THESIS ON INFORMATICS AND SYSTEM ENGINEERING

# Selected Issues of Modeling, Verification and Testing of Digital Systems

**ARTUR JUTMAN**

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

Dissertation is accepted for the defence of the Doctor of Philosophy in Engineering at Tallinn University of Technology

The commencement of the thesis will take place on the 6th of October, 2004 in Tallinn University of Technology, Ehitajate tee 5, Tallinn, Estonia

Supervisor:

Prof. Raimund Ubar, D.Sc., Academician
Department of Computer Engineering, Tallinn University of Technology

Opponents:

Michel Renovell, Ph.D.
Head of the Microelectronics Department of Laboratory of Computer Science, Automation and Microelectronics of Montpellier (LIRMM), France

Manfred Glesner, Prof. Dr. Dr.h.c.mult.
Head of Institute of Microelectronic Systems at Department of Electrical Engineering and Information Technology, Darmstadt University of Technology, Germany

Declaration

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



ISSN

ISBN

# Abstract

The research described in this thesis embraces comparatively large area in verification and testing of digital systems. It represents rather a "breadth-first" investigation touching different topics starting from design modeling and simulation and spanning to some issues of design for testability and diagnostics. The research handles both IC and PCB testing.

In each direction of the research, new appreciable results were achieved in form of problem solution, analysis, or concept development. The results were presented at international conferences and published in refereed journals. The selection of representative papers is collected and attached to the thesis.

The most important result of the research described in the first half of thesis is the study of an efficient design representation called SSBDD model and showing its advantages over commonly used gate-level netlist. The author succeeded to show, in a formal way, that this special kind of BBDs has linear complexity with respect to the number of logic gates for an arbitrary combinational circuit. Moreover, it is shown that the SSBDD model complexity is always smaller than the complexity of corresponding logic-level netlist exactly by the number of logic gates in this netlist. This is reflected in a simple equation which can be used for complexity estimation for the algorithms working with SSBDD model. Other examples of application of this equation are also considered.

During further study of the model four different logic-level simulation methods implemented on the SSBDD model were experimentally studied. It turned out that the simulation speed-up of these algorithms, in comparison with the gate-level simulation speed, is linearly proportional to the average size of macro measured in the number of gates. Further successful improvement of the simulation efficiency was achieved by experimenting with SSBDD reordering.

The most important result of research on MCM testing is the creation of a new homogeneous framework of at-speed interconnect test that touches and improves all the major aspects of this area of testing.

The key to all advantages of this framework is a combination of a novel original design of test pattern generator and response analyzer with a new deterministic test sequence called the Interleaved True/Complement Code. At first, it brings a never achieved before high level of universality, scalability, and configuration independence into the at-speed interconnect testing and diagnosis.

Second contribution of the conception is that it allows a free of aliasing or masking exact fault diagnosis, which does not even require a “golden signature” to compare with. The proposed framework takes into account all the important issues a practically efficient at-speed interconnect BIST framework must handle, such as: parallel application of test patterns, detection of both static and dynamic faults, bus contention problem etc. At the same time, the hardware implementation of the proposed framework is among the most efficient ones in terms of silicon area.

# Kokkuvõte

## Valitud teemad digitaalsüsteemide modelleerimisest, verifitseerimisest ja testimisest

Käesolevas dissertatsioonis esitatud uurimistöö hõlmab suhteliselt laia ala digitaalsüsteemide verifitseerimisest ja testist. Dissertatsiooni näol on tegemist rohkem laiuti analüüsiga, mis puudutab erinevaid skeemide modelleerimise ja simuleerimise teemasid ning käsitleb ka testitavat projekteerimist ja diagnostikat. Uurimistöö haarab nii kiipide kui ka trükkplaatide testimist.

Igas vaadeldavas uurimissuunas saavutati uudseid tulemusi probleemide lahenduste, analüüsi ja kontsepsioonide väljatöötamisel. Teadustulemused on esitatud rahvusvahelistel konverentsidel ning avaldatud eelretsenseeritavates ajakirjades. Valik esinduslikematest töödest on lisatud käesolevale dissertatsioonile lisana.

Doktoritöö esimese poole kõige olulisemaks tulemuseks on struktuurselt sünteesitud binaarsete otsustusdiagrammide (SSBDD) kui efektiivse skeemimudeli edasiarendamine ning mudeli eeliste uurimine. Autor esitas formaalse tõestuse, mille kohaselt eelpool nimetatud skeemimudel omab lineaarset keerukust skeemipunktide arvu suhtes. Veelenam, töös on näidatud, et SSBDD mudeli keerukus on alati väiksem vastavast skeemist traditsiooniliste loogikalülituste tasemel. Seda väljendab lihtne valem, mida saab kasutada SSBDD mudelil töötavate algoritmide keerukuse analüüsiks. Töös vaadeldakse ka muid rakendusi nimetatud valemile.

Töös käsitletud neli loogikasimuleerimise meetodit kujutavad endast SSBDD mudeli edasist uuringut. Läbi viidud eksperimendid näitasid, et nende algoritmide võit simuleerimise kiiruses on lineaarses sõltuvuses keskmisest SSBDD-s esinevate loogikalülide arvust. Uuritud on ka SSBDD tippude ümberjärjestamist, millega saavutati täiendav kiiruse võit simuleerimisel.

Kõige tähtsamaks töös esitatud uurimistulemuseks mitmekiibi moodulite (multi-chip modules e. MCM) testimise vallast on uue, ühtse arhitektuuri loomine seadme töökiirusel toimuvaks kiibi väljaviikude testiks, millega on parandatud eksisteerivaid meetodeid kõigis olulisemates parameetrites.

Süsteemis on kombineeritud uudne testigeneraator ja väljundväärtuste analüsaator uue determineeritud testjadaga, mida töös kutsutakse “vahelduv tõene-tastand kood” (e. Ingl. k. Interleaved True/Complement Code). Esiteks võimaldab see tuua töökiirusel toimuvale väljaviikude testi ja diagnoosi

valdkonda seninägematult kõrge universaalsuse, skaleeritavuse ja konfiguratsioonist sõltumatuse.

Teiseks võimaldab see maskeerumisvaba täpset diagnoosi, mis ei nõua isegi nn. “kuldse signatuuri” olemasolu. Väljapakutud arhitektuur arvestab kõikide praktilistele süsteemidele esitatavate nõuetega nagu testjada paralleelsus, staatiliste ja dünaamiliste rikete avastamine, siini konfliktide vältimine jne. Samal ajal kuulub väljapakutud arhitektuur kõige optimaalsemate lahenduste hulka nõutava ränipindala pindala poolest.

# Acknowledgements

I would like to begin by expressing my sincere gratitude to my father who fostered in me a deep interest in science and technology always inspiring and impressing me by his encyclopedic erudition and unreserved support in my undertakings.

My supervisor, Prof. Raimund Ubar has become an example of a scientist for me towards which I'm trying to tend. He introduced me to the topics of digital diagnostics and then provided me with necessary freedom for fulfillment of my scientific research interests.

Dr. Jaan Raik and Dr. Sven Nõmm have shown me how one should defend a Ph.D. successfully. Being my older friends, they helped me tremendously on the stony path to this degree by their advices, example, and friendly support.

Our director, Dr. Margus Kruus is like a good general who cares for his soldiers. Many thanks go to him for valuable help in critical situations and for supporting the administrative and technical side of my work.

Other people from the room IT-210, Dr. Marina Brik, Margit Aarna, Eero Ivask, Elmet Orasson, Tarmo Robal, and Marek Mandre always create a amicable milieu and deserve hearty thanks. The same holds for our friendly staff and students of Computer Engineering Department, especially Dr. Alexander Sudnitson, Sergei Devadze, Slava Rosin, Ruslan Gorjachev, and many others.

I would also like to express warm thanks to my colleagues from other universities. The fruitful cooperation with them opened new horizons through broadening my mental outlook. I'd like to especially mention Dr. Dieter Wuttke, Dr. Karsten Henke, and Jürgen Schmidt from Technical University of Ilmenau (Germany) where I spend all summers during the last six years, as well Dr. Bengt Magnhagen and Dr. Shashi Kumar from University of Jönköping (Sweden). Most of my experiments with SSBDD model and timing simulation were conducted during my visit to ESLAB of University of Linköping, which is lead by Prof. Zebo Peng, whose hospitality and valuable advices were always in time.

I'd like to thank Dr. Ben Bennetts for unintentionally inspiring me for the research in the area of Boundary Scan and interconnect testing. He later also encouraged me to continue my investigations by pointing out some really important problems for further study.

Finally, I'd like to thank all my friends and family for care, friendship, and patience.

This work was financially supported by the Estonian Science Foundation Grants No 1850, 4300, 5910, 5649, by the Royal Swedish Academy of Sciences, by Kristjan Jaak's Stipendiums, and by the European Community (Copernicus JEP 9624 VILAB).

# Contents


## PART I: OVERVIEW 1

**Introduction** **3**
1.1 Foreword 3
1.2 Motivation 4
1.3 Design and test cycle of development of a digital system 7
1.3 Contribution of selected research papers 10

**Simulation and Diagnosis of Digital Circuits Represented by SSBDDs** **13**
2.1 Overview of SSBDD Model 13
2.2 Size and Complexity of SSBDD Model 15
2.3 SSBDD-Based Simulation: Experimental Result Analysis 16
2.4 Design error diagnosis 18
2.5 Main contributions of this chapter 19

**At-Speed Interconnect Testing and Diagnosis** **21**
3.1 Introduction 21
3.2 Building blocks of at-speed interconnect BIST 22
3.3 Overall architecture of at-speed interconnect BIST and diagnosis 24
3.4 Main contributions of this chapter 26

**Conclusions** **29**

**References** **31**

## PART II: RESEARCH PAPERS 33

1. A. Jutman, "On SSBDD Model Size and Complexity", *in Proc. of 4th Electronic Circuits and Systems Conference (ECS'03)*, Bratislava, Slovakia, September 11-12, 2003, pp. 17-22. 35
2. R. Ubar, A. Jutman, Z. Peng, "Timing Simulation of Digital Circuits with Binary Decision Diagrams," *Proc. of DATE 2001 Conference*, Munich, Germany, March 13-16, 2001, pp. 460-466. 43

3. A. Jutman, J. Raik, R. Ubar, "SSBDDs: Advantageous Model and Efficient Algorithms for Digital Circuit Modeling, Simulation & Test," in *Proc. of 5th International Workshop on Boolean Problems (IWSBP'02)*, Freiberg, Germany, Sept. 19-20, 2002, pp. 157-166. 53

4. 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. 65

5. A. Jutman, "At-Speed On-Chip Diagnosis of Board-Level Interconnect Faults", in *Formal Proc. of 9th European Test Symposium (ETS'04)*, Corsica, France, May 23-26, 2004 81

**Curriculum Vitae** **89**

**Elulookirjeldus (CV)** **91**

**Selected Publications** **93**

# List of Abbreviations

ASIC – Application Specific Integrated Circuit
ATE – Automatic Test Equipment
ATPG – Automatic Test Pattern Generation
BDD – Binary Decision Diagram
BIST – Built-In Self-Test
BS – Boundary Scan
CPU – Central Processing Unit
DfT – Design for Testability
EDA – Electronic Design Automation
IC – Integrated Circuit
IEEE – Institute of Electrical and Electronics Engineers
I/O – Input/Output
ITCC – Interleaved True/Complement Code
ITRS – International Technology Roadmap for Semiconductors
LFSR – Linear Feedback Shift Register
MCM – Multi-Chip Module
MISR – Multiple Input Shift Register
MPU – Microprocessor Unit
OBDD – Ordered Binary Decision Diagram
PCB – Printed Circuit Board
RA – Response Analyzer
R&D – Research and Development
RTL – Register Transfer Level
SoC – System-on-Chip
SSBDD – Structurally Synthesized Binary Decision Diagram
TG – Test Generation
TPG – Test Pattern Generation

Part I

# Overview

# Chapter 1

# Introduction

## 1.1 Foreword

The five research papers lying in the basis of this thesis describe research in different aspects of design modeling, verification and testing. The range of considered problems is extending from digital circuit modeling to board-level embedded diagnosis. It is relatively difficult to push all these diverse topics into a single logically connected monograph. Therefore, I selected the option of collecting the most important representative papers and writing an introductory part, which attempts to bound them.

The whole work can be divided into two major parts:

a) verification and diagnosis of integrated circuits (IC);

b) testing and diagnostics of multi-chip modules (MCM).

The importance of the verification and testing in the scope of design and test cycle of electronic systems is considered in the first chapter of the thesis. This chapter is also pouring light on some aspects worse mentioning but for different reasons not included into the published articles. The other two chapters provide a detailed overview of the research topics embraced by this thesis and describe the results obtained by the author.

Chapter 2 concentrates on IC verification and diagnosis. Various simulation methods are considered with a strong shift to timing simulation as one of the most important verification domains in today's world of high frequency ICs. The process that follows the verification, the design error diagnosis, was also one of the research topics. The unifying factor for these diverse research areas is the design representation model. The unique property of this model is the ability of describing a digital circuit at a level, which is slightly above the pure logic level. The model, however, keeps the essential structural information about logic-level design, while reducing the complexity of simulation algorithms running on this model and, therefore, reducing the CPU time and, in the end, the time-to market of the final product. The author studies the complexity of this model and proposes the directions of improving its efficiency.

The third chapter introduces reader to the area of MCM and board-level testing. The research described in that chapter extends the Boundary Scan architecture so that it enables at-speed testing of interconnect faults and expands the range of testable faults to delays. New test generation algorithm, new BIST/DfT architecture, new diagnostic method and on-chip diagnostic analyzer for at-speed and functional interconnect testing are among contributions of that research possibly leading to a new improved board testing paradigm and, due to high similarity degree, a new approach to interconnect testing in a SoC.

## 1.2 Motivation

Semiconductor industry has a unique feature – improvement of its most important characteristics by the factor of two every eighteen months. This became possible due to vast investments in research and development (R&D) together with industry's ability of exponential decreasing of minimum feature sizes used in manufacturing of integrated circuits. In its turn, it results in exponential reduction of the cost per function in microprocessors and ICs [10]. Such a phenomenon holds for the last 40 years and it is commonly referred to as Moore's law.

There are several very important consequences of that trend. First of all, the living quality of the society is constantly improving via proliferation of consumer and industrial electronics, computers, and electronic communication. At the same time, the productivity of an individual and the society as a whole grows as well.

The price the industry has to pay for this progress is the increase in investments in research and development sphere. The search for new technologies in the pursuit of the Moore's law became itself a target of the industry – not a consequence as it used to be for decades. There is a strong challenging market pressure in form of customer's interest in faster, cheaper, lighter, and quality semiconductor devices.

This trend has motivated the industry to search for a reliable estimation of its R&D needs for the near future and it gave rise to collaboration of companies and professionals in form of various consortia and partnerships. The worldwide cooperation called the International Technology Roadmap for Semiconductors (ITRS) started in 1992 [3] represents an especially successful example of such a consortium. Its aim is predicting the semiconductor industry development trends and needs for the next 15 years. Reflecting the industry-wide consensus, the Roadmap provides a detailed investment guide for research institutions and companies by pinpointing areas and directions that truly need a research breakthrough.

Based on a long term analysis, the Roadmap predicts several positive trends reflected in Table 1.1, which is an excerpt from the latest edition (2003) of the Roadmap [11].

**Table 1.1 Roadmap trends in semiconductor industry (excerpt) [11]**

| | | 2005 | 2006 | 2007 | 2008 | 2009 | 2010 | 2012 | 2013 |
|---|---|---|---|---|---|---|---|---|---|
| **Printed Gate Length – MPU (nm)** | | 45 | 40 | 35 | 32 | 28 | 25 | 20 | 18 |
| **Cost per Function – MPU (microcents/transistor)** | | 24 | 17 | 12 | 8.6 | 6.1 | 4.3 | 2.5 | 1.5 |
| **On-chip Local Clock (GHz)** | | 5.2 | 6.8 | 9.3 | 11 | 12,4 | 15 | 20 | 23 |
| **Signal I/O Pads – MPU** | | 1024 | 1024 | 1024 | 1109 | 1195 | 1280 | 1365 | 1408 |
| **Signal I/O Pads – ASIC (high performance)** | | 2000 | 2100 | 2200 | 2300 | 2400 | 2400 | 2600 | 2700 |
| **Tester Cost per Pin ($/pin) – ASIC (high performance)** | **max** | 3000 | 3000 | 3000 | 3000 | 3000 | 4000 | 4000 | 4000 |
| | **min** | 1000 | 1000 | 1000 | 1000 | 2000 | 2000 | 2000 | 3000 |

The first row of the table reflects the constant growth of density of integrated circuits. The reduction of the printed gate length in microprocessors and other ICs enables designers to continuously increase the amount of functionality per chip. In its turn, it results in smaller electronic devices with reduced power consumption levels, which is a positive trend from the customer and marketing point of view.

The same trend has another positive consequence, which is reflected by the second row of the table. It is the cost per function that decreases exponentially over time and brings even more satisfaction to the customer.

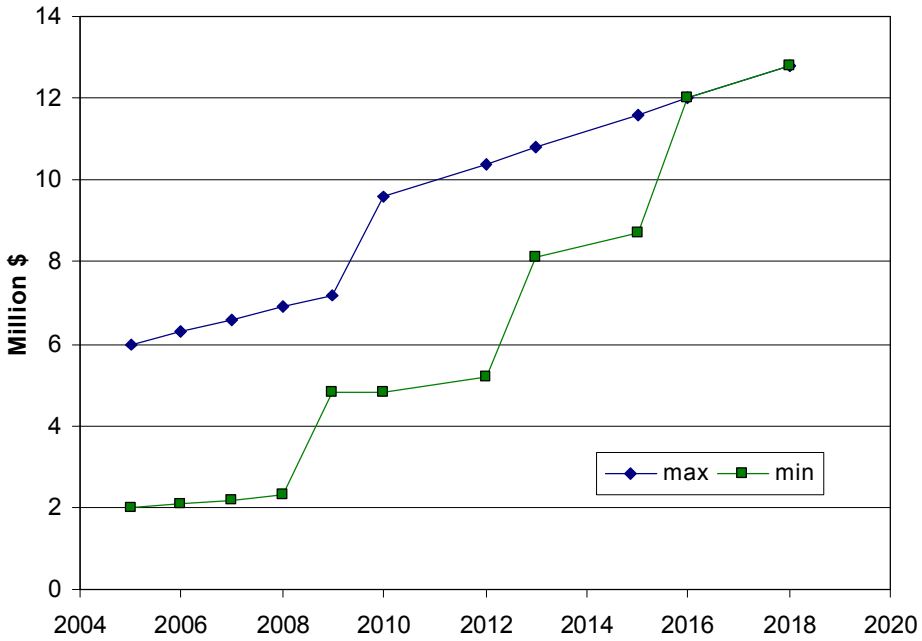



**Figure 1.1 Estimated average cost of volume testing**

The core operating frequency demonstrates the same exponential growth (row 3) allowing more operations executed per unit of time. In this way the functional density grows both in space and time domains. The major consequences of that fact is rapidly growing complexity and costs of validation, verification, and testing of ICs.

Despite the fact that the testing technology is also developing fast, the cost of testing accordingly to the Roadmap will be continuously increasing in the future. The last two rows of Table 1.1 represent the estimation of minimum and maximum cost-per-pin of a volume manufacturing tester. It is slightly but constantly growing. The timeline of signal pins per package has the same behavior. Based on this data, one can conclude that the average cost of volume testing will be rapidly increasing, which is shown in Figure 1.1. In fact, the base cost of a tester, which is around 0.5 million dollars [12], should be added to these calculations increasing the ATE cost even more.

Another severe warning in the ITRS roadmap is so called "design productivity gap" referred to the fact that the number of transistors in IC grows faster than the ability to design them consciously [8]. According to Intel, a 1981 leading edge chip contained $10^4$ transistors and required 100 designer months, which makes 100 transistors per month per designer. In 2002, such a chip contained already $150 \cdot 10^6$ transistors and required $3 \cdot 10^4$ designer months making it 5000 transistors/month per designer. Thus, the designer's load increased 50 times in 20 years (see Fig. 1.2).

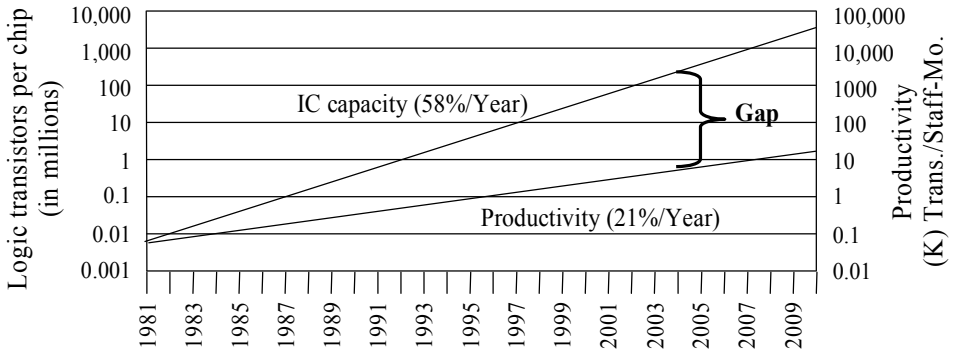



**Figure 1.2. Design productivity gap**

Electronic design automation (EDA) tools are a major force that helps designers to keep pace with this impressive trend. However, this is still not enough since the increase in design efforts from 100 to $3 \cdot 10^4$ designer/months per chip being further escalated leads to overgrown designers staff. The design cost has already grown 300 times compared to year 1981. Therefore, an intensive research in EDA area has always been among the first priorities of the industry and it will continue to be so. Since verification and testing already can take up to 70 percent of IC design cycle, it's been an area of emphasis for most of EDA vendors and many researchers [8].

The facts considered here indicate that there is a strong motivation for further R&D work to be done in semiconductor industry. The motivation for research in testing domain – in both DFT and test automation tools – is one of the strongest all over the industry. This makes the work presented in current thesis and ultimately aimed at test cost reduction during both development and manufacturing phase, being relevant in today's world. The next sections detail the areas of my research and further reveal the motivation behind it.

## 1.3 Design and test cycle of development of a digital system

Since verification and testing are tightly bounded up with design cycle, we will consider the main phases of development of electronic systems (design, synthesis, verification, and test) and point out those of them related to the research described in current thesis. Modern electronic systems consist of hardware and software. This thesis concentrates on digital hardware testing. The reason for that is the fact that majority of the hardware produced today consists of digital circuits.

Though the design process has been changing continuously to reflect the status of technology and EDA tools on the market, it still consists of three main phases: system design, logic design, and physical design [16]. Normally, each design phase is supported by EDA tools for entering, verifying, and transformation of the design to the next phase. A general picture of a design cycle is given in Figure 1.3.

The design starts from general concept or some sort of specification. Then it is entered usually at behavioral or at register transfer level (RTL). It undergoes then several synthesis steps being transformed to the lower level of abstraction. At that time, the design is being modeled by four different representations – behavioral, RTL, gate level, and layout. They are diverse projections of the same circuit. During synthesis from one representation to another, it is likely that some errors are induced. They can be caused by bugs in EDA tools, designer's misuse of the tools, or just designer's manual intervention into the synthesis process or into the final representation driven by intention to improve some parameters of the circuit, like timing, level of power consumption, etc. during design optimization steps.

The mechanism used to check the lower level representation for conformance with the higher (highest) one is called design verification. Basically, there are two different ways of verification – simulation and formal verification. Although, extensive research has been done in formal verification during the last decade, there are few practical tools available for the designer and, therefore, the simulation still remains the most popular way of design verification [16].

Several types of simulation are used during verification. The most important is functional simulation, which is primarily used to check that all blocks of the circuit perform their intended functions. Modern EDA tools allow extraction of

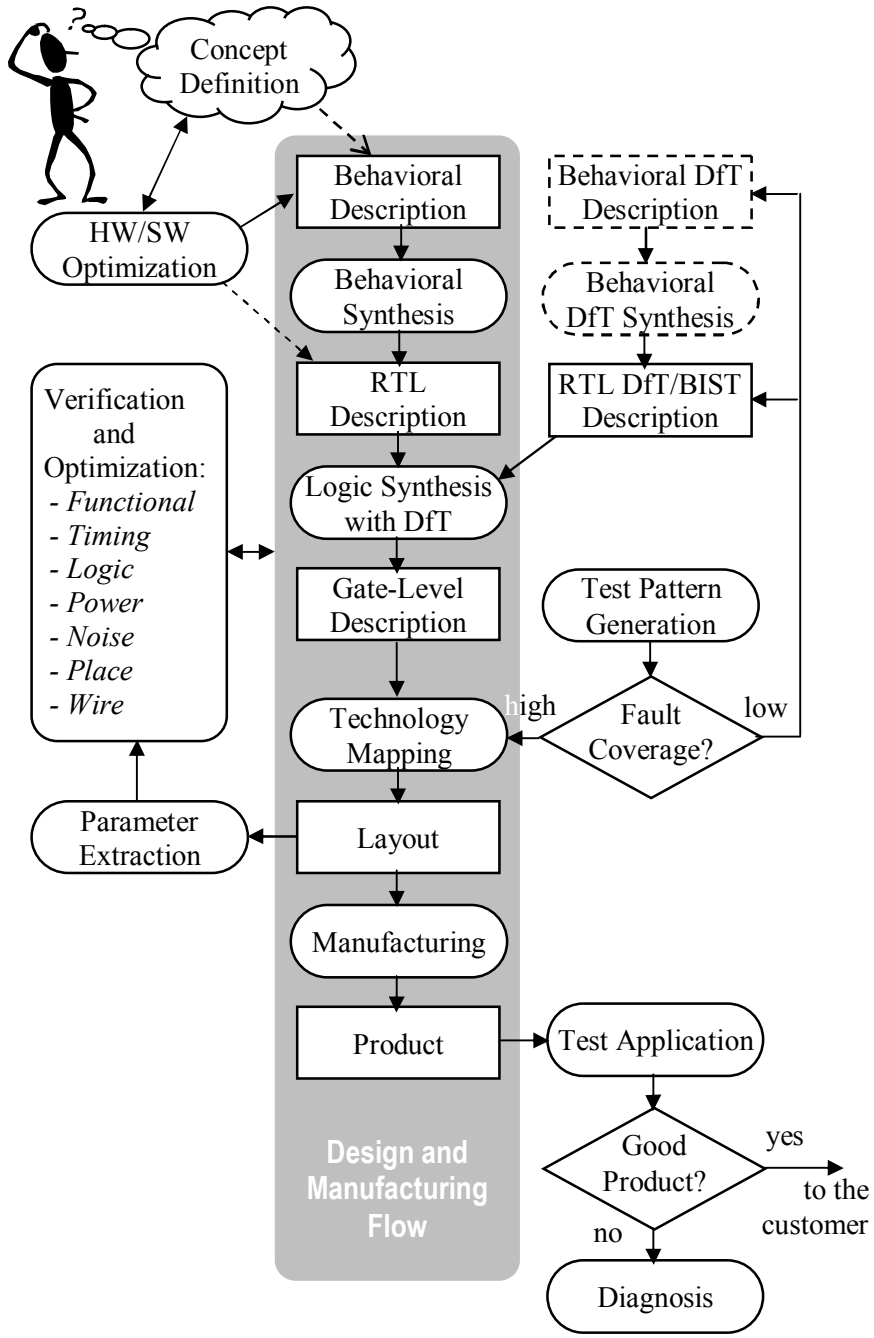



**Figure 1.3 Design and test cycle**

various parameters and, in this way, facilitate simulation under actual process conditions, which allows to build sufficient confidence in the design to launch production of the prototype.

Timing simulation is one of traditionally performed simulation types. The ultimate goal of timing simulation is to check the performance of the design at a target operating speed. This is due to the treat that in many cases the leading edge high-speed designs fail to operate correctly at speed, although they are functionally correct. Therefore, the weak points should be identified and the design must be further optimized.

There are yet more examples of simulations a design must successfully pass before starting the production cycle. For example, it might be important to estimate power consumption or heat distribution or even electromagnetic noise the future device will feature. In the same way, the reliability and testability of electronic systems are no longer a concern limited to the aerospace, military, or medical industry. Therefore, most of the designs must also pass a testability assessment phase, which is achieved through test generation and fault simulation.

Hence, various kinds of simulation are important for the quality electronic system to be produced and if simulation reveals that the design doesn't fit into some of the imposed requirements, then designers repeat one or more steps of optimization. The latter represents another possible source of errors that should be newly checked during subsequent verification step.

The ultimate goal of the verification is to check the correctness of the design. The ultimate goal of the designer, in case of error detection, is the localization and subsequent rectification of that fault. The diagnosis of functional faults is commonly referred to as design error diagnosis. Its aim is the localization of functional blocks that fail to perform their intended functions. This topic gained attention of the researchers in the last decade as the complexity of the designs on the logic level overgrown the limits of conscious analysis.

Fault diagnosis is also used in manufacturing testing when a fabricated chip is being checked for manufacturing defects. In this case, the diagnosis might be used for improving the manufacturing process.

Talking about manufacturing and assembly testing we can separate three main phases: chip-level, PCB-level, and system level. The system level testing incorporates mostly software testing since, up to this point, the hardware has normally undergone thorough chip-level and PCB-level testing and can be considered as correct. The reason for such a separation of testing phases is simple. It comes from economical reasons and called "the rule of 10" [5], which says that testing a defective IC that has been placed on board is 10 times more expensive than testing it separately. If the board is already placed into the final electronic system, then testing an escaped fault is yet 10 times more expensive. This is the reason why testing in the system-on-chip (SoC) era requires new well-structured systematic approaches [12] and places high stakes on various DfT and BIST solutions.

The PCB-level testing has been extremely simplified with the introduction of the IEEE 1149.1 Boundary Scan standard [9] in 1990. Due to Boundary Scan (BS) PCB testing turned to just an interconnect testing between chips in case when it does not involve glue logic. In the BS era, there are just two major

difficulties related to board testing. They are: a) handling of non-BS devices and b) handling of glue logic. Recently, test engineers and researchers became concerned to another very important topic – at-speed testing. Indeed, with growing due to Moore's law clock frequencies, there is less spare space left for speed parameter variations in the design. This leads to the situation where physical phenomena like crosstalk noise and other capacitance-and-inductance-related effects are becoming more and more important. The reaction time of sending and receiving buffers is also coming into the game.

## 1.3 Contribution of selected research papers

The essence of my research is thoroughly described in corresponding papers from the second part of the thesis. The papers are given rather in logical order than the chronological one.

The first paper is called "**On SSBDD Model Size and Complexity**" and presented by author at the 4th Electronic Circuits and Systems Conference (ECS'03) in Bratislava, Slovakia, in September of 2003. This paper addresses an important matter all the other papers from Chapter 2 are based on. It studies a mathematical model called Structurally Synthesized Binary Decision Diagrams (SSBDD), which is used as a circuit representation in design simulation and diagnostic algorithms described in that chapter. The most important contribution of this paper is the formal proof of the fact that differently from other BDD types, SSBDD has linear complexity with respect to the number of logic gates for an arbitrary combinational circuit. This fact was mentioned in literature before [21] but the arguments were based on experimental observations and general speculations.

It has also been shown that the size of SSBDD model generated from a logic-level netlist is always smaller than the size of the netlist exactly by the number of logic gates in that netlist. This fact placed an efficient basis for porting gate-level simulation algorithms from commonly used gate-level netlist to the SSBDD model. Although, this fact was not a surprise, its formal proof was missing up to appearing in this paper.

Several equations for exact calculation of SSBDD model size were derived. The simplest one is based on two basic parameters of combinational circuits: the number of logic gates and the number of signal lines (or half the number of un-collapsed stuck-at faults). The same equation can be used, for example, for calculation of the number of checkpoints [1] in logic-level circuits.

The next paper handles one of the most critical tasks to be solved by EDA tools – timing simulation. The paper called "**Timing Simulation of Digital Circuits with Binary Decision Diagrams**" was presented by author at the Design Automation and Test in Europe (DATE'01) conference in Munich, Germany, in March of 2001. The author contributed to a new based on SSBDD model, timing simulation algorithm, which outperforms traditional logic-level timing simulation as it should have been expected provided the previous paper was available at that time.

Driven by the desire of further improvement of the method and searching for hidden relationships between simulation speed-up and certain parameters of the SSBDD model some experimental research has been conducted, which involved other logic-level simulation algorithms like functional, multi-valued, and fault simulation. The result of this study was included into a summarizing paper that describes the model itself and its most remarkable properties along with studied simulation methods and found dependencies. The paper called "**SSBDDs: Advantageous Model and Efficient Algorithms for Digital Circuit Modeling, Simulation & Test**" was presented by author at the 5th International Workshop on Boolean Problems (IWSBP'02) in Freiberg, Germany, in September of 2002.

It turned out that the efficiency of the considered simulation algorithms depends on one of the characteristics of SSBDD model. For all of them, the simulation speed-up, in comparison with the gate-level simulation speed, was linearly proportional to the average size of macro measured in the number of gates. The main conclusion of this paper is such that circuits with larger macros transformed into SSBDDs demonstrate higher simulation speed-up reaching the maximum with pure tree-like circuits.

Further successful improvement of the simulation efficiency was achieved by experimenting with SSBDD reordering [22].

The next contribution in the scope of this thesis is related to the design error diagnosis that is applied after functional verification. The method represents a hierarchical approach where fast simulation based on SSBDDs is used first and then precise calculations on gate-level netlist are applied. Another advantage of the method is that it is based on functional test vectors used in verification and does not require computationally expensive generation of diagnostic vectors. Such a strategy possesses a remarkable speed-up in comparison to the methods published before. Another feature of this method lies in an unusual utilization of the stuck-at fault model in design error diagnosis. This research was published in Journal of Microelectronics Reliability, Pergamon Pressin, in 2000 under the title "**Design Error Diagnosis in Digital Circuits with Stuck-at Fault Model**".

The last selected paper is named "**At-Speed On-Chip Diagnosis of Board-Level Interconnect Faults**" and presented by author at the 9th European Test Symposium (ETS'04) in Corsica, France, in May of 2004. It was also selected by reviewers for publication in formal proceedings of the conference.

