THESIS ON INFORMATICS AND SYSTEM ENGINEERING C58

Hierarchical Test Pattern Generation and Untestability Identification Techniques for Synchronous Sequential Circuits

ANNA RANNASTE

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

This dissertation was accepted for the defence of the degree of Doctor of Philosophy in Computer and Systems Engineering on October 25, 2010

Supervisor: Dr. Jaan Raik

Opponents: Dr. Erik Larsson, Linköping University, Sweden
Dr. Görschwin Fey, University of Bremen, Germany

Defence: November 25, 2010

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.

/Anna Rannaste/

ISSN 1406-4731
ISBN 978-9949-23-041-9

INFORMAATIKA JA SÜSTEEMITEHNIKA C58

Hierarhilised testigenereerimise ja mittetestitavuse identifitseerimise meetodid sünkroonsetele järjestikskeemidele

ANNA RANNASTE

To my parents

Abstract

Nowadays, digital electronic systems are widely adopted and they are rapidly developing. Research on effective testing methods is needed in order to guarantee the quality of digital systems. Current thesis addresses the hierarchical test pattern generation issues. First, a novel constraint-based automated test pattern generator for Register-Transfer Level (RTL) designs is presented. The tool combines test path constraint activation with a constraint solver allowing test generation for hard to test faults in sequential digital circuits. In addition, the problem of high-level identification of an important subclass of faults: potentially testable initialization faults is discussed.

The second contribution of this research are techniques for identification of untestability for synchronous sequential circuits. Proving untestable faults is important because it helps raising the confidence in the test quality, without this proof test pattern generation would spend a lot of time trying to generate tests for faults that cannot be detected. A novel method of identifying untestable stuck-at faults in the RTL sequential circuits using model-checking is presented. Additionally, a new method for proving sequential untestability based on test generation constraints was developed.

Results obtained during the experimental studies presented in this thesis give a possibility to find "weak" spots from existing sequential circuit test generation approaches and to improve them in the future.

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

Annotatsioon

Tänapäeval on digitaalsed elektroonikasüsteemid laialt levinud ja nad arenevad väga kiirelt. Selleks, et tõsta digitaalsüsteemide kvaliteeti, tuleb välja töötada efektiivsemad testimismeetodid. Käesolevas dissertatsioonis uuritakse peamiselt hierarhilist testvektorite genereerimist puudutavaid probleeme.

Töö esitab uue, kitsendustepõhise automatiseeritud testvektorite generaatori registersiirde taseme skeemidele, mis ühendab endas testi kitsenduste formaalse genereerimise ja lahendamise. Programm võimaldab testida nn. raskesti testitavaid rikkeid. Lisaks uuritakse potentsiaalselt testitavate initsiaaliseerimisrikete identifitseerimise probleeme kõrgtasemel.

Lisaks on uurimistöö eesmärk välja töötada meetodid mittetestitavate rikete tuvastamiseks sünkroonsetes järjestikskeemides. Info mittetestitavate rikete kohta aitab tõsta testide genereerimise kiirust ning suurendab kindlust testi kvaliteedi suhtes. Loodi uudsed meetodid mittetestitavate rikete identifitseerimiseks järjestikskeemides register-siirde tasemel, kasutades verifitseerimisest tuntud mudeli-kontrolli meetodit ja kitsendustel põhinevat testigeneraatorit.

Saadud eksperimentaaltulemused annavad meile võimaluse määrata meetodite nõrgad kohad ja täiustada neid tulevikus.

Dissertatsioon tugineb valitud teaduslikele artiklitele, mis on avaldatud mitmetel rahvusvahelistel konverentsidel.

Acknowledgments

I have many people to thank for their direct or indirect contribution to this research work.

First, I would like to thank my supervisor, Dr. Jaan Raik of Tallinn University of Technology for leading me to this exciting and challenging topic. I am especially grateful to him for guiding my efforts, motivating and consulting me through my studies.

I am thankful to my other advisers, especially to Professor Raimund Ubar for helping me to make the first steps in my research activity and for giving valuable comments and remarks about this thesis.

Also I express my deep gratitude to Professor Hideo Fujiwara from Nara Institute of Science and Technology for inspiring collaboration. I appreciate highly the three academic papers published as a result of our scientific co-operation.

Special thanks to Dr. Margus Kruus, the head of Department of Computer Engineering for giving me his support in critical situations, which enable my research to achieve fruitful results.

Moreover, I would like to acknowledge the organizations that have supported my Ph.D. studies: Tallinn University of Technology, European Commission FP6 research project VERTIGO, National Graduate School in Information and Communication Technologies (IKTDK) and Estonian IT Foundation (EITSA).

Many thanks to all my colleagues and friends for their advice and enthusiastic company.

Finally, I'd like to express my gratitude to my beloved husband Aavo for his patience, care and support at every difficult moment.

I devote this thesis to my parents. Without their continuous support and love my PhD studies would not have been possible.

Anna Rannaste,

Tallinn, 2010

Table of Contents

List of Publications

Test Pattern Generation for Sequential Circuits

- Taavi Viilukas, Jaan Raik, Maksim Jenihhin, Raimund Ubar, Anna Krivenko (Rannaste). Constraint-based Test Pattern Generation at the Register-Transfer Level, *Proceedings of the 13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS'10)*, pp. 352 – 357, April 14–16, 2010, Vienna, Austria.
- Jaan Raik, Hideo Fujiwara, Anna Krivenko (Rannaste). RT-Level Identification of Potentially Testable Initialization Faults. *The Ninth IEEE Workshop on RTL and High Level Testing (WRTLT'08)*, IEEE, pp. 667-672, November 27-28, 2008, Sapporo, Japan.
- Jaan Raik, Anna Krivenko (Rannaste), Raimund Ubar. Comparative Analysis of Sequential Circuit Test Generation Approaches. *Proceedings of the Baltic Electronic Conference (BEC'04)*, pp. 225-228, Oct. 3-6, 2004 Tallinn, Estonia.

Proving Untestable Faults in Sequential Circuits at RTL

- Jaan Raik, Hideo Fujiwara, Raimund Ubar, Anna Krivenko (Rannaste). Untestable Fault Identification in Sequential Circuits Using Model-Checking. *The 17th Asian Test Symposium (ATS'08)*, IEEE, pp. 667-672, November 24-27, 2008, Sapporo, Japan.
- Jaan Raik, Raimund Ubar, Anna Krivenko (Rannaste), Margus Kruus. Hierarchical Identification of Untestable Faults in Sequential Circuits, *Proceedings of the 10th IEEE Euromicro Conference on Digital Systems Design (DSD'07)*, IEEE Computer Society, pp. 668-671, 27-31 August, 2007, Lübeck, Germany.
- J. Raik, A. Krivenko (Rannaste), T. Viilukas, M. Jenihhin, R. Ubar, H. Fujiwara. Constraint-Based Hierarchical Untestability Identification for Synchronous Sequential Circuits, (submitted to the DATE'11 conference)

List of Abbreviations

ATPG	Automated Test Pattern Generator
ADD	Assignment Decision Diagram
ADN	Assignment Decision Node
ALU	Arithmetic Logic Units
BDD	Binary Decision Diagram
DD	Decision Diagram
FSM	Finite State Machine
FU	Functional Unit
GCD	Greatest Common Divisor
HLDD	High-Level Decision Diagrams
IEEE	Institute of Electrical and Electronics Engineers
MUX	Multiplexer
PI	Primary Input
PO	Primary Output
RTL	Register Transfer Level
SSA	Single stuck – at (fault)
TTCN-3	Testing and Test Control Notation Version 3
TUT	Tallinn University of Technology

VHDL	VHSIC (Very-High-Speed Integrated Circuit) Hardware Description Language
UT	University of Tartu

Chapter 1
INTRODUCTION

The basis of this thesis is formed of five research papers representing different aspects of hierarchical test pattern generation issues and listed below:

Paper I: "Comparative Analysis of Sequential Circuit Test Generation Approaches", Jaan Raik, Anna Krivenko (Rannaste), Raimund Ubar (BEC'04)

Paper II: "Constraint-based Test Pattern Generation at the Register-Transfer Level", Taavi Viilukas, Jaan Raik, Maksim Jenihhin, Raimund Ubar, Anna Krivenko (Rannaste) (DDECS'10)

Paper III: "RT-Level Identification of Potentially Testable Initialization Faults", Jaan Raik, Hideo Fujiwara, Anna Krivenko (Rannaste) (WRTLT'08)

Paper IV: "Untestable Fault Identification in Sequential Circuits Using Model-Checking", Jaan Raik, Hideo Fujiwara, Raimund Ubar, Anna Krivenko (Rannaste) (ATS'08).

Paper V: "Constraint-Based Hierarchical Untestability Identification for Synchronous Sequential Circuits", J. Raik, A. Krivenko (Rannaste), T. Viilukas, M. Jenihhin, R. Ubar, H. Fujiwara (submitted to the DATE'11 conference).

The main emphasis is put on untestability identification techniques for synchronous sequential circuits. In addition, constraint-based hierarchical test generation techniques are presented.

This introductory Chapter first presents the motivation for the work which is followed by the formulation of the problem and the outline of main contributions. In the last part of the Chapter the structure of the thesis is described.

1.1 Motivation

Nowadays, electronic systems are widely applicable and reliability is becoming the main problem for these systems. Reliability of electronic systems is essential in military, aerospace and nuclear industries, where failures may have catastrophic consequences. Adequate testing of electronic products is required to guarantee a certain level of reliability. However, it is hard task to test contemporary electronic systems because the ever-increasing complexity. Therefore, developing new, more efficient test methods is required [40].

Electronic circuits are divided into analog and digital ones. Majority of the hardware in use today is based on digital circuits. In this work we consider digital test only. By *testing* we understand not checking the correctness of the function of the implemented circuit (functional verification), but checking for manufacturing correctness. Testing and verification are different tasks with different goals.

Fault models are needed to model the actual physical defects. Fault coverage is the fraction of modeled faults covered by a test. Nowadays, *automated test pattern generation* (ATPG) is used to automatically obtain tests with high-fault coverage for digital electronic circuits. Already for two decades, *automated test pattern generation* for *combinational circuits* was considered as a solved problem. However, almost all the digital devices produced nowadays belong to the class of *sequential circuits*, containing feedback loops that make the test generation problem extremely difficult [38].

Scannable registers are used to test complex electronic circuits. These architectures are inserted to the circuits to make internal points of the circuit controllable and observable by converting a sequential design into a pseudo-combinational one. Full-scan design is easy to accomplish and it allows application of the combinational ATPG resulting in a near-100-percent fault efficiency. Scannable registers have a number of drawbacks including performance, routing overhead and excessive amount of test data. There are three major shortcomings of using scannable registers. First one, at–speed test is costly in scanned designs.

Errors that would appear only at full speed may escape the test. Second, scannable registers increase the cost of the chip. Third, this method also causes targeting of non-functional failure modes, which results in over-testing and yield loss [39].

In this work, we consider test generation problem without implementing scan structures. Test generation can be carried out at different levels of design abstraction. Usually each design abstraction level is represented by different types of models. In this work, a hierarchical test pattern generation approach is presented, where high-level (register-transfer level) and logic-level design information is described by decision diagrams.

1.2 Problem Formulation

As it was mentioned above, the problem of automated test pattern generation for combinational digital circuit was solved already by the end of 1980s, but optimal decision for test pattern generation for sequential circuits is still not found. There have been many different approaches to generating tests for structural faults in sequential circuits proposed over the years [1-10], [20-24]. However, the problem of test pattern generation for sequential circuits is a challenge that lacks an acceptable solution despite of decades of research. The achieved fault coverages are unsatisfactory and test generation times are long for more complex circuits.

Similar to test pattern generation, the problem of identifying untestable faults in sequential synchronous circuits remains unsolved. *Untestable fault* is a fault for which no test exists. Test generation for sequential synchronous designs is a time-consuming task. Automated Test Pattern Generation (ATPG) tools spend a lot of effort not only for deriving test vectors for testable faults but also for proving that there exist no tests for the untestable faults. Because of this reason, the identification of untestable faults has been an important aspect in speeding up the sequential ATPG. The previously published works in untestability identification operate at the logic-level and, thus, they do not scale with the increasing complexity of modern designs [30-34]. Thus, it would be good if part of the untestable faults could be identified at higher abstraction levels. Identification of untestable faults allows raising the confidence in the ATPG efficiency.

The current thesis is focused on improving untestability identification techniques of synchronous sequential circuits that is the one of the major issues in the area of digital circuits testing.

In addition, this thesis also presents a register-transfer level test pattern generation for non-scan sequential circuits containing feedback loops. A novel

constraint-based automated test pattern generator for Register-Transfer Level designs is introduced. The tool combines test path constraint activation with a constraint solver. First, a deterministic algorithm that extracts constraints for activating test paths at RTL is applied. Subsequently, a constraint solving package ECLiPSe is used for assembling the tests. We showed that experiments offers short run times, increased fault coverage for hard-to-test designs with respect to earlier approaches listed above.

Finally, a new method of RT-Level identification of potentially testable initialization faults is considered.

In sequential Automated Test Pattern Generation (ATPG) based on a three-valued algebra (0,1,X) a fault is said to be hard-detected if a fault effect (0/1 or 1/0) appears at a primary output. However, not all the faults can be tested by such hard-detection model. Many faults belonging to the class of initialization faults are known to be covered only by resorting to potential detection (effect 0/X or 1/X). Existing high-level fault models assume hard-detection and therefore are not capable of handling the initialization faults.

It is obvious that any ATPG algorithm first attempts to generate hard-detection tests. As a result, high-level algorithms spend test generation time also for those faults that may only be detected potentially.

The next Sections detail the areas of my research and further reveal the motivation beind it.

1.3 Contributions

The main contributions of this thesis are summarised as follows:

The current thesis introduces several techniques to perform hierarchical test pattern generation and untestability identification for synchronous sequential circuits.

- An overview of the comparative study of ATPG methods is proposed [I]. A comparative study of test pattern generation approaches based on three tools: a genetic algorithm test generator GATEST, a deterministic logic-level tool HITEC and a hierarchical tool DECIDER. The purpose of this study was to find out, which fault types are covered by the tools implementing completely different approaches.

- A novel constraint-based automated test pattern generator for Register-Transfer Level (RTL) designs is introduced [II]. The tool combines test path constraint activation with a constraint solver. First, a deterministic algorithm that extracts constraints for activating test paths at RTL is applied. Subsequently, a constraint solving package ECLiPSe [14] is used for assembling the tests.

- This thesis also presents the problem of high-level identification of an important subclass of faults, of potentially testable initialization faults [III]. Potentially detectable initialization faults form a large subset of all the faults not testable by hard-detection [III]. Potentially testable fault identification is applicable, both, for stuck-at and high-level fault models.

- An approach of identifying of untestable faults in sequential circuits is considered [IV]. We propose using model-checking for detecting untestable stuck-at faults at the Register-Transfer Level (RTL). In particular, we present a method for formally generating PSL language assertions for proving untestable stuck-at faults in sequential synchronous designs.

- A novel method of register-transfer level untestability identification for non-scan sequential circuits containing feedback loops is introduced [V]. First, an RTL test pattern generator Decider is applied in order to extract test path extraction constraints. Then, a constraint-driven deterministic logic-level automated test pattern generator is run providing hierarchical test generation and testability proof in sequential circuits.

1.4 Thesis Organization

The presented thesis is organized in a form of overview of the research results that have been published in five scientific papers. It is divided into seven main Chapters. The thesis contains description of the investigated problem, implementation discussions, examples and conclusions. The rest of the thesis is organized as follows.

Chapter 2 provides an overview of related works in studying of untestability identification techniques and test pattern generation for synchronous sequential circuits.

Chapter 3 presents an overview of preliminaries and makes an introduction to different aspects of digital circuit. In addition, describes a classification of test pattern generation methods for sequential circuits and Register-transfer level view to circuits.

Chapter 4 forms a background information required for discussion of the further proposed approaches and gives an overview of the comparative study of ATPG methods.

Chapter 5 starts with discussion of a novel constraint-based automated test pattern generation at register-transfer level. Further, research of identification of potentially testable initialization faults at the RT-Level is introduced.

Chapter 6 presents an overview of the research results based on the selected publications. First, a new approach proposes applying model-checking for detecting untestable stuck-at faults at the register-transfer level. Finally, hierarchical untestability identification for non-scan sequential circuits containing feedback loops is considered.

Chapter 7 draws conclusions for this thesis and discusses possible directions for future work.

Chapter 2
RELATED WORKS

Current Chapter summarises the scope and provides an overview of related works.

At present, satisfactory methods for testing sequential circuits are missing. Gate-level test pattern generation for sequential circuits is a challenge that lacks an acceptable solution despite of decades of research. Therefore, the test community has turned towards higher abstraction levels. In particular register-transfer level (RTL) test generation has been regarded as a potential trade-off between functional and low-level approaches as it provides design abstraction while still retaining correspondence to the circuit structure.

A common industry practice is therefore to resort to full- or partial-scan design, where scan paths are inserted into circuit flip-flops converting a sequential design into a pseudo-combinational one. Full-scan design is easy to accomplish and it allows application of the combinational ATPG resulting in a near-100 percent fault efficiency. However, the scan path approach has a number of drawbacks including performance and routing overhead, difficulty to achieve at-speed testing, excessive amount of test data, and last but not least, yield loss because of over-testing.

A number of works have been proposed in order to tackle the problem of untestability identification. Despite of all the efforts the problem still lacks a breakthrough. At the gate-level, a number of deterministic test generation tools, both academic [1, 2] and commercial, have been implemented. None of these methods can efficiently handle sequential designs of even a couple of thousands of gates. With the further growth of the circuit size fault coverages tend to drop while run times increase rapidly.

Better performance has been obtained with simulation-based approaches. Here, genetic algorithm based methods have been widely used [6, 7, 8]. Relatively efficient results have been obtained by spectral methods [9]. However, the simulation-based methods are fast for smaller circuits only and become ineffective when the number of primary inputs and the sequential depth of the circuit increases.

Many works on functional test generation have been published in the past [3, 4]. In this field, an efficient technique based on BDD manipulation of data domain partitions has been proposed [5]. However, the fundamental shortcoming of the approaches that rely on functional fault models is that they do not offer full structural level fault coverage.

Hierarchical automatic test pattern generation (ATPG) has been a promising alternative to tackle complex sequential circuits for already more than a decade. In hierarchical RTL test generation, top-down and bottom-up strategies are known. In the bottom-up approach [20], tests generated at the low-level will be later assembled at the higher abstraction level. Such algorithms ignore the incompleteness problem: constraints imposed by other modules and/or the network structure may prevent test vectors from being assembled. Thus, while being fast, this type of approach is not really applicable for sequential circuits with difficult to test feedback loops. In the top-down approach [10], constraints are extracted at the higher level as a goal to be considered when deriving tests for modules at the lower level. This approach allows testing modules embedded deep into the RTL structure. However, as modules are often tested through highly complex constraints, their fault coverage may be compromised.

