test pattern is applied by emerging events on primary inputs of circuit (e.g. *A*, *B* and *C*). The whole simulation process consists of consequent evaluation of events (changes of signals) on good and bad copies of gates.

Evaluation of events occurred at good gates computes fault-free values of signals. If an event on output of a fault origin bad gate causes signal to be different

from fault-free value, then this bad gate become *visible* (i.e. creates fault effect). The fault origin bad gates OR_1/f_1, INV/f_2 and AND_2/f_2 in Figure 2.8 are visible.

And vice-versa, fault origin gate become *invisible* if presence of fault does not change its output value (e.g. AND_2 gate with the fault f_3 is invisible because its output coincides with fault-free value).

Propagation of the fault event by visible bad gate to the input of destination gate causes a new *fault effect gate* to be *diverged* and added to the list of bad gates (e.g. INV/f_2 diverges a new fault effect bad gate AND_1/f_2). Diverged gates propagate the effect further. On the contrary, a fault effect gate *converges* to its original good gate if signals on its inputs are indistinguishable of fault-free values. Finally, a fault becomes detected in case if the effect of this fault reaches the primary outputs of circuit (f_1 in Figure 2.8).

Concurrent fault simulator is more flexible than other fault simulation methods because the rules of events evaluation, changing bad gate visibility, diverging and converging fault effect gates could be adapted to process the circuit description on different abstraction levels as well as for handling non-standard fault models. Moreover, the elimination of unneeded computation for the parts of circuit not affected by a fault considerably increases the efficiency of the method. However storing many copies of bad gates at run time is a potential memory problem, because the size of the lists is not known prior to the simulation.

A variation of concurrent fault simulation referred as differential fault simulator [16] utilizes the analogous event-driven technique but requires minimal amount of memory for implementation. Unlike the previous method, differential fault simulation deals with single fault at a time. There exists a parallel implementation of differential fault analysis algorithm [17] that speed-ups the fault detection process.

2.3.4 Deductive fault simulation

Deductive fault simulation (firstly proposed in [18]) is completely different in comparison with the methods described above. Deductive algorithm relies on logic reasoning rather than pure simulation (however the simulation is still needed but only to compute fault-free values of signals).

In deductive fault simulation a fault set S_x is associated with each signal line x. A fault f belongs to the fault set S_x if the presence of the fault f in circuit flips the state of signal line x. Thus a presence of fault f in the set associated with primary output indicates that f changes the output response of circuit. Therefore the ultimate goal of deductive simulator is to eventually construct fault sets for primary outputs of circuit and unite them into final set of detected faults $R = \bigcup S_o, o \in PO$.

In Figure 2.9 the example of deductive fault simulation is given (with only active three faults: f_1, f_2 and f_3).

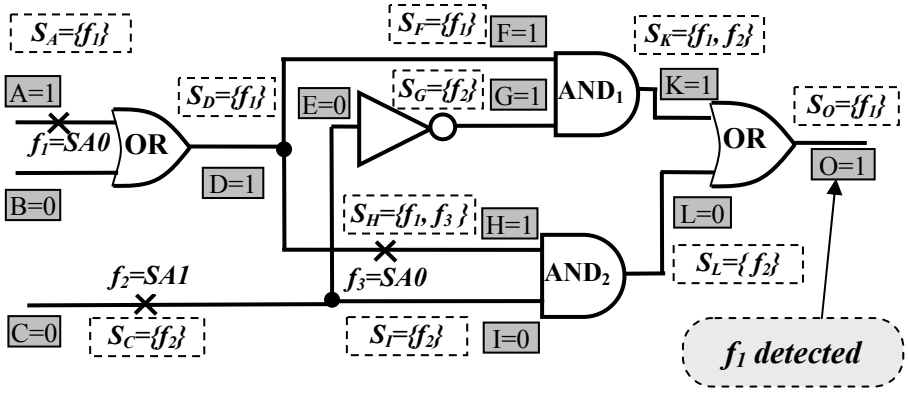


Figure 2.9: Deductive fault simulation

A procedure of deducing S_x fault sets is conducted as follows. As the first step, the initial fault sets are formed for the primary inputs (e.g. S_A and S_C) by the immediate inclusion of faults on the inputs of circuit. Next, the evaluation of gates begins in the direction from primary inputs to primary outputs. For the single gate with known fault sets on its inputs, deductive simulator derives a fault set for gate output (this process is referred as *fault propagation*). Then the fault at output is included (*activated*) in the propagated list of fault (*fault activation*).

Consider the procedure for propagation fault sets in general. During gate evaluation deductive fault simulator distinguishes between the gate inputs holding controlling values (set of gate inputs I_c) and non-controlling values (set of gate inputs I_{nc}). A controlling value of a gate is the value that defines the value of gate output: for example a controlling value for AND gate is 0 (because appearance of logic 0 at least on one of the inputs of AND gate forces its output to go into logic 0 state), a controlling value for OR gate is 1, etc. Then two cases are recognized:

1. $I_c \in \varnothing$ (i.e. all gate inputs hold non-controlling values). This means, that the fault effect observed on any of inputs will propagate to gate output (i.e. fault that belongs to any of fault sets of gate input is observable at gate output). For this case the target fault set *S'* is calculated using the following equation:

$$S' = \bigcup_{i \in I_{nc}} S_i$$

   In the example in Figure 2.9 the fault set for the output of AND_1 gate is constructed out of the sets S_F and S_G by using the formula above.

2. $I_c \notin \varnothing$ (e.g. some inputs may hold controlling value). In this case, a fault propagates through the gate only if its effect was propagated to every controlling input while self-masking effect (the appearance of the same

fault effect at any non-controlling inputs of the gate) is absent. The respective fault set *S'* for gate output is derived as:

$$S' = \bigcap_{i \in I_c} S_i - \bigcup_{j \in I_{nc}} S_j$$

The application of this rule is illustrated by deriving fault set S_O from the sets S_K and S_L.

Finally, to obtain the complete fault set *S*, the potential fault at corresponded signal line (i.e. gate output) need to be activated, i.e. added to the fault set:

$$S = S' \cup f_{at\_current\_signal\_line}$$

The described procedure continues until the construction of fault sets for all primary outputs is finished. As the last step, fault sets associated with primary outputs are united into single set of detected faults $R = \bigcup_{o \in PO} S_o$.

The deductive fault simulation is extremely powerful in comparison with simulation-based approaches due to the fact that all faults are processed in a single run (for given test pattern) avoiding re-simulations of the same circuit. However during the simulation process deductive fault simulator spends most of CPU time on logic operations over fault sets (union, intersection and complementation).

2.3.5 Critical path tracing

As an alternative to the fault simulation, critical path tracing (CPT) algorithm [19] does not conduct any simulations except true-valued one. Instead of that, critical signal lines are traced starting from primary outputs towards inputs of circuit.

During path tracing, a signal line is considered as critical if change of its value causes a flip of the state of primary output. As result, a stuck-at fault that is associated with the critical line (*SA0* if the value of signal line is logic 1, *SA1* otherwise) should be immediately added to the list of detected faults.

Critical path tracing starts with the primary output. Since primary output is essentially critical, it is added to the list of critical nets and the tracing continues for gate that drives the output. The inputs of the gate are evaluated to determine whether they affect any of critical nets or not. By the result of such evaluation, the inputs of gate may be added to list of critical nets. The tracing continues until all the nets are evaluated (or until no critical nets under evaluation remained).

The process of critical path tracing is presented in Figure 2.10 (the critical nets are marked by bold lines). Note that for exact results, CPT need to be stopped by reaching fanout. Otherwise the *fault-masking* effect could spoil the results of simulation. In the example in Figure 2.10 critical path tracing discovers signal line

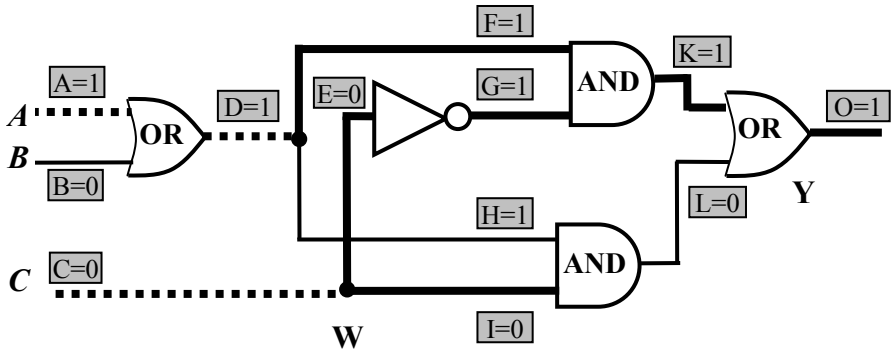


Figure 2.10: Critical path tracing technique

C to be critical in respect to primary output. However the fault effect diverges at fanout *W* continues to propagate by both fanout branches *E* and *I*. Finally, the phenomena of *fault masking* becomes apparent at reconvergency point *Y*, where fault effects on both lines *K* and *L* cancel (*mask*) each other hence stopping the fault propagation.

Critical path tracing algorithm provides linear complexity (in respect to the number of nets) within fanout free region of circuit. Unfortunately original implementation of critical path tracing cannot straightforwardly handle fanout reconvergencies hence providing only approximate fault coverage. In order to obtain exact results, critical path tracing algorithm should either be restricted by fanout free region or perform special handling of reconvergent fanouts.

2.3.6 Review of state-of-the-art

The methods presented in the previous sections can be characterized as basic types of fault simulation algorithms. However a number of sophisticated optimizations were proposed so far in order to achieve better efficiency of fault analysis. Below we will outline some of the attempts to build powerful algorithms for fault simulation.

Antreich and Schulz have proposed an accelerated parallel-pattern fault simulation method [20]. The key idea of the algorithm is to reduce the number of fanout stems to be unnecessarily re-simulated by identifying of independent fanout branches.

Harel et al. [21] suggested to use a *dominator concept* (in terms of graph theory) for improving the efficiency of PPSFP by the reduction of unnecessary simulated areas of circuit. Another proposed optimization of the paper is implementation of priority queues for maintaining the list of gates waiting for evaluation.

Another improvement of parallel fault simulation method proposed by Maamri and Rajski is based on *stem-region* analysis [22]. For each fault in reconvergent

fanout the approach determines a stem-region which limits the simulation area. The stem-region is bounded by so-called *exit lines* that form a set of disjoint cones (from exit-line to primary outputs). If fault is detected on exit line and this line is critical the further simulation is not needed. For fault anslysis inside fanout-free regions the method uses CPT technique.

Test-detect fault simulation algorithm proposed by Roth [23] requires faults on gate lines to be evaluated in a backward levelized order. Hence for each fault, the gates that occur later in the order are already considered. This gives the opportunity to stop the simulation of a fault if the propagation path of this fault is converged to a single gate. The further elaboration of this method [24], results in test-detect algorithm refined for parallel use.

Lee and Ha have proposed the efficient version of PPSFP-type simulator [25] that is exploits the idea of eliminating of unnecessarily simulated regions on early stages of fault simulation. This is achieved by examining the detectability of faults and exclusion the following regions out of simulation in case if no faults are detectable at the output of currently simulated FFR or stem region. The method also enhanced with usage of critical-path tracing inside FFRs and efficient implementation of stack of gates under evaluation.

Saab have presented parallel-concurrent fault simulator [26] that relies on the approach of concurrent simulation but simultaneously processes fault groups instead of single faults. The technique for partitioning faults into fault groups reduces time needed for processing of events in concurrent simulator.

Takahashi et al. have extended deductive fault simulation approach for the case of multiple stuck-at fault model [27]. Authors provide the solution for handling very large number of fault combinations by using Boolean functions (represented with the help of shared BDDs) thus cutting down the memory requirements.

Wu and Walker have proposed critical path tracing method [28] that allows to perform exact CPT on a circuit with reconvergencies in nearly-linear time. The method is based on traditional CPT supplemented with a special set of rules to handle various cases of reonvergencies.

The method of approximate fault analysis called fault sampling was proposed in [29] for reducing the efforts of fault simulation. The method works in conjunction with fault simulator to determine the detectability of randomly picked sample of faults (i.e. subset of fault list) and extrapolate these results by using means of probabilities theory.

Jain and Agrawal have proposed another approximate method of fault simulation. Statistical fault analysis [30] uses results of fault-free simulation for producing fault coverage estimation. During logic simulation the number of occurrences of 0- and 1-values for each signal line and number of cases when gate input is sensitized to the gate output are counted. Basing on these values, statistical

fault analyzer computes the probability of each fault to be detected. However the both approximate methods cannot provide the exact data about fault detectability.

Besides the conventional approaches, many challenges have been made to increase the speed of fault simulation by delegating part of the process to specially developed hardware accelerators [31], [32]. Many of such attempts utilize reconfigurability of FPGA to emulate the whole circuit under test in reprogrammable logic [33], [34], [35]. However these techniques require additional devices to be attached to host computer thus narrowing their applicability.

Recently a new dimension in the area of accelerating fault simulation speed is being thoroughly explored [36], [37]. The key idea of the approach is to use standard off-the-shelf hardware that is capable for parallel processing to accelerate the well-known fault simulation algorithms. Typically, graphical processing units (which likely contain hundreds of separate processing cores) are programmed for concurrent execution of basic operations needed to run simple fault simulation algorithms (e.g. parallel fault simulation or PPSFP).

2.3.7 Problems with fault simulation algorithms

Although the techniques considered in this section are equipped with sophisticated algorithms for fault simulation, they still have certain drawbacks that slow down the analysis speed. The major disadvantage of deductive fault simulation (Section 2.3.4) is the demand of complex operations on large fault sets that decrease the speed of deductive reasoning. However fault equivalence and fault dominance relationships (Section 2.2.4) can assist in reducing the total number of faults.