The paper contributes simultaneously to board-level manufacturing test generation and test application, DFT, BIST, and embedded fault detection and diagnosis for both functional and at-speed testing. The proposed framework is an extension of Boundary Scan architecture, which takes into account all the important issues a practically efficient at-speed interconnect BIST framework must handle, such as: parallel application of test patterns, detection of both static and dynamic faults, bus contention problem etc. At the same time, the hardware implementation of the proposed architecture is among the most efficient ones in terms of silicon area. Therefore, the results of this research

possibly lead to a new improved BS testing paradigm where the system is capable not only of at-speed self-testing but also of at-speed self-diagnosis.

To summarize, the research papers collected in the second part of the work describe research in different aspects of design modeling, verification and testing. The range of considered problems is extending from digital circuit modeling to board-level embedded diagnosis. The emphasis is made upon:

a) studying the SSBDD model used in IC design verification and diagnosis;

b) development of at-speed interconnect BIST framework.

These topics are further detailed in the next chapters from Part I and corresponding papers from Part II of the thesis.

Chapter 2

# Simulation and Diagnosis of Digital Circuits Represented by SSBDDs

## 2.1 Overview of SSBDD Model

The SSBDD stands for Structurally Synthesized Binary Decision Diagram, yet another mathematical representation of logic-level digital circuits. This model was developed in Tallinn University of Technology and first time published in 1976 [23] in Soviet Union (in Russian) ten years before the famous paper on Binary Decision Diagrams (BDDs) by Bryant [4] and two years before a well known article by Akers [2]. For a long time, the model was also known as Alternative Graphs [24, 25] until it was once renamed and finally called SSBDDs. This new name better reflects the essence of the model since SSBDDs are, in fact, a subclass of BDDs.

SSBDDs, however, should be considered as a very special subclass, since they are normally used in such areas of digital design and test that are quite unusual for other kinds of binary decision diagrams, which are mainly used in formal verification [6]. Due to its ability of keeping structural information about digital circuit, the SSBDD model found its application in VLSI testing as a model of logic-level digital circuits and related manufacturing defects.

One of the most important and time consuming tasks in VLSI testing is test generation which is rarely possible without fault modeling and in many cases relies on fault simulation. The SSBDD based stuck-at fault modeling, for instance, comes with an automatic fault collapsing as one of the properties of the model. Therefore, such a complexity reduction is achieved without any related computational efforts. Similar effects appear in other modeling or simulation domains.

Normally, the SSBDD model is used for representation of the structure and function of combinational logic. However, it can be supplemented by additional parameters to extend the class of modeled phenomena. Sequential logic, for instance, can be described by introducing a special delay flag into the model. By

adding timing information, one can use the model for delay fault simulation, hazard analysis, or timing simulation [14,15].

The model has been also extended to higher level of abstraction. As the result, the high-level decision diagrams (HLDDs) [19] were introduced. High-level DDs are used for describing digital designs at the RTL and behavioral level. The two models – HLDD and SSBDD have many common features, which results in similar simulation and modeling algorithms. Moreover, the combination of two models enables development of powerful hierarchical (two-level) algorithms that combine the speed of high-level and the accuracy of logic-level models. For example, a very fast hierarchical ATPG based on this idea has been recently developed in TTU [19, 20].

Similarly to other BDD classes, there is an important issue of node reordering in the SSBDD model. The reordering normally helps to reduce the size of Ordered BDD (OBDD) [6] and to achieve the canonicity (uniqueness) of representation. On the contrary, the size of SSBDD model does not depend on ordering and the ordering itself has a different meaning here. The reordering of SSBDD nodes, which is performed in a certain way, reduces CPU time for some simulation algorithms [22]. Therefore, the procedure of SSBDD model extraction from a digital circuit must take into account those operations which will be performed with this model in future. It is also possible to reorder the model later for other applications.

One of the biggest advantages of SSBDD model is its size, which is always linear to the size of logic-level netlist from which the model is initially generated [13]. With other kinds of BDDs there has always been a problem of complexity, which is exponential in many cases. However, the application domain of SSBDDs compels us to compare this model not with other BDD classes but rather with such logic-level representations of digital circuits as the gate-level netlist. This part of the work is dedicated to study of special properties of the SSBDD model, utilization of them in simulation algorithms for design verification, and analysis of the experimental results.

An example of a BDD that represents Boolean function $y = x_1 x_2 + \overline{x}_2 \overline{x}_3$ is given in Figure 2.1.

The main purpose and the main property of SSBDD model is the ability of representation of the structure of the logic circuit. For instance, the same

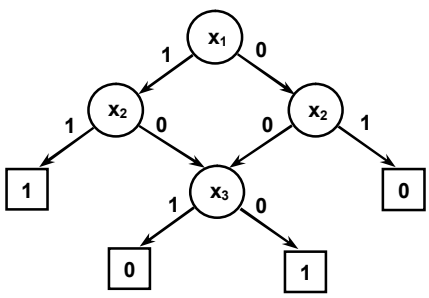



**Figure 2.1 A binary decision diagram**

Boolean function can be transformed as follows: $y = x_1 x_2 + \overline{x}_2 \overline{x}_3 = x_1 x_2 + \overline{x}_2 \overline{x}_3 + x_1 \overline{x}_3 = \overline{x}_3 (x_1 + \overline{x}_2) + x_1 x_2$. However, the traditional BDD, which is based on Shannon expansion, does not need to be transformed to reflect the change, since the Boolean function remains the same. This is exactly opposite when dealing with SSBDDs. Let us consider two combinational circuits corresponding to both forms of the mentioned Boolean function (Figure 2.2).

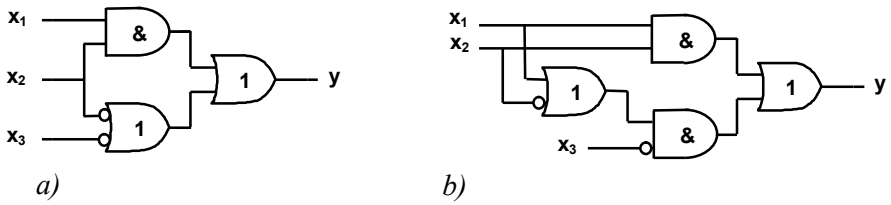



**Figure 2.2 Two realizations of the same Boolean function**

The respective SSBDD representations are given in Figure 2.3.

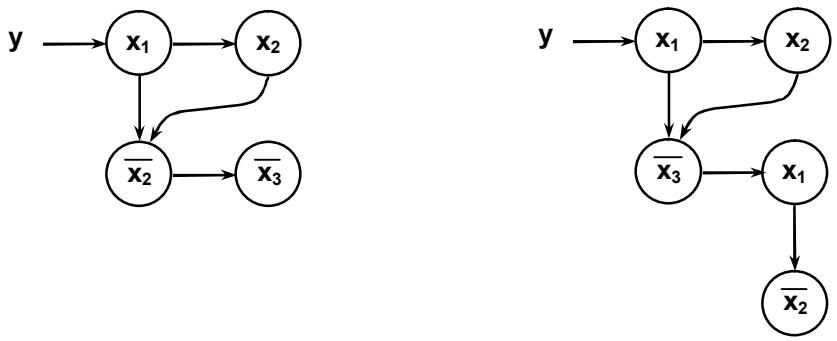



**Figure 2.3 Corresponding SSBDD representations for the circuits from Figure 2.2**

In SSBDD model, there is a *one-to-one correspondence* between non-terminal nodes of the SSBDD and signal paths in the combinational circuit. In general, this is the main factor in SSBDD's ability of keeping structural information. Such property is clearly seen from the figure above. Indeed, each node in the graph corresponds to a respective input of the combinational circuit. The way, the nodes are connected inside the graph, in its turn, reflects the function of the circuit and, at the same time, the function of gates in this circuit.

## 2.2 Size and Complexity of SSBDD Model

The fundamental result of the first paper from Part II is the following equation:

$$N_{nodes} = N_{signals} - N_{gates} \quad (1)$$

It describes the relationship between the size of SSBDD model and the size of the initial logic-level netlist this model was extracted from. Here, $N_{nodes}$ is the number of nodes in SSBDD, $N_{signals}$ and $N_{gates}$ are, correspondingly, the number of signal lines and the number of gates in the netlist. Taken into account the fact that many simulation algorithms operate with signal lines rather than gates, the netlist complexity should be measured using this parameter. A good example of such a simulation is fault simulation, where the number of uncollapsed stuck-at faults is exactly $2 \cdot N_{signals}$.

Hence, the first conclusion from this equation is that the SSBDD complexity is linear to the complexity of the netlist it is extracted from. Moreover, the size of resulting SSBDD model is less than the netlist size exactly by the number of gates in the circuit.

We can play with that equation and see that the only case when $N_{nodes} = N_{signals}$ is when $N_{gates} = 0$. This hypothetical case corresponds to a circuit consisting of a single wire. Then the simulation performance on both models should be equal. Another extreme case is when $N_{signals} = N_{gates}$, which is impossible in practice and therefore we will never receive an infinite speedup. The highest speedup will be achieved when $N_{signals} - N_{gates} = 1$. This case corresponds to a circuit that is a chain of inverters or buffers with a single primary input and single primary output.

In general, the higher the ratio $N_{gates} / N_{signals}$ the bigger the simulation speedup we can achieve moving from netlist circuit representation to SSBDD model. Hence, the longer the chain of inverters/buffers the higher the speed-up. In such a way, the equation helps to understand the advantage of SSBDD model over the gate-level netlist and reveals the nature of SSBDD-based simulation speed-up.

Another interesting application of this equation comes from the domain of fault collapsing. For those who became familiar with SSBDD model, it is easy to notice that $N_{nodes}$ is actually equal to the number of checkpoints in the circuit, which is defined as the sum of primary inputs and fanout branches [1]. Number of checkpoints multiplied by 2 can be used as a lower bound of faults remained after fault collapsing. Abramovichi et al. [1] derive, in their book, a formula for estimation of this number. The formula operates with four different parameters: number of gates, inputs, average fanout count ($f$), and the fraction of signal sources with only one fanout ($q$). It is given as

$$N_{checkpoints} = N_{inputs} + (N_{inputs} + N_{gates})(f - q) \quad (2)$$

or the lower bound of collapsed faults in the circuit is equal to

$$N_{collapsed\_faults} = 2 \cdot N_{inputs} + 2 \cdot (N_{inputs} + N_{gates})(f - q) \quad (3)$$

Now we can easily calculate this value using our equation and simple circuit parameters. Indeed

$$N_{collapsed\_faults} = 2 \cdot (N_{signals} - N_{gates}) \quad (4)$$

or since $2 \cdot N_{signals} = N_{uncollapsed\_faults}$, then

$$N_{collapsed\_faults} = N_{uncollapsed\_faults} - 2 \cdot N_{gates} \quad (5)$$

We can continue and calculate the fraction of faults eliminated. In the book [1] it is approximated as

$$1 - \frac{N_{collapsed\_faults}}{N_{uncollapsed\_faults}} = \frac{1}{1 + f - q} \quad (6)$$

From equation 5 we can get simpler formula:

$$1 - \frac{N_{collapsed\_faults}}{N_{uncollapsed\_faults}} = 1 - \frac{N_{uncollapsed\_faults} - 2 \cdot N_{gates}}{N_{uncollapsed\_faults}} = 2 \frac{N_{gates}}{N_{uncollapsed\_faults}} \quad (7)$$

## 2.3 SSBDD-Based Simulation: Experimental Result Analysis

Besides the description of SSBDD model and simulation algorithms, the main contribution of the second research paper from part II represents the experimental results that reveal clear relationship between an SSBDD parameter called the average macro size and the simulation speed-up compared to corresponding simulation algorithms running on a gate-level netlist. These results are illustrated in Figures 2.4 and 2.5.

From these figures we can see that for different circuits the speedup ranges from around 1,5 to 3,5 times (except fault simulation that shows better performance) and does not depend on the circuit size. The conclusion of that paper was that the speedup is directly proportional to the gate/macro ratio, i.e. to $N_{gates} / N_{macros}$. This result was acquired almost accidentally in a pure experimental way by observing and comparing different trends and circuit parameters. Now, when we have got equipped with equation (1) from previous section, we can, through calculation, estimate the ratio of simulation

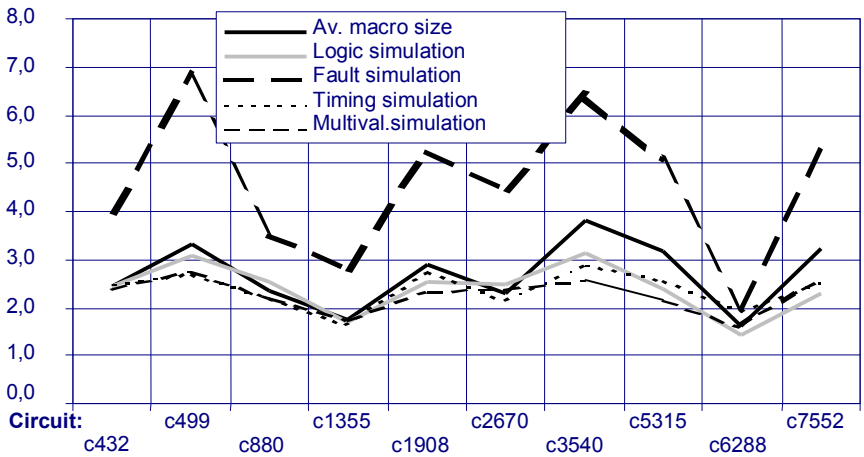



**Figure 2.4 Logic-level simulation speedup for different algorithms**

complexities for both circuit representations. The simulation speed-up must be equal to that ratio which is roughly equal to the ratio of sizes of the models i.e. $N_{signals} / N_{nodes}$.

$$\frac{N_{signals}}{N_{nodes}} = \frac{N_{nodes} + N_{gates}}{N_{nodes}} = 1 + \frac{N_{gates}}{N_{nodes}} \tag{8}$$

Hence, we have got a better estimation of the comparative simulation performance. The further refinement of that depends on implementation details of corresponding algorithms and cannot be determined by pure comparison of the models. This is illustrated by fault simulation example, which although obeys the same behavior as other algorithms, numerically it differs a lot.

Other aspects and contributions related to simulation based verification of

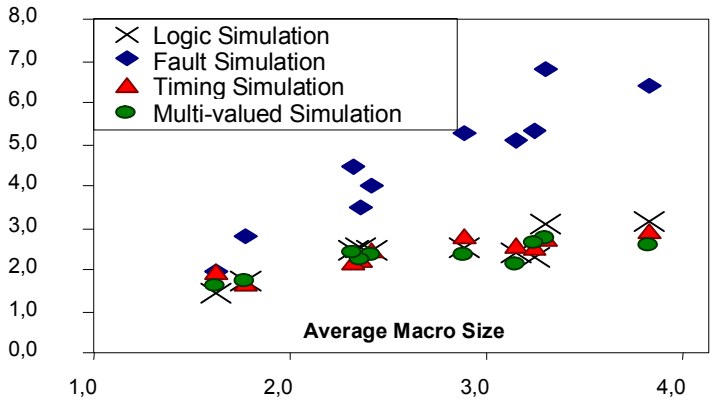



**Figure 2.5 Logic-level simulation speedup vs. average macro size**

digital circuits are considered in the second and third papers of part II.

## 2.4 Design error diagnosis

This section is devoted to the design error diagnosis method, which is very thoroughly described in the journal paper given in Part II. Hence, only the most important issues are discussed here.

The algorithm represents a hierarchical approach where fast simulation based on SSBDDs is used first and then precise calculations on gate-level netlist are applied. Another advantage of the method is that it is based on functional test vectors used in verification and does not require computationally expensive generation of diagnostic vectors. Such a strategy possesses a remarkable speed-up in comparison to the methods published before.

The comparison of CPU times of previous and current method are given in two last columns of Table 2.1. The diagnostic resolution of the method is represented by minimum and maximum values (measured in number of gates) as well as by the average suspected area of the circuit (percentage). It can be seen that very quickly the erroneous area gets localized. However, it is possible to further improve the diagnostic resolution by generation of additional diagnostic vectors with special properties. This procedure, however, will negatively affect the CPU time.

The limitation of the method lies in the error model it is based on. New methods appeared recently are more flexible since they do not require any error model. This is the driving force for the future research directed on transferring the current method to the new diagnostic paradigm.

**Table 2.1 Diagnostic resolution and CPU times for ISCAS '85 benchmarks**

| Circuit Name | Fault Cove-rage, % | Diagnostic Resolution | | | | Time, s | | | Time for [26], s |
|---|---|---|---|---|---|---|---|---|---|
| | | No. of gates | | | Av. % of Area | Test Gene-ration | Fault Analysis (average) | Total | |
| | | Min | Max | Av. | | | | | |
| c432 | 93,02 | 3 | 92 | 11,3 | **4,86** | 0,62 | 0,06 | **0,7** | 17,57 |
| c499 | 99,33 | 1 | 307 | 73,3 | **11,86** | 1,01 | 0,8 | **1,8** | 111,64 |
| c880 | 100 | 1 | 33 | 5,7 | **1,60** | 0,19 | 0,3 | **0,5** | 126,79 |
| c1355 | 99,51 | 1 | 248 | 55,2 | **10,74** | 1,35 | 0,9 | **2,3** | 241,79 |
| c1908 | 99,31 | 3 | 70 | 12,3 | **1,71** | 0,93 | 1,0 | **1,9** | 341,92 |
| c2670 | 94,97 | 69 | 218 | 91,1 | **9,14** | 3,55 | 8,5 | **12,1** | 661,91 |
| c3540 | 95,27 | 107 | 190 | 115,7 | **8,00** | 3,08 | 2,3 | **5,4** | 1513,82 |
| c5315 | 98,69 | 6 | 204 | 15,7 | **0,79** | 2,38 | 17,4 | **19,8** | 1814,04 |
| c6288 | 99,34 | 17 | 92 | 21,5 | **0,89** | 2,17 | 1,7 | **3,9** | 1895,90 |
| c7552 | 95,95 | 131 | 372 | 144,3 | **4,84** | 12,06 | 26,8 | **38,9** | |

## 2.5 Main contributions of this chapter

This chapter describes the research that contributes mainly to the study of a promising logic-level design representation called the SSBDD model. The author succeeded to show, in a formal way, that this special kind of BBDs has linear complexity with respect to the number of logic gates for an arbitrary combinational circuit. Moreover, it is shown that the SSBDD model complexity is always smaller than the complexity of corresponding logic-level netlist exactly by the number of logic gates in this netlist. Several equations for exact calculation of SSBDD model size are derived. The simplest one is based on two basic parameters of combinational circuits: the number of logic gates and the number of signal lines (or half the number of un-collapsed stuck-at faults).

Another part of this research contributes to design verification via simulation. The author directly contributed to a new, based on SSBDD model, timing simulation algorithm, which outperforms traditional logic-level timing simulation.

During further study of the problem and search for the reasons of the model efficiency four other logic-level simulation methods implemented on the SSBDD model were experimentally studied. It turned out that the efficiency of the algorithms directly depends on one of the characteristics of the underlying model. For all the considered algorithms, the simulation speed-up, in comparison with the gate-level simulation speed, is linearly proportional to the average size of macro measured in the number of gates. It was also found that the algorithm that most benefited from the usage of the SSBDD model is the fault simulation algorithm, as it utilizes SSBDD based fault collapsing together with model reduction. Further successful improvment of the simulation efficiency was achieved by experimenting with SSBDD reordering [22].

The last contribution reflected in the scope of this chapter is related to the diagnosis that comes after functional verification. Similarly to the algorithms mentioned above, the design error diagnosis method developed by the author relies on SSBDD model. The method represents a hierarchical approach where fast simulation based on SSBDDs is used first and then precise calculations on gate-level netlist are applied. Another advantage of the method is that it is based functional test vectors used in verification and does not require computationally expensive generation of diagnostic vectors. Such a strategy possesses a remarkable speed-up in comparison to the methods published before.

Chapter 3

# At-Speed Interconnect Testing and Diagnosis

## 3.1 Introduction

Since the introduction of the IEEE 1149.1 Boundary Scan (BS) standard [9] in 1990, this powerful and universal BS architecture became the main mechanism of board-level testing, which in its turn reduced to interconnect testing between the chips. It is widely accepted that in the BS era, there are just two major difficulties related to board testing. They are: a) handling of non-BS devices and b) handling of glue logic. Recently, test engineers and researchers became concerned to another very important topic – at-speed testing. Indeed, with growing clock frequencies, there is less spare space left for speed parameter variations in the design. This leads to the situation where physical phenomena like crosstalk noise and other capacitance-and-inductance-related effects are becoming more and more important. The same importance gained the delay defects in sending/receiving buffers. These facts have recently motivated a new wave of research in this area.

Due to known issues like cost and speed, the ATE equipment is not the best choice for at-speed testing of modern complex systems. One of the main messages of the 2003 version of ITRS on Test and Test Equipment [12] is the substantial shift of test technology to the area of DfT and BIST.

The adoption of standard BS architecture for at-speed testing is relatively problematic. The most difficult issue is the timing between update and capture operations which spans at least 2,5 clock cycles. There are some solutions to this problem proposed in literature [17, 18]. All of them concentrate on modification of BS architecture but none targets the problem of parallel generation of test vectors. Normally, the vectors are shifted in and responses are shifted out via a serial interface. Hence, the TPG is usually placed outside of the chip.

The last paper presented in Part II of current thesis describes a revolutionary different approach to at-speed interconnect testing comprising of a new test generation algorithm, new BIST/DfT architecture, new diagnostic method and

on-chip diagnostic analyzer for at-speed and functional interconnect testing. The proposed framework might possibly create a foundation for a new improved board testing paradigm.

## 3.2 Building blocks of at-speed interconnect BIST

Basically there are four key elements in a successful at-speed testing solution. The first is the selection of the test sequence, which should detect all the necessary timing-related faults. The second is the parallel test pattern generation, i.e. each new pattern should be formed in the TPG at each new clock cycle (like in LFSR) and without any intermediate shift operations. Third, the response analysis must also be made at-speed and in parallel. Finally, the whole solution must be efficient in terms of silicon area.

The test generation algorithm proposed in the article is called *Interleaved True/Complement Code* (ITCC) and has comparatively good properties with respect to other known algorithms. In addition to detection of all possible opens and shorts it detects delay faults and, in many cases, also crosstalk and other transient phenomena. At the same time, the length of test sequence is kept within boundaries of commonly used test generation algorithms and is equal to $2\lceil log_2(N) \rceil$ where $N$ is the number of independent nets on board. The properties of this algorithm and comparison to others are detailed in the corresponding article.

The on-chip hardware that allows automatic pattern generation accordingly to the ITCC algorithm on the transmitter side and corresponding embedded diagnostic analyzer on the receiver side are also described. The generator is illustrated in Figure 3.1a and represents a very simple structure consisting of a shift register with a single feedback. Compared to a commonly used TPG (Figure 3.1b) it provides noticeable hardware reduction.

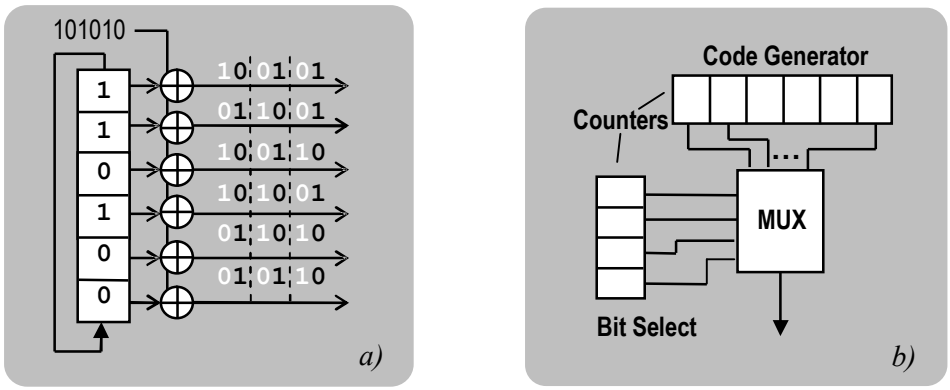



**Figure 3.1 Interconnect TPGs: new vs. traditional approaches**

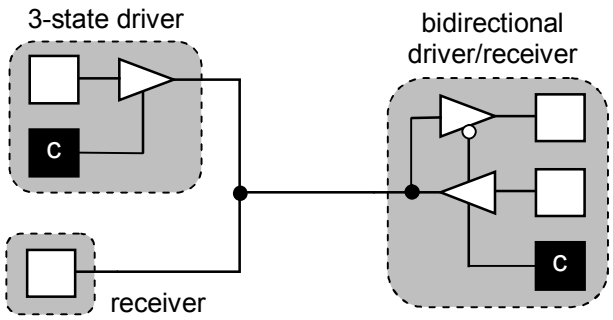



**Figure 3.2 A typical tri-state net in a board-level interconnect**

The test pattern generator should also handle the driver contention problem, which is illustrated in Figure 3.2. To avoid this problem, the TPG must take care of several drivers connected to the same net by proper activation of the corresponding control cells (denoted by "c" in Figure 3.2) so that no two drivers of the same net are ever active simultaneously. Otherwise, they might drive the net to the opposite logic values, which might even lead to a permanent damage of the system under test. A common solution to this problem, which is also utilized in current framework, is TPG partitioning as shown in Figure 3.3. The test data then represents composite vectors generated by two types of TPGs: C-TPG (for control signals) and D-TPG (for data signals).

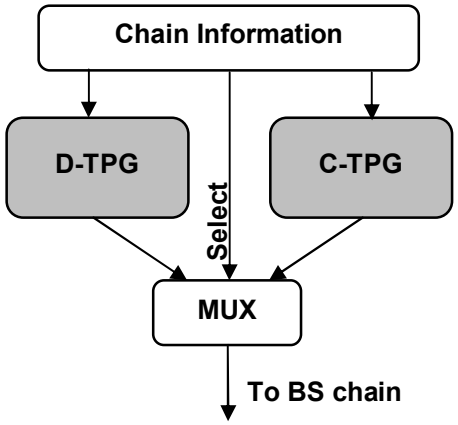



**Figure 3.3 Partitioned TPG for BS based interconnect BIST**

The embedded diagnostic response analyzer is shown in Figure 3.4 as a circuit for a separate receiver cell and in Figure 3.5a as a chain of these circuits that constitute RA chain. Due to properties of ITCC code the correct output of each RA cell is equal to 0. If the diagnostic signature shifted out from the RA chain contains 1-s, they clearly identify failing nets providing exact diagnosis. If just fault detection is required, then it can be performed on board by the circuit shown in Figure 3.5b.

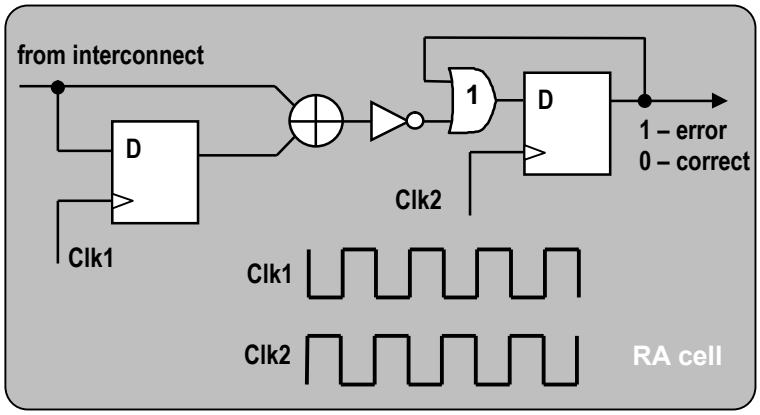



**Figure 3.4 Embedded diagnostic response analysis cell**

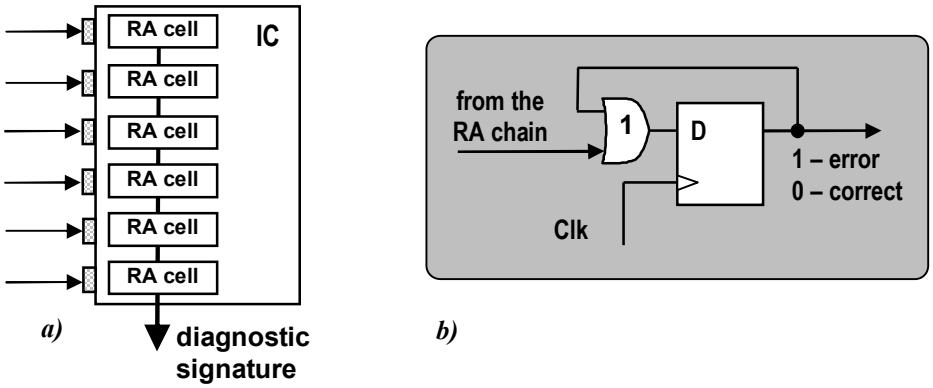



**Figure 3.5 Interconnect fault detection and diagnosis**

## 3.3 Overall architecture of at-speed interconnect BIST and diagnosis

The general structure is illustrated in Figure 3.6. It borrows most of architectural elements from the Boundary Scan architecture, which makes our approach being an extension of the BS standard. We partition the BS register into three separate scan registers: the register of input cells, output cells, and control cells. This brings the flexibility, which results in three advantages of the framework: 1) easy manipulation of control cells and, therefore, avoidance of the contention problem, 2) parallel at-speed test generation in the output scan register, and 3) on-chip at-speed precise interconnect fault diagnosis in the input scan register. At the board level, all the scan registers of all the chips should be connected into three separate scan chains (the at-speed testing mode in Figure

3.7a). In order to support the compatibility with the BS standard, there is a mode when these partitioned registers are connected into a single chain (the BS mode in Figure 3.7b). Such a switching hardware, which enables both modes, can be implemented either inside or outside the chip.

There is also an external (on-board) LFSR, which is used for generation of the initial state for the at-speed TPG. This at-speed TPG is nothing more than just a circular register formed through connecting all the corresponding output scan registers of all the chips into the scan chain. The initial state for the TPG is formed in ether the least or the most significant bit of the external LFSR. The size of the LFSR is equal to $\lceil log_2(K) \rceil$, where $K$ is the length of the scan chain. All the needed test vectors are formed in this chain just by consecutive shifting of the initial state by one bit in either direction. The diagnostic signature analysis is made in the scan chain of input cells, which hardware complexity is approximately the same as the one of the MISR.

The operation of the system is described as follows. First, the device is set into the at-speed interconnect testing mode. Then the initial state is formed in the external LFSR and shifted into the at-speed TPG, which takes $K$ clock cycles. The current interconnect configuration is selected in C-TPG and shifted into the control scan chain, which takes $C$ clock cycles, where $C$ is the length of that chain. Then, the test generation process is started. In general, $\lceil log_2(K+2) \rceil$ test patterns is generated for static faults, like opens and shorts, or $2\lceil log_2(K+1) \rceil$ test vectors for both static and delay faults. The number of needed clock cycles is equal to the number of generated test patterns in both cases. The diagnostic signature is constantly updated in the scan chain of input cells. When all the

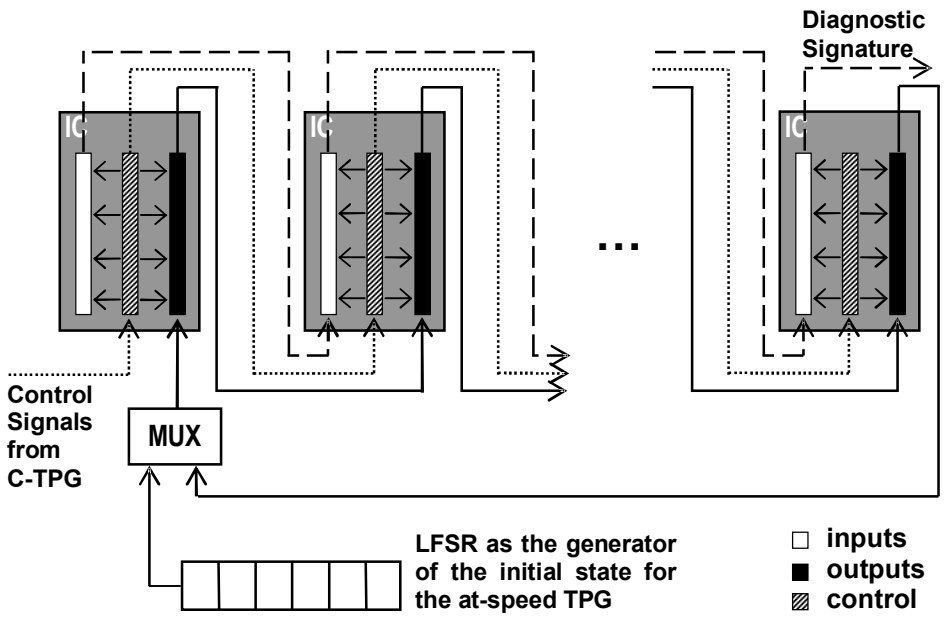



**Figure 3.6 General architecture of at-speed interconnect BIST and diagnosis**

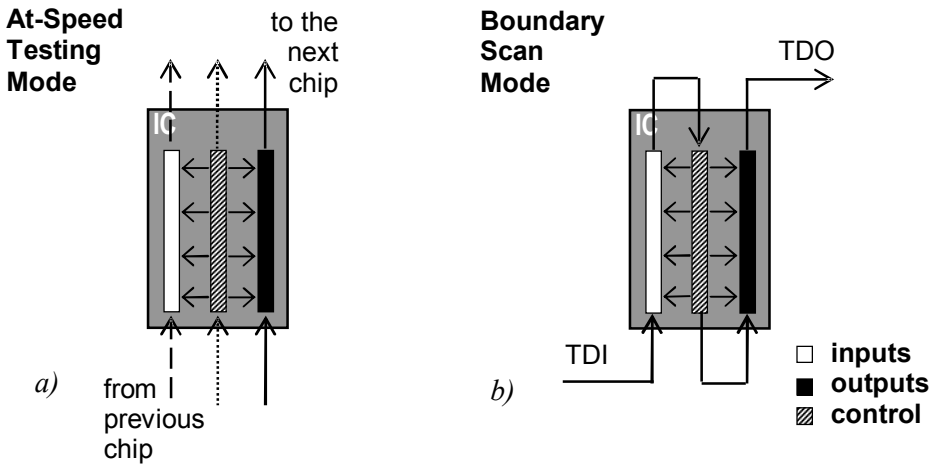



**Figure 3.7 The at-speed testing mode vs. Boundary Scan mode**

patterns are applied, the next interconnect configuration can be uploaded from the C-TPG. The updating of the initial state in the TPG is not necessary at this step. The unique deterministic patterns will be generated by TPG based on a new initial state formed inside the TPG automatically.