Early methods on bottom-up RTL testing relied on the assembly of module tests and were applicable of the simplest systems only [20]. A more solid basis for the bottom-up paradigm was laid by Ghosh in [21]. In their work, test environments are generated for each functional module of a given functional RTL circuit described in an assignment decision diagram (ADD) [22] using symbolic justification/propagation rules using a nine-valued algebra. In this method, a test sequence is then formed by substituting the corresponding test patterns into the test environment. However, regardless of the existence of some test environments, the proposed nine-valued algebra cannot always generate the test environments. To overcome this drawback, Zhang et al. upgraded the nine-valued algebra to a ten-valued algebra by taking the signal line value range into consideration. This algebra is able to generate much more test environments [23]. In [24], Zhang's approach has been further improved by introducing additional propagation rules.

Lee and Patel introduced constraint-based test pattern generation for microprocessors in [10]. [11] proposed a bottom-up approach based on a high-level decision diagram (HLDD) engine and a commercial SICStus constraint solver. As experiments show, the tool achieves lower fault coverage in comparison to a commercial gate-level ATPG. In [12], a top-down approach called Decider was introduced, which relied on random constraint solving. The method was later combined with Extended Finite State Machine based engine Laerte++ from the University of Verona, which resulted in a semi-formal setup [13]. Thus, hard-to-test faults inside the modules were not targeted. In this thesis, a constraint solving package ECLiPSe [14] has been incorporated into the Decider tool [II] providing a deterministic hierarchical test pattern generation environment.

In sequential Automated Test Pattern Generation (ATPG) based on a three-valued algebra (0, 1, X) a fault is said to be hard-detected if a fault effect (0/1 or 1/0) appears at a primary output. However, not all the faults can be tested by such hard-detection model. Many faults belonging to the class of initialization faults are known to be covered only by resorting to potential detection (effect 0/X or 1/X). It is obvious that any ATPG algorithm first attempts to generate hard-detection tests. This means wasting test generation time also for those faults that may only be detected potentially.

Experimental analysis presented in paper [III] points out that an important subclass of faults, the potentially detectable initialization faults, form a large subset of all the faults not testable by the hard-detection model. As a result of the proposed approach the confidence level of sequential ATPG can be increased. As it is pointed out in this thesis, the proposed potentially testable fault identification is applicable, both, for stuck-at and high-level fault models.

Existing high-level fault models assume hard-detection and therefore are not capable of handling the initialization faults. Thus, it would be desirable that the high-level ATPG would have knowledge about the faults that cannot be tested by the hard-detection model.

In their previous work [VI], the authors introduced a new subclass of untestable faults, called register enable stuck-on faults. However, the paper did not propose any formal method for identifying untestable register faults. Paper [IV] presents a new method that is capable of identifying such type of untestable faults. Using model-checking for detecting untestable stuck-at faults at the Register-Transfer Level (RTL) is proposed. In particular, a method for formally generating PSL language assertions for proving untestable stuck-at faults in sequential synchronous designs is presented. As a result the fault efficiency was significantly increased but still remained well below 100 per cent.

The common short-coming of all the earlier methods is that they do not annotate RTL constraints back to gate-level untestable faults. Thus, the fault efficiency (i.e. test coverage) reported by the approaches is often low, which decreases the test engineers confidence to the test. We will show in the paper [V] , in many cases, fault coverage obtained for the modules in RTL test generation decreases if path activation constraints are being ignored.

Chapter 3
PRELIMINARIES

Digital circuits have become increasingly complex, with more transistors and functionalities packed on a single chip of nearly the same physical size. Therefore testing of that complex circuit has become a major problem technically and economically.

3.1 Digital Circuits

A digital circuits can be defined (according to author Francis C. Wang "Digital circuit testing") generally as an interconnections of logic elements such as AND gates, OR gates, INVERTORS, flip-flops, and registers. It must also be able to process a set of discrete and finite-valued electrical signals.

Digital circuits may be classified as combinational or sequential.

3.1.1 Combinational vs. Sequential Circuits

In a combinational circuit, the present outputs depend only on present inputs (subject to reaction times).

A sequential circuit (Figure 3.1) consist of a combinational part and memory elements. There are also feedback loops in the circuit. The combinational part of the circuit is modeled at the Boolean gate-level. Flip-flops are treated as ideal memory elements, whose clock signal is not explicitly represented.

The two classes of circuits have different topologies.

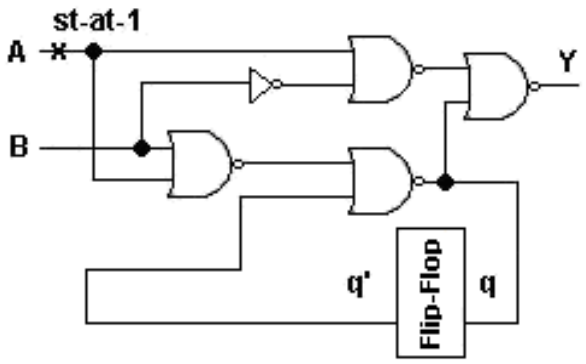


Figure 3.1 An example of a sequential circuit

The first difference between combinational and sequential circuits is that the combinational circuit does not contain memory elements. The second one is that a test for a fault in a sequential circuit may consist of several vectors. A combinational ATPG, on the other hand, is capable of generating always only a single vector for a target fault.

Sequential digital circuits may be further classified as asynchronous or synchronous.

3.1.2 Asynchronous vs. Synchronous Circuits

The outputs of a sequential circuit may be assembled into an ordered list called the state vector, or simply the state.

In an asynchronous circuit, the state can change at any time in response to changes in the inputs.

In a synchronous circuit, the state can change only at discrete times.

3.2 Fault Models

The selection of the fault model determines the efficiency of test generation and the quality of tests. Physical defects represented by mathematical abstraction mechanisms are called *fault models*. The terms *defect, fault and error* have different and specific meaning. *Defects* are physical failures that occur during manufacturing. The goal of test generation is to generate such input stimuli that all (or possibly many) defects would manifest themselves as erroneous output responses of the circuit. In general, there is no one-to-one correspondence between faults and defects but the set of *faults* belonging to the model should represent the defect combinations that are likely to occur in reality. The term *error* is related to the behaviour of the circuit. Wrong output responses of circuits by defects are referred to as errors.

Fault models can be classified according to their level of abstraction into transistor level, logic level and high level (register – transfer and behavioral level) ones. Logic level fault models are: different types of delay fault models (path delay faults, gate delay faults) and the stuck – at fault model. The most simple and at the same time the most popular logic level model is the single stuck – at (SSA) fault model, which will be presented in the Subsection 3.2.1. Its main advantages are as follows:

- it represents a large number of physical defects
- it is independent of technology
- many other fault models can be reduced to the SSA model.

3.2.1 The Stuck-at Fault Model

We assume that the circuit is modelled as an interconnected network of blocks in the stuck-at fault model. At logic level these blocks are Boolean gates. A stuck-at fault is assumed to affect only the interconnections between the gates. Each connection can have stuck-at-0 and stuck-at-1 types of faults. A line with a stuck-at-1 fault will always hold the logic value 1 irrespective of the correct logic output of the gate driving it. Accordingly, a line with a stuck-at-0 fault will always hold the logic value 0 irrespective of the correct logic output of the gate driving it.

Several stuck-at faults can be simultaneously presented in the circuit. A circuit with n lines can have 3^n-1 different stuck line combinations. Even a moderate value

of n will result in an enormous amount of multiple faults. Therefore, only single stuck-at faults are modelled. An n-line circuit has 2n single stuck-at faults and all the single stuck-at faults will also cover also most of the possible fault combinations.

The properties of single stuck-at faults are [25]:

1. only one line is faulty
2. the fault can be at an input or output of a gate
3. the faulty line is permanently stuck to 0 (1)

3.3 Classification of Test Pattern Generation Methods

Test pattern generation methods can be classified (by definition of authors Michel L. Bushnell and Vishwani D. Agrawal from the book " Essentials of Electronic Testing for Digital Memory and Mixed Signal VLSI Circuits), [26] into three categories:

Time-frame expansion. In this method a model of the circuit is created such that tests can be generated by a combinational ATPG method. This procedure is very efficient for circuits described at the Boolean gate-level. Its efficiency degrades significantly with cyclic structure, multiple-clocks, or asynchronous circuitry.

Simulation-based methods. In these methods a fault simulator and a vector generator are used to derive tests. In general, tests can be generated for any circuit that can be simulated. Also, circuits modeled at other levels (register-transfer, transistor, etc) can be treated.

RTL test generation: Test generation with a known initial state (based on hierarchical test generation, which exploits the RTL and gate – level descriptions of a circuit), symbolic test generation for microprocessors (based on symbolic test generation for microprocessors), test generation with functional fault models (uses functional fault models to speed up test generation) [27] and hierarchical test generation method DECIDER (in this method top-down and bottom-up strategies are known. hierarchical test generation is based on multi-level decision diagrams) [15]. The last method of hierarchical test generation is considered in this work.

3.4 Register-Transfer Level View to Circuits

An RTL circuit consists of a datapath and a controller (Figure 3.2). The datapath consists of a network of registers, functional units, multiplexers and buses. The controller governs the data computation in the datapath by generating appropriate load signals for the registers and select signals for the multiplexers and arithmetic-logic units (ALUs).

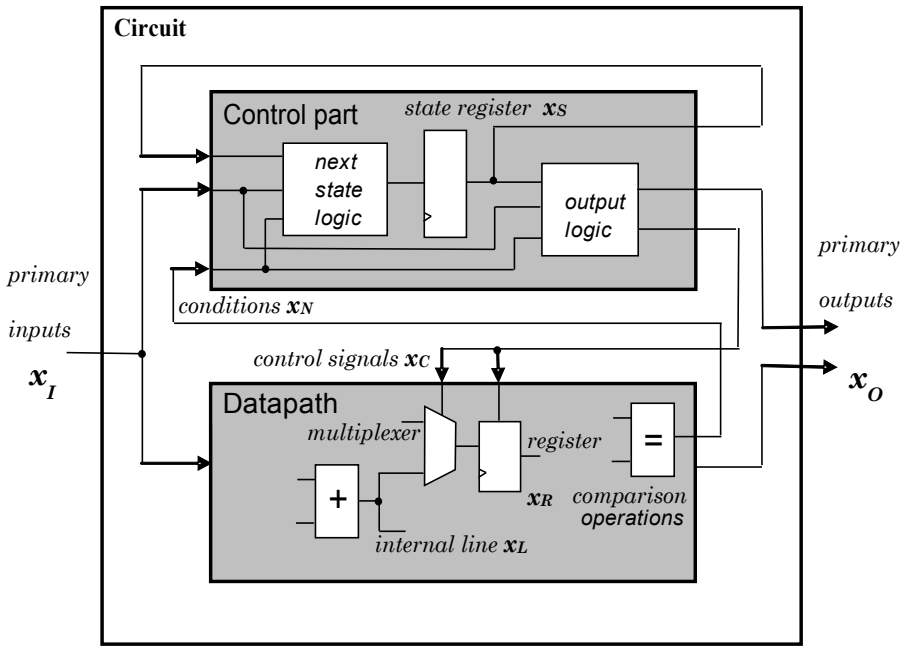


Figure 3.2 RT-level view of a digital circuit and

Here, the control part is a Finite State Machine (FSM) with a state register (variable x_S), next state logic and output logic. As input signals to the FSM are the primary inputs of the design (variables x_I), status bits originating from the datapath (variables x_N) and the previous value of the state variable x_S. Outputs of the FSM are the primary outputs of the design (variables x_O), control signals (variables x_C) and current value of x_S.

Chapter 4
COMPARATIVE STUDY OF ATPG METHODS

Current Chapter presents background information on the topics related to current research and gives an overview of the research results presented in Paper I. The first paper is entitled „**Comparative Analysis of Sequential Circuit Test Generation Approaches**". The paper was written by J. Raik, R. Ubar and the author of this thesis. It was presented at the Baltic Electronic Conference (BEC'04) in Tallinn, Estonia, in October 2004.

The paper address a comparative study of test pattern generation approaches based on three tools: a genetic algorithm test generator GATEST, a deterministic logic-level tool HITEC and a hierarchical tool DECIDER. The purpose of this study was to find out, which fault types are likely to be covered by different approaches. Additional motivation for the work was to find guidelines for improving the fault models implemented in the hierarchical test pattern generator DECIDER, which is being developed at TUT.

4.1 Case Study of ATPG Methods

The major goal of this work is the comparative analysis of three test generators for sequential circuits: the test generator GATEST based on the genetic algorithm [19], deterministic generator HITEC [18] and hierarchical generator DECIDER [15]. The first two are popular public domain programs from University of Illinois.

The latter is a software developed at Tallinn University of Technology. The research consists of finding out the methods of test pattern generation which cover faults in the circuits. At the same time the faults can be found out in various parts of the circuit.

Table 4.1. shows the comparison of fault coverages and run times achieved by each of the tools. From the table it can be seen that DECIDER (hierarchical ATPG) is the most effective tool with. 88.9 % average fault cover and short run times, followed by GATEST (genetic algorithms) with 87.9 % and HITEC (deterministic) with 76.9 % coverage. The fault list sizes for the circuits are provided in the second column of the Table.

Table 4.1 Comparison of sequential circuit test generation tools

Circuit	Faults	HITEC		GATEST		DECIDER		Total
		F.C.,%	Time,s	F.C.,%	Time, s	F.C.,%	Time, s	F.C., %
Gcd	454	81,1	169,5	**91,0**	75	89,9	129,8	91,7
Sosq	1938	77,3	728,4	79,9	739	**80,1**	129,6	80,3
Mult8x8	2036	65,9	1243	69,2	821,6	**74,7**	93,7	74,8
Ellipf	5388	87,9	2090	94,7	6229	**95,04**	1258,9	95,06
Risc	6434	52,8	49020	96,0	2459	**96,5**	150,5	96,7
Diffeq	10008	96,2	13320	96,40	3000	**97,09**	453,7	97,20
Average F.C.		76,9		87,9		**88.9**		89.3

However, the goal is not to compare the absolute results of the tools, which has been done in earlier works [15], but to find out, what regions of the circuit space are covered by the tools implementing completely different approaches. Our study reveals a number of facts previously not known about the capabilities of test tools. For example, what we noticed is that although genetic algorithm based tool

performed well on the absolute scale, it contributed little if any new faults to the two remaining tools. Also, the hierarchical test pattern generator achieves highest results for five out of six benchmark circuits but it is still far from the combined fault coverage of the three tools summed. The combined results are presented in the last column of Table 4.1.

In current study we compared three test generators on six different sequential circuit benchmarks. We examined the fault space covered by different generators in order to determine the sets of overlapping between the tools. We also carried out a more detailed analysis, identifying in which type of the circuit modules (functional units, MUXs, registers, etc.) different ATPGs covered faults. The main goal was to find out suggestions for improving the sequential test generation methods in the future.

For the experiments the following benchmark circuits have been used: *gcd* is a greatest common divisor circuit, *mult8x8* is a 8-bit multiplier, *diffeq* is a circuit implementing the differential equation calculation method, *sosq* implements sum of squares, *risc* is an ALU-based microprocessor, *ellipf* is a DSP circuit implementing an elliptical filter. *Gcd*, *ellipf* and *diffeq* belong to the HLSynth92 benchmark family [16] while the remaining three are from the VILAB set [17]. The characteristics of benchmark circuits are presented in the table 4.2.

Table 4.2 Characteristics of benchmark circuits

Circuit	Fsm	Inputs of buses	Inputs of bits	Outputs of buses	Outputs of bits	Registers	Multiplexers	Functional unit	Tested faults
Diffeq	6	5	81	3	48	7	9	5	10008
Ellipf	28	9	130	8	113	17	7	3	5388
Gcd	8	2	9	1	4	3	4	3	454
Mult8x8	8	2	17	1	16	7	4	9	2036
Risc	4	5	26	1	16	8	4	4	6434
Sosq	5	1	9	2	32	7	2	6	1938

The experiments were run on a 366 MHz SUN UltraSPARC 60 server with 512 MB RAM under SOLARIS 2.8 operating system. We work with stuck-at faults.

Figures 4.1 and 4.2 give a more detailed look to the test results of Table 4.1. Figure 4.1 shows overlappings of the faults covered by the three generators for the six example circuits. Here, "hierarchical" denotes the ATPG DECIDER [15], "genetic" stands for GATEST [19] and "deterministic" is for HITEC [18].

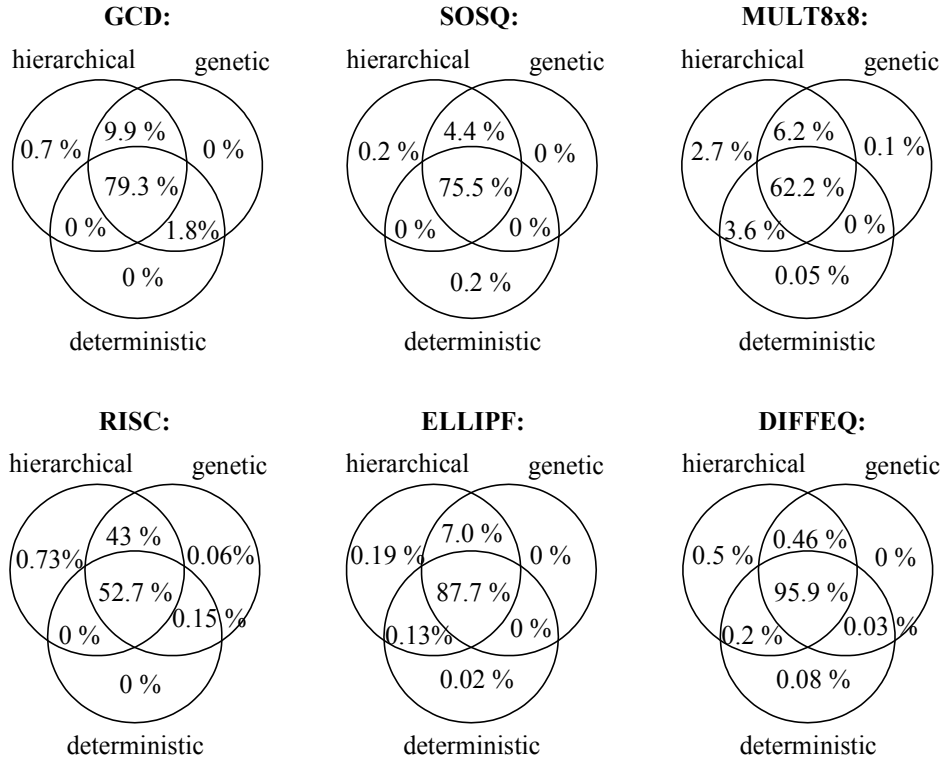


Figure 4.1 Portions of faults detected by hierarchical, genetic and deterministic ATPGs

The most important observations we can make basing on Figure 4.1 are the following:

1. While experiments in Table 4.1 indicate that DECIDER gives in most cases the highest fault coverage, we can see that there are some unique portions of faults covered by GATEST and HITEC.