Straightforward implementation of parallel-pattern fault analysis (Section 2.3.2.2) results in unnecessary simulation of the parts of circuit that are not affected by a fault. Also re-simulation is performed for the regions that have already been simulated under the same conditions (for instance, if propagation of effect of a fault converges to a single signal line). However several approaches have been proposed to reduce this overlap ([20],[21],[22],[24],[25]), they cannot pretend to completely eliminate the unneeded simulation.

In the latter sense concurrent and differential simulators (Section 2.3.3) are more efficient since they simulate only “active” parts of circuit. However these methods are generalized for any types of circuit description and fault models and do not exploit the advantage of gate-level combinational stuck-at fault simulation. In addition, the parallel versions of concurrent and differential simulators do not provide speed gain comparable with the gain obtained by PPSFP simulators.

Although critical path tracing (Section 2.3.5) is very powerful technique because it offers linear-time complexity, it cannot handle fanout reconvergencies. The proposed exact critical path tracing for arbitrary circuit [28] is based on a set of sophisticated rules and cannot be applied for parallel processing of patterns.

2.3.8 Contributions of current thesis

In this thesis we are attempting to improve deductive (Section 3.2) and parallel-pattern (Section 3.3) fault simulation approaches. In contrast with the methods described above we propose to perform fault simulation on a network of macros instead of processing logic gates (see Section 3.1).

As it was mentioned before, the speed of deduction algorithm directly depends on the number of faults taken for consideration. In the thesis we propose fault list reduction technique (Section 3.2.1) as well as the method that is intended to avoid sophisticated fault list deduction for certain parts of circuit (Section 3.2.3).

Parallel-pattern fault analysis approach that is proposed in the current work essentially incorporates parallel critical path tracing technique. In comparison with traditional CPT, the proposed method has been accelerated by parallel processing and the use of macro-level description of circuit (Section 3.3.1). Moreover, the method of parallel critical path tracing is extended for circuit with reconvergencies using a special calculation model (Section 3.3.3 and 3.3.4) that helps to escape unnecessary re-simulations.

2.4 Applicability of fault simulation

In the following subsections the main test-related tasks that require intensive use of fault analysis are outlined. This helps to reveal the relevance of the problem of fault simulation performance. The speed of fault simulator is especially crucial while solving the tasks discussed in subsections 2.4.2, 2.4.4 and 2.4.5 since the process of fault analysis is required to be performed many times in a cycle.

2.4.1 Test quality evaluation

An essential task of fault simulator is to evaluate quality of the supplied test program. The quality is measured in terms of *fault coverage* with respect to the specific fault model. Fault coverage indicates the ratio of faults discovered by test patterns in respect to the total number of faults in the model. Calculating the fault coverage and reporting the list of detected and undetected faults is the primary task of any fault simulator. In addition the fault simulators are capable to build a complete fault table with the lists of discovered faults for each test pattern. The latter is necessary for the task of diagnosis.

2.4.2 Test generation

The fault simulator is often used in conjunction with Automatic Test Pattern Generation (ATPG) process in order to verify the generated test pattern (Figure 2.11). This is especially applicable to different types of random pattern

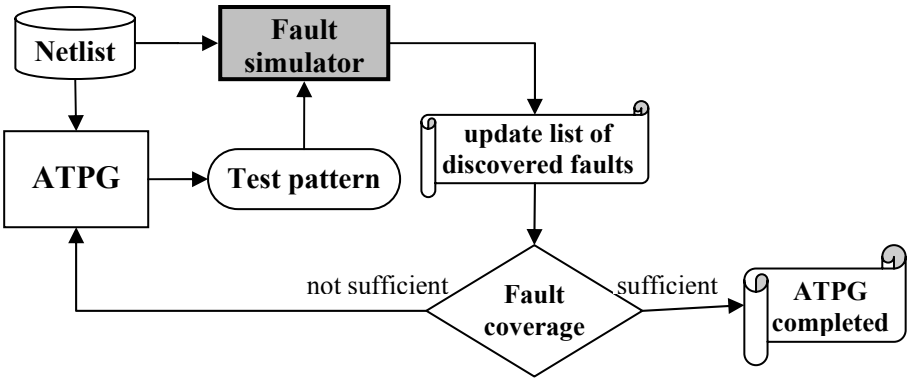


Figure 2.11: Fault simulator in conjunction with ATPG

generators (e.g. [38], [39], [40]) which typically cannot prove whether the produced test pattern detects any of new faults or not.

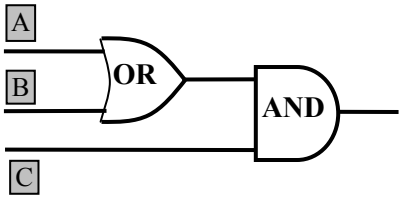
Genetic algorithms of ATPG ([41], [42]) use fault simulator to compute the fitness function that adjusts the generation process. There exist many other types of simulation-based ATPG (e.g. [43], [44]) that demand tight interaction with fault simulator to achieve the results.

Even in case of deterministic ATPG that target a specific fault (or faults), the fault simulation is applied as a kind of a post processing. In the latter case it helps to reveal the information about other faults (besides the targeted ones) possibly discovered by the test pattern.

2.4.3 Fault diagnosis and fault dictionaries

Another area of applicability of fault simulators is the problem of *diagnosis*. The testing of a device helps to decide whether the DUT is functioning correctly or not. However if the device fails to pass the tests, there is still no information about the cause of malfunction or probable location of defects. Nevertheless such info could be vital in case of repairing the system or improving the production yield.

The diagnostic information assists to determine the source and location of fault thus narrowing the suspected area and type of defect. One of the well-known methods to perform diagnosis is to use *fault dictionary* (this technique is also referred as *cause-effect analysis* [5], [4]). The fault dictionary contains a list of symptoms (i.e. failed test patterns and their output responses) and a specific fault (or group of suspected faults) associated with each symptom.



Test patterns:
t_1=101, t_2=001, t_3=011, t_4=100

Symptom				Suspected faults
t_1	t_2	t_3	t_4	
F	P	P	P	*SA0@A*
P	F	P	P	*SA1@A, SA1@B*
F	P	F	P	*SA0@C*
P	P	F	P	*SA0@B*
P	P	P	F	*SA1@C*

Figure 2.12: Fault diagnosis via fault dictionary

In Figure 2.12 a fault dictionary is constructed for sample circuit under diagnosis. During diagnostic test, the information about failed test patterns is recorded by a tester together with the test result. Then a diagnostic program searches the fault dictionary for the corresponding symptom and reports with the diagnosed fault.

A fault dictionary is straightforwardly built out of the fault table: for every fault the test patterns that detect this fault need to be stored as a fault symptom. The fault table in its turn is built by fault simulation which needs to be conducted without usage of fault dropping.

2.4.4 Test compaction

Usage of fault simulator can also help to optimize already available test program without decreasing the fault coverage. This is done for the sake of lessening the cost of test application time and the cost of storage of patterns in tester memory. The technique referred as *test compaction* is capable to achieve good results on reduction of initial test pattern set.

Test compaction exploits several fundamental ideas [11]. First, the method removes the test patterns that do not contribute into detection of new faults. In other words, if a test pattern detects faults that have been already detected by other test patterns, it can be removed without degradation of the test quality. The redundant test patterns can be identified by carrying out fault simulation of test set in reversed-order.

Another approach is based on the fact that ATPG typically targets a specific fault and thus lefts many of primary inputs undefined [45]. Hence by filling-in values for the unassigned inputs (don't care positions) test compaction algorithm attempts to increase the efficiency of the test pattern. Indeed the intelligent substitution of don't cares with the 0 and 1 can significantly raise the number of fault detected by single test pattern.

However the latter approach is heavily relied on fault simulator because it requires many runs of fault simulation over the different variations of test pattern set. This results from the fact that test compactor needs to choose the optimal substitution of undefined bits by zeros and ones.

2.4.5 Built-in self test

Built-in Self Test (BIST) is a widespread DFT technique which enables a silicon device with the additional functionality of self-testing. BIST is very attractive because it provides a number of advantages [4] including excellent test access to the internal structures of device, native at-speed testing, instant test application, low testing cost, ability to test device over its lifespan, etc.

A typical Logic BIST controller consists of a special pseudo-random test pattern generator (PRPG) and response analyzer. A version of Logic BIST that combines PRPG together with the set of pre-defined deterministic patterns is called *Hybrid BIST* approach. However the latter requires additional memory to be allocated on the device.

In the process of design of BIST controller an engineer is faced with number of challenges, e.g. the selection of optimal parameters for PRPG [5], [46] or delimitation of the bounds between deterministic and pseudo-random test [47]. As result, an exploration of the search space of BIST controller needs to be performed to achieve the better fault coverage or solve the tradeoff between the size of additional memory and the quality of test. Similarly to the previous task, the search for optimal solution normally requires many cycles of the fault simulation to be run under various conditions.

2.4.6 Contributions of current thesis

The current thesis introduces an application of fault simulation technique as well. In Paper IV presented in the last part of the thesis, fault analysis is incorporated into the procedure of composing the list of malicious faults (see Section 3.4) that is used in analysis of dependability of design.

The focus of co-authored papers ([VIII], [IX] and [X]) is a laboratory environment for research in the area of digital testing. Although, this topic have been not included into the main scope of current thesis we will try to outline the purposes and main features of the developed tools below.

The main goal of the presented tools is to study the different aspects of design and test using relatively small illustrative examples. In particular, in [VIII] the system for investigating issues of register-transfer level design and test is presented. In addition to wide set of features, it is capable to perform hierarchical fault simulation (on RTL and macro-level) in order to determine the fault coverage of functional testing. In [IX], the presented tool has been supplemented with the

possibility to measure and evaluate the effectiveness of various implementations of BIST controller.

Furthermore, the software package of laboratory tools [X] has been enhanced with the system that is capable to study the influence of real defects and draw the correlations between physical defects and fault models.

2.5 Chapter summary

The purpose of this chapter was to provide reader with the background information needed to understand the basic principles of fault analysis.

The first part has briefly specified the goal and concept of testing of digital device. Next the notations fault and defects have been explained and the fault modeling issues have been studied. The main emphasis was made on the description and properties of stuck-at fault model that is used in simulation methods described in the thesis.

The approaches described in the chapter represent the main types of fault simulation. The chapter briefly introduces with the advanced techniques that are intended to optimize fault simulation process.

It is also important to understand that fault simulation is a not standalone problem of test quality assessment. Instead, fault analysis is used as an auxiliary step of other test-oriented tasks. Indeed, several applications described in the last part of the chapter require many iterations of fault simulation for achieving the satisfactory results. The latter fact gives especially clear motivation for improving efficiency of fault simulator, since even small advance in fault analysis speed can significantly influence the overall time spent on finding the optimal solution.

Chapter 3

OVERVIEW OF RESEARCH RESULTS

This chapter gives an overview of the research results presented in Papers I-IV. The scope of the research is mainly focused on improving fault simulation methods for combinational circuits (or circuits enhanced with scan-path) with the usage of stuck-at faults for modeling defects. However, the intermediate steps of research were published in larger extent of papers (including [V], [VI] and [VII]), we have selected only four publications that contain the most important achievements.

The first paper that is called "***Efficient Single-Pattern Fault Simulation on Structurally Synthesized BDDs***" addresses the problem of single-pattern fault analysis. The paper studies the possibility to perform fault simulation on macro-level by introducing the version of deductive fault propagation algorithm refined for SSBDD model. Moreover, the paper proposes to use fault-free simulation on SSBDD for shrinking the set of initial fault list. Another contribution of the article is the idea to use topological analysis prior to fault simulation for accelerating propagation process for certain parts of the circuit. Since the implementation of the proposed method was not finished in the moment of publishing, the paper contains rather preliminary data, experimental results and estimations of the potential speed-up. The overview of Paper I together with final results is given in Section 3.2.

The next two papers are devoted to improving parallel-pattern simulation approach. The paper called "***Ultra Fast Parallel Fault Analysis on Structurally Synthesized BDDs***" presents several contributions. First, parallel critical path tracing algorithm is proposed for SSBDD graphs. This had allowed to use parallel CPT technique inside the fanout-free regions but on macro-level instead of gate-level. Another result is an approach to extend results of critical path tracing beyond

FFRs with the help of Boolean differentials. The latter requires building of a special calculation model in the phase of topological analysis of circuit. The presented method has achieved significant speed-up in comparison with other simulation tools.

The next paper that is called "***Parallel Exact Critical Path Tracing Fault Simulation with Reduced Memory Requirements***" continues to improve the technique proposed in Paper II. Two novelties are introduced: the way of optimization of calculation model and an approach for splitting the model into parts for carrying out simulation iteratively. The former results in higher simulation speed while the latter reduces overall memory consumption of fault simulator.

Since the both papers basically address the same topic and logically supplement each other, their overview is presented under the single Section 3.3.

The last paper titled "***Hierarchical Calculation of Malicious Faults for Evaluating the Fault-Tolerance***" is not directly devoted to fault simulation but rather to the application of fault analysis. In particular, the approach is intended for generation of malicious fault list used in dependability analysis. The fundamental idea of the method is to use hierarchy of Decision Diagrams to represent (and analyze) circuit on multiple levels of abstraction. This gives an opportunity to cope with the complexity problems but, at the same time, preserves the accuracy of gate-level evaluation. The brief overview of Paper IV is presented in Section 3.4.

Before proceeding with the description of the research results, it was decided to give an overview of SSBDD model that is used to represent circuit on macro level (Section 3.1). This is required since the presented algorithms rely on SSBDD graphs for carrying out fault simulation.

In addition, Section 3.5 summarizes the results of experiments carried out to compare the proposed fault simulation methods with each other and state-of-the-art tools. This is done because the experimental data published in Papers I-IV have been obtained on different platforms hence are not easily comparable.