When all the test application process is finished, the final diagnostic signature must be shifted out. *M* clock cycles is required for that, where *M* is the length of the signature analyzer. The signature represents the string of 0-s and 1-s, where the 1-s explicitly indicate the failing nets. This signature is never aliased. It should be analyzed outside the board in order to identify failing nets. Using a simple signature compactor illustrated in Figure 3.5b, it can also be further compacted onboard into a single go/no go signal for the board as a whole.

The most important property of the proposed framework is the flexibility, which allows easy reconfiguration of the board without any change neither in the TPG configuration (including the initial state), nor the signature analysis. New chips can be inserted into the scan chain or taken away; the board level routing may be totally redesigned; the only modification to the IBIST configuration, which might be required in all these cases, is the update of the external LFSR (its length and initial state) for being consistent with the modified length of the scan chain (the TPG one).

## 3.4 Main contributions of this chapter

This chapter describes a new Boundary Scan-like BIST conception for autonomous at-speed parallel testing and diagnosis of interconnects. The key to all advantages of this framework is a combination of a novel original design of test pattern generator and response analyzer with a new deterministic test sequence called the Interleaved True/Complement Code. First of all, it brings a

never achieved before high level of universality, scalability, and configuration independence into the at-speed interconnect testing and diagnosis.

Second contribution of the conception is that it allows a free of aliasing or masking exact fault diagnosis, which does not even require a “golden signature” to compare with. The proposed framework takes into account all the important issues a practically efficient at-speed interconnect BIST framework must handle, such as: parallel application of test patterns, detection of both static and dynamic faults, bus contention problem etc. At the same time, the hardware implementation of the proposed framework is among the most efficient ones in terms of silicon area. Therefore, the results of this research possibly lead to a new improved BS testing paradigm.

# Chapter 4

# Conclusions

The research described in this thesis embraces comparatively large area in verification and testing of digital systems. It represents rather a "breadth-first" investigation touching different topics starting from design modeling and simulation and spanning to some issues of design for testability and diagnostics. The research handles both IC and PCB testing.

In each direction of the research, new appreciable results were achieved in form of problem solution, analysis, or concept development. The results were presented at international conferences and published in refereed journals. The selection of five representative papers is collected and attached to the thesis.

The whole work can be divided into two major parts:

- verification and diagnosis of integrated circuits;
- testing and diagnostics of MCMs.

The importance of the verification and testing in the scope of design and test cycle of electronic systems is considered in the first chapter of the thesis. The other two chapters concentrate on the corresponding areas of research and describe the results obtained by the author.

The main contributions in verification and diagnosis of ICs are the following.

- investigation of Structurally Synthesized Binary Decision Diagrams;
- proof of linear complexity of SSBDDs;
- proof that SSBDDs are less complex than the corresponding gate-level netlist;
- experimental study of four SSBDD-based simulation algorithms;
- development and optimization of a timing simulation algorithm on SSBDDs;
- development and optimization of an SSBDD-based design error diagnosis algorithm.

These contributions provide sufficient basis for further research in this area. For example, the further optimization of simulation algorithms went in two

directions: a) increasing the average macro size and b) SSBDD reordering. The latter research has got reflected in a recent paper [22].

The most important result of research on MCM testing is the creation of a new homogeneous framework of at-speed interconnect test that touches and improves all the major aspects of this area of testing. It is based on the following contributions:

- new test generation algorithm has been developed that covers delay faults in addition to traditional opens and shorts;
- a new corresponding test generation hardware was proposed for this algorithm;
- a fault detection and a fault diagnosis hardware was proposed;
- a system-level view to the paradigm has been described.

This work contributes simultaneously to board-level manufacturing test generation and test application, DFT, BIST, and embedded fault detection and diagnosis for both functional and at-speed testing. The proposed framework is an extension of Boundary Scan architecture, which takes into account all the important issues a practically efficient at-speed interconnect BIST framework must handle, such as: parallel application of test patterns, detection of both static and dynamic faults, bus contention problem etc.

The key to all advantages of this framework is a combination of a novel original BIST/DfT hardware architecture with a new deterministic test sequence. The advantages are:

- high level of universality, scalability, and configuration independence;
- real at-speed interconnect testing;
- free of aliasing or masking exact fault diagnosis, which does not even require a “golden signature” to compare with;
- detection of both static and dynamic faults.

At the same time, the hardware implementation of the proposed architecture is among the most efficient ones in terms of silicon area. Therefore, the results of this research possibly lead to a new improved BS testing paradigm where the system is capable not only of at-speed self-testing but also of at-speed self-diagnosis.

# References


[1] Abramovici M., Breuer M.A., and Friedman A.D, *Digital systems testing and testable design*, IEEE Press, New York, 1990, 652 p.

[2] S.B. Akers, "Binary Decision Diagrams", *IEEE Transactions on Computers*, Vol. C-27, No. 6, June 1978, pp. 509-516.

[3] A. Allan, D. Edenfeld, W.H. Joyner, A.B. Kahng, M. Rodgers, Y. Zorian, "2001 Technology Roadmap for Semiconductors," *IEEE Computer*, January 2002, pp. 42-53.

[4] R. Bryant "Graph-based algorithms for Boolean function manipulation", *IEEE Transaction on Computers*, 1986, vol. C-35, pp. 677-691.

[5] M.L. Bushnell, V.D. Agrawal, *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*, Kluwer Academic Publishers, 2000, 690 p.

[6] R. Drechsler, B. Becker, *Binary decision diagrams: Theory and implementation*, Kluwer Academic Publishers, Boston, 1998, 200 p.

[7] D. Edenfeld, A.B. Kahng, M. Rodgers, Y. Zorian, "2003 Technology Roadmap for Semiconductors," *IEEE Computer*, January 2004, pp. 47-56.

[8] R. Goering, "Productivity may stumble at 100 nm," *EE Times*, September 23, 2003,
URL: http://www.eet.com/futureofsemis/designtest/OEG20030923S0041

[9] IEEE 1149.1-1990, "IEEE Standard Test Access Port and Boundary-Scan Architecture," 1990.

[10] *The International Technology Roadmap for Semiconductors, 2001 edition*. International Sematech, Austin, Texas, 2001.

[11] *The International Technology Roadmap for Semiconductors, 2003 Update*. URL: http: //public.itrs.net/

[12] *The International Technology Roadmap for Semiconductors, 2003 Update: Test and Test Equipment*. URL: http: //public.itrs.net/

[13] A. Jutman, "On SSBDD Model Size and Complexity", in *Proc. of 4th Electronic Circuits and Systems Conference (ECS'03)*, Bratislava, Slovakia, September 11-12, 2003, pp. 17-22.

[14] 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.

[15] A. Jutman, J. Raik, R. Ubar, "SSBDDs: Advantageous Model and Efficient Algorithms for Digital Circuit Modeling, Simulation & Test," in

*Proc. of 5th Int. Workshop on Boolean Problems*, Freiberg, Germany, Sept. 19-20, 2002, pp. 157-166.

[16] S. Mourad, Y. Zorian, *Principles of Testing Electronic Systems*, John Wiley & Sons, Inc., New York, 2000, 420 p.

[17] B. Nadeau-Dostie, J. Cote, H. Hulvershorn, S. Pateras, "An Embedded Technique for At-Speed Interconnect Testing", in *Proc. of Int. Test Conf. (ITC'99)*, Atlantic City, USA, Sept. 28-30, 1999, pp.431-438.

[18] S.Park, T.Kim, "A new IEEE 1149.1 boundary scan design for the detection of delay defects," in *Proc. of DATE'2000*, Paris, France, pp. 458-462.

[19] J. Raik, *Hierarchical Test Generation for Digital Circuits Represented by Decision Diagrams*, TTU Press, Tallinn, 2001, 108 p.

[20] J. Raik, R. Ubar, "High-Level Path Activation Technique to Speed Up Sequential Circuit Test Generation," *Proc. of the European Test Workshop*, Konstanz, Germany, May 25-28, 1999, pp. 84-89.

[21] J. Raik, R. Ubar, "Sequential Circuit Test Generation Using Decision Diagram Models," *Proceedings of the DATE'99 Conference*, Munich, Germany, March 9-12, 1999, pp. 736-740.

[22] R. Ubar, T. Vassiljeva, J. Raik, A. Jutman, M. Tombak, A. Peder, "Optimization of Structurally Synthesized BDDs", in *Proc of 4th IASTED Int. Conf. on Modeling, Simulation, and Optimization*, Kauai, Hawaii, USA, August 17-19, 2004, pp. 234-240.

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

[24] R. Ubar, "Beschreibung Digitaler Einrichtungen mit Alternativen Graphen für die Fehlerdiagnose," *Nachrichtentechnik/Elektronik,* (30) 1980, H.3, pp.96-102.

[25] R. Ubar, "Test Synthesis with Alternative Graphs," *IEEE D&T of Comp.* Spring 1996, pp. 48-59.

[26] A. Wahba, and D. Borrione, "A Method for Automatic Design Error Location and Correction in Combinational Logic Circuits," *Journal of Electronic Testing: Theory and Applications*, Kluwer, Vol. 8, No. 2, April 1996.

Part II

# Research Papers

# Paper 1

A. Jutman, “On SSBDD Model Size and Complexity”, *in Proc. of 4th Electronic Circuits and Systems Conference (ECS’03)*, Bratislava, Slovakia, September 11-12, 2003, pp. 17-22.

# On SSBDD Model Size & Complexity

Artur Jutman

*Dept. of Computer Engineering, Tallinn Technical University*
*Raja 15, Tallinn 12618, Estonia. E-mail: artur@pld.ttu.ee*

**Abstract.** Binary decision diagrams (BDD) have gained a wide acceptance as a mathematical model for representation and manipulation of Boolean functions in VLSI CAD. In this paper we consider a special kind of BDDs called Structurally Synthesized BDDs (SSBDDs), which have an important characteristic property of keeping information about circuit's *structure*. Despite the fact that the SSBDD model itself is not new, we show for the first time in a formal way that this model is of *linear* complexity with respect to the number of logic gates in the circuit. We present new theoretical results in a form of equations that allow exact calculation of SSBDD model size for an arbitrary combinational circuit. Using these equations, we estimate the model complexity and compare it with the complexity of the gate-level netlist. We show in a formal way, that the SSBDD model is always smaller than the netlist still keeping the structure of the circuit.

## I. Introduction

The constant increase of integration level of modern digital devices imposes high and further growing requirements on methods and algorithms used in VLSI CAD design, verification, and testing. It is clear that the efficiency of any algorithm depends heavily on the underlying mathematical model. This fact has made the search for good models for Boolean function representation and manipulation a hot topic already for several decades. In the meantime the BDD model [1] has gained a wide acceptance and became a state-of-the-art data structure in modern VLSI CAD. First works on BDDs [2, 3, 4, 5] date back to 70s and even 50s. However, the model was not widely known until Bryant proposed a new data structure called *Reduced Ordered Binary Decision Diagrams* (ROBDDs) [6] in 1986. He showed simplicity of manipulation and proved the model canonicity what made it one of the most popular representations of Boolean functions.

During the last decade, many modifications to BDD model were proposed [7]. They were mostly aimed at fighting a memory explosion problem, which limits its usability on large designs. As a result, some similar new models appeared that can represent any Boolean circuit in linear space, e.g. Boolean Expression Diagrams [8]. There is a special kind of BDDs called *Structurally Synthesized BDDs* (SSBDDs), which is also known to be of linear complexity with respect to the number of logic gates in the circuit [9,10]. However, the latter statement has always been based on intuitive speculations and experimental observations only. In this paper, for the first time, we show in a formal way that the SSBDD model always has linear complexity with respect to the number of logic gates for any arbitrary combinational circuit. Moreover, we show that this model size is always less than the size of corresponding logic-level netlist. We also provide simple equations for exact calculation of SSBDD model size. Different equations use different parameters of combinational circuit. The simplest one is based on two basic parameters only: the total number of logic gates and the total number of signal lines (or half the number of un-collapsed stuck-at faults).

Besides linear complexity, the main characteristic feature and advantage of SSBDDs compared to other classes of BDDs is their ability of keeping structure of combinational circuits. Due to this fact the SSBDD model has been proposed as a data structure for various CAD problems like fast

*deterministic* test pattern generation [9], efficient *design error* localization [11], *logic* and *multi-valued* simulation [12] for different purposes (like hazards investigation, *delay fault* analysis, and *fault cover* analysis in dynamic testing). Efficient algorithms for *timing* and *fault* simulation were proposed in [13, 14]. SSBDD model was introduced for the first time in [3, 5] as *Structural Alternative Graphs* and generalized as *multiple-valued* decision diagrams in [9]. Some more details on SSBDDs as well as their advantages in the logic-level simulation domain are considered in [10].

We continue our paper with definitions and terminology in Section 2. Section 3 is the main section where all the mentioned equations are proved and a method of the SSBDD model complexity estimation is considered. Finally, we bring conclusions in Section 4.

## II. Definitions and terminology

In the following we will define some notions that are used throughout the paper. Since the SSBDD model is aimed at representation of logic level digital circuits rather than Boolean functions we define a *combinational circuit* as a network of basic Boolean logic gates, i.e. AND, NAND, OR, NOR, inverter, and buffer. This restriction on component base is important. We state that our technique holds for any combinational circuit composed of the specified set of 6 basic logic gates. Circuits, containing other types of gates have to be transformed into this component base before SSBDD model can be generated.

*Def.1* A connection between output of a gate and input of another gate that does not contain fanout points is called a *signal line*. If a signal connection contains a fanout point then the fanout stem and all the branches are separate signal lines. All primary inputs and primary outputs are also signal lines.

It is not difficult to see that the number of signal lines is exactly a half of the number of uncollapsed stuck-at faults (two SAFs per signal line). The circuit in Fig. 1 has 3 primary inputs, 1 primary output and 8 signal lines.

*Def.2* Any part of a combinational circuit is called a *tree-like subcircuit* if it contains no fanout points inside. A tree-like subcircuit always has a single output and one or more inputs. A single logic gate or a signal line is also a tree-like subcircuit. Any combinational circuit can be partitioned into a set of tree-like subcircuits (see Figure 2). It is not difficult to notice that if the elements of such a partition are tree-like subcircuits of a maximum size, then such a partition is unique.

*Def.3* Such tree-like subcircuits of a maximum size are called *macros*.

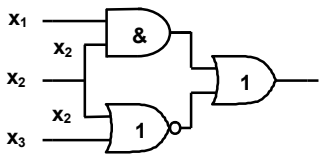



Fig 1 Combinational circuit

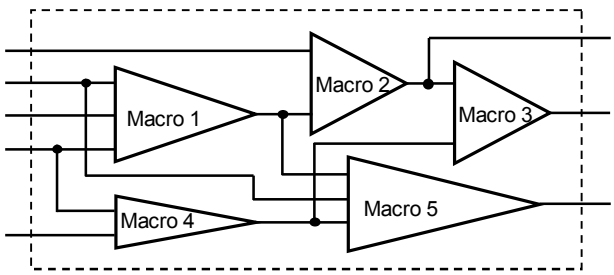



Fig 2 Partitioning of a combinational circuit

*Def.4* A BDD that represents a Boolean function $y=f(X)$ over a set of Boolean variables $X=\{x_1, x_2, \ldots, x_n\}$ is a connected directed acyclic graph $G_y=(V,E)$ with a set of nodes $V$ and a set of edges $E$ that defines mapping from $V$ to $V$. Set $V$ consists of two types of nodes: internal (non-terminal) $V^N$ and terminal $V^T$, $V=V^N \cup V^T$. A terminal node $v^T$ is labeled by a constant $\{0,1\}$ and is called *leaf*, while all non-terminal nodes $v \in M^N$ are labeled by variables $x \in X$, and have exactly two successor nodes. Let us denote the variable associated with internal node $v$ as $x(v)$, then $low(v)$ is the successor of $v$ at

the value *x(v)=0* and *high*(*v*) is the successor of *v* at the value *x(v)=1*. An example of a BDD is given in Figure 3.

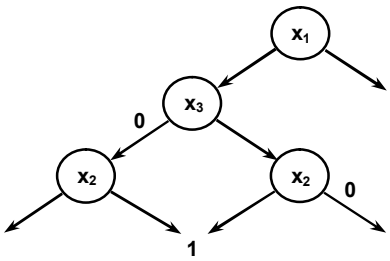



Figure 3:Binary Decision Diagram

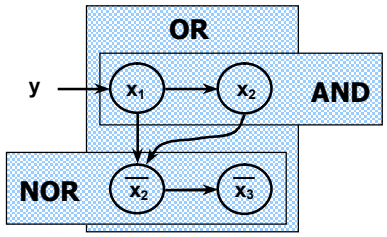



Figure 4: Illustration of the superposition principle

The majority of BDD modifications use *Shannon decomposition*: $y = \bar{x}f_{\bar{x}} + xf_x$. Here $f_x$ and $f_{\bar{x}}$ are obtained from *y=f(x)* by replacing variable *x* by value 1 and 0 correspondingly. On the contrary, SSBDDs do not rely on Shannon decomposition. They are based upon the *equivalent parenthesis form* (EPF), that is, they describe a digital circuit structurally.

*Def.5* A BDD is called *SSBDD*, if there is a *one-to-one correspondence* between non-terminal nodes of a BDD and signal paths of a combinational circuit (or a subcircuit). Non-terminal nodes of an SSBDD are labeled by indexed variables, which can be inverted or not. The SSBDD model is further defined by construction.

An SSBDD is constructed directly from a gate-level description of a combinational circuit by a graph superposition procedure. In this sense, it is equivalent to EPF generation by superposition of Boolean functions. In prior to the superposition, the circuit must be partitioned into a set of macros (Fig. 2). Each macro will be represented by its own SSBDD. That is, the whole circuit is represented not by a single SSBDD but by a set of separate SSBDDs connected by internal variables. This is the main clue why SSBDD model does not grow exponentially. It is also useful to notice that *the number of nodes in a particular SSBDD is equal to the number of inputs of the corresponding macro* and that *each node represents a whole signal path inside the macro from one of its inputs to the output*.

Figure 4 illustrates how an SSBDD for the combinational circuit from Fig. 1 is constructed. First, we set elementary BDDs: for AND and NOR gates. Then we use the superposition principle to combine them into a whole single SSBDD just as OR gate combines the two gates into a single circuit. For SSBDDs, it is agreed that the edge corresponding to the value *x(v)=0* always goes down while the edge corresponding to *x(v)=1* always goes right. Terminal nodes are not shown on this picture because it is also agreed that if *x(v)=0* and no edge goes down from the node *v*, then one gets to the terminal node labeled by 0. Similar reasoning holds in the case of missing "right" edge [9].

## III. Calculation of SSBDD model size and complexity

Let us start with combinational circuits consisting of a *single macro* only. Then, we can generalize the case for an arbitrary circuit. Finally, we are going to estimate the model complexity.

### *A. Tree-Like Subcircuits*

Let's start with macros consisting of one-input gates only. From Figure 5 it can be seen that the number of SSBDD nodes for such circuits is exactly 1 independently of the number of one-input gates. The variable in this single node will be inverted or not depending on the number of inverters in such a one-input circuit.

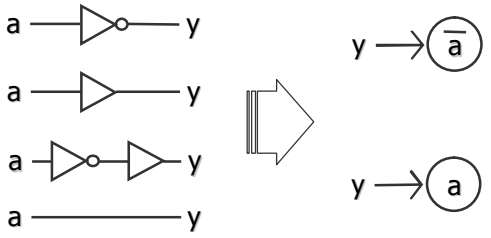



Fig 5 SSBDDs for one-input gates

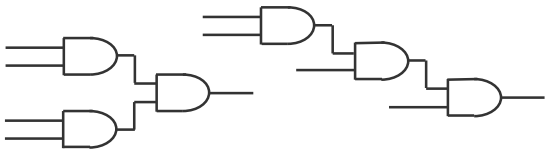


Fig 6 Macro consisting of two-input gates

The next figure shows different macros consisting of two-input gates. It is not difficult to see that the number of inputs of a tree-like circuit that consists only of two-input gates depends *on the number of gates only but not on their combination!* Indeed, if we add one more gate to any of the two circuits in Figure 6 it will add two inputs and remove one. Therefore, each additional gate adds one input in total while the very first one adds 2 inputs. Since the number of SSBDD nodes $N_{nodes}$ is equal to the number of inputs of the corresponding macro, it can be calculated using the following equation where $g$ is the number of 2-input gates in the macro: $N_{nodes} = g + 1$.

The same reasoning holds for macros consisting of 3-input gates. The only difference is that each additional gate adds 2 inputs except the very first one, which adds all 3 inputs (see Fig. 7). Then the size of such a macro can be calculated using the following equation: $N_{nodes} = 2g + 1$. If we continue increasing the number of inputs of the gates we will notice soon that the generalized equation for a macro consisting of *k-input* gates is the following: $N_{nodes} = (k-1)g + 1$.

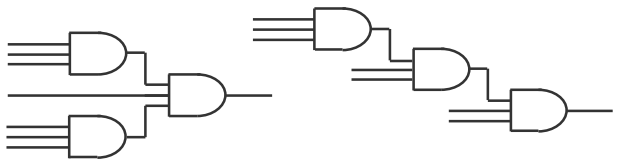


Fig 7 Macro consisting of three-input gates

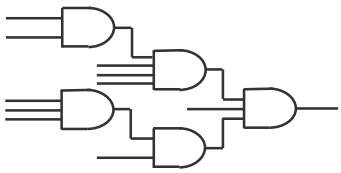


Fig 8 Macro consisting of different gates

So far we have considered quite an unusual situation where the whole circuit consists of a single macro, which in its turn consists only of gates with equal number of inputs. However, it is not difficult to generalize the equation for a macro consisting of arbitrary Boolean gates. Figure 8 clearly shows that, again, each *k-input* gate adds $k$ inputs and takes one back (except, again, the first one). Therefore, we have to take the quantities of 1-input gates $g_1$, 2-input gates $g_2$, etc. into account separately. The generalized equation for a single macro consisting of arbitrary gates is the following:

$$N_{nodes} = 0g_1 + 1g_2 + 2g_3 + 3g_4 + \ldots + (n\text{-}1)g_n + 1$$

*B. Arbitrary Combinational Circuits*

Let us again consider an arbitrary combinational circuit that may consist of an arbitrary number of macros. One must not forget here, that each primary input, which is a stem of a fanout as well as each primary output that is a fanout branch, all of them are *separate macros* having one input and one output (see Figure 2). Therefore, they will be represented by a 1-node-SSBDD.

*Theorem 1*. The total number of SSBDD nodes for an arbitrary combinational circuit is

$$N_{nodes} = 0g_1 + 1g_2 + 2g_3 + 3g_4 + \ldots + (n\text{-}1)g_n + m \quad (1)$$

where $m$ is the total number of macros and $g_i$ is the number of $i$-input gates in the circuit.

*Proof.* In order to calculate the total number of nodes in the SSBDD model for an arbitrary combinational circuit consisting of *m* macros we have to sum up SSBDD sizes for each macro:

$$N_1 = 0g_{11} + 1g_{21} + 2g_{31} + 3g_{41} + \ldots + (n\text{-}1)g_{n1} + 1$$
$$N_2 = 0g_{12} + 1g_{22} + 2g_{32} + 3g_{42} + \ldots + (n\text{-}1)g_{n2} + 1$$
$$N_3 = 0g_{13} + 1g_{23} + 2g_{33} + 3g_{43} + \ldots + (n\text{-}1)g_{n3} + 1$$
$$\vdots$$
$$N_m = 0g_{1m} + 1g_{2m} + 2g_{3m} + 3g_{4m} + \ldots + (n\text{-}1)g_{nm} + 1$$
$$N_{nodes} = 0g_1 + 1g_2 + 2g_3 + 3g_4 + \ldots + (n\text{-}1)g_n + m$$

Here, $N_j$ is the SSBDD size for j-th macro and $g_{ij}$ is the number of i-input gates in j-th macro.■

The last equation allows the exact calculation of the SSBDD model size for any arbitrary combinational circuit if some detailed information (like number of gates of each type and number of fanout points) about the corresponding logic level netlist is provided. This can be done even when the model itself is not yet generated.

*C. SSBDD Model Complexity*

Based on the last equation, let us now estimate the SSBDD model complexity. It is not difficult to notice that, in the worst case, all the gates in the circuit have the maximum possible number of inputs *n* and the number of macros is equal to the number of gates. In this case we have:

$$N_{nodes} = (n\text{-}1)g_n + g_n = n \cdot g_n - g_n + g_n = n \cdot g_n$$

That is, the worst case complexity of the SSBDD model is *O(n·G)*, where *G* is the total number of logic gates in the circuit and *n* is the number of inputs of the largest gate. In reality *n is much less than G*. Moreover, *n* can be regarded as a constant when *G* approaches to infinity. Therefore, the worst case complexity of the SSBDD model is linear with respect to the number of logic gates.

*D. Comparison of Sizes of the SSBDD Model and the Gate-Level Netlist*

The gate-level netlist size can be characterized by the number of signal lines. Let us calculate this parameter in similar way as we have calculated the SSBDD size. It not hard to see (see Fig. 8), that the number of signal lines in a single macro is as follows:

$$N_{signals} = 1g_1 + 2g_2 + 3g_3 + 4g_4 + \ldots + n \cdot g_n + 1$$

We just need to sum up all the inputs of all the gates plus one primary output of the macro. If circuit consists of an arbitrary number of macros *m*, then the equation is the following:

$$N_{signals} = 1g_1 + 2g_2 + 3g_3 + 4g_4 + \ldots + n \cdot g_n + m \quad (2)$$

If we compare equations (1) and (2) we will see that the size of SSBDD model is *always smaller* than the size of the corresponding gate-level netlist for any arbitrary combinational circuit.

*E. SSBDD Size: Simple Equation*

*Theorem 2*. The size of SSBDD model generated for an arbitrary combinational circuit is the difference between the number of signal lines and the number of gates in the circuit:

$$N_{nodes} = N_{signals} - N_{gates} \quad (3)$$

*Proof*. Let us subtract equation (1) from equation (2).

$$N_{signals} - N_{nodes} = (1g_1 + 2g_2 + 3g_3 + 4g_4 + \ldots + n \cdot g_n + m) - (0g_1 + 1g_2 + 2g_3 + 3g_4 + \ldots + (n\text{-}1)g_n + m)$$

Subtracting term by term we will have the following:

$$N_{signals} - N_{nodes} = g_1 + g_2 + g_3 + g_4 + \ldots + g_n = N_{gates}$$

Therefore: $N_{nodes} = N_{signals} - N_{gates}$ ■

This result is important because it provides very facile method of estimation of the SSBDD model size based on just two main parameters of combinational circuits: the number of gates and the number of signal lines. Moreover, as it has been discussed above, the latter parameter is exactly a half of the total number of un-collapsed SAFs in the circuit, which is usually known. It also follows from the last equation that the SSBDD model is smaller than the gate-level netlist exactly by the number of gates.

## IV. Conclusions

In this paper we showed in a formal way that a special type of BDDs called *Structurally Synthesized BDDs* (SSBDDs) always have linear complexity with respect to the number of logic gates for an arbitrary combinational circuit. Moreover, we showed that the SSBDD model complexity is even smaller than the complexity of corresponding logic-level netlist exactly by the number of logic gates.

We provide several equations for exact calculation of SSBDD model size. The simplest one is based on two basic parameters of combinational circuits: the number of logic gates and the number of signal lines (or half the number of un-collapsed stuck-at faults). It also represents a strict relationship between mentioned parameters and the number of nodes in SSBDD model.

## References


[1] R. Drechsler, B. Becker. *Binary decision diagrams: Theory and implementation*, Kluwer Academic Publishers, Boston, 1998, 200 p.

[2] C.Y. Lee. Representation of switching circuits by binary decision diagrams. Bell System Technical Jour., 38:985-999, 1959

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

[4] S.B. Akers "Binary decision diagrams" IEEE Trans. On Comp., 27:509-516, 1978

[5] R. Ubar, "Beschreibung Digitaler Einrichtungen mit Alternativen Graphen für die Fehlerdiagnose," Nachrichtentechnik/Elektronik, (30) 1980, H.3, pp.96-102.

[6] R. Bryant "Graph-based algorithms for Boolean function manipulation", IEEE Transaction on Computers, 1986, vol. C-35, pp. 677-691.

[7] A.Narayan, "Recent Advances in BDD Based Representations for Boolean Functions: A Survey", in Proc. 12th International Conference on VLSI Design, Goa, India, 1999, pp. 408-413.

[8] H.R. Andersen, H. Hulgaard "Boolean Expression Diagrams," in *Proc. 12th IEEE Symposium on Logic in Computer Science*, Warsaw, Poland, June 29-July 2, 1997, pp. 88-98.

[9] R. Ubar, "Test Synthesis with Alternative Graphs," IEEE D&T of Comp. Spring 1996, pp. 48-59.

[10] A. Jutman, J. Raik, R. Ubar, "SSBDDs: Advantageous Model and Efficient Algorithms for Digital Circuit Modeling, Simulation & Test," in Proc. of 5th Int. Workshop on Boolean Problems, Freiberg, Germany, Sept. 19-20, 2002, pp. 157-166.

[11] 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.

[12] R. Ubar, "Multi-Valued Simulation of Digital Circuits with Structurally Synthesized Binary Decision Diagrams," OPA, Gordon and Breach Publishers, Multiple Valued Logic, 1998, Vol.4, pp. 141-157.

[13] R. Ubar, A. Jutman, Z. Peng, "Timing Simulation of Digital Circuits with Binary Decision Diagrams", in Proc. of DATE 2001 Conference, München, Germany, 2001, pp. 460-466.

[14] R. Ubar, "Parallel Critical Path Tracing Fault Simulation," in Proc. of the 39. Int. Wiss. Kolloquium, Ilmenau, Germany, 1994, Band 1, pp. 399-404.

# Paper 2

R. Ubar, A. Jutman, Z. Peng, “Timing Simulation of Digital Circuits with Binary Decision Diagrams,” *Proc. of DATE 2001 Conference*, Munich, Germany, March 13-16, 2001, pp. 460-466.

# Timing Simulation of Digital Circuits with Binary Decision Diagrams

R. Ubar, A. Jutman
Tallinn Technical University, Estonia
{raiub, artur}@pld.ttu.ee

Z. Peng
Linköping University, Sweden
zpe@ida.liu.se

## Abstract

*Meeting timing requirements is an important constraint imposed on highly integrated circuits, and the verification of timing of a circuit before manufacturing is one of the critical tasks to be solved by CAD tools. In this paper, a new approach and the implementation of several algorithms to speed up gate-level timing simulation are proposed where, instead of gate delays, path delays for tree-like subcircuits (macros) are used. Therefore timing waveforms are calculated not for all internal nodes of the gate-level circuit but only for outputs of macros. The macros are represented by structurally synthesized binary decision diagrams (SSBDD) which enable a fast computation of delays for macros. The new approach to speed up the timing simulation is supported by encouraging experimental results.*

## 1. Introduction

The transition from the traditional Application-Specific Integrated Circuit (ASIC) to System-on-Chip has lead to new challenges in design methods, manufacturing, verification, and test. Timing simulation is a widely used method to verify the timing behavior of a digital design. In a synchronous digital system the timing property that is needed to be verified is that for each input vector transition the combinational logic settles to a stable state within a given clock period. One approach to ensuring this is to use delay simulation.

There are different methods to model the delays in digital circuits, including the *zero delay, unit-delay and multiple-delay* models [1]. While the zero-delay models can be used to analyze combinational circuits without memories, and unit-delay models can be used to verify the logical behavior of synchronous sequential circuits, they are inadequate for analyzing the timing behavior of digital circuits. For the timing behavior, a multiple-delay model should be used. In such a model, each circuit element is assigned a delay which is an integer multiple of a time unit. Usually separate rise and fall delays are specified. If the gate delays are not a function of the direction of the output change, we can use a *transition-independent delay* model. In the following we use a *nominal delay* model [2] with the assumption that the gate delays are known.

In the classical gate-level delay simulation [2] all the gates should be evaluated once per cycle which leads to a great amount of simulation with circuits of high complexity. In this paper, instead of gate-level simulation, we use macro-level simulation, where macros represent tree-like subcircuits (i.e. subcircuits with no reconvergent fanouts). The paths are considered only inside the macros. For this reason, we avoid the exponential explosion of the number of paths processed. When representing complex gates by macros, the number of macros is equal to the number of tree-like subcircuits in the complex gate. For example, a one-bit multiplexer is represented by a single macro.

To each path we assign a delay (or two delays in the case of transition dependent delay model). For simplicity, in this paper, without loosing the generality, we consider the one-delay case for each path. For example, assume that the subcircuit in Fig.1 is represented by a macro. This macro is characterized by 6 paths and 6 delays calculated on the basis of gate delays.

A novel method for delay simulation is developed based on Boolean derivatives and structurally synthesized binary decision diagrams (SSBDD). SSBDDs were introduced the first time in [3,4] as structural alternative graphs, and generalized for the multiple-valued decision diagrams in [5]. In [6] SSBDDs were suggested for multivalued simulation of digital circuits for different purposes like hazards investigation [7], delay fault analysis [8], and fault cover analysis in dynamic testing [9]. When using SSBDDs for representing macros, the complexity of the model will be substantially reduced compared to the gate-level approaches.

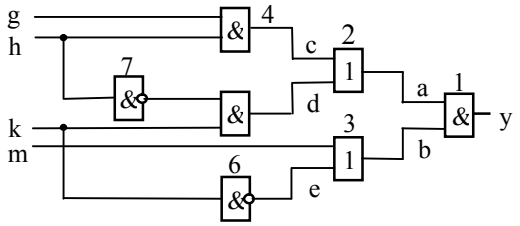



**Figure 1: Digital subcircuit**

This paper is organized as follows. Section 2 describes equivalent parenthesis forms (EPF) for a given digital circuit. In Section 3 the main considerations about timing simulation based on Boolean derivatives are given, and in Section 4 an efficient implementation of this approach on SSBDDs is described. Our algorithms are explained in detail in section 5. In Section 6 experimental results are given and finally Section 7 brings concluding remarks.

## 2. Equivalent parenthesis forms

Let us represent a digital circuit by an *equivalent parenthesis form* (EPF) synthesized by a superposition procedure directly from the gate-level description of a circuit. For synthesizing the EPF of a given circuit, numbers are first assigned to the gates and letters to the nets. Then, starting at an output and working back toward the primary inputs, EPF replaces individual literals by products of literals or sums of literals.