In fact, the union of the sets of faults covered by the three test generators gives a fault coverage that is in average 0.4 (!) per cent higher than the average fault cover of DECIDER.

2. Table 4.1 also shows that GATEST performs well in terms of the absolute fault coverage numbers. However, it fails to detect nearly any unique faults.

If we look at Figure 4.1, it can be seen that the genetic tool GATEST does not provide any new unique faults at all for four out of six benchmarks: gcd, sosq, ellipf, diffeq. HITEC, whose fault coverage numbers are roughly 11 per cent lower than GATEST's, detects much higher number of unique faults.

This leads to a conclusion that there are many 'hard-to-test' random pattern resistant faults that GATEST as a simulation-based method is not capable of detecting. While deterministic methods are known to have difficulties with larger sequential designs they could still provide useful addition in terms of detecting hard-to-test faults.

Figure 4.2 presents the distribution of achieved fault coverages by module types. Five different types are distinguished: functional unit, comparison operation, MUX, register and control part FSM. 'Total' denotes the summed result for the whole circuit. In the Figure, average values for the set of six circuits are shown.

One of the conclusions that can be made based on Figure 4.2 is that DECIDER covers well the faults in functional units, comparison operators and MUXs. These are the modules it explicitly tests (See the grey circles Figure 3.2!). However, control part FSM is poorly covered by the hierarchical tool. This means that fault models for testing control part could be useful improvement to the tool in the future.

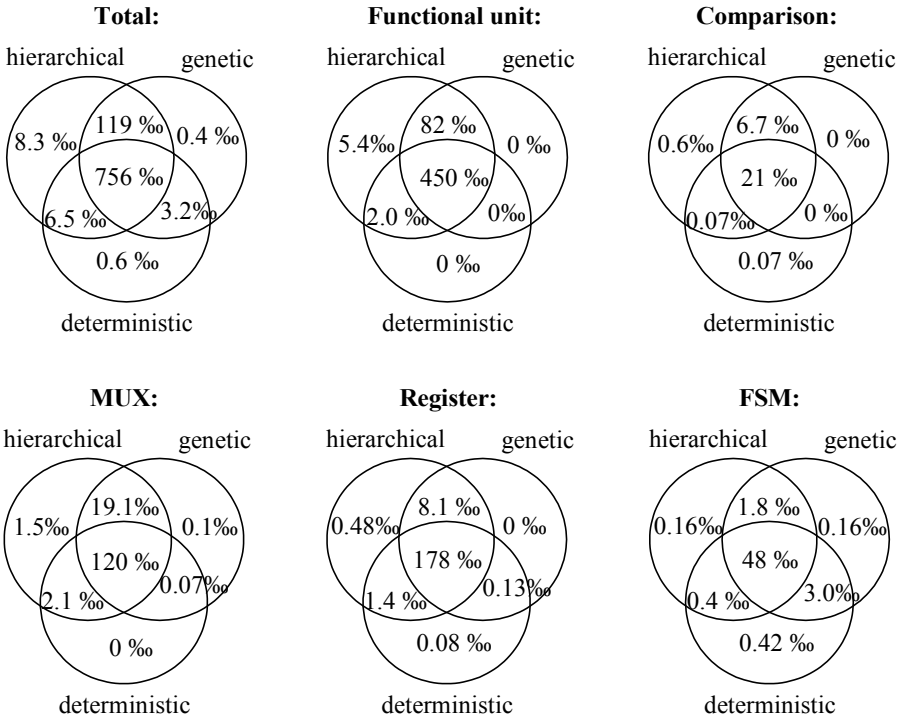


Figure 4.2 Coverage of circuit regions for the three test generators

4.2 Chapter Summary

This Chapter has provided background information needed to understand basic principles of high-level test generation.

The main contribution of the Chapter was to introduce a new method of comparison of different test generation approaches based on three tools: a genetic algorithm test generator GATEST [19], a deterministic logic-level tool HITEC [18] and a hierarchical tool DECIDER [15]. The purpose of this study was to find out, which fault types are likely to be covered by different approaches.

Experiments on a set of six sequential benchmark circuits lead to the following conclusions:

- While genetic algorithm based tool performs well in terms of the absolute fault coverage numbers, it fails to detect nearly any unique faults.
- Deterministic tool has difficulties with larger sequential designs but it is capable of detecting a portion of hard-to-test faults.
- The union of the sets of faults covered by the three test generators has a fault coverage that is in average 0.4 per cent higher than the fault cover of the best tool in the comparison: DECIDER.
- DECIDER loses fault coverage mainly in the control part FSM.

The analysis carried out will be helpful for further development of the hierarchical ATPG DECIDER. Moreover, the authors hope that the results presented here could give valuable guidelines for the developers of future test pattern generators in general.

Chapter 5
TEST PATTERN GENERATION FOR SEQUENTIAL CIRCUITS

The theoretical contribution of this Chapter gives an overview of the research results in generating test pattern for sequential circuits is presented in Papers II and III. The research described in this Chapter embraces comparatively large area and can be divided into two major parts:

a) In Section 5.1, the scope of the research is mainly focused on a novel constraint-based automated test pattern generation at Register-Transfer Level [II].

b) In addition, in Section 5.2 we concentrate on research of identification of Potentially Testable Initialization Faults at the RT-Level [III].

5.1 Constraint-Based Test Pattern Generation at the Register-Transfer Level

The second paper is entitled "**Constraint-based Test Pattern Generation at the Register-Transfer Level**" [II]. The authors of the paper were Taavi Viilukas, Jaan Raik, Maksim Jenihhin, Raimund Ubar and the author of this thesis. The paper was presented at the 13th IEEE International Symposium on Design and Diagnostics of electronic circuits and Systems (DDECS'10) in Vienna, Austria in Aprill 2010.

The paper introduces a novel constraint-based automated test pattern generator for Register-Transfer Level (RTL) designs. The tool combines test path constraint activation with a constraint solver. First, a deterministic algorithm that extracts constraints for activating test paths at RTL is applied. Subsequently, a constraint solving package ECLiPSe [14] is used for assembling the tests. Experiments on ITC99 and HLSynth92/95 benchmarks show that the proposed deterministic method offers short run times. In particular, it provides increased fault coverage for hard-to-test designs with respect to earlier approaches.

5.1.1 Concept of Path Activation Constraints

The test generation approach proposed in current thesis contains two main phases. During the first phase, high-level test path activation, an untested module is selected and for this module propagation and justification is performed as explained in Section 5.1.2. In addition, constraints for the test path are extracted. The goal of the second phase is to satisfy the constraints by using a constraint solver and to compile the test patterns by assigning the values obtained by the constraint solver to the primary input signals (See Section 5.1.3).

The high-level test generation constraints considered in paper [II] are divided into three categories. These are path activation constraints, transformation constraints and propagation constraints. *Path activation constraints* correspond to the logic conditions in the control flow graph that have to be satisfied in order to perform propagation and value justification through the circuit. *Transformation constraints*, in turn, reflect the value changes along the paths from the inputs of the high-level Module Under Test (MUT) to the primary inputs of the whole circuit. These constraints are needed in order to derive the local test patterns for the module under test. *Propagation constraints* show how the value propagated from the output of the MUT to a primary output is depending on the values of the signals in the system. The main idea here is to guarantee that fault signals will not be masked when propagated.

Let us explain the role of these constraints in test generation on an example test path activation for a circuit module shown in Figure 5.1. In the Figure there are two path activation constraints: $true = f_1(x_1,x_2)$ and $false = f_2(x_2,x_3)$. The first one is necessary to propagate the value from the output of the module to the primary output y_3 of the circuit. The latter is required for justification of the first input (D_1) of the module under test. Both these constraints are extracted from the conditional nodes traversed in the control flow graph of the circuit during high-level path activation.

The figure also presents two transformation constraints. These constraints are applied for computing the value of the corresponding module input depending on the values of primary inputs of the circuit. Finally, there is a propagation constraint, which states that the value propagated from the module to the primary output y_3 is dependent on the primary input x_6. Thus, in order to avoid fault masking the value of x_6 must be chosen such that the fault free and faulty values of D_{out} would differ. Note, that the subsets of the primary input variables included into the different types of constraints may overlap.

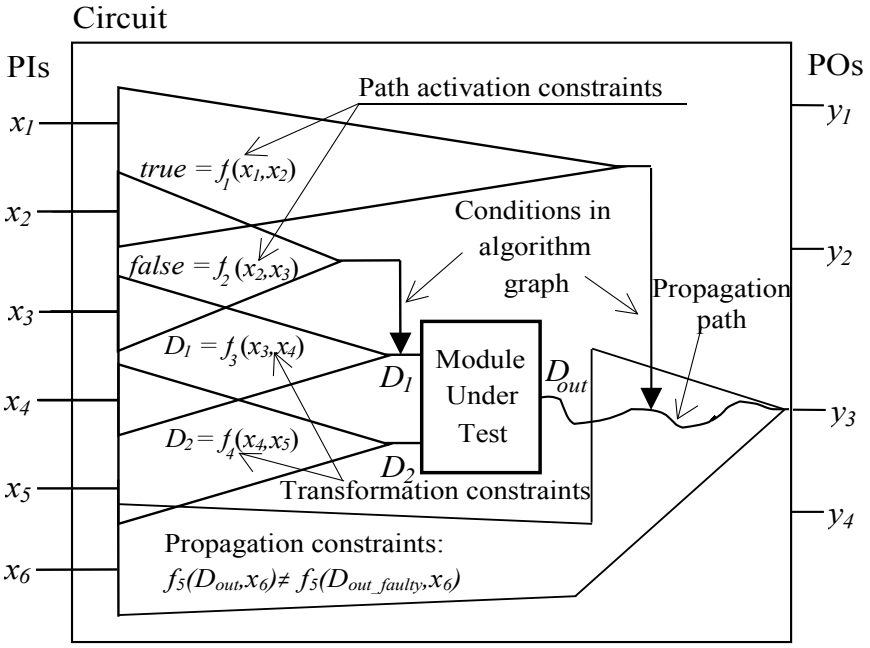


Fig. 5.1. An example of test generation constraints

In the following, the data structure and update operations of high-level test generation constraints are defined.

Definition 1: A condition C in the form $S = g(x)$, where S is an integer, Boolean or symbolic value, and $g(x)$ is an expression on a subset of variables of the model representing the system under test, is referred to as constraint.

In current approach, symbolic values that can be used for S in a constraint $S = g(x)$ are D_i and D_{out} which correspond to the values of the i-th input and the output of the current Module Under Test (MUT), respectively (See Figure 5.1).

Definition 2: Constraint $S = g(x)$ is said to be *justified* if $x \subseteq x_I$, where x_I is the set of primary inputs of the system. Otherwise, $S = g(x)$ is an *unjustified* constraint (See Section 3.4) .

If a constraint $S = g(x)$ is unjustified then all the variables in the set $x^U \subseteq x$ that are not input variables x_I are said to be *unjustified variables* of the constraint.

Definition 3: Let x^J be the set of justified variables and x^U be the set of unjustified variables of a constraint $S = g(x^J, x^U)$.

The process, where each variable $x^U{}_i$ is substituted by expressions on model variables $x'_i \subseteq x$, is refered to as *updating the constraint* $S = g(x^J, x^U)$ and it creates a new constraint $S' = g'(x^J, x')$, where g' can be regarded as a superposition of functions on a set of variables in the system model representation. Section 5.1.3 presents an example of constraint update in test path activation.

Note, that justified constraints consist of operations on primary inputs x_I and constants x_C (see Section 3.4). Furthermore, the exponential size complexity of the constraints $g(x)$ is avoided by uniting multiple occurrences of the same variable (i.e. the literals) in the constraints at each time step into one single fanout variable. Because of this, the size requirements for the constraints are linear with respect to justification time-frames and they represent a small subset of the expanded time-frame model of the circuit. Thus, the high-level test constraint extraction procedure is scalable.

5.1.2 Deterministic Test Path Activation

The high-level symbolic path activation, proposed in current thesis is a complete algorithm, i.e. if transparent paths for fault effect propagation and value justification exist, they will be activated. The algorithm has been implemented as a systematic search and therefore an inconsistency in any stage causes a backtrack and a return to the last decision. The general test generation flow is presented in Figure 5.2 [29] , [38].

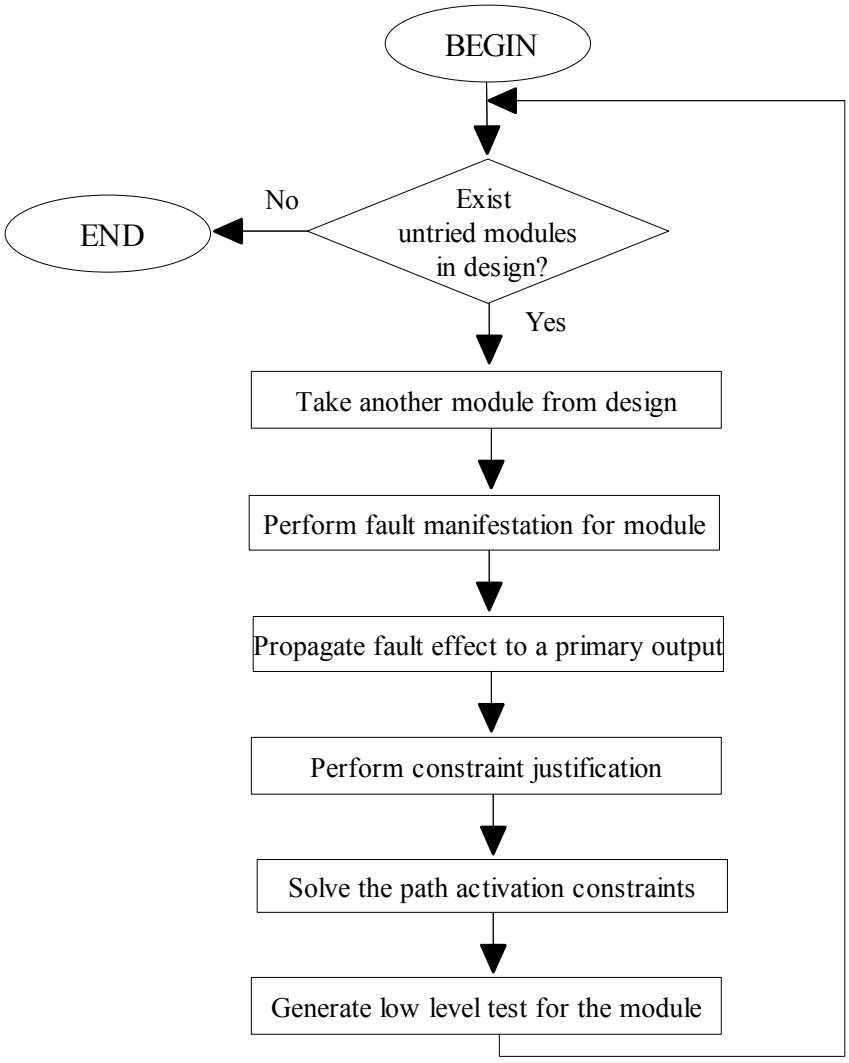


Figure 5.2 The general flow of the hierarchical test generation algoritm

In the following the propagation and justification principles of the proposed RT level ATPG are presented.

5.1.2.1 Fault Effect Propagation

The purpose of the propagation procedure is to activate a state sequence that propagates the fault effect from the output of the module under test to one of the primary outputs of the design. In current approach, propagation along single path is implemented. In order to keep track of the fault effect propagation a dedicated fault effect pointer is used. During propagation, high-level test path activation constraints are created. Fig. 5.3, presents the algorithm for fault effect propagation. In the algorithm descriptions the term consistent FSM control vector is frequently used. By this term we mean a control vector (row) in the control part's FSM state table whose control signal values are consistent with value assignments made for control signals while propagating (activating) paths in the datapath.

```
/* Fault manifestation for module M */
Create constraints from all the module inputs input_i(M)
Set fault effect pointer to node output(M)
/* Fault effect propagation */
While fault pointer is not propagated to a primary output
Let a be the node pointed by fault effect pointer
Choose the most observable fanout branch of a
  Set control signals required to transport fault effect from the
    fanout branch to the next fanout stem or register node b
    /* always only one such path exists! */
 Set fault effect pointer to b
 If exists a consistent FSM control vector then
    Choose the most observable consistent control vector
    Create constraints of corresponding FSM input vector
    If b is a register then
      move to the next time-frame
    Endif
 Else
    Backtrack
 Endif
 Endwhile
```

Figure 5.3 Fault effect propagation algorithm

5.1.2.2 Constraint Justification

Subsequent to propagation, constraint justification starts. Justification moves backward in time, starting from the clock-cycle, where propagation ended. During this process existing constraints are updated and additional path activation constraints are created. Finally, constraints solving procedure is applied to the extracted constraints and module under test is fault simulated by constraint-driven, local test data.

Nodes of the circuit, which correspond to primary inputs x_I or constants x_C are called justified nodes. All other nodes are said to be unjustified. Constraints containing unjustified nodes are referred to as unjustified constraints.

Justification step: first select previous state. Then update constraints according to control vector of this control state.

Updating the test generation constraints is defined in Section 5.1.1 and shown in more detail on an example presented in Section 5.1.3. Basically, updating a constraint can be regarded as superposition of the unjustified nodes of the constraint by new datapath nodes determined by paths activated in the datapath by current control vector.