3.1 Representation of circuit on macro-level

In contrast with the conventional approaches that use gate-level netlist for describing of a digital circuit on logical level, the methods presented in the thesis use a slightly higher level of abstraction called *macro-level*. A circuit on macro-level is described by using a special form of Binary Decision Diagrams (BDDs).

Structurally Synthesized BDDs (SSBDDs) firstly proposed in 1976 [48] have been successfully used in the field of design and test. Unlike the traditional BDDs [49], [50], the distinctive feature of SSBDD is the ability to keep the information about the structure of a modeled circuit (whereas traditional BDDs represent only a logical function). Besides this, SSBDD model has linear

complexity in respect to number of gates in original circuit (the worst-case complexity of traditional BDDs is exponential). Furthermore, in [51] it was proven that the size of SSBDD model is always smaller than the size of logic-level netlist it has been generated from.

Another property of SSBDD model is a built-in fault collapsing for stuck-at fault model [52]. The latter feature makes SSBDD be especially attractive for usage in conjunction with stuck-at fault model and avoids the performing fault collapsing explicitly. On the other hand, fault resolution remains the same as with gate-level models that use fault collapsing technique described in Section 2.2.4.

A comprehensive research that had been made to study the applicability of SSBDDs in the field of testing of digital circuits have found it feasible to use the model for tasks of test generation [53], logic and multi-valued simulation [52], timing simulation [54],[55], design error diagnosis [56], etc. The preliminary comparison of a serial fault simulation on macro and gate levels [52] had shown the potential of the SSBDD-based approach also for fault analysis. In the current thesis we attempt to extend the usage of SSBDD model for advanced fault simulation techniques.

Structurally Synthesized Binary Decision Diagram is a planar, acyclic BDD that is obtained by superposition of elementary BDDs for logic gates. While traditional BDDs are generated by Shannon's expansions that extract the function of the logic, SSBDD models extracts both, function and data about structural paths of the circuit. A digital circuit is modeled as a system of BDDs, where for each of tree-like fanout-free regions a separate SSBDD is generated.

An SSBDD G is a triple (M, X, Γ), where M is a set of nodes, $X(m)$ is a function, which defines line variables labeling the node m and $\Gamma(m, e)$ is a function, which gives the successor node of m with $X(m)=e$, $e \in \{0, 1\}$. The set of nodes $M=M_N \cup M_T$ is divided into a set of nonterminal nodes $M_N=\{m_0,..,m_k\}$ and a set M_T that contains 0- and 1-terminals nodes (m_{T0} and m_{T1} respectively).

SSBDD graphs for gate-level digital circuits are created as follows. Starting from the output of the FFR, logic gates are recursively substituted by their respective elementary BDDs. The procedure of superposition terminates in those nodes, which represent a primary input or a fanout branch.

In Figure 3.1 a logic circuit with an output line Y and its corresponding SSBDD graphs are depicted. Note, that the direction of an edge (down or right) corresponds to respective 0- or 1-label (thus labels are omitted). The illustrated terminals nodes (0- and 1-nodes) can be also omitted: the exiting the BDD downwards corresponds to 0 and rightwards to 1. In addition, SSBDD nodes can also be labeled by inverted variables (e.g. $\overline{C}_1$ in Figure 3.1).

Let us denote $\Gamma(m, e)$ by m^e. Then m^0 is the successor of m for the value $X(m)$=0 and m^1 is the successor of m for the value $X(m)$=1. By the value assignment, we say that the edge between nodes m and m^e is *activated.* Consider a situation where

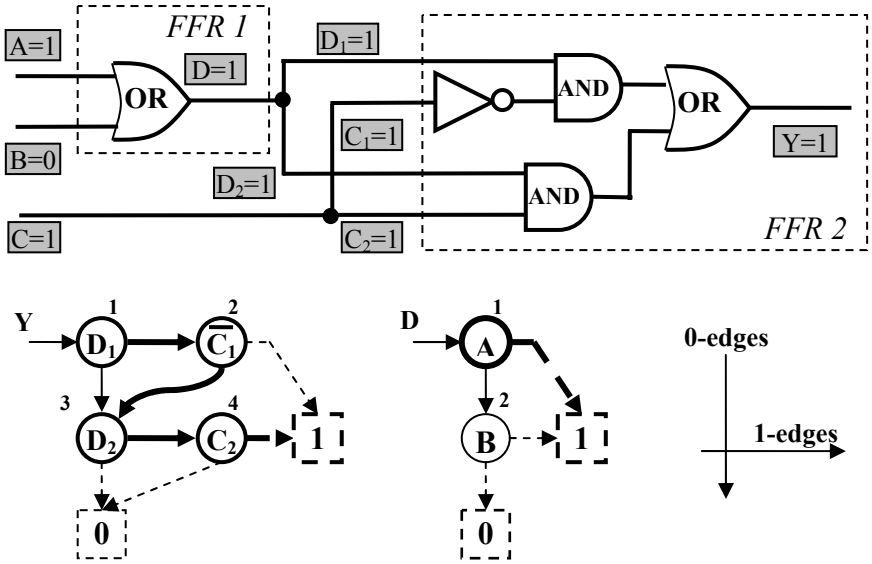


Figure 3.1: A circuit and its SSBDD representation

all the variables $X(m)$ are assigned by a Boolean vector $X^t \in \{0,1\}^n$ to some value. The edges activated by X^t form an *activated path* $l_{act}=(m_0, ..., m_T)$ from the root node m_0 to one of the terminal nodes $m_T \in M_T$. The bold edges in Figure 3.1 indicate the activated paths inside SSBDD for current input vector.

There also exists a directed path l through all non-terminal nodes. The latter path levelizes the nodes by numerical labels $n(m)$ so that each $m \in M$ has label $n(m)$ with number higher than its predecessors on path l. There is one-to-one mapping between the nodes in SSBDD and signal paths of circuit. For instance, the node D_1 in Figure 3.1 corresponds to signal path with the beginning in fanout branch D_1 to the output Y.

In SSBDD graph stuck-at faults are modeled at non-terminal nodes. The presence of a fault inside the node $m \in M_N$ permanently fixes the successor node to m^0 (down edge) for *SA0*, or m^1 (right edge) for *SA1*, regardless of the value $X(m)$. Thanks to one-to-one correspondence between the nodes and signal paths, stuck-at fault modeling on SSBDD is almost identical to fault collapsing technique described Section 2.2.4 (this has been shown in [52]).

To conclude with, we will outline the main advantages of SSBDD model. First, the model allows to rise the level of abstraction of circuit representation from the set of gates to more coarser network of macros. At the same time, the gate-level accuracy is preserved since the fault resolution remains the same as for collapsed stuck-at fault model for gate-level circuits. Moreover, the automatic fault collapsing integrated in SSBDD model, avoids the explicit checking whether a fault is included into the collapsed list or not.

3.2 Single-pattern fault simulation

This section presents a novel approach for conducting single-pattern fault simulation on SSBDD model. Although the comprehensive description of the method is provided in Paper I, this section gives an outline of key aspects and presents final results that have not been published in the original paper.

Unlike parallel-pattern fault simulation methods (Section 2.3.2.2), the presented technique is most efficient for the cases when only one (or few) patterns need to be simulated at a time. The latter condition lessens the attractiveness of parallel-pattern fault analysis because the efficiency of parallelism cannot be exploited in full extent. In particular, single-pattern fault analysis is especially relevant for simulation of synchronous sequential circuits as well as for the problems of test pattern generation and test compaction (Sections 2.4.2 and 2.4.4). For these tasks the next pattern is typically issued only after the analysis of the fault detectability of the preceding pattern. Moreover, unlike parallel-fault analysis (Section 2.3.2.1) the presented algorithm has no limitations on using of fault dropping.

In general, the proposed algorithm consists of the following steps:

1. Fault-free simulation on SSBDD and identification of fault candidates
2. Topological analysis to determines the type of reconvergency for each FFRs
3. Activation of faults and propagation of fault lists through SSBDD

3.2.1 Fault-free simulation on SSBDDs and fault list reduction

Fault-free simulation on SSBDD model is performed straightforwardly by consequent evaluation of graphs corresponding to the primary outputs of circuit. The nodes are traversed starting from the root node m_0. Each successor node m^e is selected depending on the value of variable $X(m)$ that labels the node. The simulation of each SSBDD graph ends by reaching one of the terminal nodes m_T. However if the value of non-terminal node depends on the other SSBDD (i.e. node is not a primary input) the simulation recursively proceeds to the underlying graph.

During fault-free simulation the two goals are achieved: true values are determined for internal signal lines (by path activation in SSBDD) and the candidate faults are identified. The proposed single-pattern fault simulation method rely on the logical operations with fault sets, hence the sizes of fault lists significantly influence the overall speed of the method. For this reason, keeping the number fault candidates as less as possible is vital.

We propose to reduce the number of potential faults by applying the following rules:

- As it was mentioned above, the nodes of a graph that are traversed by fault-free simulator establish an *activated path* inside the graph. Obviously,

the only faults with sites at the nodes that belong to the activated path can affect the result of graph evaluation. For instance, the fault at node *B* (Figure 3.1) cannot change the traversed path under current vector X^t thus does not influence the output value *Y*.

- In case if a SSBDD avoids recursive evaluation, all the nodes of the graph are excluded from the list of potential fault sites. However, a true-value simulation is still needed in order to ensure the correct function of fault propagation algorithms.
- If the value of successor function of a node does not equal to the output value of graph the fault at this node cannot affect the result of graph evaluation (e.g. node C_1 in Figure 3.1). This comes from the property of planarity of SSBDDs.

However for a single test pattern there is no need to consider both *SA0* and *SA1* faults in each fault site. Instead, the only single fault (i.e. the one that changes the successor node) should be taken into account.

The complete description of the algorithm is provided in Paper I.

3.2.2 Deductive fault propagation on SSBDDs

The deduction-based fault analysis procedure can be divided into two stages *fault activation* and *fault propagation* (Section 2.3.4). Fault activation finds out which of the faults with the locations in fault candidate nodes of FFR affect the output of this FFR. Fault propagation in its turn derives the fault set for the output of FFR from the lists of faults on its inputs.

The activation of faults is done straightforwardly. We can either invert the corresponded node (to imitate fault) and re-simulate SSBDD (see Paper I), or perform critical path tracing on SSBDD (see Section 3.3.1).

Below we propose deductive algorithm that is used for propagation of fault lists through FFR represented by SSBDD graph. First we define notations as follows:

- M_1 - set of nonterminal nodes at the activated path l_{act}
- $m' = m^{X(m)}$ $m'' = m^{\neg X(m)}$
- m_{TF} – terminal node that corresponds to faulty value of FFR $\{ m_{TF} \in M_T \mid X(m_{TF}) \neq f(X^t)\}$
- $S(m)$ – fault list propagated to m from previous SSBDDs.
- $L(m)$ – temporary list of faults propagated inside SSBDD to node m.
- $N(m)$ – set of nodes succeeding node m in directed path l through all the nodes of SSBDD
- R – resulting set of the faults propagated to the output of FFR.

The following observations lay in the basis of the algorithm:

1) faults in fault propagation list *S(m)* of a non-terminal node *m* that belong to activated path l_{act} are added to the temporal list *L* of a faulty successor node. However the temporal list should not contain the faults already propagated to the succeeding nodes in the graph.

$$L(m'') = L(m'') \cup S(m) \qquad (1)$$
$$L(m'') = L(m'') \cup (S(m) \setminus \bigcup_{k \in N(m)} L(k))$$

2) if a fault in temporal list of a node *m* that does not belong to activated path l_{act} is not included into fault propagation list *S* of this node, then such fault is added to the temporal fault list *L* of a true-value successor node.

$$L(m') = L(m') \cup (L(m) \setminus S(m)) \qquad (2)$$

3) if fault in temporal list of a node that does not belong to activated path is included into fault propagation list *S* of this node then such fault is added to the temporal fault list *L* of a faulty-value successor node

$$L(m'') = L(m'') \cup (L(m) \cap S(m)) \qquad (3)$$

After performing the above-listed steps for all non-terminal nodes, the fault list associated with terminal node that corresponds to the faulty value of the graph is treated as the resulting set of propagated faults:

$$R = L(m_{TF}) \qquad (4)$$

The overall algorithm of deductive for fault list propagation on SSBDD is provided below:

```
R =Ø
for each m ∈M_N
        L(m)=Ø
end for
for each m ∈M_N
        if m ∈M_1 then
                T =∪L(n), for n ∈N(m).
                L(m'')= L(m'') ∪(S(m) \ T)
        else
                L(m')= L(m') ∪(L(m) \ S(m))
                L(m'')= L(m'') ∪(L(m'') ∩S(m))
        end if
end for
R = L(m_TF)
```

The deductive propagation of faults on SSBDD has advantage over classical gate-level deductive fault simulation because it does not require evaluation of gate type and inputs. Instead of that, the propagation is done on uniform model using

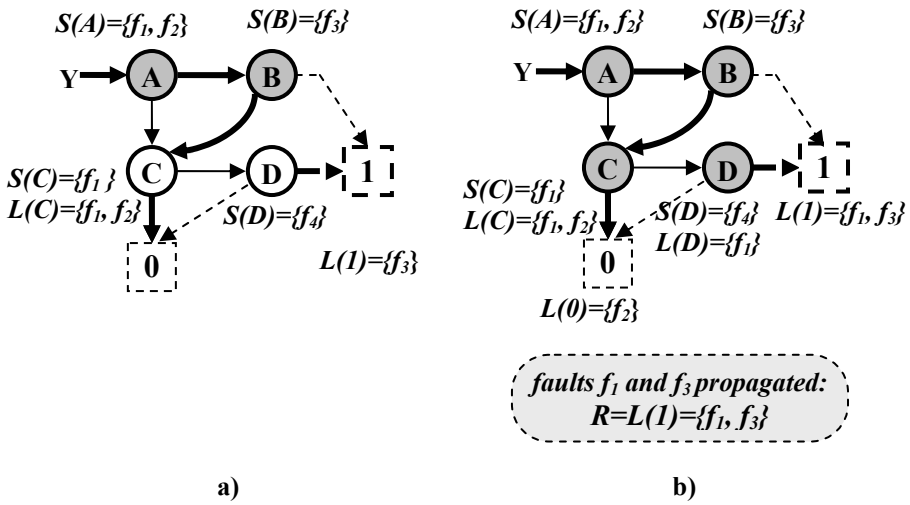


Figure 3.2: Example of deductive fault effect propagation on SSBDDs

small set of rules. Moreover, the complexity of SSBDD is always less than the complexity of corresponded gate-level netlist, thus processing of SSBDD graph will require fewer steps than evaluation of corresponded gate-list.