When an AND gate is encountered during backtracing, a product term is created in which the literals are the names of nets connected to the inputs of the AND gate. Encountering an OR gate causes a sum of literals to be formed, while encountering an inverter causes a literal to be complemented.

As an example, the procedure is illustrated by transforming the circuit in Fig.1 to its EPF:

$y = a_1 b_1 = (c_{12} + d_{12})(m_{13} + e_{13}) = (g_{124} h_{124} + f_{125} k_{125}) \wedge$
$\wedge (m_{13} + \neg k_{136}) = (g_{124} h_{124} + \neg h_{1257} k_{125})(m_{13} + \neg k_{136})$.

When creating an equation by the superposition procedure described above, the identity of every signal path from the inputs to the outputs of the given circuit will be retained. Each literal in an EPF consists of a subscripted input variable or its complement, which identifies a path from the variable to the output. From the manner in which the EPF is constructed, it can be seen that there will be at least one subscripted literal for every path from each input variable to the output. It is also easy to see that the complemented literals correspond to paths, which contain an odd number of inversions.

## 3. Equivalent parenthesis forms and timing simulation

Let us have an EPF $y = P(x_1, x_2, \ldots x_i, \ldots x_n)$ where $x_i \in X$ are literals (inverted or not), which describe the behavior of a digital circuit. Denote by $L(x_i) = (g_{i1}, g_{i2}, \ldots, g_{in})$ the signal path through the gates $g_{i1}, g_{i2}, \ldots, g_{in}$ from the output $y$ up to the input $x_i$. Denote the delay of the gate $g_{ij}$ by $d(g_{ij})$. For simplicity, here we use the same delay for all the gate inputs for both raise and fall transitions. However, this does not affect the generality of the approach.

Let us call $\partial y/\partial x_i$ as *partial Boolean derivative*. The theory of Boolean differential calculus tells that if $\partial y/\partial x_i = 1$, then a transition of the signal at input $x_i$ leads to a transition of the signal at output $y$. To take into consideration the timing aspect, we introduce a function $\partial y(t_y)/\partial x_I(t_x)$, where $\partial y(t_y)/\partial x_i(t_x) = 1$ means that the transition of $x_i$ at moment $t_x$ causes the transition of $y$ at moment $t_y$.

*Theorem 1:* Given a single transition at moment $t_x$ on the input $x_i$ with a single output of a circuit represented by EPF $y = P(x_1, x_2, \ldots x_i, \ldots x_n)$ with the path $L(x_i) = (g_{i1}, g_{i2}, \ldots, g_{in})$ from $x_i$ to $y$, the transition propagates up to $y$ with the delay

$$d(x_i \rightarrow y) = d(g_{i1}) + d(g_{i2}) + \ldots + d(g_{in}). \qquad (1)$$

iff $\partial y(t_y)/\partial x_i(t_x) = 1$ where $t_y = t_x + d(x_i \rightarrow y)$.

*Proof:* Along the definition of partial Boolean derivatives, from $\partial y/\partial x_i = 1$ (here and afterwards $t_y$ and $t_x$ for $y$ and $x$ are dropped for better readability) it follows that the value of y is depending on the value of $x_i$, hence the transition at $x_i$ propagates up to $y$. Since the path $L(x_i) = (g_{i1}, g_{i2}, \ldots, g_{in})$ along which the transition propagates is not a branch, and it also has no fanouts, no other reconverging paths can exist along which the same transition at $x_i$ could influence the value of $y$. Hence, the delay of the transition at $y$ may be produced only by the sum of the delays of the gates along the path $L(x_i)$, and the relationship (1) is valid. ■

In the general case, if transitions occur on several inputs, or a transition propagates along several reconverging paths, then the derivative $\partial y/\partial x_i$ may depend on the influence of other transitions which may result in a glitch at $y$. In other words, the value of the function $\partial y/\partial x_i = f(x_1, x_2, \ldots x_{i-1}, \ldots x_{i+1}, \ldots x_n)$ depends in this case on the literals where values are nondetermined (unknown), and the calculation of $\partial y/\partial x_i$ is impossible.

Let us introduce now the set $S_5 = \{0, 1, \varepsilon, h, U\}$ for 5-valued simulation, where $\varepsilon$ *(h)* represents a waveform having a step-up transition from 0 to a final value of 1 (step-down transition from 1 to a final value of 0), and $U$ represents undetermined (unknown) or don't care waveform. These values $\varepsilon$, $h$, $U$ are called *dynamic values*.

In the following table we give also the algebra introduced for the dynamic values $\{\varepsilon, h, U\}$ in [6]:

| $\vee$ | $\varepsilon$ | $h$ | $U$ | $\wedge$ | $\varepsilon$ | $h$ | $U$ |
|---|---|---|---|---|---|---|---|
| $\varepsilon$ | $\varepsilon$ | $U$ | $U$ | $\varepsilon$ | $\varepsilon$ | $U$ | $U$ |
| $h$ | $U$ | $h$ | $U$ | $h$ | $U$ | $h$ | $U$ |
| $U$ | $U$ | $U$ | $U$ | $U$ | $U$ | $U$ | $U$ |

**Table 1: Calculation of dynamic values**

Let us have a network with EPF $y = f(x_1, x_2, \ldots x_i, \ldots x_n)$ and a multi-valued pattern $x^t = (x^t_1, x^t_2, \ldots x^t_i, \ldots x^t_n)$ at time $t_x$ where $x^t_i \in S_5$. Denote a subset of literals with dynamic values at $t_x$ by $x_D = \{x_i \mid x^t_i \in \{\varepsilon, h, U\}\}$.

*Definition 1:* We say $max\{\partial y/\partial x_i\} = 1$ iff there is at least one combination of values 0 or 1 for nonspecified $x$'s which produce $\partial y/\partial x_i = 1$. Otherwise, $max\{\partial y/\partial x_i\} = 0$.

*Lemma 1:* The value of EPF $y = P(x_1, x_2, \ldots x_i, \ldots x_n)$ for a given network in the multivalued alphabet $S_5$ is:

$$y = \bigwedge_{x_i \in x_D \cap \{x_i \mid max\{\partial y/\partial x_i\}=1\}} x_i = \bigvee_{x_i \in x_D \cap \{x_i \mid max\{\partial y/\partial x_i\}=1\}} x_i \quad (2)$$

iff $x_D \cap \{x_i \mid max\{\partial y/\partial x_i\} = 1\} \neq \varnothing$.

*Proof:* If $max\{\partial y/\partial x_i\} = 1$ is valid for a single $x_i \in x_D$ then according to the definition of Boolean derivatives, $y = x_i$. In this case the same value of $x_i$ occurs on the output (or inverted value if $x_i$ is inverted). Suppose now that there are more than one literals $x_i \in x_D$ satisfying the condition $max\{\partial y/\partial x_i\} = 1$. In other words, it means there are more than one converging paths in the network which propagate transitions towards the output. If two paths are converging, either AND or OR of multiple values from $\{\varepsilon, h, U\}$ is possible. From the equivalence of operations AND and OR on the set $\{\varepsilon, h, U\}$, it follows that the value of $y$ can be calculated as the function of AND (or OR) of values $x_i \in x_D \cap \{x_i \mid max\{\partial y/\partial x_i\} = 1\}$. ■

Consider, for example, a transition pattern $g = k = m = 1$, $h = \varepsilon$ at the input of the circuit in Fig 1. By calculating Boolean derivatives, we find: $\partial y/\partial h_{124} = h_{1257}$, and $\partial y/\partial h_{1257} = \neg h_{124}$. Since $h_{124}$ and $h_{1257}$ have dynamic values $h_{124} = h_{1257} = h = \varepsilon$, the calculation of Boolean derivative is impossible. On the other hand, since $max\{\partial y/\partial h_{124}\} = max\{\partial y/\partial h_{1257}\} = 1$, and since $x_D \cap \{x_i \mid max\{\partial y/\partial x_i\} = 1\} = \{h_{124}, \neg h_{1257}\}$, we have $y = h_{124} \wedge \neg h_{1257} = \varepsilon \wedge \neg \varepsilon = U$. The value $U$ on the output of the subcircuit in Fig.1 means the possibility of a glitch at the given transition pattern.

*Theorem 2:* Given $|x_D| > 1$ at input pattern $x^t = (x^t_1, x^t_2, \ldots x^t_i, \ldots x^t_n)$ where $x^t_i \in S_5$, and a subset $x^*_D \in x_D$ where

$$\forall x_i \in x^*_D : (max\{\partial y/\partial x_i\} = 1) \;\&\; (d(x_i \rightarrow y) = \Delta_i), \quad (3)$$

there appears a transition on the output of a circuit $y = P(x_1, x_2, \ldots x_i, \ldots x_n)$ with the value

$$y = \bigwedge_{x_i \in x^*_D} x_i \quad (4)$$

at time $t_x + \Delta_i$ where $\Delta_i$ is calculated by formula (1).

*Proof:* Suppose there exist at least two inputs $x^t_i, x^t_j \in x^*_D$ with corresponding paths $L(x_i) = (g_{i1}, g_{i2}, \ldots, g_{in})$ and $L(x_j) = (g_{j1}, g_{j2}, \ldots, g_{jm})$ through the circuit. Suppose they have a joint path $L(g_{i,k}) = (g_{i1}, g_{i2}, \ldots, g_{i,k-1})$ starting from the output of a gate $g_{ik} \equiv g_{jk}$, $k>0$, with the transition delay $\tau = d(g_{i1}) + d(g_{i2}) + \ldots + d(g_{i,k-1})$. From (3) it follows that the transitions evoked at the inputs $x^t_i, x^t_j$ reach the inputs of the gate $g_{ik}$ at the same moment $t_{k+1} = t_x + (\Delta_i - \tau - d(g_{ik}))$. On the other hand, from the condition $x^t_i, x^t_j \in x_D \cap \{x_i \mid max\{\partial y/\partial x_i\} = 1\}$ and Lemma 1, it follows that the value of the signal at time $t_k = t_x + \Delta_i - \tau$ on the output of the gate $g_{ik}$ belongs to the set $\{\varepsilon, h, U\}$, which means a transition (where $U$ is a possible glitch). Since the path $L(g_{i,k})$ is also activated due to (3), the transition propagates to the output and shows itself at time $t_{k+1} + d(g_{ik}) + \tau = t_x + (\Delta_i - \tau - d(g_{ik})) + d(g_{ik}) + \tau = t_x + \Delta_i$. ■

*Corollary:* From Theorems 1 and 2 the following algorithm can be derived for timing simulation based on calculating Boolean derivatives of equivalent parenthesis forms.

*Algoritm 1.*

1. Calculate $\partial y/\partial x_i$ for $x_i \in x_D$ for the given transition $x^t$.
2. Take the lowest value of $\Delta_i = d(x_i \rightarrow y)$.
   If $\partial y/\partial x_i = 1$ fix the new value of $y$ for time $t_x + \Delta_i$.
   Use formula (2) to check if a glitch is present.
   Remove $x_i$ from $x_D$.
3. If $x_D = \varnothing$, stop, else repeat step 2.

## 4. Timing simulation on SSBDDs

A structurally synthesized BDD $G_y=(M,\Gamma,X)$ with a set of nodes $M$ and a mapping $\Gamma$ from $M$ to $M$ is a BDD which represents an equivalent parenthesis form $y = P(x)$ of a gate-level network. The set of nodes consists of a subset of nonterminal nodes $M^N$ and of a subset of terminal nodes $M^T$; $M = M^N \cup M^T$. There are one initial node $m_0 \in M^N$ and two terminal nodes $m^{T,e} \in M^T$, $e \in \{0,1\}$, in $M$. A one-to-one correspondence exists between nonterminal nodes $m \in M^N$ and the literals $x_i \in X$. The nodes $m \in M^N$ are labeled by subscripted input variables (or the inverted variables) which identify a path from the input to the output of the network. The terminal nodes $m^{T,e} \in M^T$ are labeled by constants $e \in \{0,1\}$. The literal $x_i \in X$ which is associated with the node $m$ is denoted by $x(m)$. The mapping $\Gamma$ defines the set of edges between the nodes of $M$ whereas $\Gamma(m) \subset M$ is a set of successors of $m$, and $m^e \in \Gamma(m)$ is the successor of $m$ for the value $x(m) = e$. A pattern $x^t$ defines a set of activated edges in $G_y$. The edge between $m$ and $m^e$ is activated when $x(m) = e$ in the pattern $x^t$. Activated edges which connect nodes $m_i$ and $m_j$ make up an activated path $l(m_i, m_j)$. The path $l(m_i, m_j)$ consists of nodes $M(m_i, m_j) \subseteq M$. An activated path $l(m_0, m^{T,e})$ is called a full activated path.

*Definition 2.* A SSBDD $G_y = (M,\Gamma,X)$ represents an equivalent parenthesis form $y = P(X)$ of a gate-level network, iff for each pattern $x^t$ a full path $l(m_0, m^{T,e})$ in $G_y$ will be activated where $y = e$.

Two-valued test pattern simulation on SSBDDs is equivalent to path tracing procedure on graphs according to the values of variables at a given test pattern. At a given pattern $x^t$, in a SSBDD $G_y$, a full path $l(m_0,m^T)$ will be activated which determines the value of $y = x(m^T)$. The simulation procedure will consist of tracing the path $l(m_0,m^T)$ and finding the value of $x(m^T)$ at the terminal node $m^T$.

For multi-valued simulation, a procedure based on calculation of Boolean derivatives on SBDDs will be now described. Denote $l(m_i,m_j) = 1$, if there exists an activated path between the nodes $m_i$ and $m_j$ at the given pattern $x^t$, otherwise, $l(m_i,m_j) = 0$.

*Theorem 3:* Given $y = P(x)$ and $x_i \in X$, the condition $dy/dx_i = 1$ for SSBDD $G_y = (M,\Gamma,X)$ where $x(m) \equiv x_i$ is equivalent to the following equation:

$$l(m_0,m) \wedge l(m^1, m^{T,1}) \wedge l(m^0, m^{T,0}) = 1. \quad (5)$$

The proof of the theorem can be found in [6].

Note, Theorem 3 can be used for calculating Boolean derivatives $dy/dx_i$ only in the case where pattern $x^t$ is two-valued, because only in this case all the paths $l(m_i,m_j)$ are activated uniquely. In the general case, when $x^t$ is a multi-valued pattern, to check the existence of a glitch, we have to generalize equation (5). The generalized case based on maximums of Boolean derivatives is considered in [6].

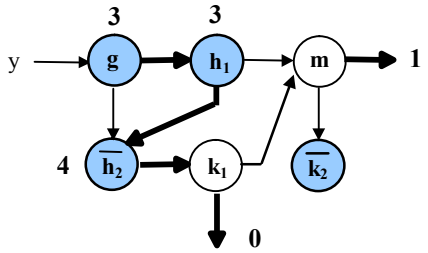



**Figure 2: SSBDD for the circuit in Figure 1**

Using SSBDDs it is possible to considerably speed up the calculations described in Algorithm 1 because it is not needed to trace all paths in equation (5) for each $x_i \in x_D$.

| Node | Path | Delay | Pattern |
|---|---|---|---|
| g | g, 4, 2, 1, y | 3 | $h$ (10) |
| $h_1$ | h, 4, 2, 1, y | 3 | $\varepsilon$ (01) |
| $\neg h_2$ | h, 7, 5, 2, 1, y | 4 | $h$ (10) |
| $k_1$ | k, 5, 2, 1, y | 3 | 0 |
| m | h, 3, 1, y | 2 | 1 |
| $\neg k_2$ | k, 6, 3, 1, y | 3 | $\varepsilon$ (01) |

**Table 2: Signal paths and delays of the example**

*Example:* An example of SSBDD for the circuit in Fig.1 is represented in Fig.2. The nodes of the graph, the corresponding paths in the circuit, and the path delays calculated by equation (1) are depicted in Table 2 (here we assume that all the gates have a unit delay).

Consider a transition pattern given in Table 2. The bold arrows (in Fig. 2) mark the activated path in the graph before the transition. The shaded nodes are those involved in the transition, i.e. where the direction of the activated path changes. For the nodes g and $h_1$ we have $max\{\partial y/\partial g\} = max\{\partial y/\partial h_1\} = 1$ [5]. Using the formula (2) we find that $g \wedge h_1 = h \wedge \varepsilon = U$ which means that at time t = 3 we may have a glitch on the output of the circuit.

## 5. The Timing Simulation Algorithms

Using the SSBDD model gives us the possibility to minimize the number of macro inputs to be processed as well as the possibility to use some SSBDD features in order to increase the timing simulation efficiency.

In this section we describe several implementations of the Algorithm 1 on the SSBDD model. First, the general algorithm is given, then we describe the single and double stack based approaches.

Given a set of multi-valued input patterns $x^t$ at the input of a macro SSBDD $G_y=(M,\Gamma,X)$, and a set of delays $\Delta=\{d^e(m_i)|\ m_i\in M,\ e\in\{\varepsilon,h\}\}$. Certain values for both raise $d^\varepsilon(m_i)$ and fall $d^h(m_i)$ delays are specified for each node. We denote a variable in the node $m_i$ as $x(m_i)$. The output of the algorithm is a single waveform for the output of each macro. The waveforms show all the transitions taking place there.

The general idea is as follows. Let the current moment of time be $t_x$ and the current pattern applied $x^t$. We are traversing the activated (before the transition) path $l(m_0,m^T)$ in the graph from the initial node $m_0$ to one of the terminal nodes $m^T$ and checking if $x(m_i)\in x_D$ in order to find the node with transition that has the minimal delay $d_{min}$. The transition in this node is the first transition that may influence the macro's output. It will happen at the moment $t_y=t_x+d_{min}$ iff $max\{\partial y/\partial(x(m_i))\}=1$. When the node is found, the current time $t_x$ is changed to $t_x+d_{min}$.

Our task now is to find the next $d_{min}$. We go back to the initial node and traverse the path from the beginning taking into account that one of the values has already been changed. However, as we are probably traversing a new path, we can find a node with delay that is smaller or equal to the previous $d_{min}$. This means that the transition in that node has also taken place and it is not interesting anymore. In general, we are not interested in all delays $d^e(m_i)<t_x$. Suppose, we are in node $m_i$, $x(m_i)\in x_D$, somewhere in the middle of the path. The delay here is $d^e(m_i)$ and somewhere before (along the path $l(m_0, m_i)$) we have already found the next minimum delay $d_{min}$. Then we will update the $d_{min}$ with $d^e(m_i)$ iff $t_x<d^e(m_i)<d_{min}$.

After we have reached a terminal node again, we are checking if it is a different one from the previously reached terminal node. If it is, we put the new transition to the output waveform labeling it with the current moment of time. We continue the graph traversal procedure until no $d_{min}>t_x$ is found. This means that all the transitions (which have influence to the macro output) in the macro have already taken place and the next vector should be taken. When all the vectors have been simulated for the given macro, a new macro is taken. The whole process stops when the whole circuit has been finished.

The above was the description of the general SSBDD-based timing simulation algorithm, which uses no stack. Note that in some cases we do not need to check all the nodes in the graph because those nodes will never lie on an activated path. To make the procedure even more efficient, we use a stack to store every encountered node along the path, with the delay which was taken as $d_{min}$. Using the stack we have no need to begin path traversal from the initial node every time. We can return to the last

node $m_s$ taken from the stack and take the $d^e(m_s)$ as the next $d_{min}$ and update it further as we start moving forward.

In the following we give the description of a single stack-based algorithm step by step.

*Algorithm 2.*

1. Initialization: $t=0$, $d_{min}=0$, $i=0$, $ptr=0$, $stack(ptr).node=0$, $stack(ptr).time=0$, macro output is undefined;
2. If $t<d^e(m_i)$ go to **3**. Otherwise take $i$ as the index of $m^0$ if $x(m_i)=\{h,0\}$ or as the index of $m^1$ if $x(m_i)=\{\varepsilon,1\}$, go to **7**;
3. If $x(m_i)\in x_D$ go to **4**. Otherwise take $i$ as the index of $m^0$ if $x(m_i)=\{\varepsilon,0\}$ or as the index of $m^1$ if $x(m_i)=\{h,1\}$, go to **7**;
4. If $ptr=0$ or $d^e(m_i)<stack(ptr).time$ go to **5**. If not, go to **6**;
5. $ptr=ptr+1$, $stack(ptr).time= d^e(m_i)$, $stack(ptr).node=i$;
6. Take $i$ as the index of $m^0$ if $x(m_i)=\varepsilon$ or as the index of $m^1$ if $x(m_i)=h$;
7. If $m_i$ is not one of the terminal nodes go to **2**. If not, go to **8**;
8. If macro output is different from the value of the terminal node we have come to, update macro output with the new transition and label it with time $t$;
9. If $ptr=0$ stop, otherwise go to **10**;
10. $t=stack(ptr).time$, $i=stack(ptr).node$, $ptr=ptr-1$, $d_{min}= stack(ptr).time$, go to **2**;

*Example:* In Fig. 3 an example to illustrate the algorithm is given for the SSBDD in Fig. 2. The input pattern and the delays are the same as in Table 2. We start from the node $g$ and go to the node $h_1$. As the stack was empty and $g$ had a transition at the given moment of time we put $g$ and its delay into the stack. The node $h_1$ has a transition but the delay in it is not smaller than that in $g$. So we continue moving forward without updating the stack. The nodes $h_2$ and $k_1$ have no transitions this time. We just pass them by. Finally we reach the terminal node $m^{T,0}$. So the initial value at the output $y$ will be 0.

We get back to the node taken from the stack (it is $g$) and go to another direction (the value in $g$ has been changed). The current moment of time is 3 now. The node $h_2$ has a transition and the transition time is greater than the current moment of time. As the stack is empty again, we put $h_2$ and the delay into the stack and move forward. Finally again we reach the same terminal node. So, the output is stable. Again we get back to the node $h_2$ taken from the stack and reach the same terminal node, which means no change of the value on the output. Since the stack is empty now, the calculation terminates.

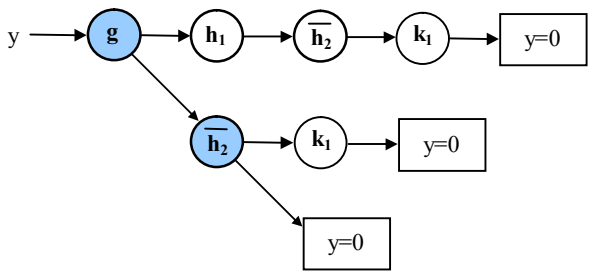



**Figure 3: Single stack based timing simulation for the SSBDD in Figure 2**

Note that despite node $k_2$ has a transition, we did not examine this macro input at all. That is, we have to check all of the macro inputs and calculate derivatives for all $x(m_i)\in x_D$ only in the worst case.

In Algorithm 2 and the example above we use a stack to return each time not to the very beginning of the graph but exactly to the node with the next transition. However, not every transition along the activated path can influence the output of the macro. In the following we give an idea how to improve the Algorithm 2 by using this feature.

Given an input pattern that activates a full path $l(m_0, m^{T,e})$, which consists of the nodes $M(m_0, m^{T,e})$. We designate $M^e(m_0, m^{T,e}) = \{ m \mid m \in M^N, m \in M(m_0, m^{T,e}), x(m) = e, e\in\{0, 1\} \}$ the set of all nonterminal nodes along the path which hold the value $e$. Similarly, the set of all the nodes along the path which hold the value $\neg e$ are designated $M^{\neg e}(m_0, m^{T,e}) = \{ m \mid m \in M^N, m \in M(m_0, m^{T,e}), x(m) = \neg e, e\in\{0, 1\} \}$. In other words we divide all the nodes along the activated path into two subsets. First one $M^0(m_0, m^{T,e})$ contains all the nodes which hold the current value 0 and another one $M^1(m_0, m^{T,e})$ contains all the nodes which hold the value 1. Terminal node $m^{T,e}$ does not belong to any of the two subsets. If the currently reached terminal node is $m^{T,0}$ then it is known that transitions in all the nodes $m \in M^1(m_0, m^{T,0})$ do not affect the output value (taking a new path, we will still reach the node $m^{T,0}$), and vise versa, for the node $m^{T,1}$ no transitions in nodes $m \in M^0(m_0, m^{T,1})$ can affect the output.

The above statement shows clearly that, standing in the terminal node $m^{T,e}$, we should consider only the nodes $m \in M^e(m_0, m^{T,e})$ as the potential sources of influence on the macro output. Therefore, we introduce a minor change to the Algorithm 2 using two different stacks for the nodes of $M^0(m_0, m^{T,e})$ and $M^1(m_0, m^{T,e})$. Standing each time at the terminal node we check only the dedicated stack for the next transition to simulate it. If there are some transitions in another stack, they will be left not simulated because they cannot affect the macro output. That is, we have to simulate all the nodes with transitions on the current active path only in the worst case.

However, certain operations and comparison of data between two stacks should be added to make the algorithm work well. This generates some overhead and in the worst case the double-stack-based algorithm may work slower than the single-stack-based one. This gave us an idea to try to use the two-stack approach only for finding the next moment of time but starting the traversal procedure from the initial node $m_0$. This helps us to avoid considerably time-consuming procedure of stack update. This means that we can win the time needed for stack update, but we lose the time needed for the path traversal from the beginning.

For different circuits all the three algorithms should give different results. It is logical to suppose that the sim-

pler algorithms should work faster for smaller macros but for bigger ones sophisticated stack-based algorithms can give better results. In the next section we will illustrate this statement by experimental data but now let us give an example to illustrate the two-stacks-based algorithm.

*Example:* Consider the same SSBDD, the same input pattern, and the same delays as in the last example. Similarly, we begin with the node g and traverse the activated path until the end but, differently from the single stack case, we store the node $g$ in one stack and the node $h_1$ in another. We do not put nodes $h_2$ and $k_1$ into the stacks similarly to the previous example. Finally we reach the terminal node $m^{T,0}$. So, the initial value at the output $y$ is 0.

In this case, only nodes with transitions 0 to 1 can affect the macro output. So, we have to check the corresponding stack. We find the node $h_1$ in this stack and go back to this node. However, at this point we cannot continue the graph traversal before we have checked another stack to see if it has a node which stands closer to the initial node and has a delay smaller than or equal to the delay in the node $h_1$. If there is such a node in another stack we have to go further to this node. This is the point where the overhead of processing of stacks is added.

In another stack we find node $g$ with the delay equal to the delay in $h_1$, so we move further to node $g$ and start the traversal of newly activated path from that point. Both stacks are empty again. As the node $h_2$ has a transition and the delay is greater than the current moment of time, we put it into one of the stacks. Node $k_1$ does not have a transition, so we pass it by and come to the same terminal node (again $y=0$).

We look at the stack, which corresponds to the situation where $y=0$ and find it to be empty. This means that the simulation is over. In Fig. 4 an illustration of the algorithm's work is given. Compared to the single-stack-based algorithm (Fig. 3) it has one step less.

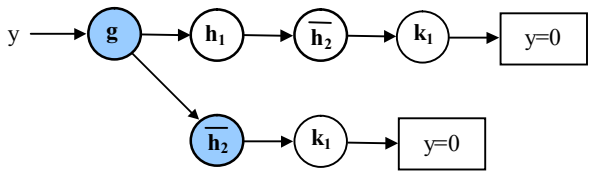



**Figure: 4. Double-stack based timing simulation for the SSBDD in Figure 2**

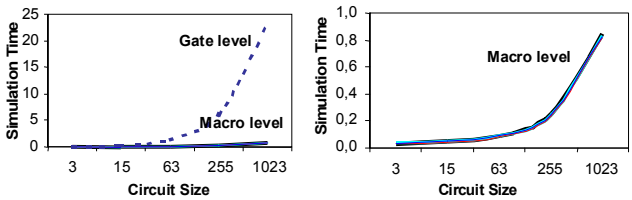



**Figure: 5, 6. Comparison of simulation times of different algorithms (single-bit transition mode)**

## 6. Experimental data

Experiments were carried out using two different types of benchmarks. The ISCAS'85 circuits were chosen since they are widely adopted benchmarks. However, the efficiency of simulation is highly dependent on the number of levels and on the number of gates in tree-like *subcircuits* represented by graphs. Therefore, we have also used 5 tree-like circuits with numbers of levels from 2 to 10 (numbers of gates from 3 to 1023). And we used two different input pattern generation modes: with a single and multiple bit transitions allowed on inputs at the same time. Experimental results presented below clearly show a noticeable speed-up of our approach.

The results for tree-like circuits are illustrated for the case of single transitions in Fig. 5 and 6 and for the case of multiple transitions in Fig. 7 and 8. Simulation time is given in seconds for 30000 random patterns. In our work we measured simulation time on both macro and gate levels. Note that the simulation time grows exponentially with the number of levels at the gate level (Fig. 5). The simulation time at the macro level (our approach) grows much slower (Fig. 5) but also not linearly (as shown in Fig. 6 with a more detailed timing scale). In Fig. 6 all the four macro-level algorithms are shown. However, there is no noticeable difference in simulation time between them.

For the multiple-bit change input pattern generation mode there is a noticeable difference in speed for the macro-level timing simulation algorithms (Fig. 8). The double-stack-based algorithm (lower thin line) is the fastest for this case and the algorithm with no stack (upper bold line) is the slowest. The timing simulation at the gate level needs the time (dashed line in Fig. 7) that grows again much faster than the time our approach requires.

Experimental results on ISCAS'85 benchmarks are given in Table 3. The first two rows show names and sizes of the benchmarks. The third row in the table indicates the two modes for pattern generation by S and M respectively. The time for 10000 input patterns simulation is given in seconds in the rows 4 to 8. Rows 4 to 7 correspond to the four macro-level algorithms and row 8 corresponds to a gate-level simulation. Rows 9 to 12 in the table (G/M Ratio) show the efficiency of the four macro-level simulation algorithms compared to the gate-level simulation algorithm. The best simulation times and speedups are shown in bold.

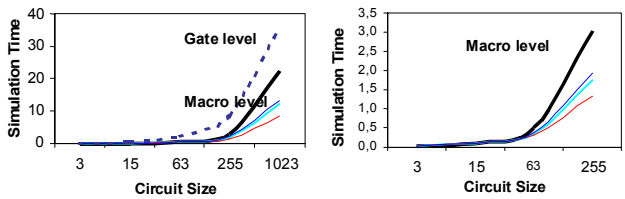



**Figure: 7, 8. Comparison of simulation times of different algorithms (multiple-bit transition mode)**

| Circuit | | 1 | c432 | | c499 | | c880 | | c1355 | | c1908 | | c2670 | | c3540 | | c5315 | | c6288 | | c7552 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Number of Gates | | 2 | 232 | | 618 | | 357 | | 514 | | 718 | | 997 | | 1446 | | 1994 | | 2416 | | 2978 | |
| Pattern Generation Mode | | 3 | S | M | S | M | S | M | S | M | S | M | S | M | S | M | S | M | S | M | S | M |
| Simulation Time for 10000 Patterns (s) | No Stack | 4 | **0,77** | 1,49 | **1,71** | **3,38** | 1,34 | 2,52 | **2,81** | **5,17** | **2,32** | **4,49** | **3,80** | 7,85 | **3,63** | 8,69 | **6,18** | 15,2 | **30,8** | 139 | **8,88** | 24,4 |
| | Single stack | 5 | 0,86 | 1,47 | 1,93 | 3,73 | 1,53 | 2,56 | 3,41 | 5,76 | 2,69 | 4,71 | 4,64 | 8,21 | 4,26 | 8,71 | 7,36 | 15,4 | 32,3 | **133** | 10,7 | 24,5 |
| | Double stack | 6 | 0,80 | **1,37** | 1,79 | 3,77 | **1,33** | **2,40** | 2,94 | 5,20 | 2,36 | 4,50 | 4,02 | **7,66** | 3,78 | **8,42** | 6,55 | **15,1** | 31,6 | 141 | 9,22 | **23,7** |
| | Double stack, no update | 7 | 0,79 | 1,46 | 1,80 | 3,53 | 1,38 | 2,58 | 2,94 | 5,44 | 2,34 | 4,59 | 4,01 | 8,00 | 3,78 | 8,81 | 6,43 | 15,6 | 33,0 | 152 | 9,26 | 24,7 |
| | Gate level | 8 | **2,16** | **3,32** | **5,30** | **9,83** | **3,26** | **5,19** | **4,86** | **8,38** | **6,98** | **11,5** | **9,24** | **15,9** | **12,9** | **23,6** | **20,1** | **37,7** | **58,7** | **272** | **28,0** | **57,1** |
| G/M Ratio | No Stack | 9 | **2,81** | 2,23 | **3,10** | **2,91** | 2,43 | 2,06 | **1,73** | **1,62** | **3,01** | **2,57** | **2,43** | 2,03 | **3,54** | 2,72 | **3,26** | 2,47 | **1,90** | 1,95 | **3,15** | 2,34 |
| | Single stack | 10 | 2,51 | 2,26 | 2,75 | 2,64 | 2,13 | 2,03 | 1,43 | 1,45 | 2,59 | 2,45 | 1,99 | 1,94 | 3,02 | 2,71 | 2,73 | 2,44 | 1,82 | **2,04** | 2,60 | 2,32 |
| | Double stack | 11 | 2,70 | **2,42** | 2,96 | 2,61 | **2,45** | **2,16** | 1,65 | 1,61 | 2,96 | 2,56 | 2,30 | **2,08** | 3,40 | **2,81** | 3,07 | **2,50** | 1,86 | 1,92 | 3,03 | **2,41** |
| | Double stack, no update | 12 | 2,73 | 2,27 | 2,94 | 2,78 | 2,36 | 2,01 | 1,65 | 1,54 | 2,98 | 2,51 | 2,30 | 1,99 | 3,40 | 2,68 | 3,13 | 2,42 | 1,78 | 1,78 | 3,02 | 2,31 |

**Table 3: Experimental results**

Note that the double stack-based and no stack-based algorithms give the best results. However, there is no big difference between the four algorithms but there is still a great speedup compared to the gate-level algorithm. The speed of simulation based on the proposed method increases up to 3,54 times for patterns with single transition and up to 2,91 times for patterns with multiple transitions.

For all the experiments we used a Sun Ultra 10 workstation with 440 MHz UltraSparc – IIi processor, 256 MB RAM, and SunOS 5.7.