At each justification step, current justification objective is chosen. In the proposed algorithm implementation the justification objective is to justify the first unjustified node from the first unjustified constraint. The algorithm for constraint justification is presented in Fig 5.4.

```
/* Constraint justification */
While exist unjustified constraints
If current time-frame is earlier than manifestation then
  Let current objective be to justify node b
  Choose the most controllable fanout, F.U. or register node a,
            which directly precedes b
  Set control signals activating path from a to b
  /* always only one such path exists! */
   If exists a consistent FSM control vector then
      Choose the most controllable consistent control vector
      Create constraints of corresponding FSM input vector
      If a is a register then
        move to the previous time-frame
      Endif
   Else
    Backtrack
   Endif
  Else
    Move to the previous time-frame
  Endif
  Update all active constraints
  Endwhile
/* Solve constraints (Section 5.1.4) */
```

Figure 5.4 Constraint justification algorithm

5.1.3 Constraint Extraction Example

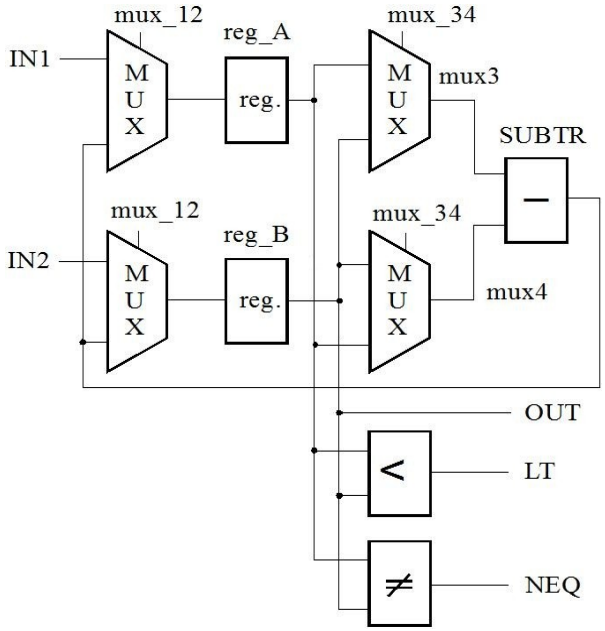
In the following, the test path activation algorithm and constraint extraction is explained basing on the example of the Greatest Common Divisor (GCD). Consider the GCD algorithm described at behavioral level in a pseudo hardware description language:

```
A := IN1;
    B := IN2;
    while (A ≠ B)
    if (A < B) then
```

```
B := B – A;
else
A := A – B;
end if;
end while;
OUT := A;
```

Let us assume that subsequent to applying high-level synthesis to the algorithm description we obtain the RTL architecture presented in Figure 5.5 This architecture consists of a datapath of 3 Functional Units (FU), 2 registers and 4 multiplexers and a control part Finite State Machine (FSM) of four states. The datapath architecture is depicted in Figure 5.5a and the control part is given as a state table in Figure 5.5b, respectively.

a)



b)

RESET	LT	NEQ	pres. state	next state	A_enable	B_enable	mux_12	mux_34
1	X	X	X	S_0	1	1	0	X
0	X	1	S_0	S_1	0	0	X	X
0	X	0	S_0	S_0	0	0	X	X
0	1	X	S_1	S_2	0	0	X	X
0	0	X	S_1	S_3	0	0	X	X
0	X	X	S_2	S_0	0	1	1	1
0	X	X	S_3	S_0	1	0	1	0

Figure 5.5 RT-level architecture of the GCD circuit

Let us explain the test generation algorithm described in Section 5.1.2 by the example of generating test paths for the module *SUBTR*.

Fault manifestation. Set all the variables to 'don't care' values. Create transformation constraints D_0=mux3, D_1=mux4. Set the fault effect pointer to variable SUBTR, i.e. y_D := SUBTR.

Fault effect propagation. Choose a datapath register that reads from the FU SUBTR. There are two possible choices: reg_A and reg_B, respectively. Let us select the first choice. Subsequently, we activate the path from SUBTR to reg_A, which results in the following variable assignments: A_enable := 1, mux_12 := 1.

Next, we have to choose a consistent FSM control vector. The only vector consistent with previous variable assignments is the one corresponding to row 7 in

the FSM state table (labeled by vector 0, X, X, S3, S0, 1, 0, 1, 0). Based on this vector we obtain the following assignments: reset:=0, B_enable := 0, mux_34 := 0, state := S3 (in current clock cycle), state := S0 (in the next clock cycle). We move to the next clock cycle and set the fault effect pointer y_D to reg_A (i.e. OUT).

We detect that the fault effect pointer points to a variable corresponding to a primary output and thus have successfully completed the fault propagation process.

Constraints justification. As there were no path activation constraints created during manifestation and propagation stages, we move backwards in terms of clock-cycles until the clock-cycle of manifestation phase is reached. We select the justification objective from the unjustified variables of the transformation constraints (D_0=mux3, D_1=mux4). Let current objective be to justify variable mux3. Due to the fact that we have already assigned mux_34 := 0 at current clock-cycle during the propagation process, then we have no choice but backtracing mux3 to reg_A. We update the constraints, obtaining D_0= reg_A, D_1= reg_B and move to the preceding clock cycle.

Without focusing on further details, we continue executing the constraint justification algorithm until the path presented in Figure 5.6 is activated as one of possible high-level path solutions.

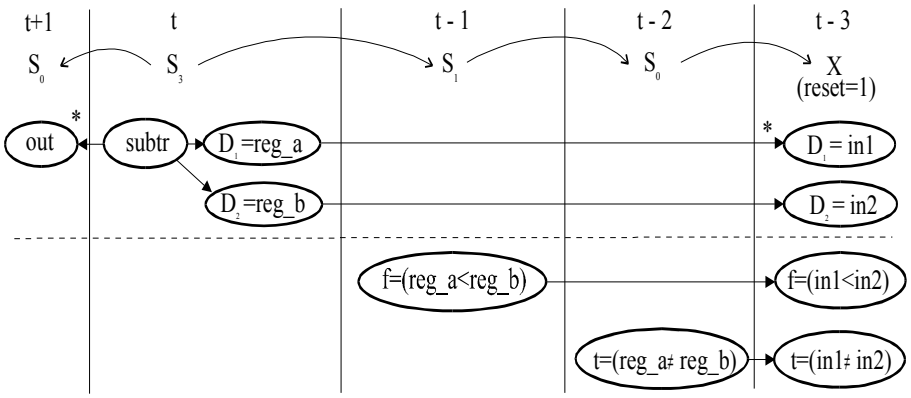


Figure 5.6 High level path activation example

In the Figure we have denoted the manifestation clock cycle by t, the i-th cycle following t is denoted by t+i and i-th cycle preceding t is denoted by t-i,

respectively. Below the clock-cycle information, the activated state sequence is provided. Then we present graphically the processes of fault propagation and extraction of transformation constraints. Decisions in the high-level path activation are marked by stars (*) in the Figure. Extraction of path activation constraints is depicted below the striped line. Here, t corresponds to Boolean value 'true' and f corresponds to 'false'. As shown in Figure 5.6 we have to apply the constraint satisfaction process to the following set of constraints: in1 < in2 is false, in1 ≠ in2 is true.

Subsequent to testing the node with the first path, backtrack occurs and the high-level path activation algorithm tries to find alternative path solutions.

5.1.4 Solving the Test Path Constraints

In the previous top-down test pattern generation algorithms by the authors [12, 13], random constraint solving was applied. In this research we have selected the open source ECLiPSe constraint solver (ECLiPSe5.10_41) to solve the test path constraints. ECLiPSe supports most of the common techniques used in solving constraint problems. It includes constraint programming, mathematical programming, local search and various combinations of the above. We have embedded the solver into the C++ code of the ATPG and use the string-based input.

As experiments presented in the following Section show the deterministic constraint solving has definite advantages over the pseudo-random method.

5.1.5 Experimental Results

In order to evaluate the impact of the deterministic constraint solving experiments on ITC99 and HLSynth92/95 benchmarks were carried out. By this moment we have included the following three circuits into the analysis: b00, 604 and gcd because these circuits contain "equal to" comparison operators which are hard to test by pseudorandom constraint solving.

Table 5.1 shows the comparison of the semi-formal approach DECIDER presented in [12] and the proposed top-down tool. Comparison has been obtained by fault simulating the test sets generated by both generators by a stuck-at fault simulator for sequential circuits. The row '# faults' of the Table shows the number of stuck-at faults in the circuit. The row '# tested' presents the number of tested faults by [12] and the proposed approach. The row 'cover., %' lists the achieved

stuck-at fault coverages. ‘time, s’ stands for the ATPG run times in seconds. Finally, the number of generated test vectors is reported in the row ‘# vect.’

Table 5.1 Comparison of semi-formal [12] and the proposed deterministic ATPG methods*

b00		b04		gcd		
[12]	**current**	**[12]**	**current**	**[12]**	**current**	
1328	1328	1488	1488	1658	1658	**# faults**
251	714	899	943	1443	1519	**# tested**
18.90	53.77	60.42	63.37	87.03	91.62	**cover, %**
0.0053	0.0044	0.002	0.011	2.72	0.02	**time, s**
534	874	574	572	4471	4756	**# vect.**

*** Note: Results in Table 5.1, Table 6.5 from Section 6.2.5 and Table 5.2 from Section 5.2.2 does not match because benchmarks were run with different synthesis tool using different options.**

It can be seen that the fault coverage improvement obtained by the deterministic constraint solving setup ranges from 3 to 34 % for the tested examples. Note, that while the fault coverages for the circuits are low, this is a usual case for the sequential ATPG because of the large number of untestable faults.

5.2 RT-Level Identification of Potentially Testable Initialization Faults

The third paper that is called "**RT-Level Identification of Potentially Testable Initialization Faults**" [III] addresses the problem of an important subclass of faults, the potentially detectable initialization faults. The authors of the paper were Jaan Raik, Hideo Fujiwara and the author of this thesis. It was presented at the Ninth IEEE Workshop on RTL and High Level Testing (WRTLT' 08) in Sapporo, Japan in November 2008.

The goal of paper was to propose high-level identification of potentially testable initialization faults. Experiments presented in the paper show that potentially detectable initialization faults form a large subset of all the faults not testable by hard-detection. As a result of the proposed approach, both, the speed as well as the confidence level of sequential ATPG can be increased.

In sequential Automated Test Pattern Generation (ATPG) based on a three-valued algebra (0,1,X) a fault is said to be hard-detected if a fault effect (0/1 or 1/0) appears at a primary output. However, not all the faults can be tested by such hard-detection model. Many faults belonging to the class of initialization faults are known to be covered only by resorting to potential detection (effect 0/X or 1/X). Existing high-level fault models assume hard-detection and therefore are not capable of handling the initialization faults. This means wasting test generation time also for those faults that may only be detected potentially.

5.2.1 Basic Definitions

Sequential ATPG and fault simulation typically relies on the 3-valued logic algebra 0, 1, and X, where X is an artificial logic value to represent the unknown or don't-care state.

Definition 1: A fault f is said to be hard-testable iff for this fault a fault effect (0/1 or 1/0) can be propagated to a primary output.

Definition 2: A fault f is only potentially testable iff it is not hard-testable and for this fault either 1/X (i.e., the fault-free value is 1 and the faulty value is unknown X) or 0/X can be propagated to a primary output.

Let us denote the set of all stuck-at faults by A, the set of hard-testable faults by D and the set of potentially testable faults by P. Relations between these three sets

is presented in Fig. 5.7. The goal of the method proposed in current thesis is to increase the fault efficiency of high-level fault models by identifying potentially testable faults from RTL. The area of faults identified by current method is depicted by the dashed circle in the Figure.

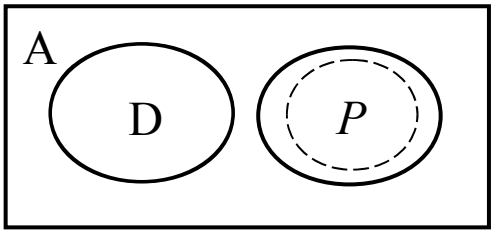


Figure 5.7 Relations between fault classes

5.2.2 Experimental Analysis of Fault Classes

Table 5.2 presents the experimental analysis of four sequential designs. The benchmarks were chosen from the HLSynth92 and HLSynth95 families and they were synthesized to RT-level from behavioral VHDL descriptions using the high-level synthesis tool SYNT from Synthesia. Subsequently, the RTL descriptions were synthesized to logic-level by Synopsys Design Compiler. The circuits were tested by two sequential ATPG tools: a simulation-based ATPG SBGEN [28] and a hierarchical ATPG DECIDER [29].

In the Table, the rows have the following meaning. Row ‘total faults’ shows the number of stuck-at faults in the circuit. Row ‘hard-detected’ gives the number of faults that were covered according to the hard-detection model. Row ‘potential-detect.’ presents the number of potentially detected faults covered by the sequential ATPG tests. This result was obtained by running a sequential stuck-at fault simulator. Row ‘uncontr./unobs.’ stands for the sum of uncontrollable and unobservable faults. These are faults, which are caused by constant inputs and unconnected gate outputs, respectively. This type of faults is very easy to identify and they are reported by most commercial and academic fault simulators. Row ‘reg. untestable’ stands for a special class of register control faults, which can be proved untestable from the RT-level as shown in [IV]. Row ‘other’ includes all the remaining faults.

Table 5.2 Fault distribution in sequential designs

Circuit	GCD	SOSQ	MULT	DIFFEQ
total faults	1760	2130	2242	10326
hard-detected	1569	1514	1417	9853
potential-detect.	**16**	**181**	**117**	**14**
uncontr./unobs.	98	275	505	320
reg. untestable	65	130	130	130
other	12	30	73	9
fault efficiency	99.32	98.59	96.74	99.91

We can make the following conclusions based on the fault distribution shown in Table 5.2. First, if we take into account the classes of uncontrollable/ unobservable, register untestable and potentially detected initialization faults then the calculated fault efficiency is high, ranging from 96.7 to nearly 100 per cent. However, since traditional high-level ATPG is not capable of identifying the untestable and the initialization faults the achieved confidence level in terms of fault efficiency would be very low. The goal of the current research is to extend RTL ATPG by potential detection capabilities in order to achieve higher fault efficiency.

5.2.3 RTL Detection of Initialization Faults

Potentially detectable initialization faults can be divided into three main groups: reset faults, control part faults and loop-counter faults. High-level detection of faults for all these groups will be discussed in more detail in this Section.

In order to present the RT-level initialization fault detection method let us introduce some definitions.

Definition 3: Registers that are either directly or through some combinational logic connected to primary outputs are refered to as the *output registers* of the design.

Definition 4: Let the control part state, which is set by activating the global reset signal be called *reset state* and the set of control signal assignments at this state be called *reset state control vector.*

Also let us assume for the sake of simplicity that the global reset signal is active high, i.e. reset=1 initializes the circuit state.

5.2.3.1 Reset Faults

First, consider the global reset signal. In order to potentially detect reset stuck-at-1 (s-a-1) fault we propose the following condition :

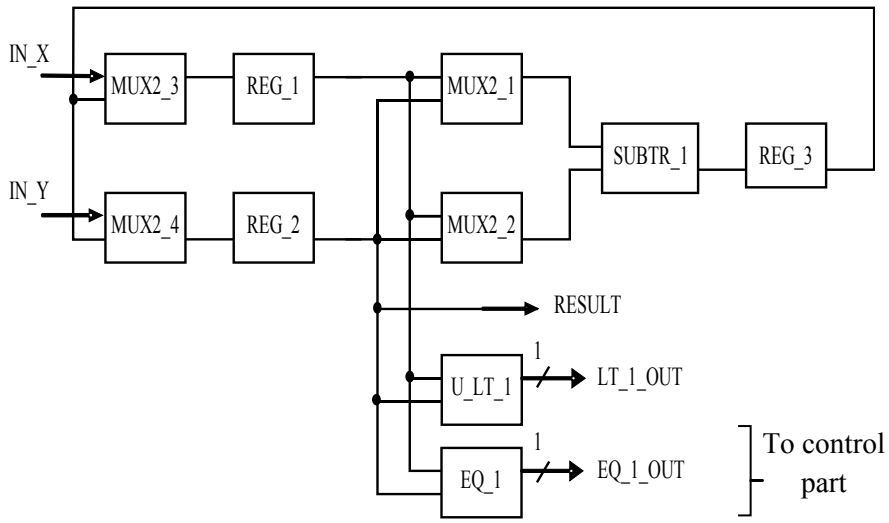
Condition 1:

- *Reset s-a-1 is potentially detectable if the control vector at the reset state neither resets nor enables any of the output registers.*

We need to check the presence or absence of register reset at all of the output registers in order to make sure that the global reset s-a-1 fault does not belong into the fault class D (See Section 5.2.1!). The condition also requires that the reset state control vector disables all the output registers, i.e. their corresponding enable signals are set to the value 0. This blocks the possibility to initialize any output register and, thus, guarantees potential detectability of reset s-a-1 fault.

Consider the RTL architecture of the Greatest Common Divisor (GCD) example shown in Figure 5.8. Fig. 5.8a presents the datapath, which contains only one output register REG_2. The first row in the state table in Fig 5.8.b shows the reset state control vector for the circuit. As it can be seen, REG_2 is not a resettable register. So, first part of Condition 1 holds. Also the second part holds because REG_2 is disabled in the reset state (Reg_2_Enable = 0). Thus, the fault Reset s-a-1 is potentially detectable in the GCD circuit.

a)



b)

RESET	EQ_1_OUTPUT	LT_1_OUTPUT	Present State	Next State	Mux_12_Address	Mux_34_Address	Reg_1_Enable	Reg_2_Enable	Reg_3_Enable
1	x	x	X	S_0	**x**	**x**	**0**	**0**	**0**
0	x	x	S_0	S_1	x	0	1	1	0
0	0	x	S_1	S_2	x	x	0	0	0
...	...	...	...	...	...	...	...	...	...

Figure 5.8 a) Datapath and b) reset state control vector

Now let us introduce the condition for identifying the fault Reset s-a-0 potentially detectable from the RTL.With Reset s-a-0 fault the control state takes a don't-care value X. It means that any control vector is valid, except the reset state one. If for each output register there exists such control vector, where it is disabled then none of these registers can be controlled using the 3-valued algebra and the value of output registers will also be X.

Condition 2:

- *Reset s-a-0 is potentially detectable if for all the output registers there exists a non-reset-state control vector where they are disabled.*

For example, the third control vector of the FSM table in Figure 5.8b disables the output register REG_2 at the same time when Reset=0. Since the value of the state register is unknown we can conclude that the value of REG_2 must also be unknown. Thus, the fault Reset s-a-0 is only potentially detectable in the GCD example.

5.2.3.2 Control Part Faults

Similar to initialization faults at the global reset there may also be potentially detectable faults in the signals of the control part FSM. For example, a stuck-at fault at a single bit in the state register may prevent initialization of the output register, etc. The RTL signals, where potentially detectable faults have to be considered include:

- control signals (FSM outputs)
- state register bits
- status bits (FSM inputs)

Let us consider each of the three cases.

Control signals. Control signals enter from the control part into the datapath and are partitioned to register enable signals and multiplexer address selects. The values for these signals are determined by the current control state and primary inputs of the design.