The preliminary version of the same algorithm that is presented in Paper I is less compact because it does not generalize the notion of temporal sets *L* also for terminal nodes.

Figure 3.2 illustrates the example of fault list propagation through SSBDD. Bold lines represent next successor for each of the node. In Figure 3.2a the state of the fault sets after evaluation of nodes *A* and *B* is depicted whereas Figure 3.2b shows the final results of execution of algorithm. At the end, the set *R* contains the faults that propagate to the output of corresponded FFR.

3.2.3 Circuit analysis and fault list propagation cases

Normally, deductive algorithm is required for the propagation of fault lists through a circuit. However in Paper I it was pointed out, that for some parts of circuit sophisticated deductive propagation can be replaced by faster methods. For this reason, we propose to perform a special analysis of topology of circuit that categorizes each of FFR into one of three types described below (see examples in Figure 3.3).

1) There is no fanout that converges at SSBDD (Figure 3.3a) hence no fault can appear on different inputs simultaneously:

$$(S(m_i) \cap S(m_j)) = \varnothing, \text{ where } m_i, m_j \in M_N,\ i \neq j$$

As result the propagation of fault lists can be performed in the moment of fault activation. If fault activated at a node changes the result of SSBDD evaluation, the fault list associated with this node is propagated through graph together with the activated fault.

2) FFR is a reconvergency of a fanout located just behind, i.e. there is no other FFR between the fanout point and this FFR (Figure 3.3b). The fault effect propagated to such fanout will indispensably affect all the nodes corresponded to reconvergent lines:

$$\begin{cases} S(m_i) = S(m_j), \text{ for } m_i, m_j \text{ that correspond to the fanout branches} \\ (S(m_i) \cap S(m_j)) = \varnothing, \text{ for the rest } m_i, m_j \in M_N,\ i \neq j \end{cases}$$

To determine whether the fault list associated with the fanout propagates to output of FFR, the simulation of graph should be repeated with the inversion of all nodes corresponding to recovergent lines. For the rest of nodes, the propagation of fault lists is made exactly as for the previous case. The description of concurrent algorithm that is used for propagation is provided in Paper I.

3) For the rest of circuit (FFRs with arbitrary reconvergencies, Figure 3.3c) deductive algorithm is applied.

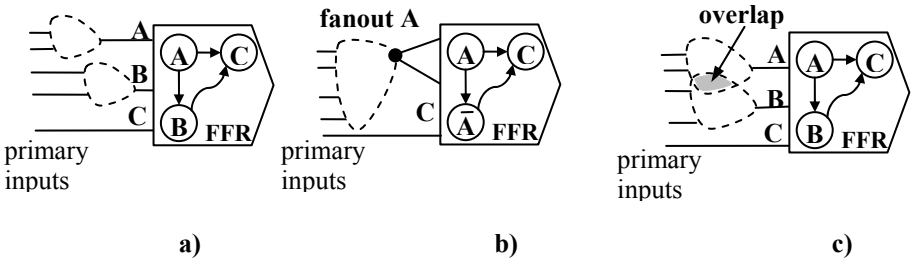


Figure 3.3: Types of reconvergencies

3.2.4 Experimental results

A number of experiments were conducted in order to confirm the feasibility of proposed approach. However, the data published in Paper I is rather result of preliminary estimation while the actual comparison of fault simulation speeds has not been included into the article. This comes from the fact that the implementation of the proposed method had not been finished prior to the publishing. In this thesis we will try to fill this gap by providing the final experimental results.

The experiments on potential fault list reduction had shown that only 40% of total faults sites (in average for ISCAS'85 benchmarks [61]) need to be considered during single-pattern fault simulation. Moreover, due to the fault collapsing

provided by the SSBDD model itself (66% in average for ISCAS'85) the number of faults reduces to 26%. Since the only one stuck-at fault per fault site (either SA1 or SA0, depending on fault-free value of signal line) need to be activated, the size of fault list collapses by 2 times, down to 13%.

The distribution of SSBDD graphs according to the types described in Section 3.2.3 is presented in Table 3.1. These are the final results in contrast to the estimation carried out in Paper I. As it can be seen from the table, the large portion of SSBDDs does not require the application of deductive algorithm. For 31% of graphs (in average) the fault list can be propagated by means of faster concurrent algorithm or directly together with the fault activation.

Table 3.1: Distribution of SSBDD graphs by type of reconvergency

circuit	total SSBDDs	SSBDDs w/o reconv.		max reconv. depth = 1		deep reconv.	
c432	96	58	60%	0	0%	38	40%
c499	187	41	22%	32	17%	114	61%
c880	151	75	50%	8	5%	68	45%
c1355	291	73	25%	0	0%	218	75%
c1908	248	79	32%	22	9%	147	59%
c2670	430	196	46%	15	3%	219	51%
c3540	378	129	34%	11	3%	238	63%
c5315	633	204	32%	43	7%	386	61%
c6288	1488	303	20%	0	0%	1185	80%
c7552	920	212	23%	5	1%	703	76%

The comparison of speed of single-pattern fault analysis method with several other fault simulators is presented in Table 3.4 (see Section 3.5). Even though in case of 10000 test patterns the results of single-pattern fault simulation are worse than for parallel-pattern approach, in the analysis of fault detectability of a single test pattern the proposed algorithm becomes more advantageous.

3.2.5 Conclusions

The main contribution of Paper I is a novel single-pattern fault simulation approach that uses SSBDD. The results, published in Paper I are basing on preliminary observations and estimate the potential of the proposed approach. However in the current thesis we have included the final measurements of the actual efficiency of this method. Finally, it turned out that the proposed optimizations are capable to reduce the list of fault candidates by 87% and avoid sophisticated deductive propagation in 39% of cases (in average). The speed gain of single-pattern algorithm (see Table 3.4) was about 3.6 times in comparison with parallel-pattern analysis for the case when the patterns are available one-by-one.

3.3 Parallel-pattern fault simulation

In this section the shortened description of parallel-pattern fault analysis methods proposed in Paper II and Paper III is given. In contrast to single-pattern approaches, parallel-pattern fault simulation is most suitable for the tasks where large amount of test patterns need to be processed. This is especially relevant for building fault dictionaries for diagnosis, evaluation of test quality or adjusting parameters of BIST controller (see Sections 2.4.1, 2.4.3 and 2.4.5).

The presented approach partially relies on SSBDD model: it uses SSBDD to perform parallel critical path tracing (CPT) inside FFRs and exploits the automatic fault collapsing provided by the model. However the techniques that are used to extend the results of critical path tracing for an arbitrary circuit are virtually independent on the underlying representation of circuit.

3.3.1 Critical Path Tracing on SSBDDs

Traditional Critical Path Tracing (CPT) technique (Section 2.3.5) is a very powerful mechanism for analyzing detectability of faults. Below we are proposing an algorithm that implements parallel critical path tracing approach for SSBDD model. The described technique has been presented in Paper II.

Parallel CPT on SSBDD is conducted as follows. At first, parallel fault-free simulation is performed to determine output value of FFR. Then, fault candidates are identified by parallel evaluation of active paths inside SSBDD. Finally, CPT itself is conducted to recognize which nodes of SSBDD (and corresponding signal lines) are critical to the output of the respective FFR. Below all the three steps are explained.

1) In order to simulate a test set $T=(t_1, \ldots, t_n)$ on G_y representing a FFR $y=f(x)$, we start from the node with the highest label $n(m)$, and repeat the vector operation for each of the nodes:

$$D(m) = (x(m) \wedge D(m^1)) \vee (x(m) \wedge D(m^0)) \qquad (5)$$

The obtained $D(m)$ values can be interpreted as a result of path activation in G_y in case if m would be a root node of G_y. Therefore $D(m_0)$ represent the value of y calculated for the root node graph G_y. Note that the initial values of terminal nodes are: $D(m_{T0})$=00...0 and $D(m_{T1})$=11…1 respectively.

2) For each test pattern $t_k \in T$, the nodes m in G_y are found which belong to the activated paths L_k. Only the nodes belonging to active path may influence the value of $y(t_k)$. Hence, only the nodes $m \in L_k$ may be the candidates for fault detection. To find the candidates for fault detection, the nodes m are processed in the direct order as follows:

$$L(m^1) = L(m^1) \vee [L(m) \wedge x(m)], \qquad (6)$$
$$L(m^0) = L(m^0) \vee [L(m) \wedge \overline{x(m)}].$$

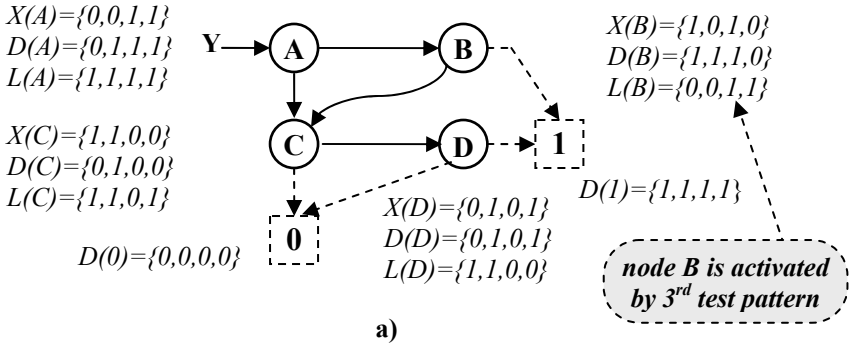
The initial values are: $L(m_0)$=11...1 and $L(m_i)$=00...0 for all other nodes $m_i \in M$. The value of $L_k(m)$=1 means that $m \in L_k$, i.e. the node m belongs to path activated by the test pattern t_k.

3) In the last stage, we carry out parallel critical path tracing to find out at which nodes of activated paths L_k the faults are detected. Detectability of faults at m in the FFR represented by G_y at the output y is calculated by using the following formula:

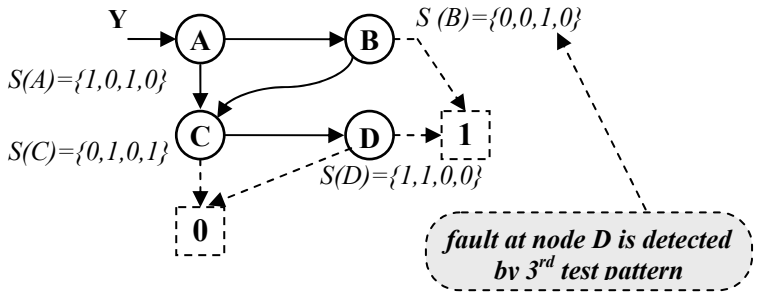
$$S(m) = L(m) \wedge (D(m^0) \oplus (D(m^1))) \qquad (7)$$

The fault at node m is detected by t_k iff the value of the vector component $S_k(m)$ is 1.

Similar to critical path tracing on gate-level, this approach has linear-time complexity in respect to the number of nodes in SSBDD graph. Likewise, the applicability of the proposed technique is restricted by the region that does not contain reconvergencies.



a)



b)

Figure 3.4: Parallel critical path tracing on SSBDDs

An example of calculating the vectors $D(m)$, $L(m)$ and $S(m)$ for 4 test patterns is illustrated in Figure 3.4. In the first part (Figure 3.4a) vectors *D(m)* and *L(m)* are calculated by using vector of values *X(m)*. The results of calculation can be interpreted as follows: $D_3(A)=1$ means that the fault-free value of output *y* is equal to 1 for the third test pattern. The value $L_3(B)=1$ indicates that node *B* is activated by third test vector.

Figure 3.4b shows the final result of CPT for given SSBDD. For example, the value *1* in third position of vector *S(B)* (i.e. $S_3(B)=1$) is interpreted as detectability of fault at *B* by 3rd test pattern. In other words, during the application of this test pattern the value of output *y* differs from the fault-free value if the value of signal line *D* has changed due to the fault.

3.3.2 Extending the results of CPT beyond FFRs

Parallel critical path tracing on SSBDD described above computes the detectability of faults inside FFR. In order to perform fault analysis on an arbitrary circuit we need to generalize these results beyond fanout-free regions. The latter is done with the aid of partial Boolean differentials. In this overview only the final results are presented, while the explanations are provided in Paper II and Paper III.

Consider fanout-free region F_y depicted in Figure 3.5a. Its schematic representation is illustrated in Figure 3.5b where the edges denote paths inside circuit without fanouts and the nodes are fanout-free regions.