## 7. Conclusions

A new approach to speed up gate-level timing simulation is proposed where, instead of gate delays, path delays for tree-like subcircuits (macros) represented by SSBDDs are used. SSBDDs capture the structure of a circuit whereas conventional BDDs does not allow that. At the same time, using SSBDDs for representing macros avoids exponential explosion of the model complexity. The number of paths in the circuit processed by delay calculation is a linear function of the number of gates.

Experiments were carried out on the ISCAS'85 benchmarks with the number of gates up to about 3000. The linear feature of the model complexity allows efficient simulation of complex realistic combinational circuits.

Four algorithms for this approach were implemented and their efficiencies compared. The timing simulation speed at the macro-level is up to 3,54 times faster compared to the gate-level simulation for the investigated set of ISCAS'85 benchmark circuits. The best among the macro-level algorithms is the double-stack based one.

The high speed of simulation is achieved on the cost of some loss of simulation data. Instead of the all waveforms for all nodes of the gate-level network, only the waveforms for the outputs of macros are calculated. This simplification is nevertheless acceptable for most industrial applications of timing simulation.


## Acknowledgements

This work has been supported partially by the Royal Swedish Academy of Sciences, Estonian Science Foundation (under Grant G4300), and the European Community (Copernicus JEP 9624 VILAB).


## References


[1] K.-T. Cheng, V.D. Agrawal. Unified Methods for VLSI Simulation and Test Generation. *Kluwer Academic Publishers,* 1989, 148 p.

[2] M. Abramovici, M.A. Breuer, A.D. Friedman. Digital Systems Testing and Testable Design. *IEEE Press,* New York, 1990, 652 p.

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

[4] R. Ubar. Beschreibung Digitaler Einrichtungen mit Alternativen Graphen für die Fehlerdiagnose. *Nachrichtentechnik/Elektronik,* (30) 1980, H.3, pp.96-102

[5] R. Ubar. Test Synthesis with Alternative Graphs. *IEEE Design & Test of Computers.* Spring 1996, pp. 48-59.

[6] R. Ubar. Multi-Valued Simulation of Digital Circuits with Structurally Synthesized Binary Decision Diagrams. *OPA, Gordon and Breach Publishers, Multiple Valued Logic,* Vol.4 pp. 141-157, 1998.

[7] R. Andrew. An algorithm for 8-valued simulation and hazard detection in gate networks. *16th Int. Symp. on Multiple Valued Logic*. Blacksburg, 1986, pp. 273-280.

[8] W. Mao, M.D. Ciletti. A variable observation method for testing delay faults. Proc. Of 27th ACM/IEEE Design Automation Conference. 1990, pp. 728-731.

[9] S. Si. Dynamic testing of redundant logic networks. IEEE Trans. on Comp,1978,Vol.C-27,No9,pp.828-832.

# Paper 3

A. Jutman, J. Raik, R. Ubar, “SSBDDs: Advantageous Model and Efficient Algorithms for Digital Circuit Modeling, Simulation & Test,” in *Proc. of 5th International Workshop on Boolean Problems (IWSBP’02)*, Freiberg, Germany, Sept. 19-20, 2002, pp. 157-166.

# SSBDDs: Advantageous Model and Efficient Algorithms for Digital Circuit Modeling, Simulation & Test

A. Jutman, J. Raik, R. Ubar
Tallinn Technical University, Estonia,
email: artur@pld.ttu.ee

**Abstract**

In this paper we sum up the research that was done during the last decade on the topic of Structurally Synthesized Binary Decision Diagrams (SSBDDs). We describe general properties of SSBDDs that make this model very efficient for circuit structure dependent methods and algorithms. In addition, we describe a deterministic test generation algorithm based on SSBDDs and four efficient simulation methods of different classes: logic simulation, multi-valued simulation, timing simulation, and fault simulation. We investigate and show the origins of their common advantages and draw conclusions, which hold for all the described algorithms. The analysis is made on the basis of experimental data acquired when applying these algorithms to ISCAS'85 benchmark circuits. The experiments unveil some new properties of these benchmarks, which we also present in our paper.

## 1 Introduction

The constant increase of integration level of modern digital circuits imposes high and further growing requirements for the methods and algorithms used in CAD design, verification, and testing. The efficiency of any algorithm usually heavily depends on the underlying mathematical model. This made the search for better models a hot topic during the last several decades.

In 1986, Bryant proposed a new data structure called *Reduced Ordered Binary Decision Diagrams* (ROBDDs) [1,2]. He showed simplicity of manipulation and proved the model canonicity, what made it one of the most popular representations of Boolean functions. This model, however, suffers from the memory explosion problem, which limits its usability on large designs. Moreover, it cannot be used as a model for such methods that require a certain degree of structural information about the design.

During the last decade, many modifications to ROBDD model were proposed. These modifications were mostly aimed at fighting the memory explosion problem but very little was done to adopt BDDs for structural problems. For such problems, the general gate-level representation is traditionally used.

In this paper we show the advantageous properties of quite a mature but not widely known model of *Structurally Synthesized Binary Decision Diagrams* (SSBDDs). This compact model preserves the structural information about the modeled circuit and utilizes the partitioning of circuit into a set of subcircuits represented each by its own SSBDD.

We investigate common properties and advantages of four methods of logic-level simulation of digital systems and a deterministic test generation algorithm, which utilize SSBDDs as the underlying mathematical model. Logic-level simulation is still one of the most often used operations on digital designs during both design and test stages. This makes it a critical issue affecting the overall cost of a project. We show the origins of the SSBDD advantages in the logic-level simulation domain.

SSBDDs have already found application as an efficient mathematical model to represent digital circuits. They were introduced the first time in [3], [4] as *Structural Alternative Graphs* and generalized as *multiple-valued* decision diagrams in [5]. This model has several critical features that make it very attractive compared to other commonly used mathematical models, such as conventional BBDs (including state-of-the-art modifications) or a gate-level netlist.

First of all, the worst-case complexity (time) of generating SSBDD model from a circuit's netlist is linear in respect to the number of logic gates (due to circuit partitioning), while it is exponential for ROBDDs. Secondly, the size of the SSBDD model is linear in respect to the circuit size (again, ROBDD can be of exponential size). Thirdly, SSBDD model implicitly preserves structural information about the circuit while other BDD models do not. It also allows Boolean operations like AND, OR, or NOT with separate SSBDDs. Finally, it even reduces the model complexity compared to the gate-level representation, as algorithms running on the SSBDD model need no

separate treatment of gates of different types (e.g. AND and OR gates are treated equally). Moreover, instead of considering each gate separately, it deals with *macros* – tree-like subcircuits (subcircuits with no reconvergent fanouts), which usually consist of several gates. Thus, each single node in an SSBDD represents a whole *signal path* from a macro input to the output of the macro. This is the most significant property, which allows development of efficient logic-level simulation algorithms. This property provides also fault collapsing for fault simulation and test generation.

Due to the above mentioned advantages the SSBDD model has been proposed as a representation model to solve various CAD problems like fast *deterministic* test pattern generation [12], efficient *design error* localization [13], *logic* and *multi-valued* simulation [6] for different purposes like hazards investigation, *delay fault* analysis, and *fault cover* analysis in dynamic testing. Efficient algorithms for *timing* and *fault* simulation were proposed in [7] and [8]. All of these methods use the advantage of the SSBDD model.

Current paper is organized as follows. In Section 2 we discuss the SSBDD model. The logic simulation is explained in Section 3, multi-valued and timing simulation methods are given in Section 4 and 5 correspondingly. Sections 6 and 7 give an overview of the SSBDD-based fault simulation and test generation. In Section 8 we discuss the experimental results and finally make concluding remarks in Section 9.

# 2 SSBDD Model: Important Properties

*Def.1* A BDD that represents a Boolean function $y=f(X)$ over a set of Boolean variables $X=\{x_1, x_2, \ldots, x_n\}$ is a directed acyclic graph $G_y=(M,\Gamma,X)$ with a set of nodes $M$ and mapping $\Gamma$ from $M$ to $M$. Set $M$ consists of two types of nodes: internal (non-terminal) $M^N$ and terminal $M^T$, $M=M^N \cup M^T$. A terminal node $m^T$ is labeled by a constant $e(m^T)\in\{0,1\}$ and is called *leaf*, while all non-terminal nodes $m\in M^N$ are labeled by variables $x\in X$, and have exactly two successor nodes. Let us denote the associated with internal node $m$ variable as $x(m)$, then $m^0$ is the successor of $m$ at the value $x(m)=0$ and $m^1$ is the successor of $m$ at the value $x(m)=1$. An example of a BDD is given in Fig.1.

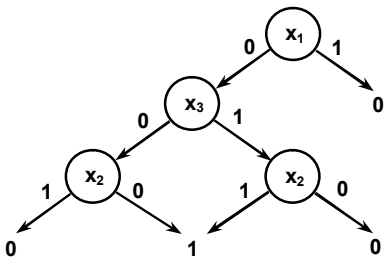



Fig.1. Binary Decision Diagram

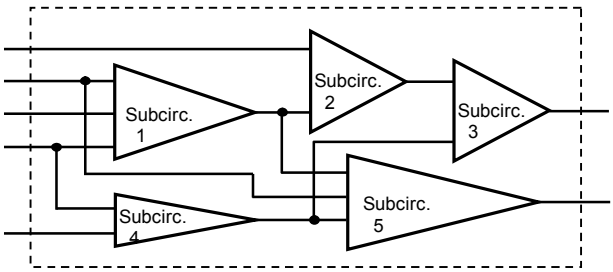



Fig.2. Combinational circuit partitioned into 5 fanout-free macros

There is a number of various kinds of BDDs have been proposed during last two decades [9]. The most of the modifications use the *Shannon decomposition* rule $y=\bar{x}f_{\bar{x}}+xf_x$. Here $f_x$ is obtained from $y=f(x)$ by replacing variable $x$ by value 1. The negative cofactor $f_{\bar{x}}$ is obtained by replacing variable $x$ by value 0. This principle, along with the enforced total *variable ordering* on the graph and the *reduction rule*, produce a BDD with a canonical property. There are other kinds of BDDs that use different decomposition rules keeping, however, the property of canonicity [9].

SSBDD model is not a canonical model. In spite of this, as we will see later, it is a natural BDD representation of a digital circuit. It does not rely on the Shannon decomposition. As the basis, it uses the *equivalent parenthesis form* (EPF), that is, describes a digital circuit structurally.

*Def.2* A BDD is called SSBDD, if there is a *one-to-one correspondence* between non-terminal nodes of the BDD and signal paths in the combinational circuit. Non-terminal nodes of an SSBDD are labeled by subscripted input variables, which can be inverted or not. In fact, SSBDD model is further defined by construction.

An SSBDD is constructed directly from a gate-level description of a combinational circuit by a graph superposition procedure. In this sense it is equivalent to EPF generation by superposition of Boolean functions. In prior to the superposition, the circuit must be partitioned into a set of tree-like fanout-free subcircuits (Fig. 2). Each such subcircuit will be represented by its own SSBDD. Therefore the whole circuit is represented not by a single SSBDD but by a set of separate SSBDDs connected by variables from extended set $X$.

Fig. 3 illustrates how an SSBDD is constructed from a combinational circuit. First, we set elementary BDDs: for AND and NOR gates. Then we use the superposition principle to combine

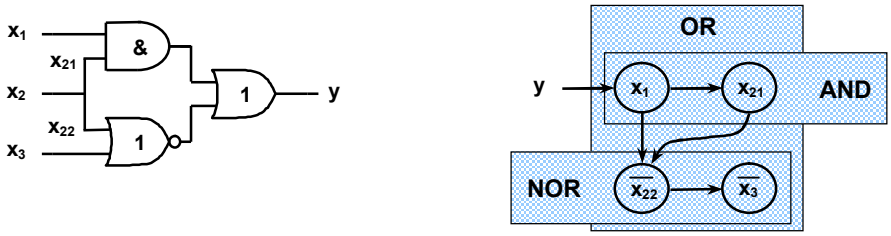



Fig.3. Illustration of the superposition principle

these into a whole single SSBDD just as OR gate combines the two gates into a single circuit. From the construction procedure it follows that it is possible to apply Boolean operations AND and OR to any two (or more) arbitrary SSBDDs. The result of this operation, a single SSBDD, will be then equivalent to the Boolean function resulting from the same Boolean operation applied to the same initial Boolean functions represented by the given (initial) SSBDDs.

Fig. 4 demonstrates two different SSBDD representations for the circuit from Fig. 3. For SSBDDs, it is agreed that the edge corresponding to the value *x(m)=0* always goes down while the edge corresponding to *x(m)=1* always goes right. Terminal nodes are not shown on this picture because it is also agreed that if *x(m)=0* and no edge goes down from the node *m*, then one gets to the terminal node labeled by 0. Similarly, in the case of missing "right" edge, one reaches the terminal node "1".

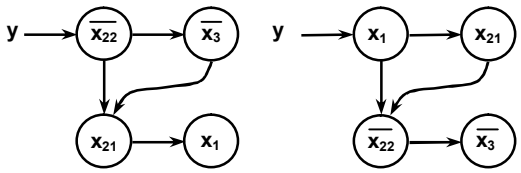



Fig. 4. Two different SSBDD representations for the combinational circuit from Fig. 3

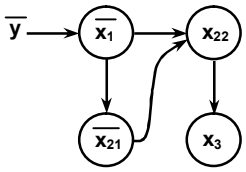



Fig. 5. An inverted SSBDD from Fig. 3

This unusual *alternative description style* used for SSBDD representation is possible due to introduction of the inversion of variables. In Fig. 4, variable $x_{22}$ and $x_3$ are inverted. Otherwise it would be impossible to represent the NOR gate using this alternative description style.

It might seem useless to introduce the new style. However, it brings many advantages successfully utilized in the algorithms described in the following sections. It is not hard to notice that the SSBDD is a *planar graph* by construction. Together with the alternative description style this property allows to substantially reduce the search space of the algorithms working on the SSBDD model. In fact, this property is analogous to the property of monotony of Boolean EPFs. From Fig. 4 it is seen that no separate treatment for different logic gates needed in SSBDD model. This must be considered as an advantage compared to gate-level netlist representation.

Now we can mention that in addition to AND and OR operations, an operation of Boolean inversion can be introduced on SSBDDs. Fig. 5 illustrates an *inverted* SSBDD in respect to the SSBDD from Fig.3. In order to invert an SSBDD we have to perform two simple operations for each node of target SSBDD. First, swap the right and down edges. Second, invert all the variables.

Definition of the three Boolean operations (AND, OR, NOT) on SSBDDs allows to apply any arbitrary Boolean operation (as a combination of the defined operations) on an arbitrary number of SSBDDs. This property allows easy manipulations with the SSBDD model as well as easy construction of an SSBDD from a Boolean function.

Unlike the example in Fig 3, the automated SSBDD model construction procedure starts from a circuit output and moves along a signal path to the direction of inputs until a fan-out point is reached. Then the next signal path is chosen. The SSBDD construction stops when all the signal paths have been traversed up to their fanout points or primary inputs. The next SSBDD is constructed in the same way. The procedure stops when the whole circuit is covered. During such a procedure, the circuit partitioning is done in a natural way with no additional computational efforts.

*SSBDD Construction Algorithm:*

***Construction (*current gate*)***
**{**
**For each input of current gate**

```
  {
    If the input is not primary or it is not a fanout
      Insert the next gate into the SSBDD
      Call Construction (next gate)
    End If
  }
Return
}
```

The latter recursive algorithm must be performed for each output and fanout point. It starts from the closest to the output (or the fanout point) logic gate and moves to the direction of inputs.

From the SSBDD model construction algorithm as well as from the definition we can derive the complexity of the model which is equal to the total number of signal paths in all the tree-like subcircuits. In the worst case this number is equal to the total number of all inputs of all the gates in the modeled circuit. In other words the size of this model has linear complexity *O(n)* in respect to the number of logic gates. Note that neither reordering nor decision making is done during the SSBDD model generation. This makes the worst case time of the model generation procedure linear in respect to the model size. Consequently, it is linear *O(n)* to the number of logic gates in the circuit as well.

Table 1. Comparison of sizes of different BDD models

| Circuit | In | Out | Gates | ROBDD [2] | FBDD [11] | SSBDD |
|---|---|---|---|---|---|---|
| c432 | 36 | 7 | 232 | 30200 | 1063 | 308 |
| c499 | 41 | 32 | 618 | 49786 | 25866 | 601 |
| c880 | 60 | 26 | 357 | 7655 | 3575 | 497 |
| c1355 | 41 | 32 | 514 | 39858 | *N/A* | 809 |
| c1908 | 33 | 25 | 718 | 12463 | 5103 | 866 |
| c2670 | 233 | 140 | 997 | *N/A* | 1815 | 1313 |
| c3540 | 50 | 22 | 1446 | 208947 | 21000 | 1648 |
| c5315 | 178 | 123 | 1994 | 32193 | 1594 | 2712 |
| c6288 | 32 | 32 | 2416 | *N/A* | *N/A* | 3872 |
| c7552 | 207 | 108 | 2978 | *N/A* | 2092 | 3552 |

Table 1 gives the model size comparison for several various BDD representations. As it is seen from the table, during the last decade there has been a substantial reduction in BDD model size from quite bulky ROBDDs to rather modest FBDDs (Free BDDs [11]). This became possible due to new sophisticated minimization algorithms and some modifications to the model itself. However, the SSBDD model still leads this race as it was twenty years ago. In average, it offers minimal model size independently of circuit function (for instance, there is no known efficient ROBDD representation for combinational multiplier). However, these two models are intended for different tasks as well. Since there is more than one SSBDD describing the same circuit, SSBDD model cannot be considered as a canonical one. This is the reason why the applications of SSBDDs and conventional BDDs are mostly different.

# 3 Logic Simulation

Two-valued *logic simulation* on SSBDDs is equivalent to path tracing procedure on graphs according to the values of variables at a given input pattern. An assignment to the variables $X$ activates a path $l(m_0, m^T)$ from the *root* node $m_0$ to a terminal node $m^T$. The simulation procedure consists in tracing the path $l(m_0,m^T)$ and evaluating the $y=f(x)$ by finding the value $e$ of the terminal node $m^T$.

*Logic Simulation Algorithm:*

```
For each SSBDD in the model
{
  Take the first node of current SSBDD
  While current node is not a terminal node
  {
    Evaluate current node and take the next node
  }
  Save the output value of current SSBDD
}
```

It is obvious that the worst case complexity of logic simulation is equal to the total number of nodes in SSBDDs. I.e. it is also linear *O(n)*.

# 4 Multi-Valued Simulation

For multi-valued simulation, we use a procedure based on calculation of Boolean derivatives on SSBDDs. Consi-der the set $S_5 = \{0, 1, \varepsilon, h, x\}$ for 5-valued simulation and a multi-valued vector $x^t = (x^t_1, x^t_2, \ldots x^t_i, \ldots x^t_n)$ for a transition period $t$. Denote a subset of literals with dynamic values at this vector by $X_D \subseteq X$, i.e.

$$X_D = \{x_i \mid x^t_i \in S_5 \cap S_D\} = \{x_i \mid x^t_i \in \{\varepsilon, h, x\}\}$$

Denote $l(m_i,m_j) = 1$, if there exists an activated path between the nodes $m_i$ and $m_j$ for a given vector $x^t$, otherwise, $l(m_i,m_j) = 0$.

*Theorem 1.* Given $y = f(x)$ and $x_i \in X$, the condition $dy/dx_i = 1$ for SSBDD $G_y = (M,\Gamma,X)$ where $x(m) \equiv x_i$ is equivalent to the following equation:

$$l(m_0,m) \wedge l(m^1, m^{T,1}) \wedge l(m^0, m^{T,0}) = 1$$

The proof of the theorem can be found in [6].

Note, Theorem 1 can be used for calculating Boolean derivatives only in the case where vector $x^t$ is two-valued, because only in this case all the paths are activated uniquely. The general case, when $x^t$ is a multi-valued vector, is considered in the following theorem.

*Def.3* Let $max\{l(m_i,m_j)\}=1$ over $x_k \in X_D$, if there exists at least one activated path between $m_i$ and $m_j$ for all two-valued assignments of $x_k \in X_D$, otherwise $max\{l(m_i,m_j)\}=0$.

*Theorem 2.* Given $y=P(X)$ and $x_i \in X$, the condition

$$\max_{x_k \in X_D} \{dy/dx_i\}=1$$

for SSBDD $G_y = (M,\Gamma,X)$ where $x(m) \equiv x_i$ is equivalent to the following equation:

$$\max_{x_k \in X_D}\{l(m_0,m)\} \wedge \max_{x_k \in X_D} \{l(m^1,m^{T,1})\} \wedge \max_{x_k \in X_D} \{l(m^0,m^{T,0})\} = 1$$

The proof of the theorem is analogous to the proof of the Theorem 1.

For calculating the maximum of a Boolean derivative and proving that $max\{dy/dx(m)\} = 1$, all dynamic values when tracing the path $l(m^1, m^{T,1})$ should be replaced by 1 and when tracing the path $l(m^0, m^{T,0})$ by 0. This follows from the property of monotony of Boolean EPFs. When tracing the path $l(m_0,m)$, all dynamic values should be replaced either by 1 or by 0 properly, so that the node $m$ can be reached. In fact, instead of sequentially calculating the maximum of derivatives separately step by step for all the nodes $m$, where $x(m) \in X_D$, we can traverse all the paths from all the nodes $m$: $x(m) \in X_D$ in both directions by a single procedure based on nested calculation of all the derivatives. The successful utilization of the described idea allowed the creation of an efficient algorithm with linear worst-case complexity $2n$.

We do not give the multi-valued simulation algorithm here because it does not have a nice compact representation. It is thoroughly described in [6] instead.

# 5 Timing Simulation

The timing simulation approach is based on the same principle of calculation of Boolean derivatives on SSBDDs. The difference between these methods is that in multi-valued simulation we are tracing paths to search for the nodes with variables having dynamic values while in timing simulation we are searching for nodes that switch in current moment of time (i.e. a notion of time is introduced). Unlike in multi-valued simulation, the complexity of the problem of timing simulation itself is NP-complete and does not depend on the underlying model. It is due to the fact that a single input transition may result in exponentially long event sequences in certain circuits. On the other hand, the worst number of events at the output of a tree-like fanout free circuit is equal to the total number of events at its inputs. This means linear complexity. The number of events in a

common circuit is somewhere in between. In fact, it is far from exponential case. Otherwise, the timing simulation of an ISCAS circuit for a modest number of input vectors would be impossible.

In the following we give a simplified timing simulation algorithm on SSBDD model. The detailed description of the algorithm, which exploits the property of monotony of Boolean EPFs and other SSBDD properties, is given in [7].

*Timing Simulation Algorithm:*

```
For each clock cycle
{
  Set current time to 0
  Perform Logic Simulation and evaluate the output
  Make list of all the nodes with transitions
  Sort list by time of transition in ascending order
  For all entries in the event list
  {
    Set current time to the value from the list entry
    Apply the specified transition
    Perform Logic Simulation and evaluate the output
    If current output value differs from previous
      Assign current time to the output transition
} }
```

We had to introduce a notion of time into SSBDD model in order to make the latter algorithm work. In this extended model a certain delay is assigned to each node. This delay is calculated as a sum of delays of all logic gates along the path represented by the corresponding node [6]. Fig. 6 illustrates this idea.

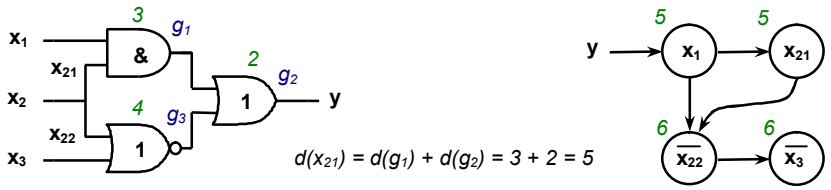



Fig. 6. Modeling delays on SSBDDs

# 6 Fault Simulation

In general, the fault simulation task is considerably complex, since several stuck-at faults can be simultaneously present in the circuit. A circuit with $n$ lines can have $3^n - 1$ different stuck line combinations [10]. Needless to say that even a moderate value of n will result in an enormous amount of multiple faults. It is a common practice, therefore, to model only single stuck-at faults. An n-line circuit has $2n$ single stuck-at faults. In the following we will see that modeling stuck-at faults on SSBDD representations allows to reduce this number further.

It is possible to minimize the number of faults to be modeled by a technique called *fault collapsing*. Traditionally, this is done by implementing the relation of *fault dominance* on the set of faults. It is said that fault $f_1$ dominates $f_2$, if any test that detects $f_2$ will also detect $f_1$. Representing stuck-at faults by faults at SSBDD nodes can be viewed as a type of fault collapsing by applying fault dominance relations along the signal paths of the circuit.

While in the gate-level descriptions we model stuck-at faults at the interconnections between the gates, in SSBDD representations the faults are considered at nodes. For example, stuck-at-0 fault at a node is modeled with the 0-edge of the node being constantly activated, regardless of the value of the variable labeling this node. As it was stated in Def. 2, each SSBDD node represents a distinct path in the corresponding fanout-free circuit. By testing all the SSBDD node faults we will consequently test all the paths in the circuit and thus all the single stuck-at faults. This ability of SSBDDs to implicitly model logic level stuck-at faults is a very important property, which distinguishes it from other classes of BDDs.

Table 2 presents the number of uncollapsed faults, collapsed faults and SSBDD faults in eight ISCAS85 circuits. As we can see from the table, the traditional fault collapsing and SSBDD representations provide almost identical results. The difference in the number of faults is at most 8

Table 2. Comparison of collapsed and SSBDD faults

| Circuit | Uncollapsed | Collapsed | SSBDD |
|---|---|---|---|
| c880 | 1550 | **942** | 994 |
| c1355 | 2194 | **1574** | 1618 |
| c1908 | 2788 | 1879 | **1732** |
| c2670 | 4150 | 2747 | **2626** |
| c3540 | 5568 | 3428 | **3296** |
| c5315 | 8638 | **5350** | 5424 |
| c6288 | 9728 | **7744** | **7744** |
| c7552 | 11590 | 7550 | **7104** |

% (in the case of c1908). SSBDD achieves in average even 2 % better compaction of the fault list than the traditional approach, reducing the fault lists in average about 1.5 times. However, this advantage occurs partly because in ISCAS models, for every signal line, at least one fault is required to be included to the fault list. In a more advanced collapsing techniques the list can be further minimized. Nevertheless, Table 2 indicates that by generating SSBDDs, the fault list will simultaneously be reduced by a considerable amount of faults.

The advantage of the SSBDD based collapsing over the traditional one is that it allows us at the same time to rise to a higher abstraction level of circuit modeling. In the traditional case we would only minimize the number of faults but would still be working at the level of logic gates.

A simple fault simulation algorithm of complexity $O((n+1)^2)$ is given in the following.

*Fault Simulation Algorithm:*

```
Perform fault-free Logic Simulation
For each SSBDD
{
  Take the first node of current SSBDD
  While current node is not a terminal node
  {
    Evaluate current node
    Insert corresponding fault
    Perform Logic Simulation to evaluate output
    If current output differs from the fault-free
      Save current fault as detected
    Remove current fault and take the next node
} }
```

More sophisticated and efficient *parallel critical path tracing fault simulation* method is described in [5]. It is based on combining the parallel backward critical path tracing inside fanout free subcircuits with parallel forward critical path tracing between fanout free subcircuits for fanout stem analysis.

The basic idea of parallel fault simulation on SSBDD models is similar to this type of simulation at gate-level. Here, traditional style SSBDD descriptions are preferable since in these we do not have to keep track of whether a variable labeling a node is inverted or not. The simulation takes place as follows. Starting from the node with the highest index value, we repeat operation:

$$\left(x(m)\,\&\,x(m^1)\right)\vee\left(\overline{x(m)}\,\&\,x(m^0)\right)$$

for each node of the SSBDD. In this operation, m denotes the current node. $m^0$ and $m^1$ are its 0 and 1-successors, respectively. x(m) denotes the value of the variable labeling the node m. The result of the simulation will be the value calculated for the root node $m_0$.

# 7 Test Generation

The test generation approach presented in this paper would be an exact equivalent of PODEM [14] if SSBDD models consisted of a system, where for each logic gate an elementary BDD corresponded. In order to compare test generation on structurally synthesized BDD and gate level models, we have implemented two types of SSBDD synthesis: one, where BDDs are synthesized for each FFR and the other, where they are synthesized for each gate. In the following, different tasks of the SSBDD-based test generation process are explained.

*Determining the D-Frontier.* Similar to classical test generation approaches, D-frontier is determined by finding all the nodes labeled by variables with the value D (or D-bar) and performing X-path check for them.

*X-Path Check.* X-path check for a circuit line with the value D (or D-bar) checks for the existence of such a path from that line to a primary output, where all the lines have the value X. At the gate-level, the outputs of each gate along corresponding paths have to be checked. In the SSBDD representations, the primitives are reduced to SSBDDs, which significantly speeds up the procedure. In SSBDD models it is possible to perform X-path check by simply checking the values of variables corresponding to the respective SSBDDs.

*Finding Boolean Derivatives.* Calculating Boolean derivative for the nodes in SSBDD is similar to determining current backtrace objective in PODEM. In PODEM, the aim of selecting current objective was to propagate the fault effect (D or D-bar) to an output of a subsequent gate. In our approach, the primitives are SSBDDs and therefore the objective is to propagate the fault effect to the output of a subsequent fanout-free region.

In order to check that Boolean derivative for a node $n$ in an SSBDD $G$ is not zero, three paths have to be traversed. One from the root node of $G$ to the node $n$. The two other paths have to be traversed downwards and rightwards from $n$, respectively. If a path from root to node $n$ can not be traversed due to the values assigned to the variables in the SSBDD, or if the two last mentioned paths overlap, Boolean derivative for node $n$ will be zero. The derivative will be zero also if the downward path exits the BDD rightwards or rightward path exits the BDD downwards. If the derivative is one, current objective will be to backtrace the variable whose value is X and that labels a node that was traversed along one of the three paths. There exists always at least one such variable. The backtrace value is determined by the traversal direction of the respective node.

The use of Boolean derivatives for choosing current objective is the reason why on SSBDDs higher fault coverages were achieved than in the case where BDD models were representing logic gates (see Table 2). Due to the fact that primitives are given on the level of FFRs rather than gates, many inconsistencies are detected earlier.

*Backtrace.* Backtrace on the SSBDD models is simple and very fast. Value $V$ is backtraced on an SSBDD $G$ as follows. Starting from the root node we traverse a path forced by variable values until the first node $n$ labeled by variable $x(n)$, which has the value X, is reached. Subsequently, we will backtrace the value, which activates node $n$ to the value $V$, on the SSBDD corresponding to the variable $x(n)$. Value $V$ is recursively backtraced on the SSBDD model until primary inputs are reached.

*D-Simulation.* D-simulation is the most time-consuming procedure in current implementation. This can be expected since it is called repeatedly after each value assignment to the primary inputs. At present, we use three-valued simulation (0, 1, X) on two vectors, one for the fault-free and the other for the faulty circuit.

*Fault Simulation.* Subsequent to each generated test, fault simulation is performed in order to determine the faults covered by the generated vector.

In the following the general algorithm of SSBDD-based test generation is given. The following definitions are used.

*Fault node* is the SSBDD node under test.

*Fault variable* is the variable labeling the fault node.

*Fault graph* is the SSBDD that contains the fault node.

*Test Generation Algorithm:*

```
While D not propagated to a PO
{
  If fault variable = X
  { Backtrace value that activates the fault }
  Else
  {
    If fault is not activated
    { Backtrack }
    Else
    {
      If the value of fault graph not D
      {
```

```
        If Boolean derivative for fault node is 0
        { Backtrack }
      }
      Else
      {
        Determine the D-frontier
        Perform backtrace with current objective
} } } }
```

# 8 Experimental Results

The experiments with the four described methods of logic-level simulation were carried out on ISCAS'85 benchmarks. We run them for each circuit on two levels of abstraction: a gate-level and SSBDD model. This allows to measure directly the effect of the chosen model on simulation time.

In all cases the speed of simulation for SSBDD model was higher than that for gate-level representation [6,7]. For logic, multi-valued, and timing simulation the average speed-up varies from about 1,5 up to almost 4 times compared to algorithms working on the gate-level netlist model (Fig. 7). The fault simulation algorithm (fat dashed line in Fig. 7) shows the most noticeable acceleration. The runtime improves up to 7 times when simulation is performed on SSBDD model.

The rise of simulation performance (and, in fact, reduction of the capacity of required memory) becomes possible due to the model complexity reduction by shifting from lower gate level to a higher macro level of fanout free subcircuits (Fig. 2). On this macro level all the required structural information is implicitly preserved due to the use of SSBDDs.

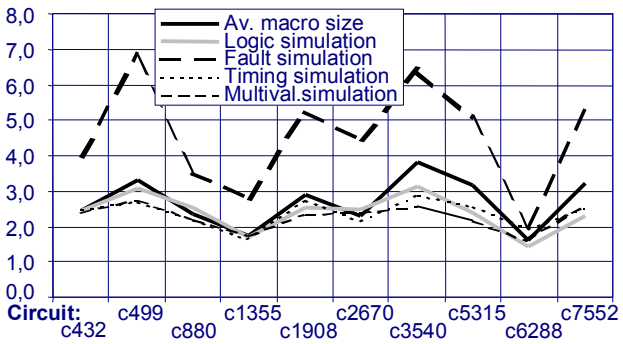



Fig. 7. Logic-level simulation speedup for different algorithms

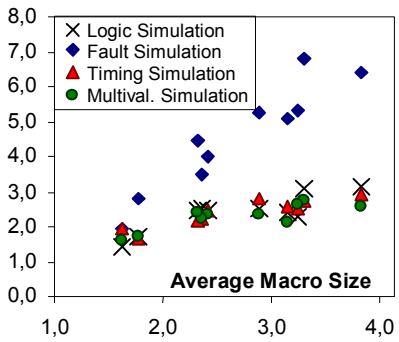



Fig. 8. Logic-level simulation speedup vs. average macro size

One can also notice a very interesting property of the methods researched, i.e. their results for all the circuits are correlated. The origin of this effect is the variance of the average size of macros (measured in the number of gates) for various circuits. This property is shown in Fig. 7 by the bold black line. The behavior of this line is also correlated to the behavior of the simulation curves. This unveils the fundamental property of the investigated methods, which is that the average simulation speed-up is directly proportional to the average size of a fanout-free subcircuit in the circuit (Fig 8).

