Linköping Studies in Science and Technology

Thesis No. 973

High-Level Test Generation and Built-In Self-Test Techniques for Digital Systems

by

Gert Jervan

INSTITUTE OF TECHNOLOGY
LINKÖPINGS UNIVERSITET

Submitted to the School of Engineering at Linköping University in partial fulfilment of the requirements for the degree of Licentiate of Engineering

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköping 2002

ISBN 91-7373-442-X

ISSN 0280-7971

Printed by UniTryck, Linköping, Sweden, 2002

High-Level Test Generation and Built-In Self-Test Techniques for Digital Systems

by

Gert Jervan

Oktober 2002
ISBN 91-7373-442-X
Linköpings Studies in Science and Technology
Thesis No. 973
ISSN 0280-7971
LiU-Tek-Lic-2002:46

ABSTRACT

The technological development is enabling production of increasingly complex electronic systems. All those systems must be verified and tested to guarantee correct behavior. As the complexity grows, testing is becoming one of the most significant factors that contribute to the final product cost. The established low-level methods for hardware testing are not any more sufficient and more work has to be done at abstraction levels higher than the classical gate and register-transfer levels. This thesis reports on one such work that deals in particular with high-level test generation and design for testability techniques.

The contribution of this thesis is twofold. First, we investigate the possibilities of generating test vectors at the early stages of the design cycle, starting directly from the behavioral description and with limited knowledge about the final implementation architecture. We have developed for this purpose a novel hierarchical test generation algorithm and demonstrated the usefulness of the generated tests not only for manufacturing test but also for testability analysis.

The second part of the thesis concentrates on design for testability. As testing of modern complex electronic systems is a very expensive procedure, special structures for simplifying this process can be inserted into the system during the design phase. We have proposed for this purpose a novel hybrid built-in self-test architecture, which makes use of both pseudorandom and deterministic test patterns, and is appropriate for modern system-on-chip designs. We have also developed methods for optimizing hybrid built-in self-test solutions and demonstrated the feasibility and efficiency of the proposed technique.

This work has been supported by the Swedish Foundation for Strategic Research (SSF) under the INTELECT program.

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Abstract

The technological development is enabling production of increasingly complex electronic systems. All those systems must be verified and tested to guarantee correct behavior. As the complexity grows, testing is becoming one of the most significant factors that contribute to the final product cost. The established low-level methods for hardware testing are not any more sufficient and more work has to be done at abstraction levels higher than the classical gate and register-transfer levels. This thesis reports on one such work that deals in particular with high-level test generation and design for testability techniques.

The contribution of this thesis is twofold. First, we investigate the possibilities of generating test vectors at the early stages of the design cycle, starting directly from the behavioral description and with limited knowledge about the final implementation architecture. We have developed for this purpose a novel hierarchical test generation algorithm and demonstrated the usefulness of the generated tests not only for manufacturing test but also for testability analysis.

The second part of the thesis concentrates on design for testability. As testing of modern complex electronic systems is a very expensive procedure, special structures for simplifying this process can be inserted into the system during the design phase. We have proposed for this purpose a novel hybrid built-in self-test architecture, which makes use of both pseudorandom and deterministic test patterns, and is appropriate for modern system-on-chip designs. We have also developed methods for optimizing hybrid built-in self-test solutions and demonstrated the feasibility and efficiency of the proposed technique.

Preface

Despite the fact that new design automation tools have allowed designers to work on higher abstraction levels, test-related activities are still mainly performed at the lower levels of abstraction. At the same time, testing is quickly becoming one of the most time and resource consuming tasks of the electronic system development and production cycle. Therefore, traditional gate-level methods are not any more practical nowadays and test activities should be migrated to the higher levels of abstraction as well. It is also very important that all design tasks can be performed with careful consideration of the overall testability of the resulting system.