At the RT-level, it is possible to potentially detect s-a-0 faults at the enable signals of the datapath registers by checking the following simple condition:

Condition 3:

- *Register enable signal s-a-0 is potentially detectable if the register is not resettable.*

In other words, enable signal s-a-0 faults at the non-resettable registers are always (!) potentially detectable. This is due to the fact that disabling an output register by setting its enable s-a-0 does not allow initialization of this register and, thus, constantly holds the value X in it.

Stuck-at-1 faults at register enable signals are either hard-detectable or untestable (See [IV, VI]).

State register bits. Stuck-at fault at the bits of the control part state register can be identified untestable if the coding of the control part FSM is known. In that case, a fault at a state register bit converts the fault-free FSM into a faulty one. In the case it will introduce illegal states (i.e. state values not present in the fault-free FSM) the fault cannot be detectable at the RTL. This is due to the fact that control vectors for illegal states are unknown at the RT-level while they have determined values at the logic-level. However, in the case if the faulty FSM introduces no illegal states then the following condition can be applied:

Condition 4:

A state register bit s-a fault is potentially detectable if its corresponding faulty FSM does include neither illegal states nor reachable states loading the output registers.

Consider the example FSM shown in Fig. 5.9a. Bold circle denotes the reset state, during two of the state transitions, its output register is loaded. Now let us see the case when the least significant bit of the state register has the fault s-a-1. In that case a faulty FSM presented in Fig. 5.9b will result. In this FSM the output register cannot be loaded. Thus, the state register bit fault is only potentially detectable. However, if the second bit of the state is s-a-1 (See Fig. 5.9c) then the faulty FSM will include a faulty state „110” and we cannot show potential detectability of this fault from the RT-level.

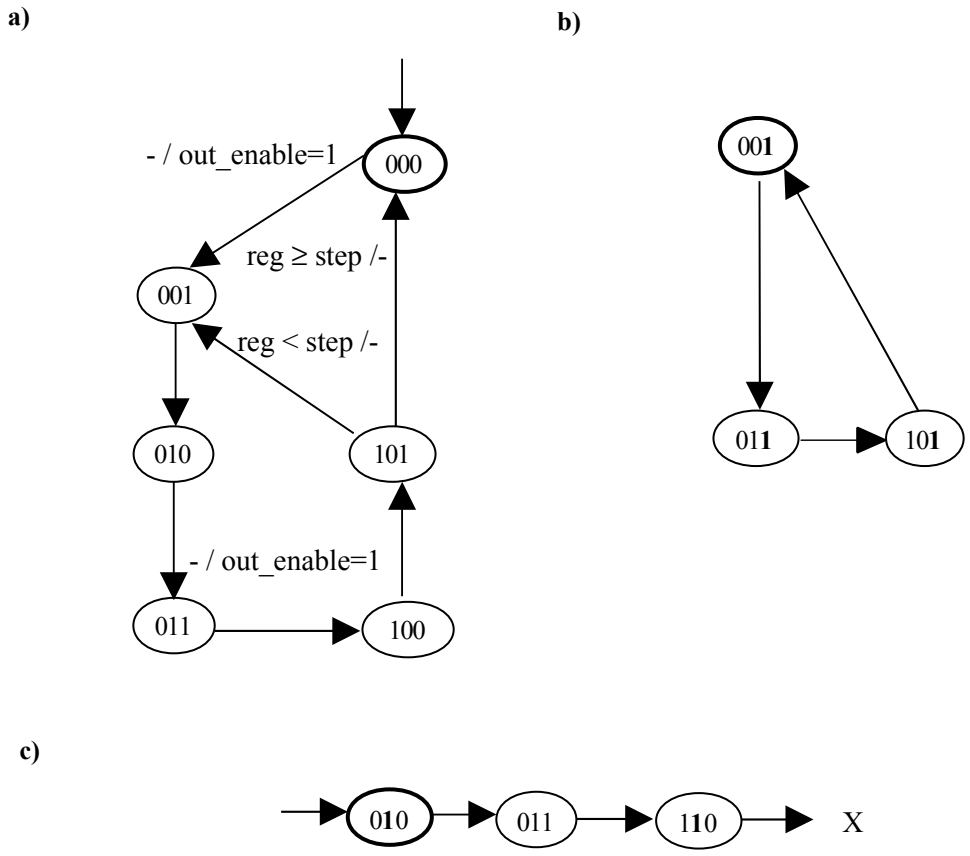


Figure 5.9 a) A fault-free FSM, b) potentially detectable state bit fault, c) faulty FSM containing illegal states

Status bits. The status bits enter from the datapath into the control part FSM. These signals represent the results of comparison operations and they control the selection of state transitions in the FSM. For example, the result of the comparison 'reg < step' is a status bit for the FSM in Fig. 5.9a.

Similar to state register bit faults, in case of stuck-at faults at status bits a faulty FSM will result where some of the branches will be excluded. Also, some of the legal states may become unreachable. However, illegal states can not result

because of status bits faults. Otherwise the condition for potential detectability of status bit faults is identical to Condition 4.

5.2.3.3. Loop-counter Faults

Loop-counters are blocks in RTL designs whose role is to implement fixed-length loops of the algorithm realized by the circuit. Output of a loop counter is a status bit (output of a comparison operator). Thus, identification of which loop-counters contain potentially testable faults is exactly identical to proving the potential testability for status bits.

5.3 Chapter Summary

The theoretical contribution of this Chapter in studying of a test pattern generation for sequential circuit was presented in Papers: "**Constraint-based Test Pattern Generation at the Register-Transfer Level**" [II] and "**RT-Level Identification of Potentially Testable Initialization Faults**" [III].

The paper [II] introduced a novel constraint-based automated test pattern generator for Register-Transfer Level (RTL) designs. The tool combines test path constraint activation with a constraint solver.

First, a deterministic algorithm that extracts constraints for activating test paths at RTL is applied. Subsequently, a constraint solving package ECLiPSe is used for assembling the tests. Experiments on ITC99 and HLSynth92/95 benchmarks show that the proposed deterministic method offers very short run times. In particular, it provides increased fault coverage which ranges from 3 to 34 % for the tested examples with respect to earlier approaches.

While the fault coverages for the circuits are low, this is a usual case for the sequential ATPG because of the large number of untestable faults. As a future work we plan to integrate untestable fault analysis for sequential circuits (will be considered in the Chapter 6 (e.g. [V]) into the constraint-based ATPG to improve fault efficiency estimation.

The paper [III] presented a new method for high-level identification of potentially testable initialization faults.

Existing high-level fault models assume hard-detection and therefore are not capable of handling such initialization faults. Furthermore, three important classes of initialization faults were identified in the thesis: reset faults, control part faults and loop-counter faults. High-level methods for potential detection of faults of the respective classes were proposed.

Experiments presented in this thesis show that potentially detectable initialization faults form a large subset of all the faults not testable by hard-detection. As a result of the proposed approach, both, the speed as well as the confidence level of sequential ATPG in terms of higher fault efficiency can be increased.

We plan to implement the potential fault detection method and include the capabilities to an RTL test pattern generator.

In each direction of the research, new appreciable results were achieved. The results were presented at international conferences.

Chapter 6

PROVING UNTESTABLE FAULTS IN SEQUENTIAL CIRCUITS AT RTL

The theoretical contribution of the second part of this Chapter in identifying untestable faults in sequential circuits is presented in Paper IV and in Paper V.

The Paper IV proposes a new approach of applying model-checking for detecting untestable stuck-at faults at the register-transfer level. The Paper V considers register-transfer level (RTL) test pattern generation for non-scan sequential circuits containing feedback loops.

6.1 Untestable Fault Identification in Sequential Circuits Using Model-Checking

The fourth paper titled **"Untestable Fault Identification in Sequential Circuits Using Model-Checking"**. However, the intermediate steps of research were published in Paper **[VI] Hierarchical Identification of Untestable Faults in Sequential Circuits**. We have selected only Paper [IV] that contain the most important achievements and continues to improve the technique proposed in Paper [VI].

Two novelties are introduced: a new approach of applying model-checking for detecting untestable stuck-at faults at the register-transfer level and a method of generating PSL language assertions for proving untestable register stuck-on faults.

The authors of the paper were Jaan Raik, Hideo Fujiwara, Raimund Ubar and the author of this thesis. The paper was presented at The Seventeenth Asian Test Symposium (ATS' 08) in Sapporo, Japan in November 2008.

In their previous work [VI], authors introduced a new subclass of untestable faults, called register input logic stuck-on faults and that it is possible to identify such faults from the register-transfer level (RTL) description of the circuit. Authors pointed out their relation to untestable gate-level stuck-at faults. Moreover, experiments show that a large subset of faults not tested by sequential ATPG fall into this category. However, the paper did not propose any formal method for identifying untestable register faults. In the paper [IV] we present a new method that is capable of identifying such type of untestable faults. We propose using model-checking for detecting untestable stuck-at faults at the Register-Transfer Level (RTL). In particular, we present a method for formally generating PSL language assertions for proving untestable stuck-at faults in sequential synchronous designs.

Experiments show that the faults identified by the method form in fact a large subset of all the untested stuck-at faults. An additional application of the method is in high-level test synthesis, where testability of sequential designs can be improved simultaneously with minimization of the circuit area.

6.1.1 Introduction

Test generation for sequential synchronous designs is a time-consuming task. Automated Test Pattern Generation (ATPG) tools spend a lot of effort not only for deriving test vectors for testable faults but also for proving that there exist no tests for the untestable faults. Because of this reason, the identification of untestable faults has been an important aspect in speeding up the sequential ATPG. The methods proposed previously are based on performing static and dynamic implications at the logic-level. Current thesis presents an approach that takes the problem of identifying untestable faults one step further: to the higher abstraction levels. We show that it is possible to very quickly find a large subset of all untestable faults before handing the untestability identification over to classical, logic-level methods.

6.1.2 Motivation for Targeting Register Faults

A special case of datapaths where register enable signals are redundant is a pipeline. In pipelines data is transported during each clock-cycle and therefore the

registers should be constantly enabled. Enable signals in pipelines are normally omitted and the registers are replaced by buffers consisting of D-flipflops.

However, there are other cases than pure pipelines, where the redundancy of enable signals is much more difficult to identify. Consider for example the Extended Finite State Machine (EFSM) representation of the Differential Equation (diffeq) benchmark shown in Fig. 6.1 In this kind of EFSM description, the nodes represent control states and the arrows represent transitions between the states. Shown on the transitions are the enabling functions (on top of the line), i.e. conditions that enable the state transition, and the update functions (below the line) that correspond to datapath register assignments.

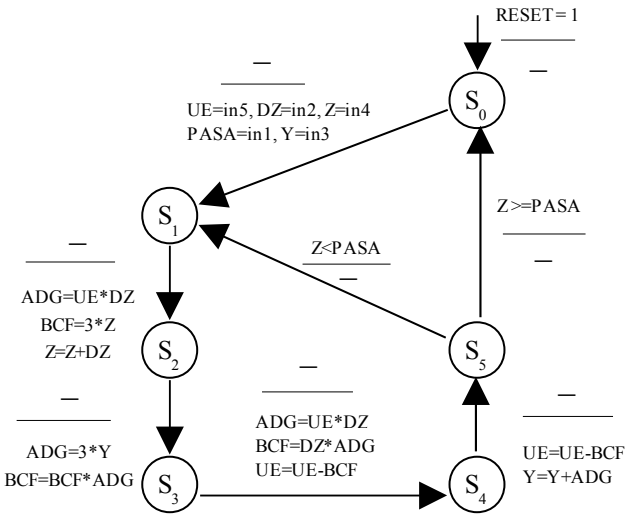


Figure 6.1 EFSM of the Diffeq benchmark

Let us focus on register ADG (in Fig. 6.1). It can be seen that this register reads during transitions $s_1 \rightarrow s_2$, $s_2 \rightarrow s_3$, $s_3 \rightarrow s_4$. It can also be seen that ADG is in turn an input for two other registers: BCF and Y (shown by grey background). The latter read ADG only during transitions $s_2 \rightarrow s_3$, $s_3 \rightarrow s_4$, $s_4 \rightarrow s_5$. Now let us assume that the enable signal of register ADG is permanently stuck on. In that case, ADG may read faulty values except between the state transitions $s_1 ... s_4$ when it is also enabled in the fault-free circuit. Note however that ADG is read always one transition later, i.e. between $s_2 ... s_5$. Thus, only fault-free values can be read from ADG and the

stuck-on fault of its enable signal is untestable. On the other hand, as an opposite example, enable in register DZ is testable because DZ is read at $s_1 \rightarrow s_2$ but DZ reads no value during one of the preceeding transitions: $s_5 \rightarrow s_1$.

The goal of this research is to introduce a formal technique for identifying such kind of untestable stuck-at faults from the RT-level. The method presented in this thesis not only allows untestable fault identification but it can also be implemented in high-level test synthesis [35-37].

In the following, conditions that are sufficient for identifying untestable faults in register enables are introduced. Later on we implement the untestable fault analysis relying on standard model-checking tools. Finally, we carry out experiments on RTL benchmarks in order to assess the relevance of register enable faults among the untestable faults in sequential designs and evaluate the efficiency of the proposed method in untestability identification.

6.1.3 Register-Transfer Level Architecture

Let us first consider the general architecture of register-transfer level (RTL) circuits. In RTL descriptions the design is partitioned into a control part (FSM) and a datapath part. The latter consists of registers R, multiplexers M and functional units (FUs) F. The former includes a state register for preserving the control state s_j from the set of states S. The set of control signals C enter from the control part into the datapath and are partitioned to register enable signals E and multiplexer address selects A. The control signals $C=E \cup A$ are determined by the current control state $s_j \in S$. The status bits B enter from the datapath into the control part FSM. These signals represent the results of comparison FUs and they facilitate the selection of state transitions in the FSM.

When a behavioral or behavioral RTL circuit is synthesized into RTL then the following two main steps are carried out by the high-level synthesis tool: 1) allocation of time-steps for operations, 2) binding of operations and variables into hardware resources: FUs, registers and multiplexers. Depending on the constraints given to the synthesis tool it may try to bind several operations into the same FU or a number of variables into the same register. At different time-steps registers obtain values from different sources (other registers, FUs or primary inputs). Thus, multiplexers to be controlled by the control part are created to select the correct source at each moment.

The general case for RTL datapaths is thus, a mux-operation-mux-register form (See example in Fig. 6.2). In other words, when moving from one register $r_{src} \in R$ to

register r_{dst} we may pass through an FU $f \in F$ whose inputs may be selected by multiplexers $M_{in} \subset M$ and we may also need an additional multiplexer $m_{out} \in M$ to allow the target register r_{dst} read from different sources. Reading new data into registers $r_i \in R$ is controlled by the control part FSM via register enable signals $e_i \in E$. Enable signals e_i are activated (i.e. e_i=1) only when the corresponding registers r_i perform a new read operation, otherwise the enable is deactivated (e_i=0). Register may also include a global reset input.

Selecting between different sources is controlled by the multiplexers $m_k \in M$ whose address signals $a_k \in A$ enter from the control part. During these states when register reads new data its multiplexer address value is specified to select the correct source. At any other state the value of the mux address is normally unspecified and this fact makes the untestability analysis of gate-level stuck-at faults from the RT-level difficult. However, in the following Section we propose a property, which allows identification of a large number of untestable faults without knowing the exact logic implementation of the control part.

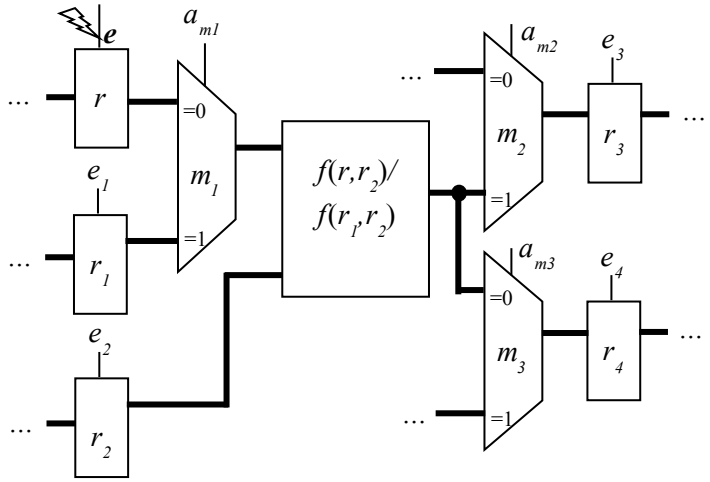


Figure 6.2 RTL datapath fragment

6.1.4 Identifying Untestable Registers

In this Section, we present a property for proving untestable register stuck-on faults implementing a commercial model-checking engine. The analysis is carried out at the register transfer level, and the untestability of control signals is formally calculated.

Let us introduce some preliminary definitions.

Definition 1: For any datapath register *r* the registers r_i whose inputs are reachable from *r* through combinational logic (multiplexers and FUs) are refered to as the *guarding registers* of register *r*. For example, the guarding registers of register *r* in Figure 6.2 are r_3 and r_4. Note, that with the presence of feedback loops register *r* itself may belong to its guarding registers r_i.

Definition 2: If the address signals a_k of multiplexers m_k are set to values that activate a path between two datapath registers r_1 and r_2 we say that the path activation condition between r_1 and r_2 holds and denote it by $\alpha_{r1,r2}=1$. Otherwise, $\alpha_{r1,r2}=0$.

For example, in Figure 6.2 the path between registers r and r_3 is selected only if the mux address signals $a_{m1}=0$ and $a_{m2}=1$. Thus, $\alpha_{r,r3}=\overline{a_{m1}} \cdot a_{m2}$.

Definition 3: Let us refer to the set of states from where a control state $s_j \in S$ can be reached within one clock-cycle as immediately preceding states of s_j. Let us denote immediately preceding states of s_j by prev(s_j).

Throughout this thesis we use the superscript notation to show at which state the signal values will be considered. For example, the value of a datapath signal v at the state s_j is denoted by v^{sj}.

Theorem 1: Let *e* be an enable signal controling a datapath register *r*, let s_j, $j=1,...,n$, $n=|S|$ be the set of control states and r_i, $i=1,...,m$ be the set of guarding registers for *r*.

If $\forall_{j=1...n} \forall_{i=1...m} e_i^{s_j} \cdot \alpha_{r,r_i}^{s_j} \rightarrow e^{prev(s_j)}$ then the register enable signal *e* stuck-at-1 fault is untestable.

In other words, the sufficient condition for untestability of the fault *e* stuck-at-1 is that for all the states s_j where a guarding register r_i (enabled by e_i) is reading from *r* (enabled by *e*) all the immediatly preceding states of s_j write values to *r*.