The same property can be used also to describe ISCAS'85 benchmarks themselves. In fact, it is hard to detect any relationship that holds with no exception for all circuits from ISCAS 85 package. For instance, circuit size is not much informative if we need to estimate a BDD generation time. This is also useless for estimation of simulation speedup we earn from SSBDD model. Now we can arrange these benchmarks by the average size of fanout-free subcircuits (macros). This order is significant at least for an evaluation of methods, which use SSBDDs as the underlying model. The order is presented in Table 3.

Table 3. Benchmarks' order according to the average macro size

| c6288 | c1355 | c2670 | c880 | c432 | c1908 | c5315 | c7552 | c499 | c3540 |
|---|---|---|---|---|---|---|---|---|---|
| 1,62 | 1,77 | 2,32 | 2,36 | 2,42 | 2,90 | 3,15 | 3,24 | 3,30 | 3,83 |

Table 4 presents experimental results that were measured on a 233 MHz Pentium II PC under Windows 95 operating system. In the table, comparative test generation results on SSBDD and

gate levels are given for two different timeout values. The achieved fault coverages were calculated by simulating faults of the test patterns on the noncollapsed gate-level model. Therefore, the coverages differ slightly from the ones of the 'classical', collapsed, fault model. As the Table shows, test generation times are 2.1 – 4.5 times faster at the SSBDD level than at the gate-level. Thus, SSBDD descriptions appear to be an efficient model for the test generation purposes.

Table 4. Comparison of SSBDD and gate-level test generation

| Circuit | Fault coverage, % | | Time, s | |
|---|---|---|---|---|
| | SSBDD | Gate | SSBDD | Gate |
| c432 | 97.33 | 87.06 | 0.10 | 0.32 |
| c880 | 100.0 | 100.0 | 0.05 | 0.11 |
| c1355 | 99.64 | 99.64 | 0.24 | 0.64 |
| c1908 | 99.75 | 99.46 | 0.22 | 0.65 |
| c2670 | 96.67 | 95.16 | 0.55 | 1.78 |
| c3540 | 95.58 | 95.24 | 0.77 | 3.47 |
| c5315 | 99.78 | 98.90 | 0.57 | 2.75 |
| c6288 | 99.80 | 99.30 | 0.60 | 1.45 |
| c7552 | 99.46 | 97.10 | 2.71 | 11.5 |

# 9 Conclusions

Common features and advantages of four logic-level simulation methods implemented on the SSBDD model are discussed in the paper. It is shown that the efficiency of the algorithms directly depends on the underlying model. The simulation speed-up in comparison with the gate-level simulation speed is linearly proportional to the average size of a macro measured in the number of gates. This statement holds for at least common realistic combinational circuits such as the ISCAS '85 benchmarks.

The algorithm that most benefited from the usage of the SSBDD model is the fault simulation algorithm, as it utilizes at the same time fault collapsing together with the model simplification.

The experiments show also the advantages of SSBDD-based TG algorithm in respect to a similar algorithm working on the gate-level netlist.

For more details on each of the discussed approaches look in [6], [7], and [8].

# References


[1] R. Bryant "Graph-based algorithms for Boolean function manipulation", *IEEE Transaction on Computers*, 1986, vol. C-35, pp. 677-691.

[2] K.S. Brace, R.L. Rudell, and R.E. Bryant, "Efficient Implementation of a BDD Package," In *Proc. of the 27th DAC*, June 1990, pp. 40-45

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

[4] R. Ubar, "Beschreibung Digitaler Einrichtungen mit Alternativen Graphen für die Fehlerdiagnose," *Nachrichtentechnik/Elektronik,* (30) 1980, H.3, pp.96-102.

[5] R. Ubar, "Test Synthesis with Alternative Graphs," *IEEE D&T of Comp.* Spring 1996, pp. 48-59.

[6] R. Ubar, "Multi-Valued Simulation of Digital Circuits with Structurally Synthesized Binary Decision Diagrams," *OPA, Gordon and Breach Publishers, Multiple Valued Logic,* 1998, Vol.4, pp. 141-157.

[7] R. Ubar, A. Jutman, Z. Peng, "Timing Simulation of Digital Circuits with Binary Decision Diagrams", in *Proc. of DATE 2001 Conference*, München, Germany, 2001, pp. 460-466.

[8] R. Ubar, "Parallel Critical Path Tracing Fault Simulation," in *Proc. of the 39. Int. Wiss. Kolloquium*, Ilmenau, Germany, 1994, Band 1, pp. 399-404.

[9] A.Narayan, "Recent Advances in BDD Based Representations for Boolean Functions: A Survey", in *Proc. 12th International Conference on VLSI Design*, Goa, India, 1999, pp. 408-413.

[10] M. Abramovici, et al., *Digital Systems Testing and Testable Design*, New York, IEEE Press, 1990, 652p.

[11] W. Günther, R. Drechsler, "Minimization of Free BDDs," In *Proc. of Asia and South Pacific Design Automation Conf.*, Hong Kong, Jan 1999, pp. 323-326

[12] J. Raik, R. Ubar, "Feasibility of Structurally Synthesized BDD Models for Test Generation," *Compendium of Papers of the European Test Workshop*, Barcelona, May 27-29, 1998, pp. 145-146.

[13] 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.

[14] P.Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits", IEEE Trans. Comput., pp. 215-222, March 1981.

# Paper 4

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.

# Design Error Diagnosis in Digital Circuits with Stuck-at Fault Model

**A. Jutman, R. Ubar**

Tallinn Technical University, Computer Engineering Department
Raja 15, 12618, Tallinn, Estonia. Fax: (+372) 620 22 53
artur@pld.ttu.ee, raiub@pld.ttu.ee

***Abstract.*** *In this paper we describe in detail a new method for the single gate-level design error diagnosis in combinational circuits. Distinctive features of the method are hierarchical approach (the localizing procedure starts at the macro level and finishes at the gate level), use of stuck-at fault model (it is mapped into design error domain only in the end), and design error diagnostic procedure that uses only test patterns generated by conventional gate-level stuck-at fault test pattern generators (ATPG). No special diagnostic tests are used because they are much more time consuming. Binary decision diagrams (BDD) are exploited for representing and localizing stuck-at faults on the higher signal path level. On the basis of detected faulty signal paths, suspected stuck-at faults at gate inputs are calculated, and then mapped into suspected design error(s). This method is enhanced compared to our previous work. It is applicable to redundant circuits and allows using incomplete tests for error diagnosis. Experimental data on ISCAS benchmark circuits shows the advantage of the proposed method compared to the known algorithms of design error diagnosis.*

## 1. INTRODUCTION

The VLSI design process is getting more and more complicated due to continuously growing sizes and complexity of the projects. Due to this fact the error emergence probability is big enough on different design stages. An error not detected at the earlier stages may cause a cost explosion of the project and significantly increase the time-to-market. Design verification and error localization, in their turn, become more and more time consuming tasks.

Most design error diagnosis approaches are either simulation-based [1, 3, 5, 7, 10, 11] or symbolic [2, 4, 8, 9, 12]. Simulation based approaches rely on a number of test vectors that differentiate the implementation and the specification. The potential error region is reduced according to simulation results of these vectors. The *symbolic* approaches do not rely on test vectors. They usually use Ordered Binary Decision Diagrams (OBDD) [14] to characterize the necessary and sufficient condition of a potential error source as a Boolean formula. The symbolic approaches are more accurate than simulation-based, however the explosion of the complexity for some classes of circuits puts practical limitations to the use of BDDs in locating design errors.

In [4, 6, 8, 15] structural approaches for design error rectification are proposed. Such approaches mostly apply verification techniques to narrow down the potential error region in the implementation. The techniques applied to search error(s) in this remaining area may be various. In [15] a heuristic called *back-substitution* is employed in hopes of fixing the error incrementally. In [4] a symbolic re-synthesis approach is used. Although these approaches are suitable for large circuits, their major drawback is that the success of the whole procedure is highly dependent on existence of structural similarity between two circuits.

In [16] and [17] general ideas and basic theoretical concepts of a new hierarchical design error diagnosis method are proposed, which is free from the restriction of the

structural approaches described above. The specification can be represented on any level of abstraction; it can be given as a truth table, as a BDD, as another gate-level circuit, or as a Boolean formula. The implementation is represented by structural BDDs (SSBDD) [22], which are generated directly from gate-level netlists. The success of the method does not depend on any structural similarity between the specification and the implementation. The idea of this new approach is refined and experimental results are presented in [18, 19].

This paper presents a further enhancement of the method that allows to extend it so that it is able to diagnose design errors using incomplete tests and that it is also applicable to redundant circuits.

The method is based on the stuck-at fault model, where all the analysis and reasoning are carried out in terms of stuck-at faults and only in the end the result of diagnosis will be mapped into the design error area. Such a treatment allows to exploit traditional ATPGs to serve the problem of design error diagnosis.

In contrast to BDD or OBDD circuit representations the complexity of SSBDD representation used in this approach grows not exponentially but only linearly [22]. In addition, SSBDD representation preserves structural information about a circuit allowing to develop fault diagnosis procedures that are more efficient for increasing the speed in error detection and localization than gate-level ones. On SSBDDs, a primary set of suspected faulty signal paths is calculated. Based on this set, a list of suspected erroneous gates is generated, which will be subsequently reduced to the minimum by using the information obtained from the test experiment.

Our approach combines both verification and error localization techniques. The information gathered on the *verification* stage is used during *localization* reducing the time needed for the whole process.

Due to the facts enumerated above, our method allows to increase the speed of design error diagnosis significantly in comparison to the known methods.

In [13] a widely adopted simple design error model is given. It includes two basic error categories: functional errors of gate elements and connection errors of signal lines. Gate errors, in their turn, are divided into gate substitution, extra/missing gate, and extra/missing inverter errors, and connection errors into extra and missing line errors. However in [7] it is shown that simple extra gate errors could be rectified using only gate substitution error model. Thus in this work, we consider only single extra/missing inverter and gate replacement error types. So, missing gate error and connection types of errors remain out of the scope of this paper. They, as well as multiple errors, are the subject of our future work.

Section 2 of the paper presents necessary definitions and terminology. The use of stuck-at faults and mapping of the diagnosis results into the design error domain are explained in Section 3. Representation of faults and signal paths on macro and gate levels is discussed in Section 4. Fault tables and vector representation of faults are described in Section 5. The algorithm for design error diagnosis is presented in Section 6 and an example of using is given in Section 7. Section 8 describes some limitations and possible improvements for the method. Experimental data is discussed in Section 9, and finally, Section 10 presents some conclusions.

## 2. DEFINITIONS AND TERMINOLOGY

Consider a circuit specification, and its implementation. The way of specification representation is not significant. Only a relationship between input patterns and output responses in the specification is important. However, without loss of generality, let the specification and the implementation be given at the Boolean level. The specification output is given by a set of variables $W = \{w_1 , w_2 , ... , w_m\}$, and the implementation output is given by a set of variables $Y = \{y_1 , y_2 , ... , y_m\}$, where m is the number of outputs. Let $X = \{x_1, x_2, ... , x_n\}$ be a set of input variables. The implementation is a gate network and Z is a set of internal variables used for connection of gates. Let S be the set of variables in the implementation $S = Y \cup Z \cup X$. The gates implement simple Boolean functions AND, NAND, OR, NOR, and NOT. An additional gate type FAN is added (one input, two or more outputs) to model fanout points (Fig. 1). It is not used in the diagnostic procedure but it is needed in several definitions below.

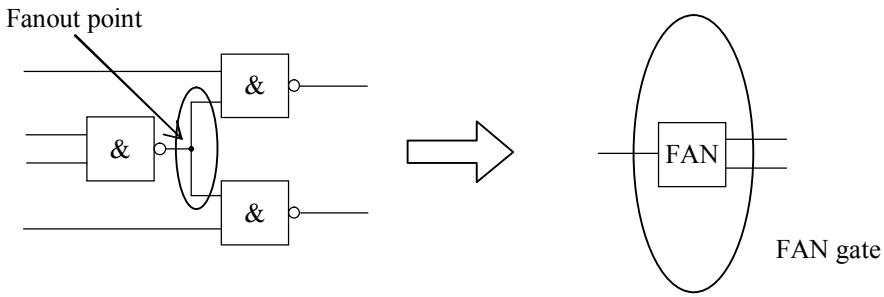



*Fig. 1. **A FAN gate***

We use two different levels for representing the implementation: gate and macro-level representations.

Let $X^F$ and $Z^F$ be subsets of inputs and internal variables that fanout (they are inputs to FAN gates). Let $Z^{FG}$ be a subset of internal variables that are outputs of FAN gates. At the gate level, the network is described by a set $NG = \{g_k\}$ of gate functions $s_k = g_k(s_k^1, s_k^2, ... ,s_k^h)$ where $s_k \in Y \cup Z$, and $s_k^j \in (Z - Z^F) \cup (X - X^F)$. At the macro-level, the network is given by a set $NF=\{f_k\}$ of macro functions $s_k = f_k(s_k^1, s_k^2, ..., s_k^p)$ in an equivalent parenthesis form (EPF) [21], where $s_k \in Y \cup Z^F$, and $S_k = \{s_k^1, s_k^2, ..., s_k^p\} \subseteq Z^{FG} \cup (X - X^F)$ are its set of inputs. In other words, a macro is a tree-structured subcircuit with no fanout points inside. It has several inputs and only one output (Fig. 2).

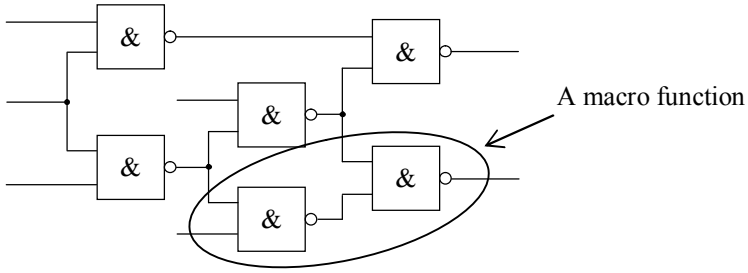



*Fig. 2. **An example of a macro function***

The following design error types are considered throughout the paper in relation to gates $g_k \in NG$.

Definition 1. *Gate replacement error*. It denotes a design error, which can be corrected by replacing the gate $g_i$ in NG with another gate $g_j$, by $g_i \rightarrow g_j$. The "$\rightarrow$" sign refers to "is replaced by".

Definition 2. *Extra/missing inverter error*. It denotes a design error, which can be corrected by removing/inserting an inverter at some input $s \in X$, or at some fanout branch $s \in Z^{FG}$: $s \rightarrow NOT(s)$.

Definition 3. *Single simple error hypothesis*. Our design error diagnosis methodology is based on a single error hypothesis, which assumes that only one error of the following types can be present in a circuit: 1) an extra/missing inverter, 2) a random gate replacement between AND, OR, NAND, and NOR gates.

It should be clear from these definitions that only inverters that are occasionally inserted/omitted at a gate input or at a primary input of a circuit are treated as extra/missing inverters. Inverter inserted/omitted at a gate output is covered by gate replacement error type (Fig. 3).


*Fig. 3.* ***Extra inverter and gate replacement errors***

Definition 4. *Test patterns*. For a circuit with n inputs, a test pattern $T'_i$ is an n-bit vector which may be binary $BN^n$ or ternary $TN^n$, where BN = {0,1} - the Boolean domain, TN = {0,1,U} - the ternary domain, where U is a don't care. Let a *test* be a set of all test patterns applied to a circuit during one test experiment as $T = \{T_1, T_2, ... T_t\}$, where t is a number of test patterns.

Definition 5. *Stuck-at fault set*. Let F be the set of stuck-at faults s/1 and s/0, where $s \in Z \cup X$.

Definition 6. *Detectable stuck-at faults*. A test pattern $T_i$ detects a stuck-at-e fault s/e, $e \in \{0,1\}$ at the output $y_j$, if applying the test pattern $T_i$ to the implementation and the specification, the result $y_j(T_i) \neq w_j(T_i)$ is observed. Let us call s/e a detectable by a test pattern $T_i$ at the $y_j$ output stuck-at fault. We denote the detection information as $\varepsilon \in \{0,1\}$, where

$$\varepsilon(T_i, y_j) = \begin{cases} 0, \text{ if the test } T_i \text{ passed at } y_j \\ 1, \text{ if the test } T_i \text{ detected an error at } y_j \end{cases}$$

and as $\delta \in \{X,0,1,D\}$, where

$$\delta(T_i, y_j) = \begin{cases} 0, \text{stuck-at 0 (s/0) is } \textit{detectable} \text{ by } T_i \text{ at } y_j \\ 1, \text{stuck-at 1 (s/1) is } \textit{detectable} \text{ by } T_i \text{ at } y_j \\ X, \text{no fault is } \textit{detectable} \text{ by } T_i \text{ at } y_j \end{cases}$$

If both types of stuck-at faults are detectable somewhere by a set of test patterns, then $\delta$=D in this place.

Denote a set of stuck-at faults detectable by a test pattern $T_i$ at an output $y_j$ as $F(T_i, y_j)$, then let

$$F(T_i) = \bigcup_{y_j \in Y} F(T_i, y_j)$$

be a set of faults detectable by a test pattern $T_i$, and

$$F(T) = \bigcup_{T_i \in T} F(T_i)$$

be a set of faults detectable by all the test patterns $T_i \in T$. A test T is complete iff F(T)=F. The method already proposed by us in [18, 19] was based on an assumption that a test is complete. This paper presents a modification of the previous method adapted to incomplete tests.

Definition 7. *Set of failing test patterns*. Let $E = \{ T_i \mid \exists j: T_i \rightarrow y_j(T_i) \neq w_j(T_i) \} \subseteq T$ be a set of failing test patterns.

## 3. MAPPING STUCK-AT FAULTS INTO DESIGN ERROR MODEL

The stuck-at fault model does not have a physical meaning in this paper. It is only used to produce, as in the case of traditional testing, a diagnosis in terms of stuck-at faults, which is then mapped into design error domain. The method of mapping follows from the proof of the following theorem given in [16] and [17].

Theorem 1. To detect a design error in the implementation at an arbitrary gate $g_k$ where $s_k = g_k(s_1, s_2, ..., s_h)$, it is sufficient to apply a pair of test patterns which detect the stuck-at faults $s_i$ /1 and $s_i$ /0 at one of the gate inputs $s_i$, i = 1,2, ... h.

From the proof the following set of corollaries was driven which describes the mapping from a stuck-at fault set to the design error model domain:

- localizing both the s/1 and s/0 faults on two or more gate inputs refers to the missing/extra inverter at the gate output, i.e. to the replacement errors: AND $\leftrightarrow$ NAND and OR $\leftrightarrow$ NOR;
- localizing s/1 faults at one or more gate inputs refers to the replacement errors: AND $\rightarrow$ OR, OR $\rightarrow$ NAND, NAND $\rightarrow$ NOR, and NOR $\rightarrow$ AND;
- localizing s/0 faults at one or more gate inputs refers to the replacement errors: AND $\rightarrow$ NOR, OR $\rightarrow$ AND, NAND $\rightarrow$ OR, and NOR $\rightarrow$ NAND;
- localizing both the s/1 and s/0 faults at one of the gate inputs $s_i$ refers to the error $s_i \rightarrow$ NOT($s_i$) at this input;
- localizing both the s-1 and s-0 faults at more than one branch of a primary input $s_i \in X^F$ refers to the error $s_i \rightarrow$ NOT($s_i$) at this input.

The mapping is summarized in Table 1. Suspected erroneous gates in implementation are given in the first column. Next several columns refer to stuck-at faults detected at the gate inputs $s_1$, $s_2$, …, $s_h$. The last column represents the correction needed to rectify the circuit.

| Gate | Stuck-at faults | | | | | | | Correction |
|---|---|---|---|---|---|---|---|---|
| | $s_1$ | | $s_2$ | | … | $s_h$ | | |
| AND | 0 | 1 | 0 | 1 | … | 0 | 1 | NAND |
| | | 1 | | 1 | … | | 1 | OR |
| | 0 | | 0 | | … | 0 | | NOR |
| | 0 | 1 | | | … | | | NOT($x_1$) |
| | | | 0 | 1 | … | | | NOT($x_2$) |
| | | | | | … | 0 | 1 | NOT($x_h$) |
| OR | 0 | 1 | 0 | 1 | … | 0 | 1 | NOR |
| | 0 | | 0 | | … | 0 | | AND |
| | | 1 | | 1 | … | | 1 | NAND |
| | 0 | 1 | | | … | | | NOT($x_1$) |
| | | | 0 | 1 | … | | | NOT($x_2$) |
| | | | | | … | 0 | 1 | NOT($x_h$) |
| NAND | 0 | 1 | 0 | 1 | … | 0 | 1 | AND |
| | 0 | | 0 | | … | 0 | | OR |
| | | 1 | | 1 | … | | 1 | NOR |
| | 0 | 1 | | | … | | | NOT($x_1$) |
| | | | 0 | 1 | … | | | NOT($x_2$) |
| | | | | | … | 0 | 1 | NOT($x_h$) |
| NOR | 0 | 1 | 0 | 1 | … | 0 | 1 | OR |
| | | 1 | | 1 | … | | 1 | AND |
| | 0 | | 0 | | … | 0 | | NAND |
| | 0 | 1 | | | … | | | NOT($x_1$) |
| | | | 0 | 1 | … | | | NOT($x_2$) |
| | | | | | … | 0 | 1 | NOT($x_h$) |

*TABLE 1.* ***Mapping stuck-at faults into design error domain***

## 4. MACROS, PATHS, AND STUCK-AT FAULT MODELING

The following fault modeling method was developed for macro-level test generation based on usage of structurally synthesized BDDs (SSBDD) as the model for tree-like subcircuits or macros [20, 21, 22]. Every macro is represented by its own SSBDD.

Definition 8. *Signal paths*. We denote $L(s_k^j)$ as a set of variables on a path from the input of the macro $s_k^j \in S_k$ to its output $s_k$.

As macros are trees, there exists a one-to-one correspondence between inputs $s_k^j \in S_k$ and the gate-level signal paths $L(s_k^j)$ in the macro. The literal $s_k^j$ in the EPF is an inverted (not inverted) variable if the number of inverters on the path from $s_k^j$ to $s_k$ is odd (even).

Definition 9. *SSBDD*. A SSBDD is a directed noncyclic graph $G_k$ with a single root node and a set of nonterminal nodes $M_k$ labeled by (inverted/not inverted) Boolean variables (arguments of the function or inputs of the tree-like subcircuit). Every node has exactly two successor-nodes, whereas terminal nodes are labeled by constants 0 or 1. The set $M_k$ represents a macro $f_k$ so that one-to-one correspondence exists between the nodes $m \in M_k$ and signal paths L(s) where $s \in S_k$. Let s(m) denote the literal at the node m in the graph $G_k$.

We also use another definition of signal paths throughout the paper: $L(m_k) = (g_k^1, g_k^2, g_k^3, \ldots, g_k^r)$ is a path corresponding to the node $m_k$ where $g_k^j$ is a gate on the path and there is a relation of order $R(g_k^a, g_k^b)$, a,b∈[1,r] where from a<b results that $g_k^b$ stands closer to the output of the macro than $g_k^a$.

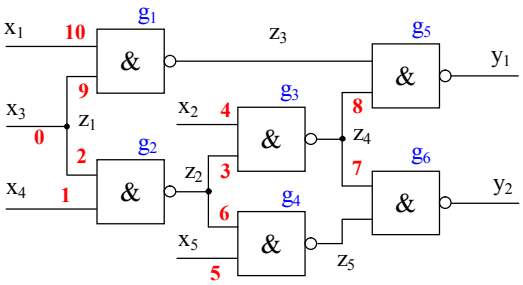



*Fig. 4.* ***Combinational circuit***

Example 1. Consider the combinational circuit from Fig. 4. It is a small circuit "c17" of ISCAS'85 benchmarks [23]. So, by previous definitions, $X=\{x_1, x_2, x_3, x_4, x_5\}$ is a set of input variables, $Y=\{y_1, y_2\}$ is a set of outputs, $Z=\{z_1, z_2, z_3, z_4, z_5\}$ represents internal variables, and $NG=\{g_1, g_2, g_3, g_4, g_5, g_6\}$ is a set of gates. The nodes of all SSBDDs are denoted in Fig. 4 by numbers 0 to 10 for nodes from $m_0$ to $m_{10}$ correspondingly. The procedure of formal synthesis of SSBDDs from gate-level networks is based on a graph superposition and described in detail in [20, 22]. In Fig. 5, a short example of macro-level SSBDD construction procedure is given. The example refers to the macro consisting of $g_4$, and $g_6$ gates.

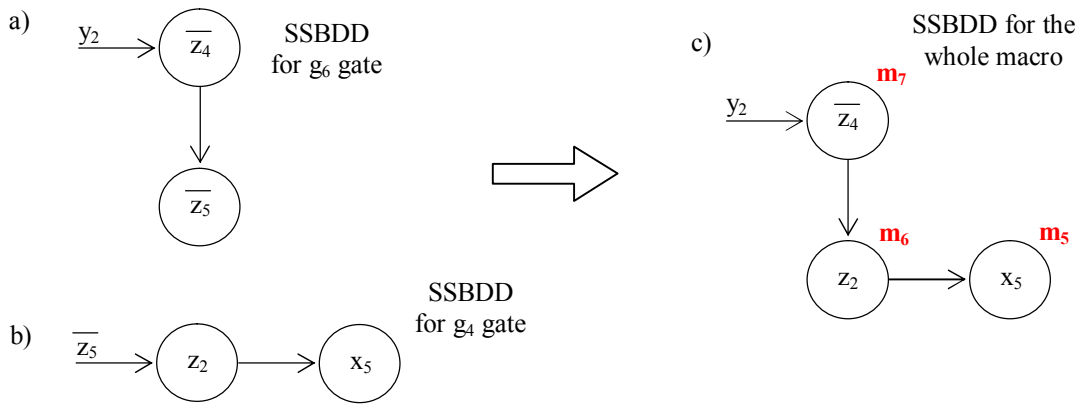



*Fig. 5.* ***Macro-level SSBDD construction***

Consider the SSBDD corresponding to internal variable $\bar{z}_5$ (Fig. 5(b)) where $z_2$ and $x_5$ are variables in nonterminal nodes, and $\bar{z}_5$ is the root node. Terminal nodes labeled by constants 0 and 1 are not shown. However, if one goes down from $z_2$ or $x_5$, one gets to the node labeled "0" that means $\bar{z}_5 = 0$. If one goes right from $x_5$, one gets to the node labeled "1" ($\bar{z}_5 = 1$). If the variable in a nonterminal node equals 0, one goes down. If it is 1, one goes to the right. For example, if $z_2$=1 and $x_5$=0, one gets to the terminal node labeled "0" ($\bar{z}_5 = 0$).

Such SSBDDs are constructed for each gate in the macro (Fig. 5). Then the superposition starts from the macro output and ends when the inputs of the macro are reached. In Fig. 5, it is shown that the node corresponding to internal variable $\bar{z}_5$ in Fig. 5(a) is substituted by the SSBDD corresponding to $\bar{z}_5$ (Fig. 5(b)) to form the SSBDD for the whole macro (Fig. 5(c)).

The following table shows how the nodes $m_k$ in SSBDD macros are related to gate-level paths $L(m_k)$ in the circuit.

| Macro | 1 | 2 | | 3 | | 4 | | | 5 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $m_k$ | $m_0$ | $m_1$ | $m_2$ | $m_3$ | $m_4$ | $m_5$ | $m_6$ | $m_7$ | $m_8$ | $m_9$ | $m_{10}$ |
| $L(m_k)$ | $\varnothing$ | $g_2$ | $g_2$ | $g_3$ | $g_3$ | $g_4,g_6$ | $g_4,g_6$ | $g_6$ | $g_5$ | $g_1,g_5$ | $g_1,g_5$ |

*TABLE 2.* ***SSBDD nodes and corresponding gate paths***

Definition 10. *Faults on signal paths, fault class*. Let $F(s_k^j/e)$ where $e \in \{0,1\}$ be a set of all faults (a fault class) in the gate-level signal path from the input $s_k^j$ of the macro $S_k$ to its output $s_k$.

A fault $s_k^j/e$ can be regarded as the representative of a fault class $F(s_k^j/e)$, since it is enough to test only the fault $s_k^j/e$ to test all the faults $F(s_k^j/e)$.

If a fault $\delta(s(m_k))$ is detected in an SSBDD node $m_k$ then a class of detected faults $F(\delta(s(m_k)))$ results at the gate level.

## 5. FAULT TABLES AND VECTORS

| $T_i$ | $s(m_k)$ | | | | | E |
|---|---|---|---|---|---|---|
| | $s(m_1)$ | $s(m_2)$ | $s(m_3)$ | ... | $s(m_q)$ | |
| $T_1$ | $\delta_{11}$ | $\delta_{12}$ | $\delta_{13}$ | ... | $\delta_{1q}$ | $\varepsilon_1$ |
| $T_2$ | $\delta_{21}$ | $\delta_{22}$ | $\delta_{23}$ | ... | $\delta_{2q}$ | $\varepsilon_2$ |
| $T_3$ | $\delta_{31}$ | $\delta_{32}$ | $\delta_{33}$ | ... | $\delta_{3q}$ | $\varepsilon_3$ |
| : | : | : | : | | : | : |
| $T_t$ | $\delta_{t1}$ | $\delta_{t2}$ | $\delta_{t3}$ | ... | $\delta_{tq}$ | $\varepsilon_t$ |

*TABLE 3.* ***A fault table***

Let us represent the set of detectable on the macro level faults F(T) as a *fault table* (Table 3) where row i corresponds to a test $T_i$ and column k corresponds to a variable

$s(m_k)$ in the node $m_k$ and $\delta_{ik} \in \{0,1,X\}$ shows what kind of fault is detectable at $m_k$ by a test pattern $T_i$.

Each row of this table ($\delta_{i1}$, $\delta_{i2}$, $\delta_{i3}$, … , $\delta_{iq}$) represents $F(T_i)$ – the set of faults detectable by a test pattern $T_i$. The last column (E) represents a set of failing test patterns, i.e. shows the results of a test experiment, whether a test detected an error or not.

Since we also need to know sets of faults detectable by a particular test pattern at each primary output $y_j$, we have to construct such fault tables for each output separately.

In this case (Fig. 6), each row ($\delta(T_i, y_j, s(m_1))$, $\delta(T_i, y_j, s(m_2))$, $\delta(T_i, y_j, s(m_3))$, … $\delta(T_i, y_j, s(m_q))$) of a table j represents $F(T_i, y_j)$ – the set of faults detectable by a test pattern $T_i$ at a primary output $y_j$.

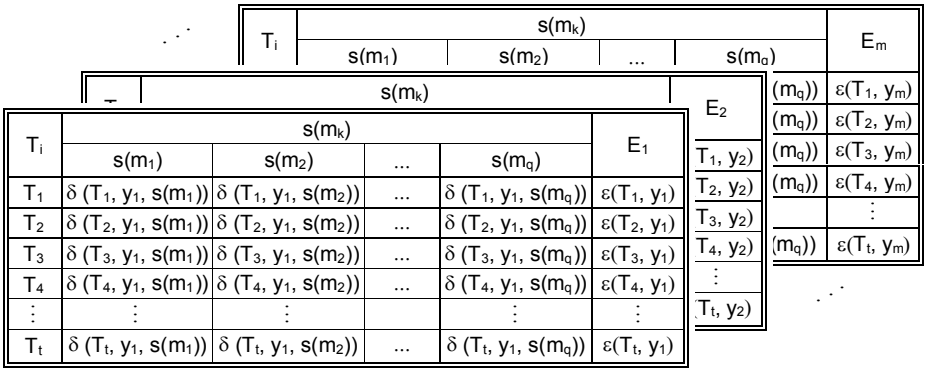



*Fig. 6.* ***Tables of faults detectable at each output***

The last column in each table shows whether a test pattern detected an error at a particular output, or not.

It is not hard to notice that each row of Table 3 as well as rows of tables in Fig. 6 can be represented in a vector form.

Definition 11. *Vector of detectable faults*. Let us denote a vector of faults detectable at the macro level by a test pattern $T_i$ as $V_M^d(T_i) = (\delta_{i1}, \delta_{i2}, \delta_{i3}, \dots, \delta_{iq})$ and let $V_M^d(T_i, y_j) = (\delta(T_i, y_j, s(m_1)), \delta(T_i, y_j, s(m_2)), \delta(T_i, y_j, s(m_3)), \dots \delta(T_i, y_j, s(m_q)))$ be a vector of faults detectable at the macro level at a particular output $y_j$ by a test pattern $T_i$.

Definition 12. At the macro level, let $V_M^c = (\delta(s(m_1)), \delta(s(m_2)), \delta(s(m_3)), \dots, \delta(s(m_q)))$ be a vector of faults guaranteed not to be present in a circuit on account of a set of failing test patterns E and a set of passed test patterns T-E. In other words, they are the faults that cannot be present in the circuit, all the other faults can be regarded to suspected or possible faults.

Let us define some operations with these vectors. We will need four types of operations: intersection ($\cap$), union ($\cup$), subtraction (–) and inversion ($\overline{\delta}$) (Fig. 7). All these operations are performed element-wise when applied to a pair of vectors.