The main objective of this thesis work has been to investigate possibilities to support reasoning about system testability in the early phases of the design cycle (behavioral and system levels) and to provide methods for systematic design modifications from a testability perspective.

The work presented in this thesis was conducted at the Embedded Systems Laboratory (ESLAB), Department of Computer and Information Science, Linköping University. It has been supported by the Swedish Foundation for Strategic Research (SSF) under the INTELECT program. Additional support was provided by the European Community via projects INCO-COPERNICUS 977133 VILAB ("Microelectronics Virtual Laboratory for Cooperation in Research and Knowledge Transfer") and IST-2000-29212 COTEST ("Testability Support in a Co-design Environment").

Our research is carried out in close cooperation with both the industry and with other, Swedish and international, research groups. We would like to mention here the very fruitful cooperation with the groups at Ericsson CadLab Research Center, Tallinn Technical University and Politecnico di Torino. Our work has also been regularly presented and discussed in the Swedish Network of Design for Test (SNDfT) meetings. This cooperation has opened new horizons and produced several results, some of which are presented in this thesis.

Acknowledgments

I would like to sincerely thank my supervisor Professor Zebo Peng for all support in the work toward this thesis. Zebo has always given me excellent guidance and I have learned a lot from him. He has also given me the opportunity and support to work with problems not directly related to the thesis, but very relevant for understanding the research organization and administrative processes. A very special thank should go to Professor Petru Eles, who has always been an excellent generator of new ideas and enriched our regular meetings with very useful remarks.

The colleagues at IDA have provided a nice working environment and I would especially like to thank the former and present members of ESLAB. They have through the past few years grown to be more than colleagues but good friends. Their support and encouragement as well as the wonderful atmosphere at ESLAB have been very important.

Many thanks also to Professor Raimund Ubar from Tallinn Technical University, who is responsible for bringing me to the wonderful world of science. The continuing cooperation with him has produced several excellent results, some of which are presented also in this thesis.

The cooperation with Gunnar Carlsson from Ericsson CadLab Research Center has provided invaluable insight into the industrial practices and helped me to understand the real-life testing problems more thoroughly.

Finally I would like to thank my parents, my sister and all my friends. You have always been there, whenever I have needed it.

Gert Jervan
Linköping, September 2002

Contents

Chapter 1
Introduction

This thesis deals with testing and design for testability of modern digital systems. In particular, we propose a novel hierarchical test generation algorithm that generates test vectors starting from a behavioral description of a system and enables testability analysis of the resulting system with very limited knowledge about the final implementation architecture.

We also propose a hybrid built-in self-test (BIST) architecture for testing systems-on-chip, which supports application of a hybrid test set consisting of a limited number of pseudorandom and deterministic test patterns, and methods for calculating the optimal combination of those two test sets.

This chapter first presents the motivation behind our work and the problem formulation. This will be followed by a summary of the main contributions together with an overview of the structure of the thesis.

1.1 Motivation

Hardware testing is a process to check whether a manufactured integrated circuit is error-free. As the produced circuits may contain different types of errors or defects that are very complex, we have to define a model to represent these defects to ease the test generation

and test quality analysis problems. This is usually done at the logic level. Test patterns are then generated based on a defined fault model and applied to the manufactured circuitry. It has been proven mathematically that the generation of test patterns is an NP-complete problem [27] and therefore different heuristics are usually used. Most of the existing hardware testing techniques work at the abstraction levels where information about the final implementation architecture is already available. Due to the growth of systems complexity these established low-level methods are not any more sufficient and more work has to be done at abstraction levels higher than the classical gate and register-transfer level (RT-level) in order to ensure that the final design is testable and the time-to-market schedule is followed.

More and more frequently designers also introduce special structures, called design for testability (DFT) structures, during the design phase of a digital system for improving its testability. Several such approaches have been standardized and widely accepted. However, all those approaches entail an overhead in terms of additional silicon area and performance degradation. Therefore it will be highly beneficial to develop DFT solutions that not only are efficient in terms of testability but also require minimal amount of overhead.