Proof: If a faulty value from register r is to be propagated to any observable output then it has to be transported via one of the guarding registers r_i. Any guarding register r_i can read the fault value only at those states s_j where $e_i^{s_j} = 1$. Thus, at the states where $e_i^{s_j} = 0$ the faulty value of r can not propagate. Furthermore, if the enable signal e_i of r_i is activated then exactly one activation condition α_r*,r_i , where r* is r or any other register that can be read by r_i, must be equal to 1 (See Section 6.1.3 for the definition of RTL architecture!). It is clear that if r* is not r then the faulty value will not propagate to r_i at the current state s_j. Thus, the prerequisite for fault propagation to a guarding register r_i at the state s_j is $e_i \cdot \alpha_{r,ri}=1$.

However, if this prerequisite is fulfilled but the register *r* is enabled at all the states *prev*(s_j) then it will contain only the fault-free value at *prev*(s_j). Thus, the fault e stuck-at-one can not be tested. ■

Note, that the property for register untestability identification introduced in Theorem 1 is only a sufficient condition for the register to be untestable. There may exist untestable register enables that do not match this condition and therefore the property is somewhat pessimistic. However, its main advantage lies in the ease of computation by formal algorithms. Experimental analysis presented in Section 6.1.6 shows that in practice the method is well capable of proving untestability in different sequential benchmarks. It is also important to stress that all register enables identified by Theorem 1 are always stuck-at untestable at the logic-level.

6.1.5 Reducing Untestability Identification to Model-Checking

This Section will discuss the technical implementation of the RTL untestable fault identification method in VHDL and PSL using Cadence IFV 05.50 model-checker. We forwarded the condition from Theorem 1 to the model-checker. If the model-checker formally proves that the condition always holds for a register r then it can be concluded that the stuck-at-1 fault of its enable signal e is untestable.

The following VHDL code with embedded PSL constructs was generated and included to the VHDL architecture description of the Design Under Test (DUT) for untestability identification of register r:

```
PROPERTIES: if (ABV_ON) generate
    begin
     write_event_<r>  <= <e_i·α_r,ri> ;

    read_event_buffer:
      process
      begin
        wait until clock'event and clock = '1' ;
          read_event_<r>  <= <e>  ;
      end process read_event_buffer ;

    -- psl ASSERT_PSL_CHECK_<r> :
    -- assert always write_event_<r> -> read_event_<r>
    -- abort(reset);

    end generate PROPERTIES;
```

The VHDL signal write_event_<r> was introduced. The signal will be equal to one when some guarding register reads from r. A dedicated VHDL process read_event_buffer was introduced to detect the time-steps when fault-free values are read to r during the previous clock-cycle. Note, that the value of read_event_<r> is equal to e but there is a one cycle delay between two signals. It has been introduced in order to simplify the PSL assertion ASSERT_PSL_CHECK_<r> by allowing a combinational property (implication) to be checked.

There are special cases of registers, which are guarded not only by other datapath registers and thus, the signal write_event_<r> must be treated differently. For registers that are inputs for FUs that generate status bits B the signal write_event_<r> is assigned to value one during those states when B is read by the control part for selecting between alternative state transitions. Moreover, for registers connected to the primary outputs of DUT write_event_<r> must be constantly tied to one.

6.1.6 Impact of Register Faults at the Gate-Level

Let us consider the impact of an untestable register enable stuck-on fault at the gate-level. Fig. 6.3 presents a typical gate-level implementation of a single bit in a datapath register. The arrows mark the untestable stuck-at faults in the register *r* whose enable signal *e* is untestable. As it can be seen, an untestable register enable causes four additional stuck-at signals to be untestable in a register implementing and-or multiplexers. Thus a total number of untestable lines in a register with untestable enable signal is 4n + 1 (Four faults per bit plus the fanout stem of the enable *e*). In the case of 32-bit register the number of untestable stuck-at faults caused by a register stuck-on fault is as high as 129. Experimental results presented in the following Section show that a large subset of all the stuck-at faults not covered by the sequential ATPG belong in fact into this particular class of faults.

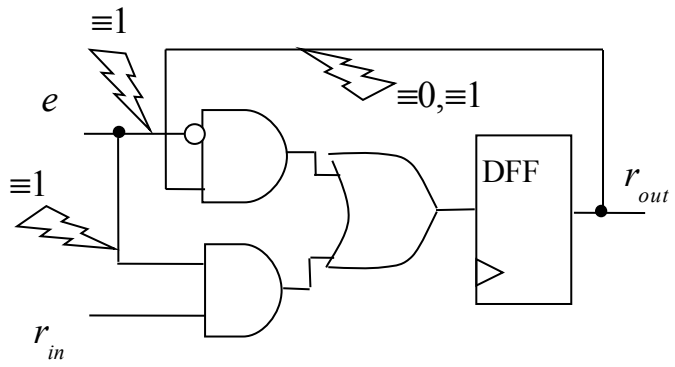


Figure 6.3 Gate-level impact of untestable e ≡ 1

6.1.7 Experimental Results

In Table 6.1, untestable fault identification experiments on four sequential designs are presented. The benchmarks were chosen from the HLSynth92 and HLSynth95 families and they were synthesized to RT-level from behavioral VHDL descriptions using the high-level synthesis tool SYNT from Synthesia. Subsequently, the RTL descriptions were synthesized to logic-level by Synopsys

Design Compiler. The same tool was applied for estimating the circuit area minimization by removal of untestable register enables.

Untestable fault identification was carried out with Cadence IFV model-checker on a SUN Sun-blade 100 Workstation with single 500 MHz UltraSPARC-IIe processor, 500 MB RAM, Solaris 2.9 OS. The circuits were tested by two sequential ATPG tools: a simulation based ATPG SBGEN [28] and a hierarchical ATPG DECIDER [29].

Table 6.1 Experimental results on identification of untestable faults

design	# faults	# tested	# untest.	# remain.	F.C., %	F.E., %	CPU time
gcd	1662	1564	65	33	94.10	98.01	2 min 56 s
sosq	1996	1514	130	352	75.85	82.36	4 min 09 s
mult8x8	2093	1417	130	546	67.70	73.91	3 min 29 s
diffeq	10098	9853	130	115	97.57	98.86	11 min 38 s

The union of the faults covered by the two test generators was chosen as the number of detected faults (column '# tested') in Table 6.1. Column '# faults' shows the total number of stuck-at faults in the circuits. Column '# untest.' shows the number of untestable register enable faults identified by the method proposed in this thesis. Column "# remain." shows the number of faults that were neither tested nor identified untestable. Columns 'F.C.' and 'F.E.' present the achieved fault coverage and fault efficiency (i.e. test coverage), respectively. Finally, column 'CPU time' gives the CPU run times for the untestability identification.

As it can be seen from Table 6.1, a large number of untestable faults has been identified by the method in a relatively short run time. Large amount of the faults not tested in the given benchmark circuits fall into the category of untestable register enable faults. An additional benefit of the approach is the increase in fault efficiency. Identification of untestable faults allows raising the confidence in the test coverage and in the efficiency of the ATPG.

6.2 Untestability Identification Driven by RT-Level Constraints

The last paper titled **"Constraint-Based Hierarchical Untestability Identification for Synchronous Sequential Circuits"** [V]. The authors of the paper were *J. Raik, T. Viilukas, M. Jenihhin, R. Ubar, H. Fujiwara* and the author of this thesis. The paper was submitted to the DATE'11 conference.

The paper considers a novel register-transfer level (RTL) test pattern generation for non-scan sequential circuits containing feedback loops. In addition, a deterministic hierarchical automated test pattern generator (ATPG) which is guided by RT-level constraints was developed. First, an RTL test pattern generator Decider is applied in order to extract test path extraction constraints. Then, the constraint-driven deterministic ATPG is run providing hierarchical test generation and testability proof in sequential circuits.

We showed by experiments that the tool is capable of quickly proving a large number of untestable faults obtaining near to 100 % fault efficiency.

In addition, our study shows that traditional, bottom-up test generation at RTL is often too optimistic due to the fact that propagation constraints have been ignored and capabilities to prove untestable faults have been missing. In this thesis we consider top-down approach of test generation.

6.2.1 Preliminaries

6.2.1.1 Assignment Decision Diagrams

Assignment decision diagram (ADD) is an acyclic graph that consists of a set of nodes that can be categorized into four types: read node, write node, operation node and assignment decision node (ADN), and a set of edges which contain the connectivity information between two nodes (Figure 6.4). A read node represents a primary input port, a storage unit or a constant while a write node represents a primary output port or a storage unit. An operation node expresses an arithmetic operation unit or a logic operation unit while an ADN selects a value from a set of values that are provided to it based on the conditions computed by the logic operation units. If one of the condition inputs becomes true, the value of the corresponding data input will be selected. Although ADD was essentially

introduced as an internal representation in the high-level synthesis process, it can be used to describe a functional RTL circuit, the controller part and the data path part of which are homogeneously represented.

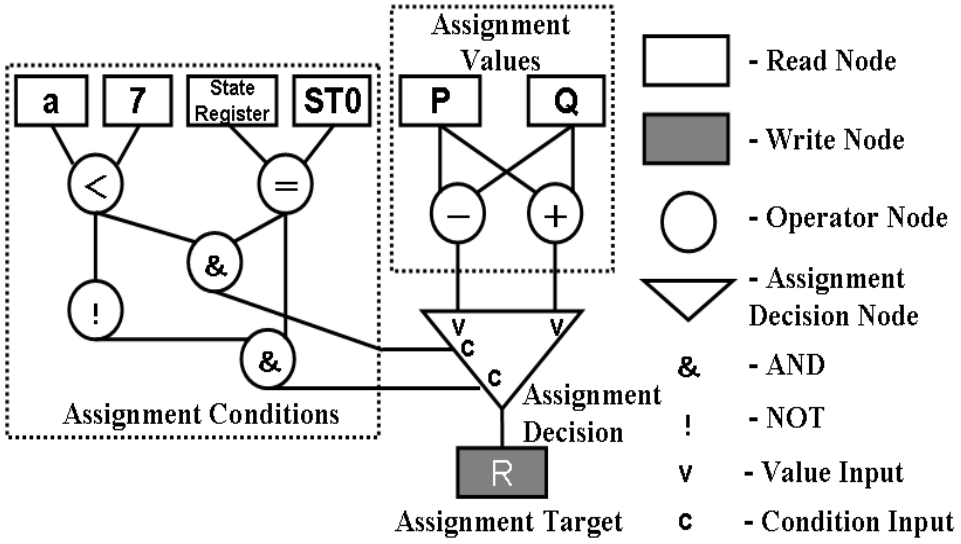


Figure 6.4 Assignment Decision Diagram (ADD)

6.2.1.2 Test Environment

When a node *N* is under test, the testability of the node is guaranteed if (a) any value can propagate from a read node corresponding to a primary input port to the input of *N*, and (b) the value at the output of *N* can propagate to a write node corresponding to a primary output port. The paths which allow (a) and (b) to occur are called *justification path* and *propagation path*, respectively. Justification and propagation can be done through symbolic processing that utilizes nine-valued algebra. The series of symbols obtained from the symbolic processing that activates justification and propagation paths is known as the *test environment* for the node under test.

For a given node under test, its test sequence is generated by first extracting a test pattern from the *test set library* and by substituting the test pattern for the test environment. The test set library is obtained beforehand by first simply taking a gate-level circuit whose functionality is the same as that of the node under test,

then generating the test patterns for all faults in the circuit using a combinational ATPG algorithm.

6.2.1.3 Multi-valued Algebra for Test Propagation

The nine symbols of Ghosh's nine-valued algebra, each of which can be assigned true or false, are as follows:

- *Cg*(*v*): variable *v* can be set to any value.
- *C0*(*v*): variable *v* can be set to 0.
- *C1*(*v*): variable *v* can be set to 1.
- *Ca1*(*v*): all bits of variable *v* can be set to 1's.
- *Cq*(*v*): variable *v* can be set to a constant.
- *Cz*(*v*): variable *v* can be set to high impedance *Z*.
- *Cs*(*v*): state variable *v* can be set to a specific state.
- *O*(*v*): any fault effect at variable *v* can be observed.
- *O'*(*v*): fault effect of *D'* can be observed for a single bit variable v.

To generate a test environment, first an objective has to be set. In order to achieve the test environment objective, the test sequence for each ADD can be generated through the following two phases using the justification/ propagation rules defined in [23] and briefly explained in an example in Section 6.2.2:

Phase 1: Generate the test environment of the node under test.

Phase 2: Generate the test sequence of the node under test by substituting the test patterns of the gate-level circuit corresponding to the node under test for the test environment.

In many cases the propagation rules of the multi-valued algebra are unable to generate test environment for a module even if test for this module exists. Furthermore, the generated environment may decrease the fault coverage for the module under test because the network constraints are not taken into account when creating the local test set to be substituted into the environment.

6.2.1.4 The Concept of Test Path Constraints

The constraint-driven deterministic test generation approach proposed in current thesis contains two main phases. During the first phase, constraints for setting up a test path to test an RTL module are extracted. The second phase generates deterministic tests to the low-level module taking into account the path constraints. This guarantees high fault coverage for the module under test and also allows keeping track of the untestable faults.

We apply RTL ATPG Decider [12] in order to extract the constraints for accessing the Module Under Test (MUT). Decider activates as many sets of constraints as there are test paths for that module in a bounded limit of clock-cycles. In [II] formal satisfaction method for the test path activation constraints has been included into the tool. However, the work in [II] does not consider the problem of testing the modules in a deterministic manner at the low-level. The purpose is to process the set of constraints in order to derive conditions for a dedicated logic-level ATPG in proving untestability.

In order to extract the RTL constraints for MUT, a test path activation tool Decider is applied. The high-level test generation constraints considered by Decider are divided into three categories. These are path activation constraints, transformation constraints and propagation constraints. *Path activation constraints* correspond to the logic conditions in the control flow graph that have to be satisfied in order to perform propagation and value justification through the circuit. *Transformation constraints*, in turn, reflect the value changes along the paths from the inputs of the high-level MUT to the primary inputs of the whole circuit. These constraints are needed in order to derive the local test patterns for the module under test. *Propagation constraints* show how the value propagated from the output of the MUT to a primary output is depending on the values of the signals in the system. The main idea here is to guarantee that fault effect will not be masked when propagated.

Note, that the extracted constraints consist of operations on primary inputs and constants. Furthermore, the exponential size complexity of the constraints is avoided by uniting multiple occurrences of the same variable (i.e. the literals) in the constraints at each time step into one single fanout variable. Because of this, the size requirements for the constraints are linear with respect to justification time-frames and they represent a small subset of the expanded time-frame model of the circuit. Thus, the high-level test constraint extraction procedure is scalable in terms of memory space requirements.

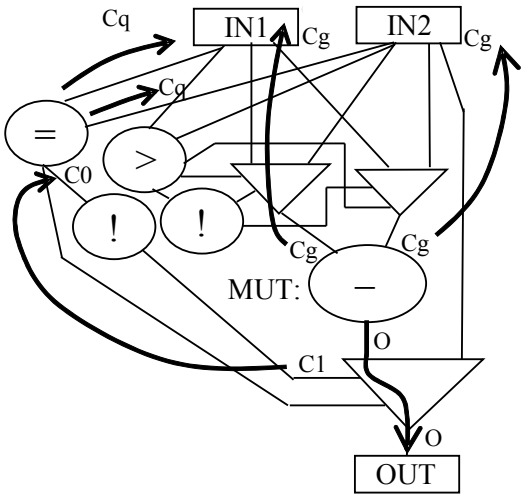
6.2.2 Generating Test Environments Under Control and Data Dependencies

In this Section we describe the problem of setting up constraint-driven test environments for RTL modules in the case of dependencies between data and control signals. We present a motivating example explaining the shortcomings that are common to the previous test environment generation approaches that ignore the effect of test path constraints.

Consider as an example, a simplification of the ADD for the Greatest Common Division (GCD) benchmark presented in Figure 6.5a. Without loss of generality in this ADD the control state information has been removed in order to improve the readability of the diagram.

Assume that our task is generating a test environment for the subtraction module (MUT) in the Figure. It can be seen that the output value of MUT will be propagated to the primary output OUT only if the first value input of the corresponding assignment decision is 1. When we justify the symbols at the MUT inputs according to the propagation rules presented in Section 6.2.1.3, then the strict interpretation of these rules would lead into a contradiction. Assume the both inputs of the MUT are set to Cg according to the rule in Figure 6.5. In order to propagate the output value of the MUT the first control input of the ADN preceding the OUT is set to C1. The justification rules for the equality operator "=" require Cq from the IN1 and IN2 read nodes.

a)



b)

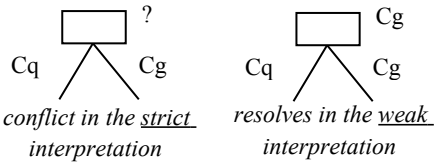


Figure 6.5 Test environment generation for GCD

That leads to conflict at the read nodes in the strict interpretation of the Ghosh's multi-value algebra for test propagation (please refer to Figure 6.5b). However, the weak interpretation (also used in [24]) would still allow the following test environment: IN1=Cg and IN2=Cg. Note, that in current situation the weak rules are preferable since they at least allow testing part of the MUT while the strict rules would not generate the test environment at all.

To summarize, the strict interpretation of Ghosh's algebra lead to overly pessimistic results because tests for some MUTs are aborted due to justification conflicts. On the other hand, the weak interpretation is too optimistic and can also lead to loss of fault coverage because some of the test patterns that are expected to cover faults in the MUT do not propagate.

Consider the case where in a bottom-up scenario we have a deterministic test T_q generated for the MUT reaching the maximum fault coverage W_q for the module. Then, we use top-down approach and generate the test environment for the module and substitute T_q into the test environment. Due to the test path constraints the actual fault coverage that can be achieved for MUT inside the network is W_a, which is generally lower than the fault coverage W_q. However, when we fault simulate T_q substituted into the test environment in bottom-up approach, we obtain a fault coverage W_r, where :

$$W_r \leq W_a \leq W_q \qquad (1)$$

In other words, the bottom-up approach may lose some fault coverage with respect to the top-down one because the set of the tests to choose from is restricted to T_q. If the local test generation algorithm for MUT had had knowledge about the test path constraints it would have generated a different test T_d, whose fault coverage would have been equal to W_a. Furthermore, the remaining faults inside

MUT would have been proven untestable. Thus, a deterministic ATPG taking into account the test path constraints is necessary in order to achieve maximum fault coverage and also to prove untestability within sequential networks. Experiments with the constraint-driven deterministic ATPG presented in Section 6.2.5 show that the difference between the coverages W_r and W_a may be even as high as 8-14 per cent of stuck-at coverage.

In the next Section we show how the test constraints can be efficiently included into the test environment in order to allow high-fault coverage testing of the modules and also to provide proof for sequentially untestable stuck-at faults.