Let us define Y_kX as the sensitivity of output *y* to the signal change at line *x* (which is a k^{th} input of FFR). Then $Y_1X=1$ iff change of *x* flips the value of *y* (in terms of Boolean derivatives described in Paper II this corresponds to the notation $\frac{\partial y}{\partial x}=1$). The value Y_kX can be obtained by evaluation of expression (7), with *m* corresponding to k^{th} input of FFR (see previous subsection).

a) **b)**

Figure 3.5: Critical path tracing in FFR

To compute the sensitivity Y_1Z for the case of two consecutive FFRs (Figure 3.6), the sensitivities Y_1X and X_2Z need to be calculated first and the final sensitivity Y_1Z is computed by the conjunction:

$$Y_1Z = Y_1X \wedge X_2Z \qquad (8)$$

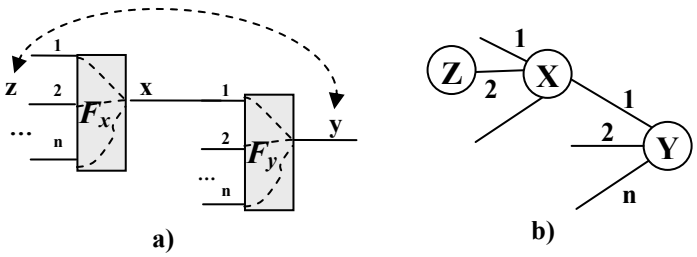


Figure 3.6: Consecutive FFRs

In Figure 3.7 the case of reconvergent fanout is illustrated. If variables $x_j...x_n$ are not influenced by x, then according to the theorem proven in Paper II, the derivative is calculated by using the following expression:

$$\frac{\partial y}{\partial x} = y \oplus F_y(x_1 \oplus \frac{\partial x_1}{\partial x},...,x_i \oplus \frac{\partial x_i}{\partial x}, x_j,...,x_n) \qquad (9)$$

Assuming there are no reconvergencies between x and x_1 as well as between x and x_2, then the values $\frac{\partial x_1}{\partial x}$ and $\frac{\partial x_i}{\partial x}$ are obtained by using of (7) or (8). Similarly we can denote the results of (9) in shorter form:

$$YX = F_y(Y_1X,\ Y_iX) \qquad (10)$$

The resulting value YX will reflect the sensitivity of output y to the presence of fault at fanout x.

This result can be generalized for the case of nested reconvergencies and consequently be applied to an arbitrary circuit with any set of reconvergencies (see Paper II).

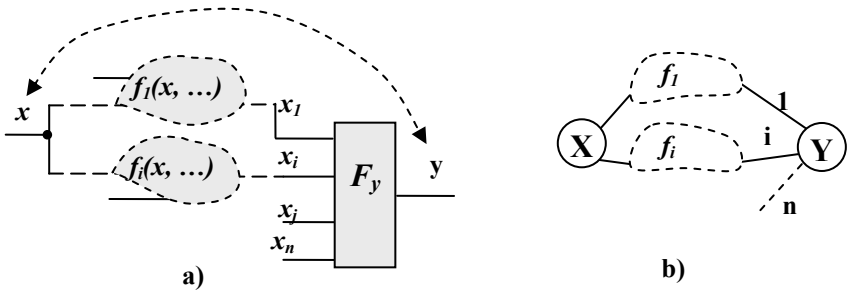


Figure 3.7: Reconvergent fanout

3.3.3 Construction of calculation model

As it was already mentioned, for an arbitrary design parallel critical path tracing needs to be augmented with some sort of parallel simulation (see Section 2.3.2) for analyzing circuit beyond FFRs. Indeed, the latter typically brings in an unnecessary re-simulation of already simulated gates.

However several optimizations have been proposed in order to reduce the amount of redundantly performed simulation. The basic solution is to restrict the area simulated for detection of a fault by the gates physically reachable from the fault site. The more advanced methods help to identify the "stop-points" where the parallel fault analysis can be interrupted [20],[22],[23],[24] or prune the unnecessary simulated regions [21].

In this section we are presenting a distinctive method for fault simulation of circuit with reconvergent fanouts. In particular, we propose to explicitly construct calculation model that is used to compute the detectability of each fanout. This approach not only avoids the unneeded simulation of the area that is not reachable by fault effect, but also tends to lessen the number of repeated calculations. The formulas described in the previous subsection are used as basic building blocks of calculation model for computing sensitivities between signal lines.

Two types of calculation model have been proposed: non-optimized model (Paper II) and the optimized one (Paper III).

Let us consider a construction of non-optimized model first. A non-optimized model is constructed for each of the primary outputs separately thus resulting in the set of sub-models.

Consider the reconvergency graph $G=(N, \Gamma)$ of example circuit in Figure 3.8 where N is the set of nodes and Γ represents the following mapping on set N:

- $\Gamma(x) \subset N$ is a set of successor nodes of node $x \in N$,
- $\Gamma^{-1}(x) \subset N$ is a set of predecessors of node $x \in N$,
- $\Gamma^{-1}(x_i) \subset N$ is a predecessor node connected to the input i of the node $x \in N$,
- $\Gamma^{*}(x) \subset N$ is a transitive closure of $\Gamma(x)$, and
- $\Gamma^{*-1}(x) \subset N$ is a transitive closure of $\Gamma^{-1}(x)$.

By topological analysis the sets of reconvergencies FS_y (for every primary output y) and the set of converging points (CP) are determined first.

$$FS_A=\{1,2,3\},\ FS_B=\{0,2,3\},\ CP=\{A, B, C\}$$

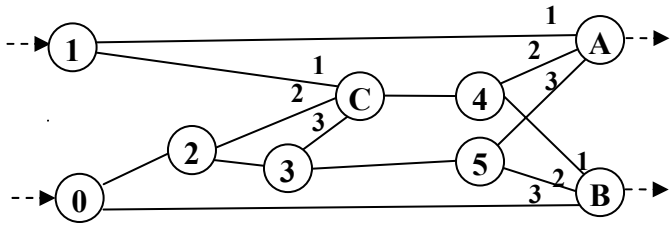


Figure 3.8: Reconvergency graph for example circuit

Then the formulas are constructed in the following way. For each primary output *y* the nodes are evaluated consequently in the direction from the output to primary inputs. The following steps are performed for each node *n* under evaluation:

1) For each i^{th} input of corresponding FFR the sensitivity formula N_iM is created, where node $m=\Gamma^{-1}(n_i)$.

   Example: for node *A* (Figure 3.8) three formulas A_11, A_24, A_35 are created. Because there are no reconvergencies between *n* and *m*, the sensitivity formulas also correspond to sensitivities of *A1*, *A4* and *A5*.

   If a node *n* does not belong to set FS_y, then for each $m \in \Gamma(n)$ the formula is constructed by using the rule (8) of two consecutive paths *(n,m)* and *(m,y)*:

   $$YN=YM \wedge MN.$$

   Note, because node *m* has been evaluated before node *n*, the sensitivity formula *YM* had been already created.

   Example: consider node *C* (while construction of sub-model for output *B*). The formula $BC=B4 \wedge 4C$ is constructed in order to compute the sensitivity of node *B* in respect to fault at node *C*.

2) If a node *n* belongs to set FS_y, then the converging paths are traced back towards primary outputs and for each of converging point *m* the respective formulas corresponding to (9) are created. The arguments for the formulas are constructed directly during the trace (in the similar way as for the previous cases but with exception is that sensitivity is computed in respect to inputs of converging FFR instead of primary output).

   Example: consider analysis of node *1* (while constructing of sub-model for output *A*). This node has the only converging point at FFR corresponding to primary output *A*. Tracing back to primary output *A* results in two converging paths *(1,A)* and *(1,C),(C4),(4,A)*. Correspondingly, the formula $A1=F_A(A_11, C1 \wedge 4C \wedge A_24)$ is constructed during this step.

Finally the computed sensitivities are united for all of primary outputs in order to compute the general detectability of fanouts. For instance, the detectability of fault at node *1* can be expressed as a union of sensitivities *A1* and *B1*.

The complete calculation model constructed for reconvergency graph in Figure 3.8 is presented in Table 3.2. In the left and right parts of the table, the formulas relating to sub-models for primary outputs *A* and *B* are given respectively.

The first column of each part indicates the number of step during the evaluation of the model. Note that some formulas require several steps for evaluation (here we treat the calculation of single sensitivity, each operation of conjunction or disjunction and evaluation of formula for reconvergent fanout as separate steps). However different steps are processed with unequal amount of time, the total number of steps in the model help to roughly estimate its complexity. The second column of each part contains the name of a node which evaluation resulted in the construction of formula in the third column.

Table 3.2: Calculation model

For output A			For output B		
step	***n***	**sensitivity formula**	**step**	***n***	**sensitivity formula**
1	***A***	$A_1 1$ (A1)	**27**	***B***	$B_1 4$ (B4)
2	***A***	$A_2 4$ (A4)	**28**	***B***	$B_2 5$ (B5)
3	***A***	$A_3 5$ (A5)	**29**	***B***	$B_3 0$
4	***5***	$5_1 3$ (53)	**30**	***C***	$BC=B4 \wedge 4C$
5	***4***	$4_1 C$ (4C)	**31**	***3***	$B3=F_B(\mathbf{C_3 \wedge 4C} \wedge B4, 53 \wedge B5)$
6	***C***	$AC=A4 \wedge 4C$	**35**	***2***	$B2=F_B(C_2 2 \wedge 4C \wedge B4, 32 \wedge 53 \wedge B5)$
7	***C***	$C_1 1$ (C1)	**40**	***1***	$B1=BC \wedge C1$
8	***C***	$C_2 2$	**41**	***0***	$B0=F_B(20 \wedge \mathbf{C2} \wedge 4C \wedge B4,$ $20 \wedge 32 \wedge 53 \wedge 5B, B0)$
9	***C***	$C_3 3$ (C3)			
10	***3***	$A3=F_A(\mathbf{C3 \wedge 4C} \wedge A4, 53 \wedge A5)$	**Union of sensitivities for all outputs**		
14	***3***	$3_1 2$ (32)			
15	***2***	$C2=F_C(C_2 2, 32 \wedge C_3 2)$	**48**		$0=A0 \cup B0$
17	***2***	$A2=F_A(C2 \wedge 4C \wedge A4, 32 \wedge 53 \wedge A5)$	**49**		$1=A1 \cup B1$
22	***2***	$2_1 0$ (20)	**50**		$2=A2 \cup B2$
23	***1***	$A1=F_A(A_1 1, C1 \wedge 4C \wedge A4)$	**51**		$3=A3 \cup B3$
26	***0***	$A0=A2 \wedge 20$	**52**		$C=AC \cup BC$
			53		$4=A4 \cup B4$

The consequent evaluation of formulas eventually gives us the sensitivity values for all fanouts in the circuit. Note that all calculations could be carried out on vectors instead of single values thus making the model be fully suitable for parallel-pattern processing.

The key advantage of the proposed method is an ability to effectively handle nested reconvergencies. For example, the calculation of detectability of fault in

fanout node *0* on primary output *B* (step 41), also involves the calculation of detectability of the same fault in converging node *C*. However, instead of re-simulation of corresponded FFR, the result of evaluation of sensitivity formula *C2* is used (step 15).

3.3.4 Construction of optimized model

The calculation model presented above has the evident drawback as it allows relatively large part of computations to be repeated. In particular it comes out of the fact that the process of construction is done separately for each of the primary outputs. In addition while converging paths are traced back the arguments of sensitivity formulas for reconvergent fanouts are likely to overlap. For example, the operation of conjunction $C3 \wedge 4C$ is performed several times (steps 10 and 31 in Table 3.2). Hence, the research conducted in [VI], [VII] was aimed to further optimization of the model.

In [VI] it was proposed to concurrently construct a unified model for all primary outputs. Furthermore, the concepts of activity and sensitivity vectors have been proposed in [VII] together with the method which lessens the overlap in calculations. Finally, the approach was generalized in [III] (Paper III).

Below we will describe the process of construction of optimized calculation model. First, the additional topological analysis is performed in order to build the following sets of nodes for reconvergency graph:

- *OUT* – set of nodes corresponded to primary outputs
- *RO* – set of nodes that corresponds to reconvergent fanouts
- For each $x \in RO$, the set *RI(x)* is also constructed which contains the convergent nodes of fanout *x*.

Optimized model is constructed for all outputs jointly. For this purpose, all the nodes *N* of reconvergency graph *G* are levelized and put into ordered set *N**. First, the ordered set *N** includes all primary outputs of circuit. Then, each node $n \in N$ is included into set *N** as soon as $\Gamma(n) \in N^*$ becomes valid.

For the example of reconvergency graph in Figure 3.8 the following sets are constructed:

$$OUT=\{A, B\},\ RO=\{0, 1, 2, 3\},$$

$$RI(0)=\{B\},\ RI(1)=\{A\},\ RI(2)=\{C,B,A\},\ RI(3)=\{B,A\},$$

$$N^*=\{A,B,5,4,C,3,2,1,0\}$$

The formulas of optimized calculation model are distinguished by their types as:

- *activity vectors* – express the sensitivities of internal nodes of reconvergency graph,

- *partial sensitivity vectors* – express detectability of fault at one of the primary outputs,
- *full sensitivity vectors* that express the global detectability of fault on any of primary outputs.

For each node n of the levelized set N^* the formulas are constructed in the following way (this procedure is briefly described in Paper III and more thouroughly in [VII]).

Step 1: Construction of activity vectors.

Let define set M as $M=\Gamma(n)$, set P_{nested} of pairs (x,y), $P_{nested}=\varnothing$ and set $M'=\varnothing$.

Step 1a:

for each node $m \in M$ and $y \in \Gamma(n)$ construct the activity vector $Y_iN=Y_iM \wedge MN$ if there exist nodes $x \in RO$ and $y \in RI(x)$ such that for subgraph G' consisting of nodes $N'=(\Gamma^{*-1}(y) \cup y) \cap (\Gamma^*(x) \cup x)$:

a) $m \in N'$, $n \in N'$ (i.e. nodes n and m belongs to converging path between x and y)

b) G' does not contain other nested reconvergency subgraph $G''(N'', \Gamma)$, $N'' \subset N$, with $n \in N''$, $m \in N''$ except that the reconvergencies that are formed by pairs of nodes in set P_{nested}.