Most of the DFT techniques require external test equipment for test application. BIST technique, on the other hand, implements all test resources inside the chip. This technique does not suffer from the bandwidth limitations which exist for external testers and allows to apply at-speed tests. The disadvantage of this approach is that it cannot guarantee sufficiently high fault coverage and may lead to very long test sequences. Therefore a hybrid BIST approach that is implemented on-chip and can guarantee high fault coverage can be very profitable when testing modern systems-on-chip (SoC).

1.2 Problem Formulation

The previous section has presented the motivation for our work and indicated also the current trends in the area of digital systems testing.

The aim of our work is twofold. First, we are interested in performing test pattern generation and testability analysis as early as possible in the design process and, secondly, we would like to propose a BIST strategy that can be used for reducing the testing effort for modern SoC deigns.

To deal with the first problem we would like to develop a method that allows generation of test vectors starting directly from an implementation independent behavioral description. The developed method would have an important impact on the design flow, since it would allow us to deal with testability issues without waiting for the structural description of the system to be ready. For this purpose high-level fault models and testability metrics should also be investigated in order to understand the links between high- and low-level testability.

Since BIST structures are becoming commonplace in modern complex electronic systems, more emphasis should be put into minimization of costs caused by insertion of those structures. Our second objective is to develop a hybrid BIST architecture that can guarantee high test quality by combining pseudorandom and deterministic test patterns, while keeping the requirements for BIST overhead low. We are particularly interested in methods to find the optimal combination of those two test sets as this can lead to significant reductions of the total test cost.

1.3 Contributions

The main contributions of this thesis are as follows:

- **A novel hierarchical test pattern generation algorithm at the behavioral level.** We propose a test generation algorithm that works at the implementation-independent behavioral level and requires only limited knowledge about the final implementation architecture. The approach is based on a hierarchical test generation method and uses two different fault models. One fault model is used for modeling errors in the system behavior and the other is related to the failures in the final implementation. This allows us to perform testability evaluation of the resulting system at the early stages of the design flow. Also it can identify possible hard-to-test modules of the system without waiting for the final implementation. In this way, appropriate DFT structures can be incorporated into the early design to avoid the time-consuming testability-improvement task in the later design stages. We perform experiments to show that the generated test vectors can be successfully used for detecting stuck-at faults and that our algorithm, working at high levels of abstraction, allows significant reduction of the test generation effort while keeping the same test quality.
- **A hybrid built-in self-test architecture and its minimization.** We propose to use, for self-test of a system, a hybrid test set which consists of a limited number of pseudorandom and deterministic test vectors. The main idea is to first apply a limited number of pseudorandom test vectors, which is then followed by the application of the stored deterministic test set specially designed to shorten the pseudorandom test cycle and to target the random resistant faults. For supporting such a test strategy we have developed a hybrid BIST architecture that is implemented using mainly the resources available in the system. As the test length is one of the very important parameters in the final test cost, we have to

find the most efficient combination of those two test sets, while not sacrificing the test quality. In this thesis we propose several different algorithms for calculating possible combinations between pseudorandom and deterministic test sequences and provide a method for finding the optimal solution.

1.4 Thesis Overview

The rest of the thesis is structured as follows. Chapter 2 discusses briefly a generic design flow for electronic systems, the VHDL language and the decision diagrams which are used in our test generation procedure and, finally, some basic concepts concerning hardware test and testability.

Chapter 3 describes our hierarchical test pattern generation algorithm. It starts with an introduction to the related work, which is followed by a more detailed discussion of behavioral level decision diagrams. Thereafter we describe selected fault models and present our test pattern generation algorithm. The chapter concludes with experimental results where we demonstrate the possibility to use our approach for early testability analysis and its efficiency for generating manufacturing tests.