Note that the definition of the inversion may seem a little bit confusing, but it becomes clear if we say that we use it propagating stuck-at faults along the signal path. If a gate on the path is inverting (NOT, NOR, NAND) then the detected stuck-at fault at the input of the gate is inverted at the output. If no faults are detected (X) at

the input, there is nothing to invert and no faults are detected at the gate output as well. Similarly, if both types of faults (D) are detected at the gate input, the separate inversions of each fault give us again both types of faults (D) detected at the output.

| $\delta_1$ \ $\delta_2$ | $\delta_1 \cap \delta_2$ | | | | $\delta_1 \cup \delta_2$ | | | | $\delta_1 - \delta_2$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | X | 0 | 1 | D | X | 0 | 1 | D | X | 0 | 1 | D |
| X | X | X | X | X | X | 0 | 1 | D | X | X | X | X |
| 0 | X | 0 | X | 0 | 0 | 0 | D | D | 0 | X | 0 | X |
| 1 | X | X | 1 | 1 | 1 | D | 1 | D | 1 | 1 | X | X |
| D | X | 0 | 1 | D | D | D | D | D | D | 1 | 0 | X |

| $\delta$ | $\overline{\delta}$ |
|---|---|
| X | X |
| 0 | 1 |
| 1 | 0 |
| D | D |

*Fig. 7.* ***Operations with fault vectors***

## 6. ALGORITHM DESCRIPTION

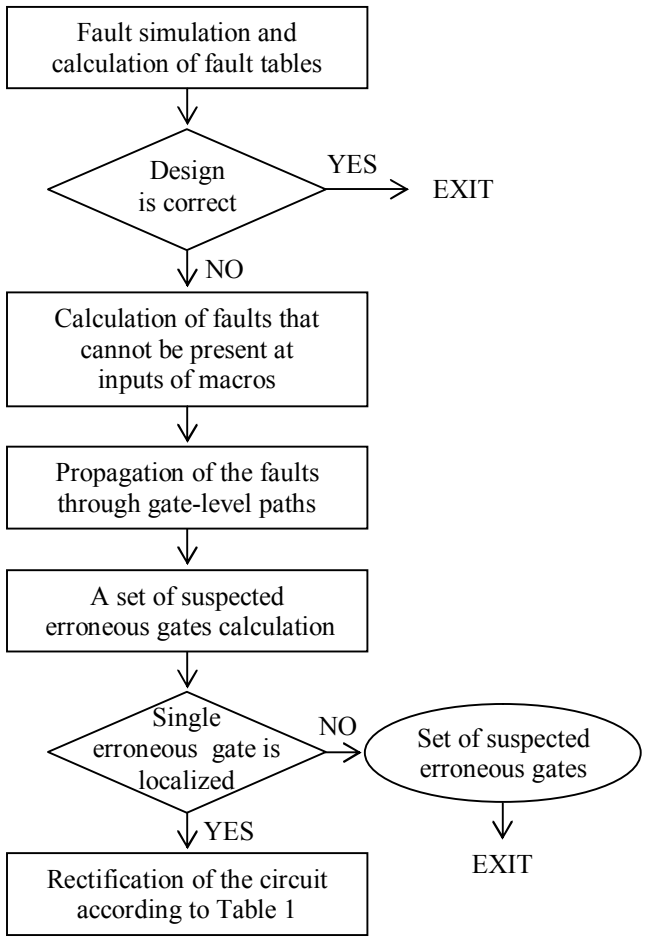



*Fig. 8.* ***Overall diagnostic algorithm***

The diagnostic procedure we described in [18, 19] is modified in this paper in order to adapt it to incomplete tests and to be applicable to circuits with redundancy. The

proposed design error diagnosis algorithm is implemented and works on two different levels of abstraction: macro (SSBDD) and gate level. The algorithm itself is very simple. This is due to underlying SSBDD circuit representation and techniques based on it. The stuck-at fault model is used during almost the whole diagnostic process instead of design error model. It helps to avoid complicated techniques for error localization. The overall algorithm is presented in Fig. 8.

Firstly, during the stage of verification through fault simulation, fault tables are constructed (Fig. 6) for each output. Then the macro-level diagnosis is done. It is based on the information which vector passed at which output, which not. The result represents a vector of faults guaranteed not to be present in a circuit. This vector is computed using the following formula:

$$V_M^c = \left( \bigcup_{T_i \in E} \left[ \bigcup_{y_j : \varepsilon(T_i, y_j)=1} V_M^d(T_i, y_j) - \bigcap_{y_j : \varepsilon(T_i, y_j)=1} V_M^d(T_i, y_j) \right] \right) \bigcup \left( \bigcup_{T_i \in E} \left[ \bigcup_{y_j : \varepsilon(T_i, y_j)=0} V_M^d(T_i, y_j) \right] \right) \bigcup \left( \bigcup_{T_i \in T-E} V_M^d(T_i) \right) \quad (1)$$

This formula represents a modification of Theorem 1 from [19]. In contrast to [19], we do not calculate a set of suspected faults first and then subtract a set of faults that guaranteed not be present in a circuit. We suspect everything first and calculate only a vector of faults guaranteed not to be present in the circuit. This allows to use incomplete tests for diagnosis and to include the faults not covered by the test into the suspected area.

Every test pattern can either detect an error or not, and if it does, it can detect the fault at one or more outputs. Due to this fact, the formula (1) consists of three main parts. The first part of the formula (1)

$$\bigcup_{T_i \in E} \left[ \bigcup_{y_j : \varepsilon(T_i, y_j)=1} V_M^d(T_i, y_j) - \bigcap_{y_j : \varepsilon(T_i, y_j)=1} V_M^d(T_i, y_j) \right]$$

refers to outputs where the error has been detected (faulty outputs) and calculates detectable by a faulty test vector $T_i \in E$ faults that cannot be present in a circuit, as they are not included in an intersection of detectable faults by faulty outputs:

$$\bigcap_{y_j : \varepsilon(T_i, y_j)=1} V_M^d(T_i, y_j)$$

This is due to the fact that a single design error can be only present in this intersection. The second part of (1)

$$\bigcup_{T_i \in E} \left[ \bigcup_{y_j : \varepsilon(T_i, y_j)=0} V_M^d(T_i, y_j) \right]$$

addresses the faulty vectors but correct outputs. It is a union of faults that cannot be present in a circuit, as they are detectable by a faulty vector $T_i \in E$ at a primary output $y_j$ while this output is correct. The last portion of (1)

$$\bigcup_{T_i \in T-E} V_M^d(T_i)$$

is a union of all faults detectable by all test patterns $T_i \in T\text{-}E$ that has not detected an error at all.

The total union of these three parts is a set of all faults that cannot be present in a circuit. It means that the real error (or a combination of stuck-at faults corresponding to the real error) is located somewhere in the area that is a complement of $V_M^c$ to the whole stuck-at fault set F. The size of this remaining area depends on diagnostic properties of test patterns, fault coverage, and presence of redundancy in a circuit.

The macro-level diagnosis is complete at this step. The final result on this stage is a vector $V_M^c$ where each element $\delta(s(m_k))$ shows what kind of fault cannot be present in an SSBDD node $m_k$.

From this step gate-level diagnosis for localization of erroneous gate(s) begins. Since every node at the macro level corresponds to a certain path (a set of gates) $L(m_k)=(g_k^1, g_k^2, g_k^3, ..., g_k^r)$ on the gate level (see Table 2), we have to propagate a fault in the node through all the gates in the path inverting it each time we meet an inverting gate (NOT, NOR, or NAND).

Algorithm 1.

Let $s(m_k)$ be a variable in a node $m_k$, $L(m_k) = (g_k^1, g_k^2, g_k^3, ..., g_k^r)$ be a path corresponding to the node $m_k$, $\delta(s(m_k))$ be a fault in the SSBDD node $m_k$, and $\delta(g_k^j)$ be a fault at the output of a gate $g_k^j$. We propagate a fault through the path as follows:

1. $$\delta(g_k^1) = \begin{cases} \delta(s(m_k)), \text{if the gate } g_k^1 \text{ is either OR or AND} \\ \overline{\delta}(s(m_k)), \text{if the gate } g_k^1 \text{ is either NOT, NOR or NAND} \end{cases}$$

2. $$\delta(g_k^j) = \begin{cases} \delta(g_k^{j-1}), \text{if the gate } g_k^j \text{ is either OR or AND} \\ \overline{\delta}(g_k^{j-1}), \text{if the gate } g_k^j \text{ is either NOT, NOR or NAND} \end{cases}$$

3. Repeat step 2 while $2 \le j \le r$ ν

Using this algorithm we construct vectors of faults $V_G^c(m_k) = (\delta(g_1), \delta(g_2), \delta(g_3), \ldots, \delta(g_p))$ for each path $L(m_k)$, where $\delta(g_i) = \{X,0,1,D\}$ is a fault that is guaranteed not to be present at the output of a gate $g_i \in NG$ and p is a number of gates in the whole circuit. Every $\delta(g_i)$ from $L(m_k)$ is calculated by the following rule:

$$\delta(g_i) = \begin{cases} X, \text{if } g_i \notin L(m_k) \\ \text{calculate by Algorithm 1, if } g_i \in L(m_k) \end{cases}$$

A union of all these vectors $V_G^c(m_k)$ gives us a vector of all stuck-at faults that cannot be present in a circuit:

$$V_G^c = \bigcup_{m_k \in M_k} V_G^c(m_k)$$

It means that the combination of stuck-at faults caused by a real design error is located among all other stuck-at faults (not covered by $V_G^c$).

Let $V_G^s = (\delta(g_1), \delta(g_2), \delta(g_3), \ldots, \delta(g_p))$ is a vector of *suspected* faults at gate outputs for every gate $g_i \in NG$ in a circuit, where $\delta(g_i) = \{X,0,1,D\}$ is the fault suspected at the

output of a gate $g_i$. This vector calculation will be the final step of the gate-level diagnosis before stuck-at-fault-to-design-error mapping.

First of all, we initialize the vector $V_G^s$ by placing $\delta(g_i) = D$ in every position of $V_G^s$. It means that we initially suspect every gate to be erroneous and therefore any combination of stuck-at faults is suspected at the output of every gate in a circuit. After that we update $V_G^s$ subtracting $V_G^c$ from it.

Now we know all stuck-at faults suspected at gate outputs and at inputs of every macro. Thus, we know combinations of suspected stuck-at faults at gate inputs and we can refer to Table 1 to determine how to rectify the circuit. However, if the set of erroneous gates contains more than one gate (the error has not been located exactly), the mapping cannot be performed.

## 7. EXAMPLE

Consider a test with 5 patterns that is applied to the inputs of the circuit in Fig. 4. Suppose now that test patterns $T_1$ and $T_5$ fail at both outputs that results in $E=\{T_1,T_5\}$ and $\varepsilon(T_1, y_1)= \varepsilon(T_1, y_2)= \varepsilon(T_5, y_1)= \varepsilon(T_5, y_2)=1$ while other $\varepsilon(T_i, y_j)=0$. The vectors of detectable faults $V_M^d(T_i, y_j)$ and error detection information for such a case are presented in the following fault tables (Table 4,5) for outputs $y_1$ and $y_2$ correspondingly.

| $T_i$ | $s(m_k)$ | | | | | | | | | | | $E_1$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $m_0$ | $m_1$ | $m_2$ | $m_3$ | $m_4$ | $m_5$ | $m_6$ | $m_7$ | $m_8$ | $m_9$ | $m_{10}$ | |
| $T_1$ | 0 | 0 | 0 | 1 | X | X | X | X | 0 | X | 1 | 1 |
| $T_2$ | 1 | X | X | X | 1 | X | X | X | 0 | 1 | X | 0 |
| $T_3$ | 0 | X | X | X | X | X | X | X | X | 0 | 0 | 0 |
| $T_4$ | 1 | X | 1 | 0 | 0 | X | X | X | 1 | X | X | 0 |
| $T_5$ | X | X | X | 0 | 0 | X | X | X | 1 | X | X | 1 |

*TABLE 4.* ***Fault table for $y_1$***

| $T_i$ | $s(m_k)$ | | | | | | | | | | | $E_2$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $m_0$ | $m_1$ | $m_2$ | $m_3$ | $m_4$ | $m_5$ | $m_6$ | $m_7$ | $m_8$ | $m_9$ | $m_{10}$ | |
| $T_1$ | 0 | 0 | 0 | 1 | X | X | 1 | 0 | X | X | X | 1 |
| $T_2$ | X | X | X | X | 1 | 1 | X | 0 | X | X | X | 0 |
| $T_3$ | X | 1 | X | X | X | 0 | 0 | X | X | X | X | 0 |
| $T_4$ | 1 | X | 1 | X | X | X | X | X | X | X | X | 0 |
| $T_5$ | X | X | X | 0 | 0 | X | X | 1 | X | X | X | 1 |

*TABLE 5.* ***Fault table for $y_2$***

From that, according to (1), we create a vector of faults that cannot be present in the circuit on the macro level (macro-level diagnosis). First, compute the part of (1) concerning fault detection information by outputs for each faulty vector:

$$\bigcup_{y_j:\varepsilon(T_1,y_j)=1} V_M^d(T_1,y_j)=(0,0,0,1,X,X,1,0,0,X,1)$$

$$\bigcap_{y_j:\varepsilon(T_1,y_j)=1} V_M^d(T_1,y_j)=(0,0,0,1,X,X,X,X,X,X,X)$$

$$\bigcup_{y_j:\varepsilon(T_1,y_j)=1} V_M^d(T_1,y_j) - \bigcap_{y_j:\varepsilon(T_1,y_j)=1} V_M^d(T_1,y_j)= (X,X,X,X,X,X,1,0,0,X,1)$$

$$\bigcup_{y_j:\varepsilon(T_5,y_j)=1} V_M^d(T_5,y_j)=(X,X,X,0,0,X,X,1,1,X,X)$$

$$\bigcap_{y_j:\varepsilon(T_5,y_j)=1} V_M^d(T_5,y_j)=(X,X,X,0,0,X,X,X,X,X,X)$$

$$\bigcup_{y_j:\varepsilon(T_5,y_j)=1} V_M^d(T_5,y_j) - \bigcap_{y_j:\varepsilon(T_5,y_j)=1} V_M^d(T_5,y_j)= (X,X,X,X,X,X,X,1,1,X,X)$$

$$\bigcup_{T_i\in E}\left[\bigcup_{y_j:\varepsilon(T_i,y_j)=1} V_M^d(T_i,y_j) - \bigcap_{y_j:\varepsilon(T_i,y_j)=1} V_M^d(T_i,y_j)\right]=(X,X,X,X,X,X,1,D,D,X,1)$$

The second part of (1) cannot be calculated because $T_1$ and $T_2$ vectors failed at each output. However, the third part of (1) can be successfully calculated as a union of all faults detectable by vectors corresponding to test patterns that have not detected an error:

$$\bigcup_{T_i\in T-E} V_M^d(T_i)= (D,1,1,0,D,D,0,0,D,D,0)$$

The total union gives us $V_M^c$=(D,1,1,0,D,D,D,D,D,D,D). It provides information that both stuck-at 0 and stuck-at 1 cannot be present in node $m_0$, stuck-at 1 cannot be present in $m_1$ but stuck-at 0 is possible, and so on.

We will turn to gate-level diagnosis now. The best is the case when an exactly located erroneous gate is the result of the diagnosis, and there exists a combination of suspected stuck-at faults at the gate inputs that clearly defines the type of detected design error (Table 1). However, sometimes the final result represents just a set of suspected erroneous gates with some suspected stuck-at fault combinations at their inputs. The explanation of possible reasons for that is given in the next chapters.

At the gate-level diagnosis stage we use Algoritm 1 and Table 2 to find a vector $V_G^c(m_k)$ for every path $L(m_k)$. The union of them is:

$$V_G^c=(D,0,D,D,D,D)$$

Subtracting it from initial $V_G^s$=(D,D,D,D,D,D) we have

$$V_G^s=(X,1,X,X,X,X)$$

It means that no stuck-at fault is suspected at $g_1$, $g_3$, $g_4$, $g_5$, and $g_6$ outputs and these gates are correct, and stuck-at 1 is suspected at $g_2$ output. As stuck-at 0 faults are suspected at nodes $m_1$ and $m_2$ that are inputs of the gate $g_2$, then according to Table 1 (the case of NAND gate) it refers to the design error NAND → OR. To correct the design, the NAND gate $g_2$ should be replaced by an OR gate.

## 8. LIMITATIONS AND IMPROVEMENT OF THE METHOD

The first thing that can be noticeable is that XOR and XNOR gate substitutions are not included in the error model. It is because some of such cases may not be detected at all even if we have 100% fault coverage. Theorem 1 says that it is sufficient to apply a pair of test patterns detecting stuck-at faults $s_i$ /1 and $s_i$ /0 at one of the gate inputs to detect a design error in the implementation at an arbitrary gate. This statement is not applicable to XOR and XNOR gates. However, for some cases, we can extend the mapping table (Table 1) for XOR and XNOR gates, too.

Consider an OR gate $g_k$ where $s_1$, $s_2$,...,$s_h$ are its inputs and $s_k$ is its output. Let $T_{OR,0}$ = $\{s_i=1, \forall j, j\neq i: s_j=0\}$ be a test pattern used to test $s_i$ /0 fault and $T_{OR,1}$ = $\{\forall i, i=1,2,\ldots,h: s_i=0\}$ – to test $s_i$ /1 fault. Suppose there should be a XNOR gate in implementation instead of the OR gate $g_k$. In such case, both suck-at faults will be detected at $s_i$ and, similarly, at each gate input. Such situation is referred to OR→NOR substitution in Table 1. So, extending our gate substitution model into XNOR gate cases makes it impossible to detect the exact substitution type. However, it gives us an opportunity to try several possible substitutions to identify the correct one. The worse case is when we can't detect a substitution error at all.

Consider the same OR gate and the same test vectors $T_{OR,0}$ and $T_{OR,1}$. Suppose now the correct gate is XOR. These test vectors can't discover any difference between the two gates. This means that the error will not be detected at all or it will be detected occasionally by some other test pattern that is generated to detect some other stuck-at faults. This leads to a situation where wrong place will be suspected to be erroneous.

Consider an AND gate now. The test pattern pair for AND gate is $T_{AND,0}$ = $\{\forall i, i=1,2,\ldots,h: s_i=1\}$ to test $s_i$ /0 fault and $T_{AND,1}$ = $\{ s_i=0, \forall j, j\neq i: s_j=1\}$ to test $s_i$ /1 faults. Both vectors detect AND→XOR error if *h* is even or AND→XNOR if *h* is odd. However, AND→XOR error will not be detected at all if *h* is odd and AND→XNOR if *h* is even. Similar examples can be constructed for NOR and NAND gates too.

The situation described above restricts the substitution model of our method to four types of simple gates (AND, OR, NAND, NOR). However, it can be extended to some cases of XOR/XNOR gate substitution (Table 6).

| Gate | Stuck-at faults | | | | | | | Correction |
|---|---|---|---|---|---|---|---|---|
| | $s_1$ | | $s_2$ | | … | $s_h$ | | |
| AND | 0 | 1 | 0 | 1 | … | 0 | 1 | even XOR, odd XNOR |
| OR | 0 | 1 | 0 | 1 | … | 0 | 1 | XNOR |
| NAND | 0 | 1 | 0 | 1 | … | 0 | 1 | odd XOR, even XNOR |
| NOR | 0 | 1 | 0 | 1 | … | 0 | 1 | XOR |

*TABLE 6.* ***Mapping table extension***

The cases opposite to discussed, namely when a gate is wrongly implemented as a XOR/XNOR, are more simple to detect and locate because every XOR/XNOR gate can be represented as a network of simpler gates and each simple gate can be tested separately. In [7], an example is given how a XOR gate replacement error can be rectified using only simple gate replacement model (Fig. 9). Thus, our model also covers a gate replacement by XOR/XNOR gates.

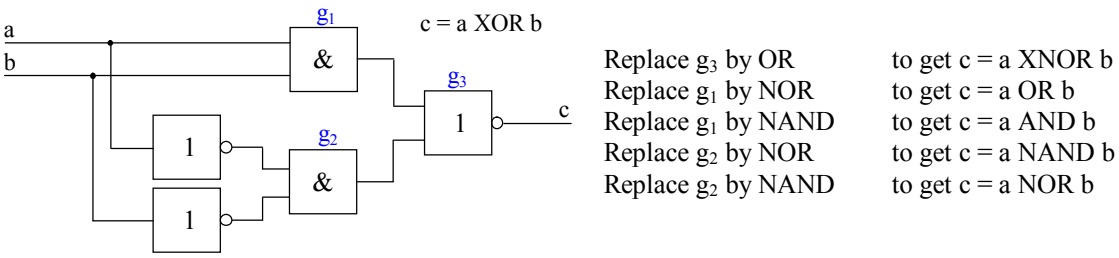
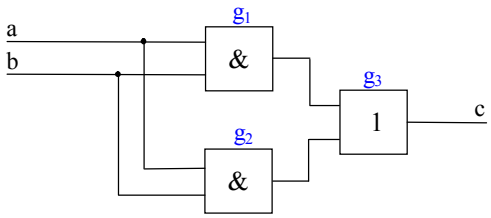



*Fig. 9.* ***Representation of a XOR gate [7]***

There is also a restriction on usage of our method if a design error causes redundancy. Consider a circuit in Fig. 10. Let the gate $g_2$ have NOR function but occasionally be substituted by an AND gate. The function of the whole circuit is AND now (c = a&b). The test for a/0 and b/0 is $T_{AND,0} = \{1,1\}$, for a/1 is $T_{AND,a1} = \{0,1\}$, and for b/1 is $T_{AND,b1} = \{1,0\}$. Table in Fig.10 shows that this complete test is not capable to detect the given gate substitution error. Such design errors can only be detected occasionally by the test patterns that are intended to test other design errors, as it was described for OR→XOR case.




| a | b | c=a&b | c=(a&b)∨(a∨b) |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 |

*Fig. 10.* ***Redundancy effect***

## 9. Results and Analysis of Experiments

The goals of the experiments described here were twofold:

- to compare the efficiency (the speed of fault localization) of the new diagnostic approach in comparison to previous results
- to evaluate the design error *diagnostic* properties of test patterns generated by traditional gate-level ATPGs for only stuck-at fault *detecting* purposes

Diagnostic experiments were carried out on ISCAS'85 benchmarks. Columns 2,3,4 in Table 7 give information about size of the benchmarks in terms of input, output and gate quantities. Note that the number of gates given in the table may be different from the real gate count because of XOR and XNOR gates representation by a number of simpler gates (Fig. 9).

Experiments were carried out in the following way. We took a netlist of an ISCAS'85 circuit and treated it as a wrong implementation. Then we generated SSBDD model

from it and created test patterns for detecting stuck-at faults. After that, using an error insertion tool, we created a "correct" specification and simulated it with the same test patterns to find the difference between output responses of the specification and the implementation. Using this information, the error analysis tool produced a diagnosis. Then, we took the initial "wrong" SSBDD again and created new "correct" specification with error insertion tool. After that the whole process was repeated. In other words, we had one "wrong" implementation and many "correct" specifications.

The fault coverage (column 6) of the test patterns created by the test generator described in [22] and the test generation time in seconds (column 11) are presented in Table 7. Experiments were carried out on the computer platform Sun SparcServer 20 (2 x Super Sparc II microprocessors, 75MHz) with Solaris 2.5.1 operating system.

The number of experiments carried out for each circuit are shown in column 5. For the cases marked by star (*), one random error for each gate per experiment was inserted. For the other cases, all possible single gate errors were simulated and analyzed – one error per experiment.

| Circuit Name | Number of | | | | Fault Cove-rage, % | Suspected Erroneous Gates | | | | Time, s | | | Time for [7], s |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Number | | | Av. % of Total | Test Gene-ration | Fault Analysis (average) | Total | |
| | In-puts | Out-puts | Gates | Experi-ments | | Min | Max | Av. | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| c432 | 36 | 7 | 232 | 671 | 93,02 | 3 | 92 | 11,3 | 4,86 | 0,62 | 0,06 | 0,7 | 17,57 |
| c499 | 41 | 32 | 618 | 1622 | 99,33 | 1 | 307 | 73,3 | 11,86 | 1,01 | 0,8 | 1,8 | 111,64 |
| c880 | 60 | 26 | 357 | 1144 | 100 | 1 | 33 | 5,7 | 1,60 | 0,19 | 0,3 | 0,5 | 126,79 |
| c1355 | 41 | 32 | 514 | 1830 | 99,51 | 1 | 248 | 55,2 | 10,74 | 1,35 | 0,9 | 2,3 | 241,79 |
| c1908 | 33 | 25 | 718 | 1922 | 99,31 | 3 | 70 | 12,3 | 1,71 | 0,93 | 1,0 | 1,9 | 341,92 |
| c2670 | 233 | 140 | 997 | 997* | 94,97 | 69 | 218 | 91,1 | 9,14 | 3,55 | 8,5 | 12,1 | 661,91 |
| c3540 | 50 | 22 | 1446 | 1446* | 95,27 | 107 | 190 | 115,7 | 8,00 | 3,08 | 2,3 | 5,4 | 1513,82 |
| c5315 | 178 | 123 | 1994 | 1994* | 98,69 | 6 | 204 | 15,7 | 0,79 | 2,38 | 17,4 | 19,8 | 1814,04 |
| c6288 | 32 | 32 | 2416 | 2416* | 99,34 | 17 | 92 | 21,5 | 0,89 | 2,17 | 1,7 | 3,9 | 1895,90 |
| c7552 | 207 | 108 | 2978 | 2978* | 95,95 | 131 | 372 | 144,3 | 4,84 | 12,06 | 26,8 | 38,9 | |

*TABLE 7.* ***Diagnostic results for ISCAS '85 benchmark circuits***

The efficiency in the speed of diagnosis (columns 11, 12, 13) is compared to the results of [7] (column 14). However, it should be noted that the algorithm from [7] is implemented in Prolog language that is comparatively slow. The total time of diagnosis (column 13) in this work consists of two components: test generation time (column 11) and fault diagnosis (column 12).

The numbers in columns 7, 8, and 9 show, correspondingly, the minimal, maximal, and average diagnostic resolutions (numbers of suspected gates) reached by the tests. However, it should be noted that the diagnostic resolution of this method is not excellent for some circuits. The possible reasons are:

- the test patterns used for diagnosis were not originally generated for diagnostic purposes but just to *detect* an error
- redundancy, present in several circuits

To reach higher diagnostic resolution, additional test patterns should be generated. For this purpose, for example, the method of [7] can be used. Since the suspected area (column 10) for diagnostic search is reduced from 100% to from 0,79% (in the best

case) to 11,86% (in the worst case), the combination of the method proposed in the present work with some other method of additional diagnostic patterns generation, should reach significant improvements.

## 10. CONCLUSIONS

The method described in this paper has the following distinct features:

- The whole procedure takes place hierarchically at two different levels:
  - macro level, where the error detection and localization of the suspected erroneous SSBDD nodes are carried out
  - gate level for erroneous gate localization and exact specification of the design error

  Exploiting the hierarchy allows to combine the efficiency of working at the higher level (for error detecting) with the accuracy (needed for error diagnosis) at the lower level.
- Working with the stuck-at fault model allows to base on a single error hypothesis, which actually means working with several error hypothesis from design error model [13] in parallel.
- Compared to our previous work [18, 19] the method is extended in order to be capable to provide a solution even if the test is not complete and/or a portion of redundancy is present in a circuit.

The method has good chances for further improvement due to its high efficiency in speed and, therefore, the possibility of use in combination with some other diagnostic test generation and rectification techniques.

Experimental data is provided to demonstrate the efficiency of the method.

Our future research in this field is directed to improvement of the diagnostic resolution of the method. For this purpose additional test pattern generation heuristics can be worked out or several existing approaches can be used. It is also necessary to pay attention to the cases of multiple errors, complex gates, and line errors. The use of word level DDs seems to be very efficient in design error diagnosis at higher functional levels like RTL or behavioral ones.

**Acknowledgements** – This work has been supported by the COPERNICUS project No 977133 VILAB and by the Estonian Science Foundation grant G-1850. Authors appreciate the work of Jaan Raik for helping to carry out the experimental work.

## REFERENCES

1. Veneris A, Venkataraman S, Hajj IN, Fuchs WK. Multiple Design Error Diagnosis and Correction in Digital VLSI Circuits. In: Proc. IEEE VLSI Test Symposium, April 1999, pp. 58B63.
2. Hoffmann DW, Kropf T. Using BDD-based Decomposition for Automatic Error Correction in Combinatorial Circuits. Technical Report, University of Karlsruhe, available at http://goethe.ira.uka.de/~hoff.

3. Huang SY, Cheng KT, Chen KC, Cheng DI. ErrorTracer: A Fault Simulation Based Approach to Design Error Diagnosis. In: Proc. International Test Conference, November 1997, pp. 974B981.
4. Huang SY, Chen KC, Cheng KT. Incremental Logic Rectification. In: Proc. VLSI Test Symposium, April 1997, pp. 134B139.
5. Wahba A, Borrione D. Connection errors location and correction in combinational circuits. In: Proc. European Design and Test Conference, ED&TC-97, Paris, France, March 1997.
6. Huang SY, Chen KC, Cheng KT. Error Correction Based on Verification Techniques. In: Proc. Design Automation Conference, June 1996, pp. 258B261.
7. Wahba A, Borrione D. A Method for Automatic Design Error Location and Correction in Combinational Logic Circuits. Journal of Electronic Testing: Theory and Applications 1996;8:113B127.
8. Lin CC, Chen KC, Chang SC, Marek-Sadowska M, Cheng KT. Logic Synthesis for Engineering Change. In: Proc. Design Automation Conference, June 1995, pp. 647B652.
9. Chung PY, Wang YM, Hajj IN. Logic design error diagnosis and correction. IEEE Transactions on VLSI Systems 1994;3:320B332.
10. Tomita M, Yamamoto T, Sumikawa F, Hirano K. Rectification of Multiple Logic Design Errors in Multiple Output Circuits. In: Proc. 31st Design Automation Conference, 1994, pp. 212B217.
11. Tamura KA. Locating Functional Errors in Logic Circuits. In: Proc. 26th Design Automation Conference, June 1989, pp. 185B191.
12. Madre JC, Coudert O, Billon JP. Automating the Diagnosis and the Rectification of Design Errors with PRIAM. In: Proc. ICCAD'89, 1989, pp. 30B33.
13. Abadir MS, Ferguson J, Kirkland TE. Logic Design Verification via Test Generation. IEEE Transactions on Computers, January 1988, pp. 138B148.
14. Bryant RE. Graph-based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers, 1986;C-35:667B691.
15. Brand D, Drumm A, Kundu S, Narrain P. Incremental Synthesis. In: Proc. International Conference on Computer Aided Design, 1994, pp. 14B18.
16. Ubar R, Borrione D. Localization of Single Gate Design Errors in Combinational Circuits by Diagnostic Information about Stuck-at Faults. In: Proc. 2nd Int. Workshop on Design and Diagnostics of Electronic Circuits and Systems, Szczyrk, Poland, September 1998, pp. 73B79.
17. Ubar R, Borrione D. Generation of Tests for Localization of Single Gate Design Errors in Combinational Circuits Using the Stuck-at Fault Model. In: Proc. 11th IEEE Brazilian Symposium on IC Design, Rio de Janeiro, Brazil, Sept. October 1998, pp. 51B54.
18. Ubar R, Jutman A. Hierarchical Design Error Diagnosis in Combinational Circuits by Stuck-at Fault Test Patterns. In: Proc. 6th International Conference Mixed Design of Integrated Circuits and Systems, Kraków, June 1999, pp. 437B442.

19. Jutman A, Ubar R. Design Error Localization in Digital Circuits by Stuck-At Fault Test Patterns. In: Proc. 22nd International Conference on Microelectronics, 19-22 Sept. 1999 (to appear).
20. Ubar R. Test Synthesis with Alternative Graphs. IEEE Design and Test of Computers, Spring, 1996, pp. 48B59.
21. Ubar R, Raik J. Multi-Valued Simulation with Binary Decision Diagrams. In: Proc. IEEE European Test Workshop, Cagliari, May 1997.
22. Raik J, Ubar R. Feasibility of Structurally Synthesized BDD Models for Test Generation. Proc. IEEE European Test Workshop, Barcelona, May 1998, pp.145B146.
23. Brglez F, Fujiwara H. A neutral netlist of 10 combinatorial benchmark circuits and a target translator in FORTRAN. In: International Symposium on Circuits and Systems, Special Session on ATPG and Fault Simulation, 1985.

# Paper 5

A. Jutman, “At-Speed On-Chip Diagnosis of Board-Level Interconnect Faults”, in *Formal Proc. of 9th European Test Symposium (ETS’04)*, Corsica, France, May 23-26, 2004

# At-Speed On-Chip Diagnosis of Board-Level Interconnect Faults

Artur Jutman
*Tallinn University of Technology*
artur@pld.ttu.ee

## Abstract

*This article describes a novel approach to fault diagnosis suitable for at-speed testing of board-level interconnect faults. The approach is based on a new parallel test pattern generator and a specific fault detecting sequence. The test sequence has tree major advantages. At first, it detects both static and dynamic faults upon interconnects. Secondly, it allows precise on-chip at-speed fault diagnosis of interconnect faults. Third, the hardware implementation of both the test generator and the response analyzer is very efficient in terms of silicon area.*

## 1. Introduction

The problem of board level interconnect test has been thoroughly studied in the past. Most methodologies proposed during the last decade tend to rely on the Boundary Scan (BS) technique [1] as a powerful and universal test access mechanism. The related problem – the problem of Interconnect BIST (IBIST) is usually approached in the same way. The major difference between known methods of IBIST is usually defined by the manner they treat several important issues like:

- code used for test generation (e.g. Walking Sequence, Counting Sequence, True/Complement Code etc.)
- hardware implementation of both the test pattern generator (TPG) and the response analyzer (RA)
- covered defect types (fault models) - as a result of selected code and hardware implementation
- handling of the multiple driver contention problem

There are also some other important issues like the power consumption level or the ground bounce problem [2].

The widely accepted interconnect fault model usually handles static defects like opens and shorts. The literature, however, agrees on the fact that the dynamic defects like transition (delay) faults, crosstalk, or switching noise are becoming very important due to higher clock speeds and lower supply voltages [3]. Unfortunately, these latter effects do not show up in practice at the testing frequencies or the frequencies the Boundary Scan operates on. The nature of this drawback originates from the fact that the Boundary Scan architecture was not designed for at-speed testing and diagnosis of timing-related faults. For instance, the interval between the update of test stimulus and capture of the response data spans at least 2,5 test clock cycles. Another drawback, which also results from the Boundary Scan nature, is the need for a long shift operation between two consecutive test vectors. This shift operation depends on the BS chain length and may take hundreds of clock cycles to complete. These two facts together limit or totally prevent efficient testing of timing-related defects by standard interconnect test methods. Therefore, the conception of *at-speed* interconnect test and BIST has become very important lately [4], [5].

The simplest way to approach the problem is to place a standard LFSR (Linear Feedback Shift Register) as a test stimuli generator on one side of interconnect wires and a MISR (Multiple Input Shift Register) as a response compactor on the other side. This approach has been successfully adopted by the industry due to its simplicity. However, such a solution cannot be easily standardized as, for example, the BS due to the following reasons. First, the LFSR as the TPG is not easily scalable since its efficiency depends on characteristic polynomials, which should be selected separately for each board configuration. The same hold for the MISR which fault signature also depends on the configuration. Secondly, there is the aliasing probability for MISR signatures, which might prevent fault detection. It may also reduce the fault diagnosis resolution of the method. Third, there is always a question of how many pseudo-random test patterns would be enough to detect all possible static and timing-related faults. Usually, the LFSR is set for generation of several thousands of patterns, which is usually considered as a good enough test length. Hence, this solution is not the most efficient one in terms of test application time.