For constructed activity vector $Y_iN=Y_iM \wedge MN$ the i is selected with accordance of $\Gamma^{-1}(m_i)=n$.

If $y \in OUT$ and $y \notin RI(n)$, then constructed activity vector is also partial sensitivity vector Y_iN.

Example: Consider the example in Figure 3.8, in case of evaluation of node C $(n=C)$. Then, the node 4 is the sole node m in set M. The $y=A$, $x=3$ and $y=B$, $x=3$ can be selected to satisfy the conditions a) and b). As result two activity vectors (for nodes A and B) are constructed: $A_2C=A4 \wedge 4_1C$ and $B_1C=B4 \wedge 4_1C$. As nodes A and B also belong to set of primary outputs and not the converging points of C, the constructed activity vectors are also partial sensitivity vectors AC and BC.

Step 1b: for subgraphs G' selected in the previous step with $x=n$, the activity vector $YN=F_Y(...)$ is constructed. Again, if $y \in OUT$ this activity vector is also represents partial sensitivity vector. The pair (x,y) is added to set P_{nested} and the node y is added to set M'.

Example: Consider the evaluation of node 3. During Step 1a the following subgraphs have been selected $G'(2,A)$, $G'(2,B)$, $G'(3,A)$ and $G'(3,B)$. Thus activity vectors $B3=F_B(B_13, B_23)$ and $A3=F_A(A_23, A_33)$ are constructed. As node

A and *B* also belong to set of primary outputs the constructed activity vectors are also partial sensitivity vectors *A3* and *B3*.

To handle nested reconvergencies the steps *1a* and *1b* are iteratively repeated with new set $M=M'$ and $M'=\varnothing$.

Step 1c: for $x=\Gamma^{-1}(n_k)$ activity vectors N_kX are constructed for all k. If $n \in OUT$ then the constructed vectors are also partial sensitivity vectors.

Example: Consider the evaluation of node *C*. Three activity vectors $C_1 1$, $C_2 2$ and $C_3 3$ are constructed.

Step 2: Construction of partial sensitivity vectors.

for all nodes $y \in OUT \cap \Gamma^*(n)$, the partial sensitivity vectors are constructed (by using formulas (8) and (10)) for *y* that were not handled in the Step 1.

Step 3: Construction of full sensitivity vector. For each node *n*, full sensitivity vector is constructed by the following formula:

$$\cup(Y_kN),\ y_k \in OUT \mid y_k \in \Gamma^*(n)$$

Example: For node *1* the full sensitivity vector is constructed as a union of partial sensitivity vectors *A1* and *B1*: $1=A1 \cup B1$.

The resulting optimized calculation model for reconvergency graph in Figure 3.8 is presented in Table 3.3. The formulas marked by "*" are sensitivity vectors whereas others are activity vectors.

It can be easily seen from the comparison of Table 3.2 and Table 3.3 that the optimized calculation model requires fewer steps (41 versus 52) for evaluation hence speeding up the whole fault simulation process.

Table 3.3: Optimized calculation model

step	*n*	formula	step	*n*	formula
1	***A***	$A_1 1$ (A1*)	**21**	***3***	$B_2 3 = B_2 5 \wedge 5_1 3$
2	***A***	$A_2 4$ (A4*)	**22**	***3***	$B3^* = F_B(B_1 3, B_2 3)$
3	***A***	$A_3 5$ (A5*)	**23**	***3***	$3^* = A3 \cup B3$
4	***B***	$B_1 4$ (B4*)	**24**	***3***	$3_1 2$
5	***B***	$B_2 5$ (B5*)	**25**	***2***	$C_3 2 = C_3 3 \wedge 3_1 2$
6	***B***	$B_3 0$	**26**	***2***	$C2 = F_C(C_2 2, C_3 2)$
7	***5***	$5^* = A5 \cup B5$	**27**	***2***	$B_1 2 = B_1 C \wedge C2$
8	***5***	$5_1 3$	**28**	***2***	$B_2 2 = B_2 3 \wedge 3_1 2$
9	***4***	$4^* = A4 \cup B4$	**29**	***2***	$B2^* = F_B(B_1 2, B_2 2)$
10	***4***	$4_1 C$	**30**	***2***	$A_2 2 = A_2 C \wedge C2$
11	***C***	$A_2 C = A_2 4 \wedge 4_1 C$ (AC*)	**31**	***2***	$A_3 2 = A_3 3 \wedge 3_1 2$
12	***C***	$B_1 C = B_1 4 \wedge 4_1 C$ (BC*)	**32**	***2***	$A2^* = F_A(A_1 2, A_2 2)$
13	***C***	$C^* = AC \cup BC$	**33**	***2***	$2^* = A2 \cup B2$
14	***C***	$C_1 1$	**34**	***2***	$2_1 0$
15	***C***	$C_1 2$	**35**	***1***	$A_2 1 = A_2 C \wedge C_1 1$
16	***C***	$C_3 3$	**36**	***1***	$A1^* = F_A(A_1 1, A_2 1)$
17	***3***	$A_2 3 = A_2 C \wedge C_1 3$	**37**	***1***	$1^* = A1 \cup B1$
18	***3***	$A_3 3 = A_2 5 \wedge 5_1 3$	**38**	***0***	$A0^* = A2 \wedge 2_1 0$
19	***3***	$A3^* = F_A(A_2 3, A_3 3)$	**39**	***0***	$B_2 0 = B_2 2 \wedge 2_1 0$
20	***3***	$B_1 3 = B_1 C \wedge C_1 3$	**40**	***0***	$B0^* = F_B(B_2 0, B_3 0)$
			41	***0***	$0^* = A0 \cup B0$

3.3.5 Reducing the memory requirements for fault simulation

Obviously, the presented method of parallel-pattern fault simulation needs extra memory for storing the formulas of calculation model. This issue can easily restrict the applicability of the approach with circuits of a large size. Figure 3.9 illustrates the empirical study of the sizes of optimized calculation models for the selected benchmarking circuits. It can be seen from the diagram that the memory consumption is growing with the circuit size.

In Paper III, a method has been proposed that is able to reduce the impact of memory requirements by splitting the calculation model into several parts. The algorithm that is described in Section 4 of Paper III is rather straightforward but it confirms the viability of the idea (see experimental data in Section 5 of Paper III).

The approach proposes to partition the set of primary inputs *PI* into a number of non-overlapping subsets $B_i \subset PI$. For each of the constructed subsets B_i, a *partial reconvergency* graph $G_i \subset G$, where set N_i consists of nodes $x \in N_i$ for which the following condition is satisfied: $B_i \cap \Gamma^{*-1}(x) \neq \varnothing$.

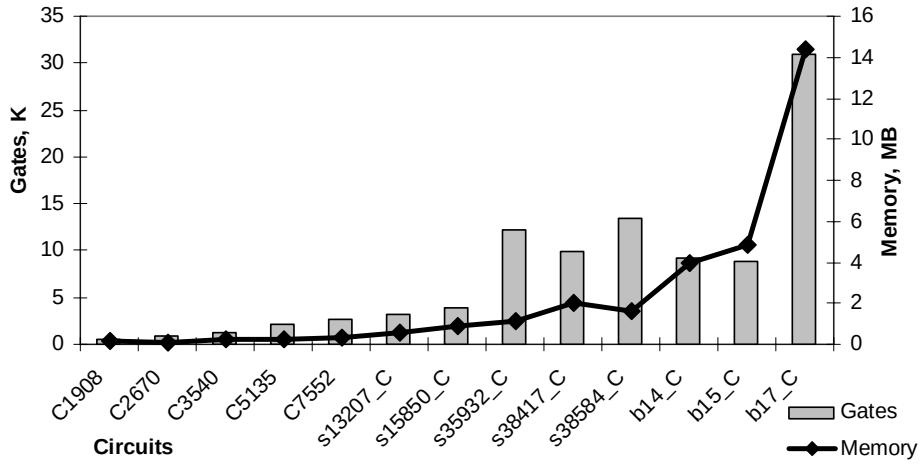


Figure 3.9: Memory requirements for fault simulation

The fault simulation with reduced memory requirements is performed in several iterations. Each pass starts with the creation of formulas for reconvergency sub-graph G_i and construction of a *partial calculation model*. During the evaluation of the model, the detectability of fanouts corresponding to nodes $x \in N_i$ is determined. Then the partial model is deleted, freeing the memory for construction of formulas for the next partial reconvergency sub-graph G_{i+1}.

It is very likely that an overlap occurs between pairs of sets N_i and N_j of partial reconvergency sub-graphs. As result, this method introduces a certain overhead: the construction and evaluation of some formulas that correspond to the nodes falling into the overlapped area is done several times.

Let us consider the schematic illustration of the circuit structure in Figure 3.10. The set of primary inputs for this example is $PI=\{i_1, i_2, i_3\}$ and the example partition of inputs is $\pi = \{\overline{i_1, i_2}; \overline{i_3}\}$. This partition splits the circuit structure into two overlapping slices and leads to construction of two partial reconvergency sub-graphs (G_1 and G_2) with the overlapped set of nodes $N_{12}=N_1 \cap N_2$.

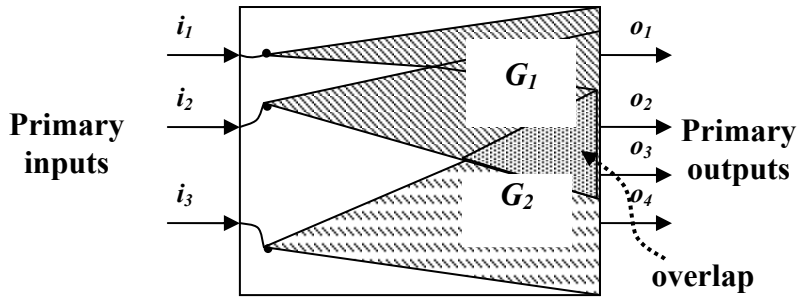


Figure 3.10: Schematic representation of circuit structure

Certainly there is no need to construct full sensitivity vectors and partial sensitivity vectors for the nodes belonging to N_{12} twice. Instead, the sensitivity vectors need to be constructed only once either for G_1 or for G_2. However, the activity vectors still are constructed for both cases as they can be used in calculation of sensitivities for nodes not belonging to N_{12}. The latter, is the source of overhead introduced by the splitting of calculation model.

As for the current implementation, no analysis is conducted to find out which of the activity vectors corresponding to the nodes in overlapping area can be constructed only for specific partial reconvergency graphs. However such analysis that is planned for future work will certainly decrease the overhead. Further investigation is also required for the ways of selection of partition π to minimize the size of overlapped area. However, for currently conducted experiments, the partition π was selected randomly taking into account only the amount of available memory.

3.3.6 Overview of experimental results

The method of parallel-pattern simulation that uses non-optimized model has demonstrated speed-up in fault analysis in comparison with commercial tools (see Paper II). However, the experiments presented in [VI] had shown improved scalability because of construction of joined model for all primary outputs.

The final approach presented in Paper III had achieved the best results due to optimization of repetitive formulas inside the calculation model. The conducted experiments (section 5 of Paper III) had shown the advantage of the method in comparison with other fault simulators. The achieved speed gain was about 4.8 and 53 times (in average) for the circuits selected from the different benchmark sets. The overall comparison of the both simulation methods is given in Section 3.5.

Considering the memory reduction, the carried out preliminary experiments had illustrated the feasibility of this approach for sort of the circuits. For example, the size of memory needed for simulation of the largest circuit of ISCAS'89 [63] decreased almost by four times while the overhead in simulation speed was less than two times. However the experiments also shown that the current approach is not universal: for certain circuits the size of memory used by fault simulator reduced insignificantly (less than by 25%). Nevertheless the average decrease of memory consumption is more than half and the average cost of such reduction is the degradation of fault simulation speed by 2.5 times.

3.3.7 Conclusions

The fundamental result of Paper II and Paper III is a novel approach for parallel exact critical path tracing for combinational circuits or scan-path designs. The main

idea of the method is to perform topological analysis of circuit in order to create a model for calculating of Boolean differentials. Thanks to the parallelism and optimization of the calculation model, the speed of fault simulation was considerably increased in comparison with other fault analysis techniques.

In case of very large circuits, the proposed technique can require large amount of extra memory for storing the formulas of calculation model. To reduce the memory consumption, the method for splitting the simulation process into several iterations was proposed. However, the latter method requires further improvement.

3.4 Hierarchical calculation of fault injection sites

The topic of research that is described in Paper IV is different from the issues discussed above since it concerns the problem of dependability analysis. However the presented method is tightly bounded with fault analysis technique.

Generally speaking, a dependable system that is equipped with capabilities for error detection and recovering typically requires a comprehensive testing of fault tolerance mechanisms along with the conventional test. Fault injection technique is the one of the methods for evaluation of the quality of fault tolerance mechanism of a system. The method suggests to perform injection of faults and study the results of system behavior in the presence of faulty component.

A simulation-based fault injection technique considered in the paper is used for analysis of dependability during the phase of the design of digital systems. However there is a problem of the selection of the subset of faults for injection and further study. In general, a presence of a fault not necessarily results in the erroneous behavior of a system (such faults are considered as not critical). But for testing of fault tolerance facilities, the critical (*malicious*) faults (that lead to malfunction of system on application level) are definitely preferable for evaluation.

In contrast with the methods that are using HDL-based approaches (e.g. [58], [59]), the key idea of the paper is to use multi-level circuit modeling with the help of Decision Diagrams (DD). In particular, High-Level Decision Diagrams (HLDDs) are used for description of circuit on a higher abstraction level while SSBDD represent circuit on lower-level. As result, the method excellently copes with the complexity (in contrast to pure gate-level models) and provides gate-level accuracy at the same time.

However the presented approach requires conversion of HDL description of system into hierarchical DD-model, the current paper is not focused on this issue. The study on this topic was carried out in [60].