In Chapter 4 we present our hybrid BIST architecture for testing systems-on-chip. We describe the main idea of the hybrid BIST and propose methods for calculating the total cost of such an approach together with methods to find the optimal solution. The chapter is concluded with experimental results to demonstrate the feasibility of our approach.

Chapter 5 concludes this thesis and discusses possible directions for our future work.

Chapter 2
Background

In this chapter a generic design flow for electronic systems is presented first. It is then followed by a discussion of the test and verification related activities with respect to the tasks which are part of this design flow. Additionally, design representations on different abstraction levels are discussed and, finally, some basic concepts concerning test and testability are introduced.

2.1 Design Flow

Due to the rapid advances in technology and the progress in the development of design methodologies and tools, the fabrication of more and more complex electronic systems has been made possible in recent years. In order to manage complexity, design activities are moving toward higher levels of abstraction and the design process is decomposed into a series of subtasks [45], which deal with different issues. This thesis will focus on the hardware part of electronic systems. We will therefore not discuss here aspects related to the hardware/software co-design of embedded systems, nor the development of software components.

The design process of a complex hardware system typically consists of the following main tasks:

1. System-level synthesis: The specification of a system at the highest level of abstraction is usually given by its functionality and a set of implementation constraints. The main task at this step is to decompose the system into several subsystems (communicating processes) and to provide a behavioral description for each of them, to be used as an input for behavioral synthesis.
2. Behavioral synthesis starts out with a description, specifying the computational solution of the problem, in terms of operations on inputs in order to produce the desired outputs. The basic elements that appear in such descriptions are similar to those of programming languages, including control structures and variables with operations applied to them. Three major subtasks are:
   - Resource allocation (selection of appropriate functional units),
   - Scheduling (assignment of operations to time slots), and
   - Resource assignment (mapping of operations to functional units).

   The output of the behavioral synthesis process is a description at a register-transfer level (RTL), consisting of a datapath and a controller. The datapath, which typically consists of functional units (FUs), storage and interconnected hardware, performs operations on the input data in order to produce the required output. The controller controls the type and sequence of data manipulations and is usually represented as a state-transition table, which can be used in later synthesis stages for controller synthesis.
3. RT-level synthesis then takes the RTL description produced by the previous step, which is divided into the datapath and the controller, as input. For the datapath, an improvement of

resource allocation and assignment can be done, while for the controller actual synthesis is performed by generating the appropriate controller architecture from the input consisting of states and state transitions.

4. Logic synthesis receives as input a technology independent description of the system, specified by blocks of combinational logic and storage elements. It deals with the optimization and logic minimization problems.
5. Technology mapping has finally the task of selecting appropriate library cells of a given target technology for the network of abstract gates produced as a result of logic synthesis, concluding thus the synthesis pipeline. The input of this step is a technology independent multi-level logic structure, a basic cell library, and a set of design constraints.

According to the current state of the art, for verification, designs are simulated on different abstraction levels. Testability issues are currently just becoming incorporated into the standard design-flows, although several testability techniques, like scan and self-test, are well investigated and ready to be used. At the same time, testing is one of the major expenses in the integrated circuit (IC) development and manufacturing process, taking up to 35% of all costs. Test, diagnosis and repair cost of complex electronic systems reaches 40-50% of the total product realization cost and very soon the industry might face the challenge that test of a transistor is more expensive than manufacturing it [28].

2.2 VHDL and Decision Diagrams

Throughout the design flow a system is modeled at different levels of abstraction. At higher levels of abstraction it contains fewer details and is therefore easier to handle. By going towards lower levels of abstraction, more details will be added and the model will become more implementation dependent.

In the following section a hardware description language and a particular model of computation, which are relevant for this thesis, will be shortly discussed.

2.2.1 System and Behavioral Specifications

A design process typically starts from an implementation independent system specification. Among the synthesis tasks at the system level are the selection of an efficient implementation architecture and also the partitioning of the specified functionality into components, which will be implemented by hardware and software, respectively.