6.2.3 Generating the Constraint-based Test Environment

In this Section we explain extracting the test path constraints for a MUT. We show how to compute the constraint-based test environment from the set of test constraints.

Consider Figure 6.6, which gives the full set of constraints for the MUT from the example of Figure 6.6. In other words, the MUT can only be tested using one of the two test paths presented in Figure 6.6a and 6.6b. The two paths are identical except for the fact that the primary inputs IN1, IN2 are swapped in them.

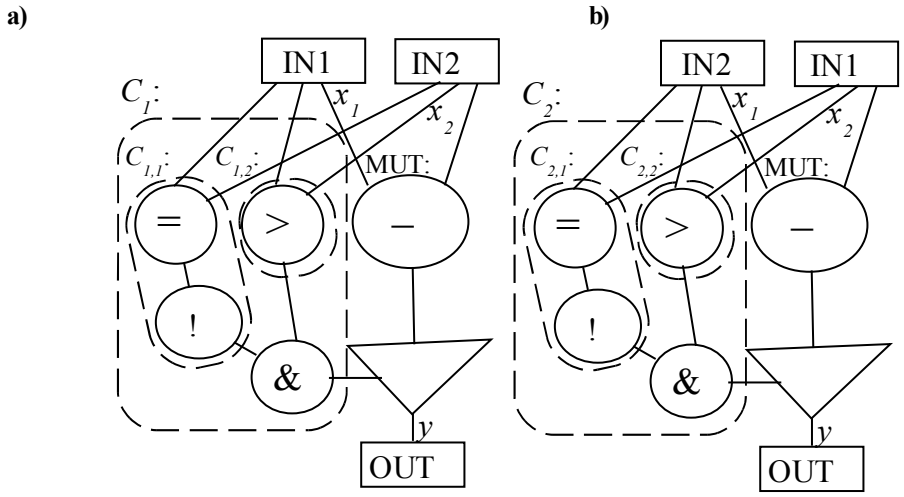


Figure 6.6. Full set of test path constraints for MUT

Note, that from the point of view of accessing the MUT these two environments are equivalent. It is irrelevant which primary input is used in applying the test patterns when representing the constraint-based test environment for proving untestability. Therefore, we denote the value justified from the i-th input of MUT by x_k and the value propagated from the MUT output by y.

The constraints C_1 and C_2 both consist of two sub-constraints $C_{1,1}$, $C_{1,2}$ and $C_{2,1}$, $C_{2,2}$, respectively. $C_{1,1}$ (which is equivalent to $C_{2,1}$) states that x_1 must not be equal to x_2. $C_{1,2}$ (equivalent to $C_{2,2}$) states that x_1 must be greater than x_2. Since all the sub-constraints within a constraint should hold simultaneously they be combined using the conjunction operator. In turn, all the constraints are combined using the disjunction operation because any one of the test paths may be used for accessing the MUT. In general case for constraints C_i each consisting of sub-constraints $C_{i,j}$ the constraint-environment for proving sequential untestability is calculated using the following formula:

$$\bigvee_i \bigwedge_i C_{i,j}. \qquad (2)$$

Subsequent to combining the test path constraints constraint minimization is performed. For the example in Figure 6.6 we obtain:

$$(x_1 \neq x_2) \wedge (x_1 > x_2) \vee (x_1 \neq x_2) \wedge (x_1 > x_2) = x_1 > x_2.$$

Figure 6.7 shows the constraint-based environment resulting for testing the MUT of the example presented in Figure 6.5. In the next Section we propose a gate-level ATPG that relies on this kind of constraint-based environment to perform hierarchical untestability identification and test pattern generation.

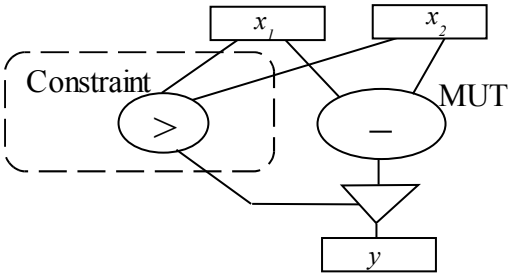


Figure 6.7 Constraint-based test environment the MUT

6.2.4 Constraint-Driven Deterministic ATPG

As it was mentioned above, the proposed ATPG method consists of two steps. First, the test constraints are extracted at RTL as explained in the previous Section. As a second step, a constraint-driven deterministic ATPG is run as discussed here. An example of a constraint-based environment was shown in Figure 6.7. The constraint is converted to the gate-level by applying logic synthesis and the MUT is instantiated from the design, in order to ensure that the gate-level structure tested exactly matches the one of MUT embedded to the RTL network. Then a gate-level ATPG is executed [28], which identifies sequentially untestable faults in the MUT.

Figure 6.8 presents the corresponding test flow. First, RTL ATPG is run in order to derive high-level test path constraints. The constraints are minimized as shown in the previous Section, translated into VHDL and synthesized to logic-level using Synopsys Design Compiler.

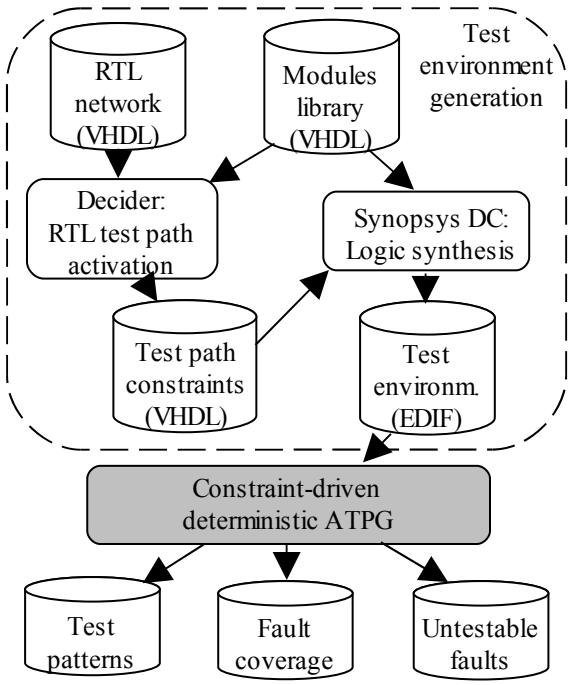


Figure 6.8. Top-down constraint-based hierarchical ATPG flow

Subsequently, the gate-level ATPG is run. As a result we obtain the list of sequentially untestable faults in the MUT as well as test patterns for the whole design. Experiments presented in the next Section show that the proposed constraint-based method obtained 100 per cent fault efficiency for all the considered modules at the same time when the symbolic approach proposed in [23] is too optimistic, loosing 8-14 per cent of fault coverage in the modules.

6.2.5 Experimental Results

In order to evaluate the hierarchical untestability identification and test generation method, experiments on HLSynth92 and HLSynth95 benchmarks were run. In addition, to compare the solution with the traditional bottom-up approach (e.g. [23]) and assess its fault efficiency, a detailed case-study was carried out.

Table 6.2 presents the characteristics of the example circuits used in test pattern generation experiments. The following benchmarks were included to the test experiment: a Greatest Common Divisor (GCD), an 8-bit multiplier (MULT8x8), and a Differential Equation (DIFFEQ). In the Table, the number of single stuck-at faults, the number of primary input and primary output bits, the number of registers, multiplexers and functional units are reported, respectively.

Table 6.2 Benchmark characteristics

circuit	# faults	PI bits	PO bits	# reg.	# mux	# FU
gcd	472	33	16	3	4	3
mult8x8	2356	17	16	7	4	9
diffeq	10326	81	48	7	9	5

In Table 6.3, comparison of test generation results of three ATPG tools on the hierarchical benchmark designs are presented. This comparison was carried out in order to show the time needed for extracting the constraint-based environment as explained in Section 6.2.3. The tools include a gate-level deterministic ATPG Hitec [2], a genetic algorithm based Gatest [6], hierarchical ATPG Decider.

Columns ‘F.C., %’ give the single stuck-at fault coverages of the test patterns generated. Columns ‘time, s’ stand for test generation run-times obtained on a 366MHz Sun UltraSparc60 server with 512MB RAM under Solaris 2.8 operating system.

Table 6.3 Comparison of sequential ATPG

circuit	HITEC		GATEST		DECIDER	
	F.C., %	time, s	F.C., %	time, s	F.C., %	time, s
gcd	81.1	169.5	**91.0**	75	89.9	129.8
mult8x8	65.9	1243	69.2	821.6	**74.7**	93.7
diffeq	96.2	13.320	96.40	3000	**97.09**	453.7

Table 6.4 shows experiments of the deterministic constraint-driven ATPG developed in this thesis. The experiments present comparison of the proposed method to the bottom-up paradigm [23]. For a reference, the modules were tested by the ATPG in a stand-alone mode. As a result a test sequence T_q yielding 100 % stuck-at fault coverage W_q was obtained. The proposed top-down constraint-driven ATPG reached fault coverage W_a which was less than W_q because of the constraints when accessing the module under test in the network. However, the fault efficiency of the proposed approach was 100 % for all the modules.

When T_q was substituted to the test environment in a bottom-up manner then fault coverage W_r was reached, which was always lower than W_a because some of the tests were invalidated by sequential dependencies. In fact, W_r was considerably lower (by 8-14 %) for all the four modules analyzed.

The test environment synthesis from VHDL to logic-level using Synopsys Design Compiler remained almost constant and was around 5 to 10 s per module while the deterministic constraint-based ATPG spent less than 0.02 s per module under test. The synthesis and test experiments were carried out on a Sun-Fire-V250 station with 1.28 GHz sparcv9 processor under Solaris 2.9 OS.

Table 6.4 Constraint-driven ATPG vs. bottom-up RTL test

Circuit	gcd	mult8x8			diffeq	
Module	**SUB**	**ADD2**	**ADD3**	**SUB2**	**MUX3**	**MUX4**
W_q, %	100	100	100	100	100	100
W_a, %	**95.74**	**86.64**	**55.88**	**85.33**	**75.00**	**75.00**
W_r, %	85.11	72.49	47.06	74.07	64.71	64.71
ATPG, s	0.01	0.01	< 00.1	0.02	< 0.01	< 0.01
synthesis, s	5.38	5.33	9.52	5.25	5.10	5.10

Table 6.5 presents detailed statistics of the circuits analyzed. The Table lists the total number of stuck-at faults in the whole circuit, the number of detected faults, number of unobservable/uncontrolllable faults, the number of faults proven sequentially untestable by the proposed constraint-based approach and finally the number of all the remaining faults. The experiments show the efficiency of the constraint-driven engine in untestability identification. Though the method covers quickly untestable faults caused by sequential untestability in the considered modules with 100 % fault efficiency, there remains a number of faults which are still neither tested nor proven untestable. Some of these remaining faults can be tested or proven untestable by traditional approaches at the logic-level.

Table 6.5. Distribution of faults*

	gcd	**mult8x8**	**diffeq**
# total faults	472	2356	10326
# detected faults (hard/potentially detected + reg. untestable)	439	1737	9867
# unobs./uncontr.	28	195	252
# untest. w constr.	**4**	**156**	**68**
# remaining	1	268	139

*** Note: Table 6.5 and Table 5.2 from Section 5.2.2 does not match because some of benchmarks were synthesized by different conditions.**

6.3 Chapter Summary

In this Chapter the overview of the research results published in Papers: **"Untestable Fault Identification in Sequential Circuits Using Model-Checking"** [IV] and in **"Constraint-Based Hierarchical Untestability Identification for Synchronous Sequential Circuits"** [V]. The overview has been presented together with the overall experimental results.

This Chapter describes the research that contrbutes mainly to the study of a an Untestable Faults in Sequential Circuits at RTL Level tehniques**.**

The first contribution is Untestable Fault Identification in Sequential Circuits Using Model-Checking and proposes a new approach of quick identification of untestable logic-level stuck-at faults from the register transfer level. The novelty of the approach lies in using an existing commercial model-checking tool for the untestability analysis. In particular, a technique for formally generating PSL language assertions for proving untestable stuck-at faults in sequential synchronous designs was developed. Experiments on well-known sequential benchmarks showed that as much as 20-60 per cent of faults not detected by sequential ATPG were identified untestable in a short run time by the approach.

The proposed untestable fault identification may also be implemented in high-level test synthesis. It was shown that by removing the redundant enable signals in average 5 per cent of the circuit area could be saved. An additional effect of the identification of untestable register enable faults lies in reducing yield loss.

The second one considers a new method and tool for register-transfer level (RTL) test pattern generation for non-scan sequential circuits containing feedback loops. A deterministic hierarchical automated test pattern generator (ATPG) guided by RT-level constraints is proposed. First, an RTL test pattern generator Decider is applied in order to extract test path extraction constraints. Then, the constraint-driven deterministic ATPG is run providing hierarchical test generation and testability proof in sequential circuits.

In addition, our study shows that traditional test generation at RTL based on symbolic test environment generation is too optimistic due to the fact that constraints in accessing the modules under test have been ignored. Experiments showed that bottom-up strategies caused a decrease of stuck-at fault coverage up to the range of 8-14 % in the modules tested. This short-coming was overcome by the

proposed constraint-based method which obtained 100 per cent fault efficiency for all the modules considered.

To the best of our knowing this is the first method that can prove sequential untestability starting from the RTL.

In each direction of the research, new appreciable results were achieved. The resuls were also presented at conferences.

Chapter 7
CONCLUSIONS

This thesis has presented several techniques to perform hierarchical test pattern generation and untestability identification for synchronous sequential circuits that is the one of the major issues in the area of digital circuits testing.

In this Chapter the main contributions of the work are outlined and points out open problems and the perspectives for future research. In addition, the contribution list in research done by the author of this thesis is added.

7.1 Thesis Contribution

The main contributions of the presented work are summarized below.

Test Pattern Generation for Sequential Circuits

- An overview of the comparative study of ATPG methods has been proposed [I]. A comparative study of test pattern generation approaches based on three tools: a genetic algorithm test generator GATEST [19], a deterministic logic-level tool HITEC [18] and a hierarchical tool DECIDER [15]. The purpose of this study was to find out, which fault types are covered by the tools implementing completely different approaches.

Experiments on a set of six sequential benchmark circuits lead to the following conclusions:

While genetic algorithm based tool performs well in terms of the absolute fault coverage numbers, it fails to detect nearly any unique faults.

Deterministic tool has difficulties with larger sequential designs but it is capable of detecting a portion of hard-to-test faults.

The union of the sets of faults covered by the three test generators has a fault coverage that is in average 0.4 per cent higher than the fault cover of the best tool in the comparison: DECIDER.

DECIDER loses fault coverage mainly in the control part FSM.

The analysis carried out was and will be helpful for further development of the hierarchical ATPG DECIDER. Moreover, the authors hope that the results presented here could give valuable guidelines for the developers of future test pattern generators in general.

- A novel constraint-based automated test pattern generator for Register-Transfer Level (RTL) designs has been introduced [II]. The tool combines test path constraint activation with a constraint solver. First, a deterministic algorithm that extracts constraints for activating test paths at RTL is applied. Subsequently, a constraint solving package ECLiPSe [14] is used for assembling the tests.

Experiments on ITC99 and HLSynth92/95 benchmarks showed that the proposed deterministic method offers short run times. In particular, it provides increased fault coverage for hard-to-test designs with respect to earlier approaches.

While the fault coverages for the circuits are low, this is a usual case for the sequential ATPG because of the large number of untestable faults. As a future work we plan to integrate untestable fault analysis for sequential circuits presented in this thesis into the constraint-based ATPG to improve fault efficiency estimation.

- This thesis also presented the problem of high-level identification of an important subclass of faults, of potentially testable initialization faults [III]. Existing high-level fault models assume hard-detection and therefore are not capable of handling such initialization faults.

Experiments showed that potentially detectable initialization faults form a large subset of all the faults not testable by hard-detection. As a result of the proposed approach, both, the speed as well as the confidence level of sequential ATPG can be increased. We plan to implement the potential fault detection method and include the capabilities to an RTL test pattern generator.

Proving Untestable Faults in Sequential Circuits at RTL

- An approach of identifying of untestable faults in sequential circuits has been considered [IV] . We proposed using model-checking for detecting untestable stuck-at faults at the Register-Transfer Level (RTL). In particular, we presented a method for formally generating PSL language assertions for proving untestable stuck-at faults in sequential synchronous designs.

Experiments showed that the faults identified by the method form in fact a large subset of all the untested stuck-at faults. It was shown that by removing the redundant enable signals in average 5 per cent of the circuit area could be saved. An additional effect of the identification of untestable register enable faults lies in reducing yield loss.

- A novel method of register-transfer level (RTL) test pattern generation for non-scan sequential circuits containing feedback loops has been introduced [V]. In addition, a deterministic hierarchical automated test pattern generator (ATPG) which is guided by RT-level constraints was developed. First, an RTL test pattern generator Decider is applied in order to extract test path extraction constraints. Then, the constraint-driven deterministic ATPG is run providing hierarchical test generation and testability proof in sequential circuits.

The proposed method is capable of quickly proving a large number of untestable faults obtaining near to 100 % fault efficiency. In addition, our study showed that traditional test generation at RTL is often too optimistic due to the fact that propagation constraints have been ignored and capabilities to prove untestable faults have been missing. Experiments showed that bottom-up strategies may cause a decrease of stuck-at fault coverage up to the range of 8-14 % in the modules under test. To the best of our knowing this is the first method that can prove sequential untestability starting from the RTL.

In conclusion, numerous challenging problems and open issues pave the road towards test generation for sequential circuits, but we believe that this remains the primary research direction for the next few years.

7.2 Author's Contribution

This Subsection provides the contribution list in research done by the author of this thesis.

The author of this thesis was involved in all stages of the research: studying problems, preparation, development, testing and presenting the research results.

Paper I „Comparative Analysis of Sequential Circuit Test Generation Approaches“ deals with a comparative study of test pattern generation approaches based on three tools: a genetic algorithm test generator GATEST, a deterministic logic-level tool HITEC and a hierarchical tool DECIDER.

The author of this thesis was responsible for performing the experimental analysis based on comparative study of test pattern generation approaches using three test generation tools and studying which fault types are likely to be covered by different approaches.

In addition, obtained results were included to author's Bachelor work:

> 2004, Comparative Analysis of Sequential Circuit Test Generation Approaches, B.Sc., supervisor Dr. Jaan Raik, Tallinn University of Technology, Faculty of Information Technology

Paper II „Constraint-based Test Pattern Generation at the Register-Transfer Level” introduces a novel constraint-based automated test pattern generator for Register-Transfer Level (RTL) designs. The tool combines test path constraint activation with a constraint solver. First, a deterministic algorithm that extracts constraints for activating test paths at RTL is applied. Subsequently, a constraint solving package ECLiPSe [14] is used for assembling the tests.