The comprehensive description of the proposed method together with the illustrative examples is provided in Paper IV.

3.5 Overall experimental results

In this section the overall results of experiments for both single-pattern and parallel-pattern fault analysis methods are presented and compared with other fault simulators. Below we are illustrating only the final results while the more detailed information can be found in Papers I-III.

The experiments have been carried out on single UltraSPACR IV+ 1500MHz platform under control of SunOS operating system. The results for linear critical path tracing method by Wu et al. published in [28] had been obtained on Pentium 2.8GHz processor. For every circuit, 10000 test patterns were simulated by each tool. The fault dropping was deactivated hence complete fault table was obtained in each case. No memory constraints were applied during fault analysis.

The first column of Table 3.4 contains the circuits selected from three different benchmark sets: ISCAS'85 [61], [62], ISCAS'89 [63] and ITC'99 [64], [65]. The sequential circuits of ISCAS'89 and ITC'99 were substituted by their combinational versions (with cut-out flip-flops). The next column represents the size of each circuit (in terms of the number of 2-input gate equivalents).

The next group of columns contains the number of seconds spent by each of fault simulators for building fault table. Columns 3 and 4 illustrate the simulation results of proposed parallel-pattern simulator with optimized and non-optimized calculation models respectively. The results of single-pattern fault analysis are presented in Column 5.

For the illustrative purposes, Column 6 contains the results of parallel-pattern simulation in case if test patterns are available one-by-one (i.e. Column 6 contains the results of Column 3 multiplied by 32). This allows the "normalized" comparison between parallel-pattern and single-pattern approaches.

As for comparison with other methods, four different fault simulators were selected. FSIM (Column 7) is an efficient PPSFP simulator described in [25] which was modified for usage without fault dropping. C1 and C2 (Columns 8 and 9) are state-of-the-art commercial parallel-pattern simulators that are incorporated into the test development toolsets from major CAD vendors. The simulation results obtained in [28] are provided in Column 10.

The last row presents an average speed-up achieved by parallel-pattern fault simulator (Column 3) in comparison with others. Except for Column 6, the average speed gain of single-pattern fault simulation is presented when comparing with the results of Column 6.

According to Table 3.4, the proposed parallel-pattern fault simulator had shown the best results in terms of simulation time. However in case of single-pattern applications of fault simulator the method described in Section 3.2 becomes more efficient.

Table 3.4: Overall experimental results

Circuit	Size, gates	Proposed methods				Commercial and academic fault simulators			
		Parallel-pattern (optim.)	Parallel-pattern (non-opt)	Single-pattern	Parallel-pattern /single	FSIM	C1	C2	Wu
c1355	518	0.3	0.4	2.5	9.6	0.2	1.7	9	638
c1908	618	0.4	0.6	2.5	11.2	0.6	3.0	12	638
c2670	883	0.4	0.5	4.1	13.1	0.8	2.2	24	555
c3540	1270	0.9	1.3	5.0	28.5	2.0	7.4	43	763
c5315	2079	0.8	0.9	9.4	24.3	1.4	5.6	57	1254
c6288	2384	7.4	14.8	26.0	237.4	12.1	27.8	284	4267
c7552	2632	1.2	1.9	16.3	37.8	2.7	8.1	88	1467
s13207	3214	2.0	2.6	32.0	64.6	N/A	5.6	70	N/A
s15850	3873	2.7	5.2	47.7	85.8	N/A	12.1	111	N/A
s35932	12204	5.7	6.4	433.4	183.7	N/A	23.6	390	N/A
s38417	9849	7.0	11.1	267.9	225.0	N/A	31.4	310	N/A
s38584	13503	6.4	9.3	336.0	205.4	N/A	23.2	320	N/A
b14	9150	14.5	35.9	78.2	463.0	N/A	49.2	N/A	N/A
b15	8877	26.6	48.3	116.3	849.9	N/A	39.1	N/A	N/A
b17	31008	77.8	233.3	1152.3	2488.6	N/A	117.7	N/A	N/A
Average speed gain by parallel-pattern fault simulator		1	1.6	17.1	3.6*	1.5	4.7	43	1189

3.6 Chapter summary

In this chapter the overview of the research results published in Papers I-IV has been presented together with the overall experimental results.

The first part of the chapter provides a description of SSBDD model that is used by the proposed fault simulation methods. Next, the main contributions and results of research have been described. This had included the shortened description of single-pattern and parallel-pattern fault simulation algorithms as well as a brief introduction to an approach for dependability analysis. Finally, the results of experiments with the proposed methods have been presented and the comparison with other approaches has been made.

Chapter 4

CONCLUSIONS

Fault simulation is a widely used task in the flow of design of digital systems. Although the primary goal of fault simulation is to estimate the quality of test program, it is incorporated into many other test-related tasks as an auxiliary step. Numerous methods of fault simulation for different fault models, circuit types and abstraction levels have been proposed so far.

The presented work mainly addresses the problem of fault simulation of combinational circuits on well-known stuck-at fault model. Several novel approaches that attempt to increase the efficiency of fault simulation (in terms of CPU time and memory consumption) have been proposed in the thesis.

In the following sections the main contributions of the work are outlined and the perspectives for future research are discussed.

4.1 Contributions

The contributions of the presented work are summarized below.

The fault simulation algorithms introduced in the current thesis work with macro-level circuit descriptions represented with the help of Structurally Synthesized Binary Decision Diagrams. This had allowed to exploit the advantages of SSBDD model that is successfully utilized in other test-related problems also for fault simulation. The usage of higher abstraction level instead of gate-level ensures the immediate gain of circuit evaluation speed but keeps the accuracy of evaluation in conformity with gate-level models. Prior to this work, only basic fault simulation techniques were realized for SSBDD model, e.g. serial fault analysis and pure PPSFP. However they had proven to be more efficient than their gate-level

analogues. The current work makes one step forward introducing more sophisticated algorithms of fault analysis on Structurally Synthesized BDDs.

A novel single-pattern fault simulation approach has been proposed. The method is based on introduced deductive fault list propagation algorithm through SSBDD graphs. The topological pre-analysis of a circuit under simulation is carried out in order to determine circuit parts that do not require deductive reasoning but rather simpler propagation algorithms. In addition, the technique for shrinking the list fault candidates is proposed for the sake of accelerating the overall simulation speed. Together, the drawn ideas form an efficient single-pattern fault simulator.

The experiments have shown that the proposed optimization technique is able to reduce the size of list of potential faults to only 13% of uncollapsed size. Moreover, thanks to the topological pre-analysis, it became evident that deductive propagation can be avoided for ~39% of SSBDD graphs in circuit (in average). However the achieved simulation speed is mediocre in comparison with the proposed parallel-pattern approaches, single-pattern simulation is very suitable for the tasks that demand simulation of test patterns one-by-one. In the latter case, the fastest parallel-pattern approach is outperformed by 3.6 times.

The thesis introduces a parallel-pattern fault simulation method. Two novelties are proposed here: the parallel critical path tracing algorithm on SSBDD model and the technique for simulating circuit beyond fanout-free regions with the help of Boolean differentials. The conducted experiments have shown that the proposed method overcomes other fault simulation techniques. In particular, the speed-up is 4.7 and 1.5 times in comparison with commercial and academic tools respectively.

The problem of the memory consumption has been studied for the case of parallel-pattern fault simulation and an approach for reducing the memory requirements has been presented. However the proposed idea is not universal, the experiments had shown its viability for certain types of circuits. The best achieved result was in cutting down memory consumption by 73% while the speed of simulation decreased only by 1.7 times. The average results are 48% and 2.5 times respectively.

An approach of dependability analysis with the help of Decision Diagrams has been proposed. The method targets construction of a list of malicious faults that are intensively used in fault tolerance analysis. The idea is based on using hierarchy of DDs for representation of circuit on multi-levels: High-Level DD model for register-transfer level and SSBDDs for gate-level. The presented conception enables to avoid the complexity problems of pure logic-level description, but at the same time performs malicious fault analysis with the gate-level accuracy.

4.2 Future work

This section outlines the most important issues that require further investigation for improving of the proposed techniques.

First of all, the experiments presented in the current thesis had been carried out on benchmark circuits of relatively small size: the largest circuit under simulation contains less than hundred of kilo gates. Obviously, we plan to measure the speed of both proposed methods by using industrial-sized benchmarks that consist of millions of gates. This is required for study of the scalability of the proposed techniques.

The experiments that reveal the influence of fault dropping have been left out of scope of current thesis. However, since this technique is frequently used to accelerate fault analysis such experiments have to be additionally conducted to find out which of the methods provides better performance with fault dropping.

The memory requirements of the proposed parallel-pattern simulation algorithm certainly demand further investigation. The abovementioned method of relaxing memory constraints provides rather straightforward solution and leaves a room for further improvements.

At the same time the conception of splitting of fault simulation model into parts can be also applied for parallelization of fault simulation over several workstations (or several processor cores). The usage of capabilities of such parallelization could bring the presented fault simulation approach to a new level of performance.

Certainly a standalone gate-level fault analysis typically is not a major issue in the digital design field nowadays. Instead, various applications demanding efficient fault simulation engine have to be tried. One of the promising ways is to go on higher levels of abstraction and build a fault simulation tool that uses hierarchical approach. In this case, the presented methods can be utilized as an efficient lower-level simulator. Another direction is to acquire more experience in acceleration of other simulation-dependent tasks by incorporation of fast fault analysis algorithms.

The presented technique for creation of lists of malicious faults demonstrates the one of possible applications of fault analysis. However, the further development of this direction requires the creation of a tool that is capable to handle all levels of hierarchy: HLDD and SSBDD. Then, more comprehensive experiments need to be carried out to estimate the effectiveness of the proposed method.

References

Co-authored papers:

[I] J. Raik, R.Ubar, S.Devadze and A.Jutman. Efficient Single-Pattern Fault Simulation on Structurally Synthesized BDDs. – Proc. of 5th European Dependable Computing Conference, Hungary, 2005, pp.332-344.

[II] R. Ubar, S. Devadze, J. Raik, A. Jutman. Ultra Fast Parallel Fault Analysis on Structurally Synthesized BDDs. – Proc. of 12th IEEE European Test Symposium, Germany, 2007, pp. 131-136.

[III] S. Devadze, R. Ubar, J. Raik, A. Jutman. Parallel Exact Critical Path Tracing Fault Simulation with Reduced Memory Requirements. – Proc. of 4th IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era, Egypt, 2009

[IV] R. Ubar, S. Devadze, M. Jenihhin, J. Raik, G. Jervan, P. Ellervee. Hierarchical Calculation of Malicious Faults for Evaluating the Fault Tolerance. – Proc. of 4th IEEE International Symposium on Electronic Design, Test & Applications, China, 2008, pp. 222-227

[V] S. Devadze, J. Raik, A. Jutman, R. Ubar. Fault Simulation with Parallel Critical Path Tracing for Combinational Circuits Using Structurally Synthesized BDDs. – Proc. of 7th IEEE Latin-American Test Workshop, Argentina, 2006, pp.97-102.

[VI] R. Ubar, S. Devadze, J. Raik, A. Jutman. Parallel Fault Backtracing for Calculation of Fault Coverage. – Proc. of 43rd International Conference on Microelectronics, Devices and Materials and the Workshop on Electronic Testing, Slovenia, 2007, pp. 165-170.

[VII] R. Ubar, S. Devadze, J. Raik, A. Jutman. Parallel Fault Backtracing for Calculation of Fault Coverage. – Proc. of 13th Asia and South Pacific Design Automation Conference, South Korea, 2008, pp. 667-672.

[VIII] S.Devadze, A.Jutman, A.Sudnitson, R.Ubar, H.-D.Wuttke. Teaching Digital RT-Level Self-Test Using a Java Applet. – Proc. of 20th IEEE Conference NORCHIP'2002, Denmark, 2002, pp.322-328.

[IX] R.Ubar, A.Jutman, S.Devadze, H.-D. Wuttke. Bringing Research Issues into Lab Scenarios on the Example of SOC Testing. – Proc. of Int. Conference on Engineering Education, Portugal, 2007, pp. 170-171.

[X] R.Ubar, A.Jutman, M.Kruus, E.Orasson, S.Devadze, H.-D.Wuttke. Learning Digital Test and Diagnostics via Internet. International Journal of Emerging Technologies in Learning. International Journal of Online Engineering, Vol.3, No.1, 2007, pp. 1-9.

Other references:

[1] G. Moore. Cramming More Components onto Integrated Circuits. – Reprint from IEEE proceedings on Electronics, Vol. 38, No. 8, 1965.

[2] R. W. Keyes, The Impact of Moore's Law. – IEEE Solid-State Circuits, Issue: Sept 2006.

[3] International Technology Roadmap for Semiconductors, 2007-2008.

http://public.irts.net [*June 2009*]

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

[5] L.-T. Wang, C.-W. Wu, X. Wen. VLSI Test Principles and Architectures. Elsevier, 2006.

[6] S. Mourad, Y. Zorian. Principles of Testing Electronic Systems. Wiley Interscience, 2000.

[7] S. Kundu, S.T. Zachariah, Y.-S. Chang, C. Tirumurti. On modeling crosstalk faults. – IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2005, vol. 24, issue 12, pp 1909-1915.

[8] R. Desineni, K. N. Dwarkanath, R.D. Blanton. Universal Test Generation Using Fault Tuples. – Proc. of the IEEE International Test Conference, 2000, pp.812-819

[9] V.K. Agarwal, A.F.S. Fung. Multiple Fault Testing of Large Circuits by Single Fault Test Sets. – IEEE Transactions on Circuits and Systems, vol CAS-28, 1981, pp. 1059-1069.

[10] R. C. Aitken. Defect-Oriented Testing. In series: D. Gizopoulos, Advances in Electronic Testing, Springer, 2006.