After the initial system specification and system synthesis steps the hardware part of the system is described at a behavioral level. A behavioral specification captures only the behavior of the design and does not contain information about its final implementation, such as structure, resources and timing. In our approach we use for the behavioral synthesis the CAMAD high-level synthesis system [15], developed at Linköping University. It accepts as an input a behavioral specification given in S'VHDL [14], a subset of VHDL. For test generation purposes the S'VHDL specification will be converted into a Decision Diagram model. In the following, a short overview of both VHDL and Decision Diagrams is given.

2.2.2 VHDL

The IEEE Standard VHDL hardware description language has its origin in the United States Government's Very High Speed Integrated Circuits (VHSIC) program, initiated in 1980. In 1987 the language was adopted by the IEEE as a standard; this version of VHDL is known as the IEEE Std. 1076-1987 [29]. A new version of the language, VHDL'92 (IEEE Std. 1076-1993) [30], is the result of a revision of the initial standard in 1993.

VHDL is designed to fill a number of needs in the design process. It allows multi-level descriptions and provides support for both a

behavioral and a structural view of hardware models with their mixture in description being possible.

S'VHDL [14] is defined as a subset of VHDL with the purpose of using it as input for high-level hardware synthesis. It is designed to accommodate a large behavioral subset of VHDL, particularly those constructs relevant for synthesis and to make available most of VHDL's facilities that support the specification of concurrency.

2.2.3 Decision Diagrams

A Decision Diagrams (DD) (previously known also as Alternative Graph) [57], [58] may represent a (Boolean or integer) function $y=F(X)$ implemented by a component or subcircuit in a digital systems. Here, y is an output variable, and X is a vector of input variables of the represented component or subcircuit.

In the general case, a DD that represents a function $y=F(X)$ is a directed, acyclic graph with a single root node. The nonterminal nodes of a DD are labeled with variables and terminal nodes with either variables, functional subexpressions or constants. Figure 1 shows, as an example, a fragment of an RT-level datapath and its corresponding DD representation.

When using DDs to describe complex digital systems, we have, at the first step, to represent the system by a suitable set of interconnected components (combinational or sequential ones). At the second step, we have to describe these components by their corresponding functions which can be represented by DDs. DDs which describe digital systems at different levels may have special interpretations, properties and characteristics. However, for all of them, the same formalism and the same algorithms for test and diagnosis purposes can be used, which is the main advantage of using DDs. In the following subsections some examples of digital systems and their representation using DDs will be given.