Additional motivation for the work in **Paper I** was to find guidelines for improving the fault models implemented in the hierarchical test pattern generator DECIDER, which is being developed at TUT. Based on received results a Master of Science work was defended by the author:

2005, Experimental analysis of hierarhical test generator DECIDER, M.Sc., supervisor Dr. Jaan Raik, Tallinn University of Technology, Faculty of Information Technology

This was the motivation for the future research that was presented in **Paper II**.

The author of the thesis was responsible of performing experimental analysis using deterministic method.

In Paper III „RT-Level Identification of Potentially Testable Initialization Faults" the idea is to introduce the problem of an important subclass of faults, the potentially detectable initialization faults.

The author of this thesis studied the problem of identification of potentially testable initialization faults and presented the proposed method at the workshop:

The Ninth IEEE Workshop on RTL and High Level Testing (WRTLT'08), IEEE, pp. 667-672, November 27-28, 2008, Sapporo, Japan.

A new approach of applying model-checking for detecting untestable stuck-at faults at the register-transfer level is introduced in **Paper IV.** In particular, we presented a method for formally generating PSL language assertions for proving untestable stuck-at faults in sequential synchronous designs.

The contributions of the author of the thesis are performing the experimental research and analysis of identification of Untestable Faults in Sequential Circuits using model-checking. The intermediate steps of research were published in **Paper VI "**Hierarchical Identification of Untestable Faults in Sequential Circuits". **Paper IV** contains the most important achievements and continues to improve the technique proposed.

In addition, the results of the proposed research were presented by the author at the conference:

The 17th Asian Test Symposium (ATS'08), IEEE, pp. 667-672, November 24-27, 2008, Sapporo, Japan.

The Paper V „Constraint-Based Hierarchical Untestability Identification for Synchronous Sequential Circuits“ considers register-transfer level (RTL) test pattern generation for non-scan sequential circuits containing feedback loops.

The contribution of the author of this thesis was to study the problem of Untestability Identification for Synchronous Sequential Circuits more deeper proceed from **Paper IV** and **VI**.

The author was developing a new method of register-transfer level (RTL) test pattern generation for non-scan sequential circuits containing feedback loops, performing experimental analysis using the deterministic hierarchical automated test pattern generator (ATPG) which is guided by RT-level constraints.

In this research we went to one step further by identifying from experiments that the tool is capable of quickly proving a large number of untestable faults obtaining near to 100 % fault efficiency.

References

Co-authored papers:

[I] J. Raik, A. Krivenko (Rannaste), R. Ubar. Comparative Analysis of Sequential Circuit Test Generation Approaches. Proc. of the Baltic Electronic Conference, pp. 225-228, Tallinn, Estonia, Oct. 3-6, 2004.

[II] Taavi Viilukas, Jaan Raik, Maksim Jenihhin, Raimund Ubar, Anna Krivenko (Rannaste) "Constraint-based Test Pattern Generation at the Register-Transfer Level", *Proceedings of the 13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS'10)*, April 14–16, 2010, Vienna, pp. 352 – 357.

[III] Jaan Raik, Hideo Fujiwara, Anna Krivenko (Rannaste). RT-Level Identification of Potentially Testable Initialization Faults. *The Ninth IEEE Workshop on RTL and High Level Testing (WRTLT 2008)*, IEEE, pp. 667-672, November 27-28, 2008, Sapporo, Japan.

[IV] Jaan Raik, Hideo Fujiwara, Raimund Ubar, Anna Krivenko (Rannaste). Untestable Fault Identification in Sequential Circuits Using Model-Checking. *The 17th Asian Test Symposium (ATS'08)*, IEEE, pp. 667-672, November 24-27, 2008, Sapporo, Japan.

[V] J. Raik, A. Krivenko (Rannaste), T. Viilukas, M. Jenihhin, R. Ubar, H. Fujiwara. Constraint-Based Hierarchical Untestability Identification for Synchronous Sequential Circuits, *(submitted to the DATE'11 conference)*

[VI] Jaan Raik, Raimund Ubar, Anna Krivenko (Rannaste), Margus Kruus. Hierarchical Identification of Untestable Faults in Sequential Circuits, *Proceedings of the 10th IEEE Euromicro Conference on Digital Systems Design (DSD'07),* IEEE Computer Society, pp. 668-671, 27-31 August, 2007, Lübeck, Germany.

Other References :

[1] H.-K.T Ma, S. Devadas, A.R. Newton, A. Sangiovanni-Vincentelli, "Test generation for sequential circuits", IEEE Trans. on CAD, Vol. 7, No. 10 pp. 1081-1093, Oct. 1988.

[2] T. M. Niermann, J. H. Patel, "HITEC: A test generation package for sequential circuits", Proc. European Conf. Design Automation (EDAC), pp.214-218, 1991.

[3] D. Brahme, J. A. Abraham, "Functional Testing of Micro-processors", IEEE Trans. Comput., vol. C-33, 1984.

[4] A. Gupta, J. R. Armstrong, "Functional fault modeling", 30th ACM/IEEE DAC, pp. 720-726, 1985.

[5] F. Ferrandi, F. Fummi, D. Sciuto, "Implicit Test Generation for Behavioral VHDL Models," Int. Test Conf., pp. 587-596, 1998.

[6] E. M. Rudnick, et al. "Sequential circuit test generation in a genetic algorithm framework", Proc. DAC, pp. 698-704, 1994.

[7] . Corno, P. Prinetto, et al., "GATTO: A genetic algorithm for automatic test pattern generation for large synchronous sequential circuits", IEEE Trans. CAD, pp.991-1000, Aug. 1996.

[8] M. S. Hiao, E. M. Rudnick, J. H. Patel, "Sequential circuit test generation using dynamic state traversal", Proc. European Design and Test Conf., pp. 22-28, 1997.

[9] A. Giani, et al., "Efficient Spectral Techniques for Sequential ATPG," Proc. IEEE DATE Conf., March 2001, pp. 204-208.

[10] J. Lee, J.H. Patel, "Architectural level test generation for microprocessors", *IEEE Trans. CAD*, pp.1288-1300, Oct. 1994

[11] G. Jervan et al., "High-Level and Hierarchical Test Sequence Generation", *IEEE HLDVT*, Cannes, 2002.

[12] J. Raik, R. Ubar, "Fast test pattern generation for sequential circuits using decision diagram representations", *JETTA*, Kluwer, 16(3), 2000.

[13] G. Di Guglielmo et al., "On the Combined Use of HLDDs and EFSMs for Functional ATPG", *East-West Design & Test Symposium*, Yerevan, 2007.

[14] The ECLiPSe Constraint Programming System http://eclipse-clp.org/

[15] J. Raik, R. Ubar, "Targeting Conditional Operations in Sequential Test Pattern Generation", *European Test Symposium*, 2004.

[16] HLSynth92 benchmark directory at URL: http: //www.cbl.ncsu.edu/pub/Benchmark_dirs/HLSynth92/

[17] E. Gramatova, et al., "FUTEG Benchmarks," Technical Report COPERNICUS JEP 9624 FUTEG No9/1995.

[18] T. M. Niermann, J. H. Patel, "HITEC: A test generation package for sequential circuits", *Proc. European Conf. Design Automation (EDAC)*, pp.214-218, 1991.

[19] E. M. Rudnick, J. H. Patel, G. S. Greenstein, T. M. Niermann, "Sequential circuit test generation in a genetic algorithm framework", *Proc.* DAC, pp. 698-704, 1994.

[20] B. T. Murray, J. P. Hayes, "Hierarchical test generation using precomputed tests for modules", *Proc. ITC*, pp.221-229, 1988.

[21] I. Ghosh, M. Fujita, "Automatic Test Pattern Generation for Functional RTL Circuits Using Assignment Decision Diagrams", *DAC*, pp. 43-48, 2000.

[22] V. Chayakul, D. D. Gajski, L. Ramachandran, "High-Level Transformations for Minimizing Syntactic Variances", DAC, pp. 413-418, June 1993.

[23] L. Zhang, I. Ghosh, M. Hsiao, "Efficient Sequential ATPG for Functional RTL Circuits", *Int. Test Conf.*, pp.290-298, 2003.

[24] H. Fujiwara, C. Y. Ooi, Y. Shimizu, "Enhancement of Test Environment Generation for Assignment Decision Diagrams", *WRTLT*, 2008.

[25] K. C. Y. Mei, "Bridging and stuck-at faults", *IEEE Trans. Comput.*, vol. C-23, no 7, pp. 720-727, July 1974

[26] L. Bushnell, A. Agrawal, "Essentials of Electronic Testing for Digital Memory & Mixed – Signal VLSI Circuits", Kluwer academic publishers, Boston / Dordrecht / London, 2000

[27] Niraj Iha and Sandeep Gupta, "Testing of digital systems" Cambridge : Cambridge University Press, c2003

[28] Turbo Tester test tools, URL: http://www.pld.ttu.ee/tt/

[29] J. Raik, R. Ubar, T. Viilukas, M. Jenihhin. Mixed Hierarchical-Functional Fault Models for Targeting Sequential Cores. J. of Systems Architecture, Elsevier, 2008.

[30] V. D. Agrawal and S. T. Chakradhar, "Combinational ATPG theorems for identifying untestable faults in sequential circuits," *IEEE Trans Comput.-Aided Des. Integr. Circuits Syst.*, vol. 14, no. 9, pp. 1155–1160, Sep. 1995.

[31] M. A. Iyer, D. E. Long, and M. Abramovici, "Identifying sequential redundancies without search," *in Proc. 33rd Annu. Conf. DAC*, LasVegas, NV, Jun. 1996, pp. 457–462.

[32] Q. Peng, M. Abramovici, and J. Savir, "MUST:Multiple stem analysis for identifying sequential untestable faults," *in Proc. Int. Test Conf.*, Atlantic City, NJ, Oct. 2000, pp. 839–846.

[33] D. E. Long, M. A. Iyer, and M. Abramovici, "FILL and FUNI: Algorithms to identify illegal states and sequentially untestable faults," *ACM Transact. Des. Automat. Electron. Syst.*, vol. 5, no. 3, pp. 631–657, Jul. 2000.

[34] H.-C. Liang, C. L. Lee, and E. J. Chen, "Identifying untestable faults in sequential circuits," *IEEE Des. Test. Comput*., vol. 12, no. 3, pp. 14–23, Sep. 1995.

[35] Michiko Inoue, Takeshi Higashimura, Kenji Noda, Toshimitsu Masuzawa, Hideo Fujiwara: A High-Level Synthesis Method for Weakly Testable Data Paths. *Asian Test Symposium* 1998: 40-45

[36] Marie-Lise Flottes, R. Pires, Bruno Rouzeyre: Alleviating DFT Cost Using Testability Driven HLS. *Asian Test Symposium* 1998: 46-51

[37] M.L. Flottes, R. Pires, and B. Rouzeyre, "Analyzing Testability from Behavioral to RT Level", *in Proc. European Design & Test Conf.*, 1997, pp. 159–165.

[38] J. Raik, "Hierarchical Test Generation for Digital Circuits Represented by Decision Diagrams", TUT press, 2001

[39] Maxwell, P. Et al., Comparing functional and structural tests, *ITC 2000*. 3-5 Oct. 2000 Page(s):400 – 407.

[40] A. Krivenko, "Experimental analysis of hierarchical test generator DECIDER", Master thesis, TUT , 2005

Curriculum Vitae in English

Personal data

Name	Anna Rannaste (Krivenko)
Date and place of birth	05.04.1981, ESTONIA
Citizenship	Estonian

Contact information

Address	Raja 15, Tallinn 12618, ESTONIA
Phone	+372 51 80673
E-mail	anna.rannaste@gmail.com

Education

2010 –	Entered MBA studies in Euroakademie, Master of Business Administration
2005 –	Entered PhD studies in Tallinn University of Technology, Computer Engineering

2004 – 2006	Engineering Educator, TUT, UT
2004 – 2005	M.Sc. in Computer Engineering, TUT
1999 – 2004	B.Sc. in Computer Engineering, TUT

Professional employment

2009 –	Elvior OÜ, Software Engineer, Reading the training course „TTCN-3 Testing Language“ at TUT. Subject: „Software automated testing“
2007 – 2009	CODA Eesti OÜ, Software Engineer
2005 – 2007	Ixonos OÜ, Test Team Coordinator, Group Manager

Language

Estonian	Good
English	Good
Russian	Native
German	Average

Defended theses

2005, Experimental analysis of hierarhical test generator DECIDER, M.Sc., supervisor Dr. Jaan Raik, Tallinn University of Technology, Faculty of Information Technology

2004, Comparative Analysis of Sequential Circuit Test Generation Approaches, B.Sc., supervisor Dr. Jaan Raik, Tallinn University of Technology, Faculty of Information Technology

Honours & Awards

"ICT Doctoral School" scholarship for PhD students, 2009

"Tiger University" grant for ICT PhD students, Estonian Information Technology Foundation (EITSA), 2008

"DAAD Sprachkursen" grant for High School German Language Course, 2004

Research interests

Natural Sciences and Engineering, Computer Sciences: High-level Test Pattern Generation for Synchronous Sequential Circuits

Scientific work

Papers

1. Jaan Raik, Anna Krivenko (Rannaste), Raimund Ubar. Comparative Analysis of Sequential Circuit Test Generation Approaches. *Proceedings of the Baltic Electronic Conference (BEC'04)*, pp. 225-228, Oct. 3-6, 2004 Tallinn, Estonia.
2. Jaan Raik, Raimund Ubar, Anna Krivenko (Rannaste), Margus Kruus. Hierarchical Identification of Untestable Faults in Sequential Circuits, *Proceedings of the 10th IEEE Euromicro Conference on Digital Systems Design (DSD'07)*, IEEE Computer Society, pp. 668-671, 27-31 August, 2007, Lübeck, Germany.
3. Jaan Raik, Hideo Fujiwara, Anna Krivenko (Rannaste). RT-Level Identification of Potentially Testable Initialization Faults. *The Ninth IEEE Workshop on RTL and High Level Testing (WRTLT'08)*, IEEE, pp. 667-672, November 27-28, 2008, Sapporo, Japan.
4. Jaan Raik, Hideo Fujiwara, Raimund Ubar, Anna Krivenko (Rannaste). Untestable Fault Identification in Sequential Circuits Using Model-Checking. *The 17th Asian Test Symposium (ATS'08)*, IEEE, pp. 667-672, November 24-27, 2008, Sapporo, Japan.

5. Taavi Viilukas, Jaan Raik, Maksim Jenihhin, Raimund Ubar, Anna Krivenko (Rannaste). Constraint-based Test Pattern Generation at the Register-Transfer Level, *Proceedings of the 13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS'10)*, pp. 352 – 357, April 14–16, 2010, Vienna, Austria.

6. J. Raik, A. Krivenko (Rannaste), T. Viilukas, M. Jenihhin, R. Ubar, H. Fujiwara. Constraint-Based Hierarchical Untestability Identification for Synchronous Sequential Circuits, (*submitted to the DATE'11 conference*).

Conference presentations

Jaan Raik, Hideo Fujiwara, Anna Krivenko (Rannaste). RT-Level Identification of Potentially Testable Initialization Faults. *The Ninth IEEE Workshop on RTL and High Level Testing (WRTLT'08)*, November 27-28, 2008, Sapporo, Japan.

Jaan Raik, Hideo Fujiwara, Raimund Ubar, Anna Krivenko (Rannaste). Untestable Fault Identification in Sequential Circuits Using Model-Checking. *The 17th Asian Test Symposium (ATS'08)*, November 24-27, 2008, Sapporo, Japan.

Projects

1. Design and Test of Digital Systems (SF0142508s03).

2. Design of Reliable Embedded Systems (SF0140041s08).

3. Digital System Verification and Test Using High-Level Decision Diagrams (ETF7068).

4. Hardware Functional Verification and Debug (ETF8478).

5. Self-Testing Digital Systems (ETF5910).

Elulookirjeldus

Isikuandmed

Nimi	Anna Rannaste (Krivenko)
Sünniaeg ja –koht	05.04.1981, EESTI
Kodakondsus	Eesti

Kontaktandmed

Aadress	Raja 15, Tallinn 12618, EESTI
Telefon	+372 5180673
E-post	anna.rannaste@gmail.com

Hariduskäik

2010 – ...	Magistrant, Ärijuhtimine (MBA), Euroülikool
2005 – ...	Doktorant, Arvutitehnika Instituut, Tallinna Tehnikaülikool
2004 – 2006	Tehnikaõpetaja, Tartu Ülikooli ja Tallinna Tehnikaülikooli koostöö
2004 – 2005	Tehnikateaduste magister, Arvuti- ja Süsteemitehnika eriala, Tallinna Tehnikaülikool

1999 – 2004	Tehnikateaduste bakalaureus, Arvuti- ja Süsteemitehnika eriala, Tallinna Tehnikaülikool

Teenistuskäik

2009 –	Elvior OÜ, tarkvarainsener, Lugesin kursust „TTCN-3 Testimis keel“ TTÜ. Aine: „Tarkvara automatiseeritud testimine“
2007 – 2009	CODA Eesti OÜ , tarkvarainsener
2005 – 2007	Ixonos OÜ, testimismeeskonna koordinaator, grupijuht

Keelteoskus

Eesti keel	Hea
Inglise keel	Hea
Vene keel	Emakeel
Saksa keel	Keskmine

Kaitstud lõputööd

2005, Hierarhilise test generaatori DECIDER eksperimenaalne analüüs, M.Sc., juhendaja Dr. Jaan Raik, Tallinna Tehnikaülikool, Infotehnoloogia teaduskond

2004, Järjestikskeemide testigeneraatorite võrdlev analüüs., B.Sc., juhendaja Dr. Jaan Raik, Tallinna Tehnikaülikool, Infotehnoloogia teaduskond

Teaduspreemiad ja tunnustused

IKT doktorikooli stipendium doktorantidele, 2009

"Tiigriülikooli" stipendium IKT doktorantidele (EITSA), 2008.a

DAAD saksa keele kursuse stipendium, 2004.a

Teadustöö põhisuunad

Loodusteadused ja tehnika, Arvutiteadused (Registersiirde taseme järjestikskeemide automatiseeritud testvektorite generatsioon)

Teadustegevus

Projektid

1. Digitaalsüsteemide disain ja test (SF0142508s03).
2. Isetestivad digitaalsüsteemid (ETF5910).
3. Kõrgtaseme otsustusdiagrammidel põhinevad digitaalsüsteemide verifitseerimis- ja testimismeetodid (ETF7068).
4. Riistvara funktsionaalne verifitseerimine ja silumine (ETF8478).
5. Töökindlate sardsüsteemide disain (SF0140041s08).

Teadusartiklite, konverentsiettekannete loetelu on toodud ingliskeelses elulookirjelduses.