[11] M. Abramovici, M.A.Breuer, A.D. Friedman. Digital systems testing and testable design. IEEE Press. 1990.

[12] A. Miczo. Digital Logic Testing and Simulation. Wiley Interscience, 2003.

[13] S. Seshu. On an Improved Diagnosis Program. – IEEE Trans. on Electronic Computers, 1965, pp. 76-79

[14] J.A. Waicukauski, E.B. Eihelberger, D.O. Forlenza, E. Lindbloom, T. McCarthy. Fault Simulation for Structured VLSI. VLSI Systems Design, 1985, pp. 20-32.

[15] E.G. Ulrich, T. Baker. The Concurrent Simulation of Nearly Identical Digital Networks. – Proc. of 10th Design Automation Workshop, 1973, pp. 145-150.

[16] W.-T. Cheng, M.-L. Yu. Differential Fault Simulation - A Fast Method using Minimal Memory. – Proc. of 26th Design Automation Workshop, 1989, pp. 424-428.

[17] T.M. Niermann, W.-T. Cheng. J. H. Patel. PROOFS: A Fast, Memory-Efficient Sequential Circuit Fault Simulator. – IEEE Transactions on Computer-Aided Design, Vol. 11, No. 2, 1992, pp. 198-207.

[18] D.B. Armstrong. A Deductive Method for Simulating Faults in Logic Circuits. – IEEE Trans. on Computers, Vol. C21, No. 5, 1972, pp. 464-471.

[19] M. Abramovici, P.R. Menon, D.T. Miller. Critical Path Tracing - an Alternative to Fault Simulation. – in Proc. of 20th Design Automation Conference, 1983, pp.214-220.

[20] K.J. Antreich, M.H. Schulz. Accelerated Fault Simulation and Fault Grading in Combinational Circuits. – IEEE Trans. on Computer-Aided Design, Vol. 6, No. 5, 1987, pp. 704-712.

[21] D. Harel, R. Sheng, J. Udell. Efficient Single Fault Propagation in Combinational Circuits. – Proc. of International Conference on Computer-Aided Design, 1987, pp. 2-5.

[22] F. Maamri, J. Raiski. A Method of Fault Simulation Based on Stem Regions. – IEEE Trans. on Computer-Aided Design, Vol. 9, No. 2, 1990, pp. 212-220.

[23] J.P. Roth, W. G. Bouricius, P. R. Schneider. Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits. – IEEE Trans. on Electronic Computers, Vol. 16, No. 10, 1967, pp. 567-579

[24] B. Underwood, J. Ferguson. The Parallel-Test-Detect Fault Simulation Algorithm. – Proc. of International Test Conference, 1989, pp.712-717.

[25] H.K. Lee, D.S. Ha. An efficient, forward fault simulation algorithm based on the parallel pattern single fault propagation. – Proc. of International Test Conference, 1991, pp. 946-955.

[26] D. Saab. Parallel-Concurrent Fault Simulation. – IEEE Trans. on VLSI Systems, Vol. 1, No. 3, 1993, pp.356-364.

[27] N. Takahashi, N. Ishiura, S.Yajima. Fault Simulation for Multiple Faults by Boolean Function Manipulation. – IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 13, No. 4, 1994, pp. 531-535.

[28] L. Wu, D.M.H. Walker. A Fast Algorithm for Critical Path Tracing in VLSI Digital Circuits. – Proc. of 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2005, pp. 178-186.

[29] M.G. McNamer, S.C. Roy, H.T. Nagle. Statistical Fault Sampling. – IEEE Trans. on Industrial Electronics, Vol. 36, No. 2, 1989, pp.141-150.

[30] S.K. Jain, V.D. Agrawal. Statistical Fault Analysis. – IEEE Design and Test of Computers, vol. 2, no. 1, January/February, 1985, pp. 38-44.

[31] D. Cock, A. Carpenter. A Proposed Hardware Fault Simulation Engine. – Proc. of the European Conference on Design Automation, 1991, pp. 570-574.

[32] W. Hahn, A. Hagerer, R. Kandlbinder. Hardware-Accelerated Concurrent Fault Simulation: Eventflow Computing versus Dataflow Computing. – Proc. of 4th Asian Test Symposium, 1995, pp. 107-111.

[33] S. Kang, Y. Hur, S.A. Szygenda. A Hardware Accelerator for Fault Simulation Utilizing a Reconfigurable Array Architecture. – VLSI Design, Vol. 6, No. 2, 1996, pp. 119-133.

[34] A. Parreira, J.P. Teixeira, A. Pantelimon, M.B. Santos, and J.T. de Sousa. Fault Simulation Using Partially Reconfigurable Hardware. – Lecture Notes in Computer Science, Vol. 2778, 2003, pp.839-848.

[35] P. Ellervee, J. Raik, V. Tihhomirov. Environment for Fault Simulation Acceleration on FPGA. – Proc of 9th Biennial Baltic Electronic Conference, Estonia, 2004, pp. 217-220.

[36] K. Gulati, S.P. Khatri. Towards Acceleration of Fault Simulation using Graphics Processing Units. – Proc. of the 45th annual Design Automation Conference, 2008, pp. 822-827.

[37] K. Gulati, S.P. Khatri. Fault Table Generation Using Graphics Processing Units. – Proc. of the 16th International Test Synthesis Workshop, 2009.

[38] H.D. Schnurmann, E. Lindbloom, R.G. Carpenter. The Weighted Random Test-Pattern Generator. – IEEE Trans. on Computers, Vol. 24, Issue 7, pp. 695-700.

[39] R. Lisanke, F. Brglez, A.J. Degeus, D. Gregory. Testability-Driven Random Test-Pattern Generation. – IEEE Trans. on Computer-Aided Design, Vol. CAD6, No. 6, 1987, pp. 1082-1087.

[40] H.J. Wunderlich. Multiple Distributions for Biased Random Test Patterns. – IEEE Trans. on Computer-Aided Design, Vol.9, No.6, 1990, pp. 584-593.

[41] M. Srinivas, L.M. Patnaik. A Simulation-Based Test Generation Scheme Using Genetic Algorithms. – Proc. of 6th Int. Conference on VLSI Design, 1993, pp. 132-135.

[42] E.M. Rudnick, J.H. Patel, G.S. Greenstein, T.M. Niermann. Sequential Circuit Test Generation in a Genetic Algorithm Framework. Proc. of Design Automation Conference, 1994, pp. 698-704.

[43] T.J. Snethen. Simulator-oriented fault test generator. – Proc of 14th Design Automation Conference, 1977, pp. 88-93.

[44] 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, 1989, pp. 131-138

[45] S. Kajihara, K. Miyase. On identifying don't care inputs of test patterns for combinational circuits. – Proc. of International Conference on Computer-Aided Design, 2001, pp. 364-369.

[46] A. Jutman, I. Aleksejev, J. Raik, R. Ubar. Reseeding using compaction of pre-generated LFSR sub-sequences. – Proc of 15th Int. Conference on Electronics, Circuits and Systems, 2008, pp. 1290-1295.

[47] G. Jervan, P. Eles, Z. Peng, R. Ubar, M. Jenihhin. Test time minimization for hybrid BIST of core-based systems. – Proc of 12th Asian Test Symposium, 2003, pp. 318-323.

[48] R. Ubar. Test Generation for Digital Circuits Using Alternative Graphs (in Russian). – Proc. of Tallinn Technical University, No.409, 1976, pp.75-81.

[49] S.B. Akers. Binary Decision Diagrams. – IEEE Trans. on Computers, Vol. C27, No. 6, 1978, pp. 509-516.

[50] R. Drechsler, B. Becker. Binary decision diagrams: Theory and implementation, Kluwer Academic Publishers, 1998.

[51] A. Jutman. On SSBDD Model Size and Complexity. – Proc of 4th Electronic Circuits and Systems Conference, Slovakia, 2003, pp. 17-22.

[52] A. Jutman, J. Raik, R. Ubar. SSBDDs: Advantageous Model and Efficient Algorithms for Digital Circuit Modeling, Simulation & Test. – Proc. of 5th Int. Workshop on Boolean Problems, Germany, 2002, pp. 157-166.

[53] R. Ubar. Test Synthesis with Alternative Graphs. – In IEEE Design and Test of Computers, 1996. pp. 48-57.

[54] R. Ubar, A. Jutman, Z. Peng. Timing Simulation of Digital Circuits with Binary Decision Diagrams. – Proc. of Design Automation and Test in Europe Conference, Germany, 2001, pp. 460-466.

[55] A. Jutman, R. Ubar. Application of Structurally Synthesized Binary Decision Diagrams for Timing Simulation of Digital Circuits. – Proc. of the Estonian Academy of Sciences, Engineering, Vol. 7/4, 2001, pp. 269-288.

[56] A. Jutman, R. Ubar. Design Error Diagnosis in Digital Circuits with Stuck-at Fault Model. – Journal of Microelectronics Reliability, Pergamon Press, Vol. 40, No. 2, 2000, pp.307-320.

[57] A. Jutman, A. Peder, J. Raik, M. Tombak, R. Ubar. Structurally synthesized binary decision diagrams. – Proc. of 6th International Workshop on Boolean Problems, Freiberg, 2004, pp.271-278.

[58] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, J. Karlsson. Fault injection into VHDL models: the MEFISTO tool. – Proc 24th Int. Symposium on ault-Tolerant Computing, 1994, pp. 66-75.

[59] R. Leveugle1, K. Hadjiat. Multi-Level Fault Injections in VHDL Descriptions: Alternative Approaches and Experiments. – Journal of Electronic Testing, Vol. 19, No. 5, 2003.

[60] M. Jenihhin. Simulation-Based Hardware Verification with High-Level Decision Diagrams. – Ph.D. Thesis, Tallinn University of Technology, 2008.

[61] F. Brglez, H. Fujiwara. A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran. – Proc. of the International Test Conference, 1985, pp. 785-794.

[62] ISCAS85 Combinational Benchmark Circuits in 'Bench' Format. Department of Computer Engineering, University of Illinois.

http://courses.ece.illinois.edu/ece543/iscas85.html [June 2009]

[63] F. Brglez, D. Bryan, K. Kominski. Combinational Profiles of Sequential Benchmark Circuits. – Proc. Int. Symposium on Circuits and Systems, 1989, pp. 1929-1934.

[64] F. Corno, M.S. Reorda, G. Squillero. RT-level ITC'99 benchmarks and first ATPG results. – IEEE Design & Test of Computers, Vol. 17, No. 3, 2000, pp. 44-53

[65] ITC99 Benchmarks, Combinational Gate-Level Versions. CAD Group, Politecnico di Torino

http://www.cad.polito.it/tools/itc99.html *[June 2009]*

RESEARCH PAPERS

Paper I

The article was published in the *Lecture Notes in Computer Science, Vol. 3463, 2005*.

Paper II

This paper was published in the *Proceeding of the 12th IEEE European Test Symposium (ETS 2007)*.

Paper III

This paper was published in the *Proceeding of IEEE International Symposium on Design & Technology of Integrated Systems in Nanoscale Era (DTIS 2009)*

Paper IV

This paper was published in the *Proceeding of IEEE International Symposium on Electronic Design, Test and Applications (DELTA 2008)*

Curriculum Vitae

Personal Data

Name	Sergei Devadze
Date of birth	05.01.1981
Place of birth	Estonia
Citizenship	Estonian

Contact Data

Address	Raja 15, Tallinn, 12618
Phone	+372 6202262
E-mail	serega@pld.ttu.ee

Education

2004 – …	Ph.D. Student, Department of Computer Engineering, Tallinn University of Technology
2003 – 2004	M.Sc. in Computer Engineering, TUT
1999 – 2003	B.Sc. in Computer Engineering, TUT

Carrier

2007 – …	Researcher at Department of Computer Engineering, TUT
2004 – 2005	ELIKO Competence Centre in Electronics-, Info- and Communication Technologies, R&D Engineer
2001 – 2006	Engineer at TUT

Academic Degree

Master of Science in Computer Engineering, TUT,

"Web-Based System for Finite State Machines Decomposition"

Awards

2001	1st prize at the contents of student works by Estonian Ministry of Education and Research
2004 – 2007	Scholarship of Estonian Information Technology Foundation (EITSA)

Research topics

Fault simulation, optimization of board-level testing, decision diagrams, decomposition of finite state machines

Elulookirjeldus

Isikuandmed

Nimi	Sergei Devadze
Sünniaeg	05.01.1981
Sünnikoht	Eesti
Kodakondsus	Eesti

Kontaktandmed

Aadress	Raja 15, Tallinn, 12618
Telefon	+372 6202262
E-post	serega@pld.ttu.ee

Hariduskäik

2004 – …	doktorant, Arvutitehnika Instituut, Tallinna Tehnikaülikool
2003 – 2004	tehnikateaduste magister, Arvuti- ja süsteemitehnika eriala, TTÜ
1999 – 2003	tehnikateaduste bakalaureus, Arvuti- ja süsteemitehnika eriala, TTÜ

Teenistuskäik

2007 – …	teadur, Arvutitehnika Instituut, TTÜ
2004 – 2005	arendusinsener, ELIKO OÜ Tehnoloogia arenduskeskus
2001 – 2006	insener, Arvutitehnika Instituut, TTÜ

Teaduskraad

Tehnikateaduste magister, Arvuti- ja süsteemitehnika, TTÜ

"Automaatide dekompositsiooni veebikeskkond"

Teaduspreemiad

2004 – 2007	Eesti Infotehnoloogia Sihtasutuse (EITSA) stipendium
2003	Eesti Haridus- ja Teadusministeeriumi korraldatud üliõpilaste teadustööde konkursi esimene preemia tehnikateaduste valdkonnas

Teadustegevus

Rikete simuleerimine, trükkplaatide testimise optimeerimine, otsusediagrammid, lõplike automaatide dekompositsioon

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. **Makism 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.