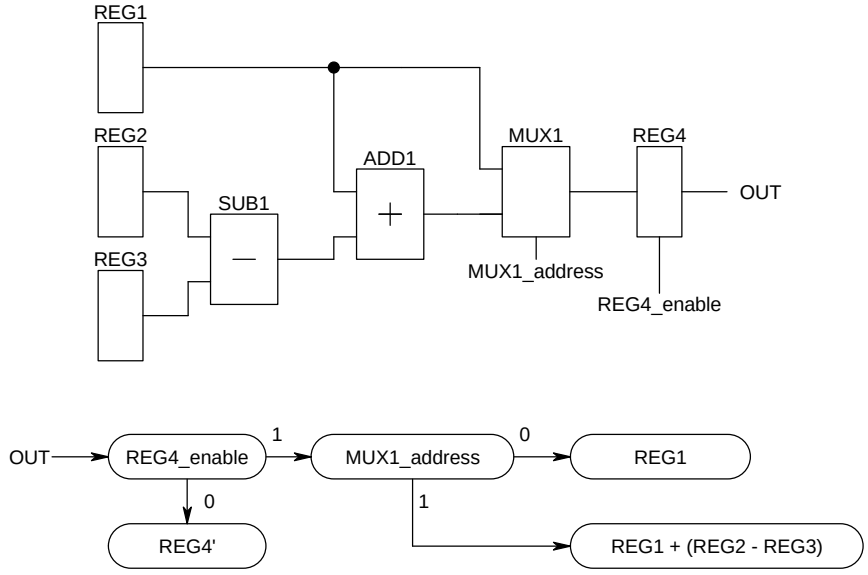


Figure 1. A datapath fragment and its DD representation

2.2.3.1 Gate-Level Combinational Circuits

Each output of a combinational circuit is defined at the gate-level by some Boolean function, which can be represented as a DD. The nonterminal nodes of this type of DD are labeled by Boolean variables and have consequently only two output branches. The terminal nodes are labeled by logical constants {0, 1}, or Boolean variables.

This type of DDs is called Binary Decision Diagrams (BDD), and there exists a special type of BDD, called Structurally Synthesized Binary Decision Diagrams (SSBDD). In SSBDDs there exists an one-to-one relationship between the DD nodes and the signal paths in the corresponding combinational circuit. This property of SSBDDs is very important because it allows us to generate tests for structural faults in circuits. An example of a combinational circuit and its

superposition-based construction of the corresponding SSBDD is given in Figure 2.

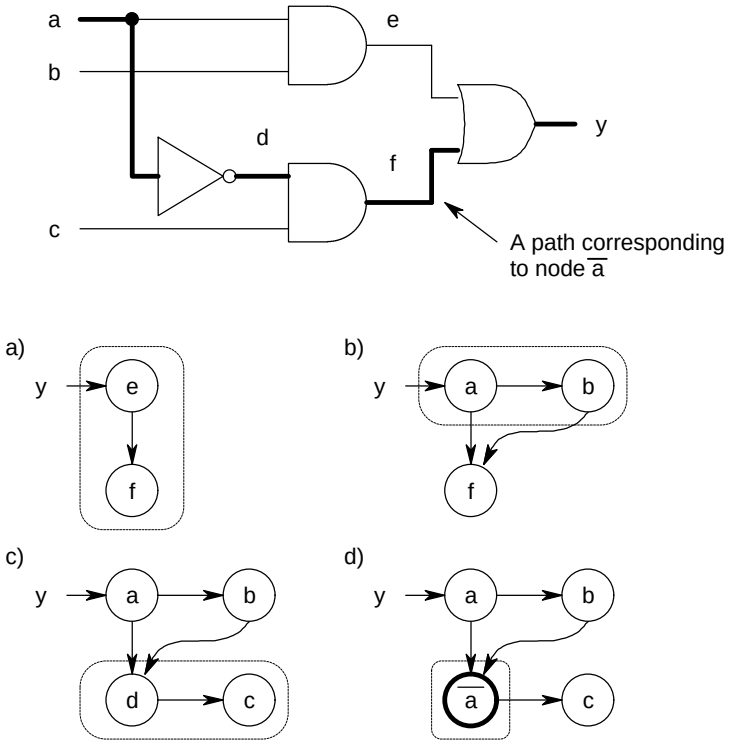


Figure 2. SSBDD for a combinational circuit

By convention the right-hand edge of an SSBDD node corresponds to 1 and the lower edge to 0. In addition, terminal nodes holding constants 0 and 1 are omitted. Therefore, exiting the SSBDD rightwards corresponds to y=1 (y denotes the output), and exiting the SSBDD downwards corresponds to y=0.

The SSBDD construction process starts from the output gate of the circuit. We replace every gate with its corresponding BDD representation. For example in Figure 2a the BDD of the output OR-gate is depicted. By using superposition, starting from the output gate, we can compress all BDDs in a tree-like subcircuit into one

single SSBDD. This process is illustrated in Figures 2b – 2d, where at every stage one internal node is replaced with its corresponding BDD. The final SSBDD is depicted in Figure 2d. As mentioned earlier, in an SSBDD there exists an one-to-one relationship between nodes and signal paths in the corresponding circuit. This situation is illustrated in Figure 2d, where node $\bar{a}$ corresponds to the highlighted path in the original circuit.