Another direction of research in this area is focused on modification of the standard BS cells so that the resulting architecture is capable of sending the test vector and capturing the response within a single clock cycle [6]. There are approaches for single as well as for multiple clock schemes. Such a solution complicates the test application mechanism of BS and brings additional silicon area. It should also be pointed out, that the test data is shifted serially into the BS chain. At first, the desired vector is shifted in, then it is applied to the nets and

finally the response is shifted out and analyzed. This is repeated for each pattern and requires at least $T \bullet L$ clock cycles, where $T$ is the test length and $L$ is the length of the BS chain. At first, $L$ is a large number usually ranging between $10^2$ and $10^3$, which results in very long test application times compared to the initial test length $T$. Secondly, this framework does not directly support the *at-speed testing* of interconnect.

The problem of long test application time has also been studied and some solutions have been proposed in literature [7]. The common idea, which bounds them, is the usage of limited shift operations between two patterns, that is, a one-bit shift in the best case. All of these solutions, however, suffer from the issue referred as a multiple driver contention problem when several drivers feed the same bus simultaneously with different values. This happens when test data, while being shifted, unintentionally activates conflicting drivers by setting up the corresponding values in control cells situated in the same scan chain and interleaved with data cells.

All the mentioned problems resulted in the fact that there are still very few methodologies, which could be effectively used for at-speed testing of timing-related interconnect faults [3].

This paper describes a novel approach to fault diagnosis suitable for at-speed testing of board-level interconnect faults. The approach is based on a new parallel test pattern generator and a specific fault detecting sequence called the *Interleaved True/Complement code*. The test sequence has tree major advantages. At first, it detects both static and dynamic faults upon interconnects. Secondly, it allows precise on-chip at-speed fault diagnosis of interconnect faults without any aliasing or masking. Third, the fault signature needs to be shifted out from the RA just once per test session. Finally, the hardware implementation of both the test generator and the response analyzer is very efficient in terms of silicon area.

The proposed framework also reduces the test application time from original $T \bullet L$ clock cycles to $T$ clock cycles, plus less than $L$ clock cycles needed for shifting out the final fault signature.

The next section of the article gives an overview of known test generation methods for interconnect testing and describes a new sequence for testing timing-related faults called the Interleaved True/Complement code. Section 3 gives an overview of widely accepted test application scheme for interconnect BIST and a new original hardware implementation of the TPG and the RA for at-speed IBIST. Conclusions are given in the end of the article.

## 2. Test generation algorithms

In interconnect test it is of common practice to consider the following defects: *a)* shorts between multiple nets and *b)* opens upon single or multiple nets. Shorts are usually modeled by wired-AND (logic 0 dominates) or wired-OR (logic 1 dominates) faults while opens are represented by stuck-at-1 or stuck-at-0 faults.

In 1974, Kautz showed that in order to test $N$ independent nets for all possible shorts one needs $\lceil \log_2(N) \rceil$ test patterns [8]. The main idea was to assign a unique code word to each net. Then, in case of a wired-OR (wired-AND) short between any two or more lines the number of 1s (0s) will be increased and the initial code will be distorted. The algorithm proposed by Kautz is called the *Counting Sequence* algorithm (Fig. 1a).

It has been shown later [9] that stuck-open faults could also be tested by this algorithm but then the all-1 and all-0 patterns should be avoided. This means that the counter should start from 1 rather than from 0 and it should finish before the all-1 pattern appears. Therefore, the number of required patterns increases in some cases (as it is shown in Fig. 1b) and becomes equal to $\lceil \log_2(N+2) \rceil$. Such a test called the *Modified Counting Sequence* guarantees the *detection* of all possible opens and shorts with a minimal number of test patterns. However, it does not guarantee the accurate *diagnosis* due to aliasing problem. Aliasing appears when some faulty response is equal to some correct response. Then it is not clear if that wire yielding the correct response is also faulty or not.

Later, Wagner [10] proposed an approach where the Counting Sequence is applied two times: once in its original form and then inverted. This algorithm is called the *True/Complement* algorithm and is illustrated in Fig. 1c. The code words applied to each line contain equal

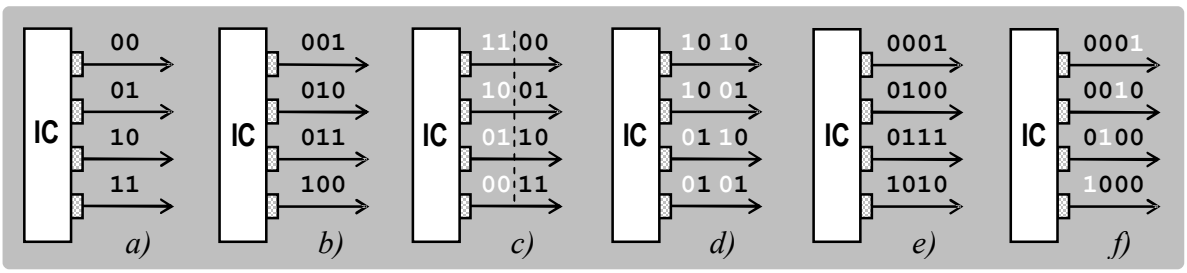



**Figure 1. Different test pattern generation algorithms**

quantities of 0s and 1s. These quantities are always increased or decreased when nets are getting shorted or broken. The lost balance between the number of 0s and 1s unambiguously guarantees the diagnosis of all shorts and opens without aliasing. In this approach all-0 and all-1 codes are not forbidden anymore, because they are getting complemented later, which makes the test length being equal to $2\lceil log_2(N) \rceil$.

It could be easily noted that if such a code is applied in a proper way, all the interconnect delay faults will be also captured. However, then the first and the last codes should still be avoided, since they do not produce both 1→0 and 0→1 transitions needed to sensitize both transition faults [6]. In our framework we are going to use a new algorithm, which we call an *Interleaved True/Complement* sequence. In the interleaved sequence the complemented vector comes right after its true-valued counterpart. We call such pair of vectors the *True/Complement Couple*. In figure 1d they are shown in white and black colors respectively. Such a code preserves all the diagnostic properties of its predecessor (the True/Complement code) but at the same time it has several additional useful features.

*Theorem 1*. The Interleaved True/Complement sequence of length $n \geq 4$ provides both 1→0 and 0→1 transitions on the interconnect lines.

*Proof*. At first, it is clear, that each true/complement couple always activates one of the two transitions upon any selected wire. Let this transition be 0→1. Then there are exactly two possible cases for the second couple at the same wire: 1→0 and 0→1, where the first case automatically yields the second required transition. In the second case, the required 1→0 transition appears *between* the two couples, i.e. 0→**1→0**→1. The same holds for the case, when the first transition is 1→0. For any sequence longer than 4, both required transitions are contained in the first 4 vectors and then repeated with each additional couple■

In fact, all the four possible cases for the four-pattern Interleaved True/Complement sequence are illustrated in Figure 1d.

The fact that each true/complement couple of vectors always drives a wire to both the 0 and the 1 values within two adjacent clock cycles allows efficient design of the response analyzer, which becomes independent and produces exact diagnosis being totally isolated of other wires (unlike it is in MISR for example).

For crosstalk testing it might be important that there were both complemented values upon adjacent wires somewhere in the sequence, i.e. at least once the first wire should be set to 0 while the second was set to 1 and once - vice versa.

*Theorem 2*. The Interleaved True/Complement sequence always guarantees that at least once both complemented values 0/1 and 1/0 appear upon any two pair of independent interconnect nets.

*Proof*. For the Counting Sequence, it is known that the code words upon any two nets are different at least in one bit position. Let the difference be 0/1. In the True/Complement sequence each bit in the code word has its complemented counterpart. This yields the corresponding 1/0 pair of complemented values upon the same two nets■

It could be shown in a similar way that the Hamming distance between any two code words in the True/Complement code is at least 2. It is well known from the coding theory, that the larger the Hamming distance, the higher the probability of detection and diagnosis of faulty signals. This holds especially for the harmful influence exerted upon the interconnect wires by crosstalk, switching noise, or other electromagnetic phenomena. This is also important for such short faults where neither the 1 nor the 0 driver is dominant.

The test length of the proposed Interleaved True/Complement sequence is the same as the original True/Complement one has, i.e. $2\lceil log_2(N) \rceil$.

There is another algorithm known to yield code words with a Hamming distance of at least 2, and at the same time requiring less test patterns [11]. It is called the LaMa algorithm and it generates the test of length $\lceil log_2(3N+2) \rceil$. This algorithm is similar to the Modified Counting one but increments by 3 rather than 1 (see Figure 1e). This algorithm, however, is not directly suitable neither for interconnect delay fault testing, nor for crosstalk. The former is true because the two required transitions (0→1 and 1→0) upon an interconnect wire are not guaranteed.

**Table 1. Properties of TPG Algorithms**

| | **Counting Sequence** | **Modified Counting** | **True/Compl Algorithm** | **Interleaved True/Compl** | **LaMa Algorithm** | **Walking Sequence** |
|---|---|---|---|---|---|---|
| **Number of vectors** | $\lceil \log_2(N) \rceil$ | $\lceil \log_2(N+2) \rceil$ | $2\lceil \log_2(N) \rceil$ | $2\lceil \log_2(N) \rceil$ | $\lceil \log_2(3N+2) \rceil$ | N |
| **Hamming distance (min)** | 1 | 1 | 2 | 2 | 2 | 2 |
| **Defect detection** | shorts | shorts, opens | shorts, opens, (delays) | shorts, opens, delays | shorts, opens | shorts, opens, (delays) |
| **Diagnostic properties** | bad | bad | good | good | average | best |

The testing of crosstalk is not guaranteed due to the fact that the two bit positions where the two code words are different could have the same difference, i.e. both 0/1 or both 1/0. This case is illustrated in Figure 1e where the code words upon the second and the third wire have only the 0/1 difference in both bit positions. The fault diagnosis properties of the LaMa algorithm are similar to the ones of the Modified Counting method with the only difference in the Hamming distance.

Another algorithm - the Walking Sequence is probably the simplest one to automatically generate. There are two types – walking 0 and walking 1. The latter is illustrated in Figure 1f. In a slightly modified way, the walking sequence keeps most important diagnostic properties like the Hamming distance and the required transitions for delay fault testing. At the same time, however, it is one of the longest test sequences used in practice, since it features the length equal to *N*.

The summary of properties of the discussed algorithms is given in Table 1. There are also other test sequences proposed by different authors (for example [7] and [15]) but they are mostly modifications to these basic ones.

It is seen from the table that there are three algorithms with comparatively good fault detection and diagnostic properties. These are both True/Complement algorithms and the Walking Sequence. Two of them require additional vectors in order to be able to detect delays upon all the nets. This makes the Interleaved True/Complement code be the shortest sequence to detect also delay faults in addition to commonly considered shorts and opens. The diagnostic properties of the Walking Sequence are the best among these codes, but its length is the largest one.

## 3. At-Speed Interconnect BIST Hardware

Since the introduction of the IEEE 1149.1 Boundary Scan (BS) standard the powerful and universal BS architecture became the main mechanism of the interconnect test. There are quite a lot of publications [4,6,7,12-14] suggesting efficient combination of the BS and IBIST. In most cases, on-board TPGs are used for feeding the BS shift register with test patterns. In [7], for instance, authors extensively study the problem and suggest different IBIST hardware realizations for the True/Complement and the walking 1 sequence.

Most of later works [12-14] consider the problem of multiple driver contention in the tri-state environment and on-board fault diagnosis. Usually they apply the True/Complement sequence modified in a certain way to avoid the contention. The test data then represents composite vectors [12] generated by two types of TPGs: C-TPG (for control signals) and D-TPG (for data signals) [13] (see Fig. 2a).

The general structure of the D-TPG is given in Fig. 2b. It consists of two counters. The first one generates the Counting Sequence while the second one selects current bit (current vector). The vectors are then generated serially and fed into the Boundary Scan chain intermixed with control signals coming from the C-TPG. As it was previously discussed such a technique needs in general $T \bullet L$ clock cycles to complete, where $T$ is the test length and $L$ is the length of the BS chain. Moreover, it does not directly support the *at-speed testing* of interconnect.

Basically there are four key elements in a successful at-speed testing solution. The first is the selection of the test sequence, which should detect all the necessary timing-related faults. The second is the parallel test pattern generation, i.e. each new pattern should be formed in the TPG at each new clock cycle (like in LFSR) and without any intermediate shift operations. Third, the response analysis must also be made at-speed and in parallel. Finally, the whole solution must be efficient in terms of silicon area. This section describes the hardware implementation of such a TPG for the Interleaved True/Complement code and the RA hardware used for fault diagnosis. The TPG forms test vectors right on chip (not on board) replacing

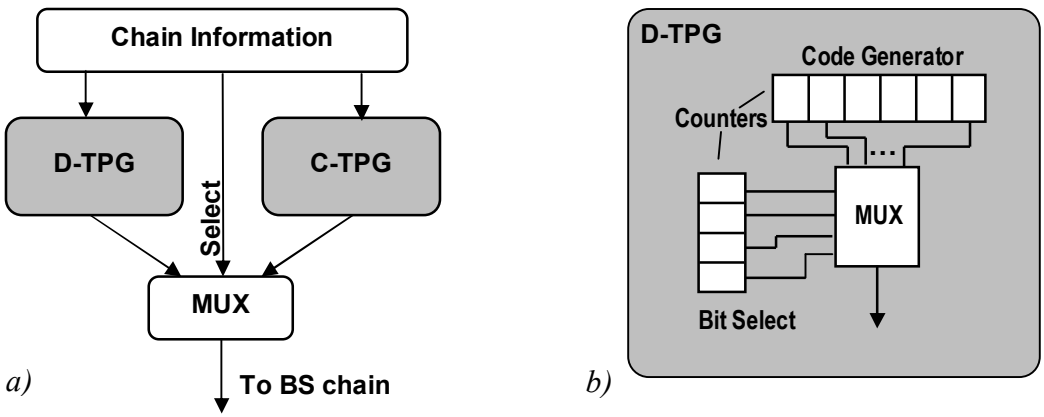



**Figure 2. Partitioned serial TPG for BS based IBIST: a traditional approach**


**Figure 3. Serial vs. parallel test generation**

and enhancing the functionality of D-TPG. The control data should still be formed in C-TPG as it is done in [13]. The main idea of the TPG architecture is based on a circular shift register and its proper initialization. The fault diagnosis is made right on-chip and only shifted out at the end of the test session. Compared to other known techniques, current approach provides the minimal hardware cost at the shortest test application time.

The main idea behind the TPG is illustrated in Fig. 3. Let us first consider the parallel generation of the Modified Counting Sequence. Fig. 3a illustrates the way it is generated by the counter in D-TPG. The arrow shows the direction of test generation, which is orthogonal to the direction of test application (Fig 3e). Therefore, each parallel test vector (the vector to be applied to the nets) needs to be generated bit by bit (Fig. 2b) each time before it is applied. Hence, the common counter-based approach (Fig. 2b) is not applicable here.

It is well known that the *fully configured* LFSR generates exactly the same $2^k$ different vectors except the all-zeros (see Fig. 3b). Since, all-zeros along with all-ones are forbidden code words in the Modified Counting Sequence, there is nothing to loose. However, as the code words coming from LFSR are of different order compared to the ones generated by counter, the all-ones code still might appear in the middle of the sequence and invalidate the test. Therefore the all-ones vector should be taken as the initial state that will be just skipped. In this case all the

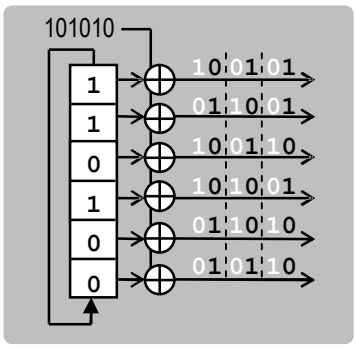



**Figure 4. At-speed TPG**

rest sequence will be valid (Fig. 3c). In this way, an LFSR, being cheaper in hardware can replace the counter for TPG purposes (also in BS based approaches). However, still it does not fit for parallel test pattern generation.

When one examines the patterns (columns) in Fig. 3c, one may notice that the middle one is exactly the rightmost one, but shifted circularly by one bit upwards. The same holds for the leftmost pattern compared to the middle one. It can be proved that by placing the leftmost (or the rightmost) vector into the circular shift register (without additional feedbacks) and applying $\lceil log_2(N+2) \rceil$ clocks the resulting test sequence generated in parallel will be exactly the same as the orthogonal one, generated by the corresponding LFSR.

In this way we achieve two goals: *a)* parallel test vector generation, *b)* reduction of TPG hardware cost compared to the counter-based approach. The resulting test maintains all the properties of the Modified Counting Sequence, which guarantees the detection of all the considered fault types except delay faults. Moreover, now it can be generated and applied at-speed.

The IBIST TPG hardware realization for the Interleaved True/Complement sequence is illustrated in Fig. 4. First, the shift register gets initialized in the same way as it was discussed above. Then it works at half the frequency and holds each vector for two clocks. The outputs of the

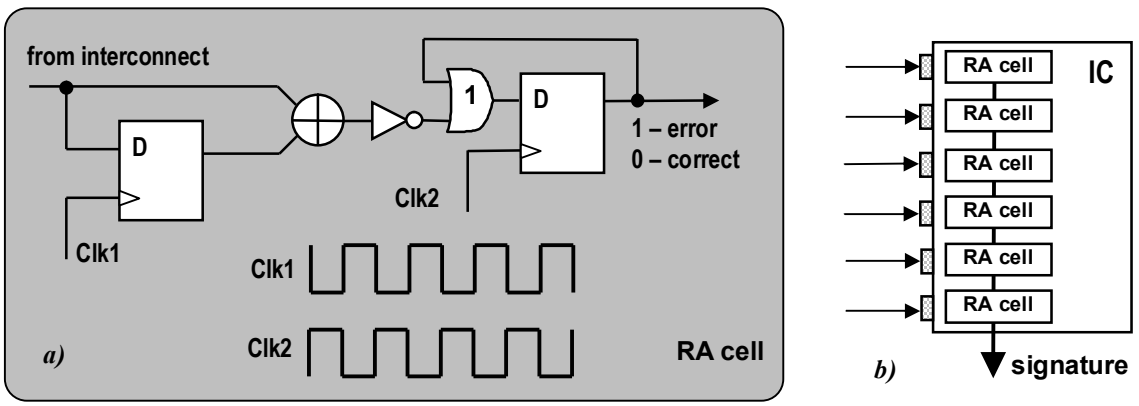



**Figure 5. Response analysis hardware**

register are connected to XOR gates where they get inverted at each second clock by an external signal 010101… coming from an oscillator. The black and the white figures upon the interconnect lines represent the true and complemented values of the corresponding vector. In Fig. 4, it can be clearly seen that the hardware complexity of the TPG is approximately the same as of the LFSR.

Now we are ready to approach the main result of this work – the board configuration independent at-speed precise fault diagnosis. As it was discussed above, the True/Complement Couples always contain a pair of 0 and 1 on each interconnect line. In case of a fault, some of such couples on a faulty line will be broken in such a way that they will contain either only 0-s or only 1-s. Therefore, for successful fault diagnosis it is enough to consider each couple separately of other couples. At the same time, each net can be considered independently.

Figure 5a shows possible hardware implementation of such a separate response analyzer for a single net. It consists of two D flip-flops driven by two antiphased clock signals. In this way the first part of the couple is stored in the first D flip-flop in order to be XOR-ed with the second part as soon as it comes along the interconnect line. The opposite values (the correct behavior) produce logic '1' at the output of the XOR gate, which becomes 0 after the inverter. At the same time the second D flip-flop becomes sensitive and that 0 will be captured inside.

As soon as there will be equal values in a couple (the erroneous behavior) the first part of the circuit will generate the error signal, which is equal to logic '1'. This error signal will be then captured inside the second flip-flop and it will be kept there until the end of the test session. In such a way, each faulty interconnect line will be explicitly identified by logic '1' in the corresponding RA cell. After the application of all the test patterns the final fault signature should be shifted out as shown in Fig 5b.

## 4. Conclusions

The main goal of this paper is to propose a new diagnostic solution for interconnect BIST, which is suitable for at-speed testing and for detection both static and dynamic faults. The idea is based on a new interconnect test sequence called the Interleaved True/Complement code and original hardware implementation of the TPG and the RA.

The TPG is capable of generating test vectors in parallel similarly to the pseudorandom vectors formed in LFSR. The main idea behind it is based on using a circular shift register and its proper initialization. The major advantage of this TPG compared to LFSR is the test lengths and the fault detection and diagnosis properties originated from the Interleaved True/Complement code. At the same time the vectors are generated in parallel, which reduces the testing time from original $T \bullet L$ clock cycles to T clock cycles. This generator forms test vectors right on chip (not on board) replacing and enhancing the functionality of D-TPG commonly used in IBIST.

Presented framework guarantees detection and diagnosis of short, open, and delay faults on interconnect lines by at-speed generated and applied test sequence of length $2\lceil log_2(N+1) \rceil$. The fault diagnosis is made right on-chip and only shifted out at the end of the test session.

The hardware implementation of both the test generator and the response analyzer is very efficient in terms of silicon area.

## Acknowledgments

This work was supported by the Estonian Science Foundation Grants No 5910 and 5649.

## References

[1] IEEE 1149.1-1990, "IEEE Standard Test Access Port and Boundary-Scan Architecture," 1990.

[2] E.J. Marinissen, B. Vermeulen, H. Hollmann, R.G. Bennetts, „Minimizing Pattern Count for Interconnect Test under a Ground Bounce Constraint," in *IEEE D&T of Comp*., March-April 2003, pp. 8-18.

[3] R.Pendurkar,A.Chatterjee,Y.Zorian,"Switching Activity Generation with Automated BIST Synthesis for Performance Testing of Interconnects," *IEEE Trans CAD/ICS*, vol.20, no.9, 2001.

[4] B.Nadeau-Dostie,et.al,"An Embedded Technique for At-Speed Interconnect Testing," in *Proc.ITC'99*, pp.431-438.

[5] A.Attarha,M.Nourani,"Testing Interconnects for Noise and Skew in Gigahertz SoC," in *Proc of ITC'2001*, pp.305-314.

[6] S.Park, T.Kim, "A new IEEE 1149.1 boundary scan design for the detection of delay defects," in *Proc. DATE'2000*, pp. 458-462.

[7] A.J.Hassan,J.Rajski,V.K.Agrawal,"Testing and Diagnosis of Interconnects…," *Proc. ITC'88*, 1988, pp.126-137.

[8] W.H. Kautz, "Testing of Faults in Wiring Interconnects," *IEEE Trans. Computers*, vol. 23, no. 4, 1974, pp. 358-363.

[9] P. Goel and M.T. McMahon, "Electronic Chip-in-Place Test," *Proc. ITC 82, IEEE Press*, 1982, pp. 83-90.

[10] P.T. Wagner, "Interconnect Testing with Boundary Scan," *Proc. Int'l Test Conf., IEEE Press*, 1987, pp. 52-57.

[11] L. Eerenstein and M. Muris, *Method for Generating Test Patterns to Detect an Electric Shortcircuit, a Method for Testing Electric Circuitry While Using Test Patterns So Generated…,* US patent 5636229, Washington, D.C., 1997.

[12] C.Su, S.W.Jeng, Y.T.Chen, "Boundary scan BIST methodology for reconfigurable systems," in *Proc. ITC'98*, *IEEE Press*, 1998, pp. 774-783.

[13] W.Feng,F.J.Meyer,F.Lombardi,"Novel control pattern generators for interconnect testing…," in *Proc. Int'l Symp. Defect and Fault Tolerance in VLSI*, 1999, pp. 112-120.

[14] C.Su,W.Tseng,"Configuration Free SoC Interconnect BIST Methodology," in *Proc. ITC'2001*, pp.1033-1038.

[15] W.T. Cheng, J.L Lewandowski, E. Wu, "Diagnosis for Wiring Interconnects," in *Proc. ITC'90*, pp. 565-571.

# Curriculum Vitae

**1. Personal Data**

| | |
|---|---|
| **Name** | Artur Jutman |
| **Date of birth** | 12.01.1976 |
| **Citizenship** | Estonian |
| **Marriage status** | married |
| **Children** | no |

**2. Contact data**

| | |
|---|---|
| **Address** | Oismae tee 146-19, Tallinn 13511, Estonia |
| **Telephone** | +372 620 2253 |
| **E-mail** | artur@pld.ttu.ee |

**3. Educational history**

1999 Entered Ph.D. studies in Tallinn Technical University (TTU), Computer Engineering (CE)
1999 Received M.Sc. degree from TTU, CE
1997 Entered Master studies in TTU, CE
1993 Entered TTU faculty of CE

**4. Languages**

| | |
|---|---|
| English | Good |
| Russian | Good |
| Estonian | Good |

**5. Professional experience**

2001- TTU, Dept. of CE, researcher
2001-2001 TTU, Dept. of CE, senior engineer
1999-2001 TTU, Dept. of CE, engineer

**6. Academic degree**

Master of Science in Computer Engineering,
"Design Error Diagnosis in Digital Circuits",
TTU, 1999

| | | |
|---|---|---|
| 7. | **Organizational membership** | Member of the Council of the European Association for Education in Electrical and Information Engineering (EAEEIE) |
| **8.** | **Awards** | 2003 – Outstanding paper award at MIXDES conference in Lodz, Poland<br>2002 – Scholarship by Development Foundation of TTU<br>2000 – I$^{st}$ prize at the contest of student works by Estonian Ministry of Education |
| **9.** | **Research interests** | Fault modelling<br>Embedded self-test<br>Test set optimization<br>Binary decision diagrams<br>Design error diagnosis |
| **10.** | **Other projects** | Development of concepts and tools for e-learning |

# Elulookirjeldus (CV)

**1. Isikuandmed**

| | |
|---|---|
| **Nimi** | Artur Jutman |
| **Sünniaeg ja -koht** | 12.01.1976, Tallinn |
| **Kodakondsus** | Eesti |
| **Perekonnaseis** | abiellus |
| **Lapsed** | ei ole |

**2. Kontaktandmed**


| | |
|---|---|
| **Aadress** | Õismäe tee 146-19, Tallinn 13511, Eesti |
| **Telefon** | +372 620 2253 |
| **E-posti aadress** | artur@pld.ttu.ee |


**3. Hariduskäik**

| **Õppeasutus** | **Lõpetamise aeg** | **Haridus** |
|---|---|---|
| Tallinna 5. Keskkool | Juuni 1993 | keskharidus |
| Tallinna Tehikaülikool | Juuni 1999 | teh. magister |

**4. Keelteoskus**

| **Keel** | **Tase** |
|---|---|
| Vene | Kõrg |
| Inglise | Kõrg |
| Saksa | Alg |

**5. Täiendõpe** –

**6. Teenistuskäik**

| | |
|---|---|
| 2001- | TTÜ, Arvutitehnika Instituut, teadur |
| 2001-2001 | TTÜ, Arvutitehnika Instituut, vaneminsener |
| 1999-2001 | TTÜ, Arvutitehnika Instituut, insener |

**7. Kaitstud lõputööd** Magistritöö “Digitaalskeemide disainivigade diagnostika” , Tallinna Tehnikaülikool, 1999

| | |
|---|---|
| **8. Auhinnad** | Silmapaistev artikkel MIXDES'2003 konverentsil Poolas, juunil 2003. a.<br>Tallinna Tehnikaülikooli Arengufondi stipendium aastal 2002.<br>Eesti üliõpilaste teadustööde riikliku konkursi esimene preemia aastal 2000. |
| **9. Teadustöö põhisuunad** | Digitaalskeemide simuleerimine ja verifitseerimine<br>Binaarsed otsustusdiagrammid<br>Digitaalsüsteemde isetestimine<br>Disainivigade diagnostika<br>Testprogrammide optimeerimine |
| **10. Teised uurimis-projektid** | Distantsõppe ja e-õppe metoodika ja vahendite väljatöötamine |

# Selected Publications


R. Ubar, A. Jutman, "Hierarchical Design Error Diagnosis in Combinational Circuits by Stuck-at Fault Test Patterns," *Proc. of 6th International Conference Mixed Design of Integrated Circuits and Systems (MIXDES'99)*, Krakow, Poland, June 17-19, 1999, pp. 437-442.

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.

R. Ubar, A. Jutman, Z. Peng, "Improving the Efficiency of Timing Simulation in Digital Circuits by Using Structurally Synthesized BDDs", *Proc. of IEEE Conference NORCHIP'2000*, Turku, Finland, November 6-7, 2000, pp. 254-261.

A. Jutman, R. Ubar, Z. Peng, "Timing Simulation of Digital Circuits with Binary Decision Diagrams," *Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE'01)*, Munchen, Germany, March 13-16, 2001, pp. 460-466.

J. Raik, A. Jutman, R. Ubar, "Fast Static Compaction of Test Sequences Using Implications and Greedy Search" *Proc. of IEEE European Test Workshop (ETW'01)*, Stockholm, Sweden, May 29 - June 1, 2001, pp. 207-209.

A. Jutman, R. Ubar, "Laboratory Training for Teaching Design and Test of Digital Circuits," in *Proc 8th International Conference on Mixed Design of Integrated Circuits and Systems (MIXDES'01)*, Zakopane, Poland, June 21-23, 2001, pp. 521-524.

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.

R.Ubar, A.Jutman, E.Orasson, J.Raik, T.Evartson, H.-D.Wuttke, "Internet-Based Software for Teaching Test of Digital Circuits," in *Proc. 4th European Workshop on Microelectronics Education (EWME'02)*. Parador de Baiona, Spain, May 23-24, 2002, pp. 317-320.

J. Raik, A. Jutman, R. Ubar, "Fast and Exact Static Compaction of Sequential Circuit Tests Based on Branch-and-Bound Techniques," in *Informal*

*Digest of 7th IEEE European Test Workshop (ETW'02)*, Corfu, Greece, May 26-29, 2002, pp. 19-20.

A. Jutman, J. Raik, R. Ubar, "SSBDD Model: Advantageous Properties and Efficient Simulation Algorithms," in *Informal Digest of 7th IEEE European Test Workshop (ETW'02)*, Corfu, Greece, May 26-29, 2002, pp. 345-346.

S. Devadze, A. Jutman, A. Sudnitson, R. Ubar, "Web-based training system for teaching basics of RT-level Digital Design, Test, and Design for Test," in *Proc. of 9th International Conference on Mixed Design of Integrated Circuits and Systems (MIXDES'02)*, Wroclaw, Poland, June 20-22, 2002, pp. 699-704.

J. Raik, A. Jutman, R. Ubar, "Fast Static Compaction of Tests Composed of Independent Sequences: Basic Properties and Comparison of Methods" in *Proc. of 9th IEEE International Conference on Electronics, Circuits and Systems (ICECS'02)*, Dubrovnik, Croatia, Sept. 15-18, 2002, Vol. 2, pp. 445-448.

A. Jutman, J. Raik, R. Ubar, "SSBDDs: Advantageous Model and Efficient Algorithms for Digital Circuit Modeling, Simulation & Test," in *Proc. of 5th International Workshop on Boolean Problems (IWSBP'02)*, Freiberg, Germany, Sept. 19-20, 2002, pp. 157-166.

S. Devadze, A. Jutman, A. Sudnitson, R. Ubar, H-D. Wuttke, "Teaching Digital RT-Level Self-Test using a Java Applet," in *Proc. 20th IEEE Conference NORCHIP'2002*, Copenhagen, Denmark, November 11-12, 2002, pp.322-328.

A. Jutman, A. Sudnitson, R. Ubar, "Web-based Applet for Teaching Boundary Scan Standard IEEE 1149.1" in *Proc. of 10th International Conference on Mixed Design of Integrated Circuits and Systems (MIXDES'03)*, Lodz, Poland, June 26-28, 2003, pp. 584-589.

A. Jutman, A. Sudnitson, R. Ubar, H.-D.Wuttke "Java Applets Support for an Asynchronous-Mode Learning of Digital Design and Test," in *Proc. of 4th International Conference on Information Technology Based Higher Education and Training (ITHET'03)*, Marrakech, Morocco, July 7-9, 2003, pp. 397-401.

A. Jutman, "On SSBDD Model Size and Complexity", in *Proc. of 4th Electronic Circuits and Systems Conference (ECS'03)*, Bratislava, Slovakia, September 11-12, 2003, pp. 17-22.

M.Aarna, E.Ivask, A.Jutman, E.Orasson, J.Raik, R.Ubar, V.Vislogubov, H.-D.Wuttke, "Turbo Tester - Diagnostic Package for Research and Training", in *Scientific-Technical Journal "Radioelectronics & Informatics"*. KNURE. Vol. 3(24), 2003, pp.69-73.

A. Jutman, R. Ubar, H.-D. Wuttke, "Overview of E-Learning Environment for Web-Based Study of Testing and Diagnostics of Digital Systems", in *Proc. of 5th European Workshop on Microelectronics Education (EWME'04)*, Lausanne, Switzerland, April 15-16, 2004, pp. 173-176.

A. Jutman, “Shift Register Based TPG for At-Speed Interconnect BIST”, in *Proc. of 24th International Conference on Microelectronics (MIEL’04)*, Nis, Serbia and Montenegro, May 16-19, 2004, pp. 751-754.

A. Jutman, “At-Speed On-Chip Diagnosis of Board-Level Interconnect Faults”, in *Formal Proc. of 9th European Test Symposium (ETS’04)*, Corsica, France, May 23-26, 2004.

A. Jutman, A Sudnitson, R. Ubar, H.-D. Wuttke, “E-Learning Environment in the Area of Digital Microelectronics”, in *Proc. of Int. Conf. on Information Technology Based Higher Education and Training (ITHET’04)*, Istanbul, Turkey, May 31 – June 2, 2004, pp. 278-283.

R. Ubar, T. Vassiljeva, J. Raik, A. Jutman, M. Tombak, A. Peder, “Optimization of Structurally Synthesized BDDs”, in *Proc of 4th IASTED Int. Conf. on Modeling, Simulation, and Optimization*, Kauai, Hawaii, USA, August 17-19, 2004, pp. 234-240.

A. Jutman A. Peder J. Raik M. Tombak R. Ubar “Structurally synthesized binary decision diagrams,” in *Proc. of 6th International Workshop on Boolean Problems (IWSBP’04)*, Freiberg, Germany, Sept. 23-24, 2004, pp. 271-278.