2.2.3.2 Digital Systems at the Register Transfer Level

In Boolean DD descriptions the DD variables have Boolean (i.e. single bit) values, whereas in register-transfer level DD descriptions, in general, multi-bit variables are used. Traditionally, on this level a digital system is decomposed into two parts – a datapath and a control part. The datapath is represented by sets of interconnected blocks (functional units), each of which can be regarded as a combinational circuit, sequential circuit or a digital system. In order to describe these blocks, corresponding types of DDs can be used.

The datapath can be described as a set of DDs, in the form where for each register and for each primary output a DD is used to capture the corresponding digital function. Here, the non-terminal nodes represent the control signals coming from the control part and terminal nodes represent signals of the datapath, i.e. primary inputs, registers, operations and constants. For example, Figure 3 depicts a fragment of a datapath, which consists of one register, one multiplexer and one FU, and the corresponding register-oriented DD. Signals S_i and OUT_i are control signals coming from the control part.

The control part is described usually as a FSM state table. The state table can be represented by a single DD where non-terminal nodes represent current state and inputs for the control part (i.e. logical conditions), and terminal nodes represent the next state logic and control signals going to the datapath. Figure 4a shows a fragment of a FSM state table with corresponding DD representation given in Figure 4b. The example shows the situation when the system is in state “S3” and INPUT1=1.

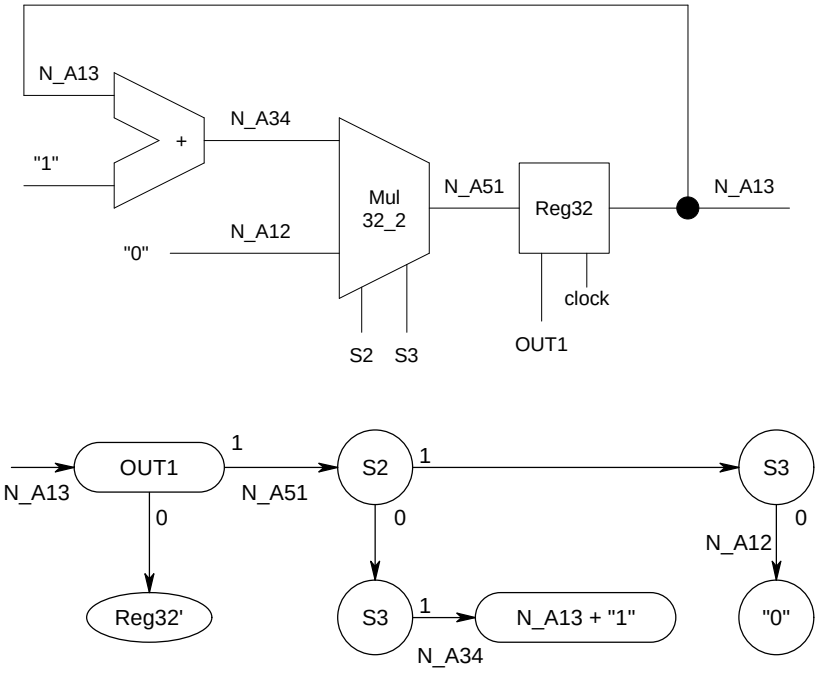


Figure 3. DD representation of a datapath

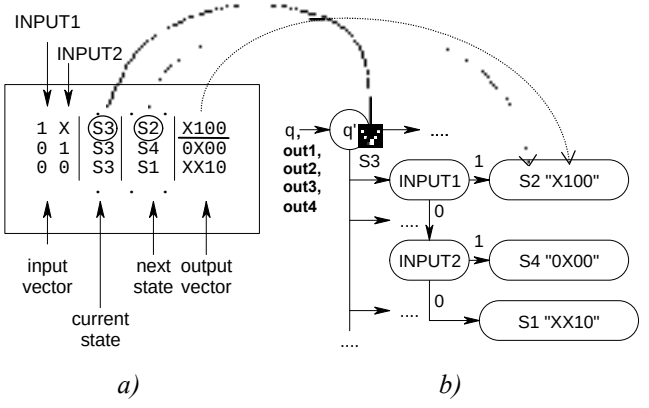


Figure 4. DD representation of a FSM

2.2.3.3 Digital Systems at the Behavioral Level

In the case of systems at the behavioral level, DDs describe their behaviors instead of their structures. The variables in the nonterminal nodes can be either Boolean (describing flags, logical conditions, etc.) or integer (describing instruction words, control fields, etc.). The terminal nodes are labeled with constants, variables (Boolean or integer) or by expressions for calculating integer/Boolean values. The number of DDs, used for describing a digital system, is equal to the number of output and internal variables used in the behavioral description.

More details about using DDs to describe digital systems at the behavioral level will be given in chapter 3.

2.3 Digital Systems Testing

Reliable electronic systems are not only needed in the areas where failures can lead to catastrophic events but also increasingly required in all application domains. A key requirement for obtaining reliable electronic systems is the ability to determine that the systems are error-free [7].

Although electronic systems contain usually both hardware and software, the main interest of this thesis is hardware testing and especially digital hardware testing. Hardware testing is a process to detect failures primarily due to manufacturing defects as well as aging, environment effects and others. It can be performed only after the design is implemented on silicon by applying appropriate stimuli and checking the responses. Generation of such stimuli together with calculation of the expected response is called test pattern generation. Test patterns are in practice generated by an automatic test pattern generation tool (ATPG) and typically applied to the circuit using automatic test equipment (ATE). Due to the increasing speed of systems and external tester bandwidth limitations, there exist approaches where the main functions of the external tester have

been moved onto the chip. Such practice is generally known as built-in self-test (BIST).

Test pattern generation belongs to a class of computationally difficult problems, referred to as NP-complete [27]. Several approaches have been developed to handle test generation for relatively large combinational circuits in a reasonable time. Test generation for large sequential circuits remains, however, an unsolved problem, despite rapid increase of computational power. According to [22], available test techniques can be classified into the following categories:

1. Functional testing, which relies on exercising the device under test (DUT) in its normal operational mode, and consequently, at its rated operational speed;
2. Testing for permanent structural faults (like stuck-at, stuck-open, bridging faults) that do not require the circuit to operate at rated speed during test;
3. Testing based on interactive fault analysis in which faults are derived from a simulation of the defect generation mechanisms in an integrated circuit (IC) (such faults tend to be permanent and do not require the circuit to be tested at rated speed);
4. Testing for delay faults that require the circuit to operate at rated speed during test;
5. Current measurement based testing techniques, which typically detect faulty circuits by measuring the current drawn by the circuit under different input conditions while the circuit is in the quiescent state.

2.3.1 Failures and Fault models

A failure is defined as an incorrect response in the behavior of the circuit.

According to [22] there are two views of failures:

- Physical/Design domain: defects (they produce a deviation from specification)
  - On the device level: gate oxide shorts, metal-to-polysilicon shorts, cracks, seal leaks, dielectric breakdown, impurities, bent-broken leads, solder shorts and bonding.
  - On the board level: missing component, wrong component, miss-oriented component, broken track, shorted tracks and open circuit.
  - Incorrect design (functional defect).
  - Wearout/environmental failures: temperature related, high humidity, vibration, electrical stress, crosstalk and radiation (alpha particles, neutron bombardment).
- Logical domain: faults (structural faults). A fault is a model that represents the effect of a failure by means of the change that is produced in the system signal.
  - Stuck-at faults: single, multiple.
  - Bridging faults: AND, OR, non-feedback and feedback.
  - Delay faults: gate and interconnect.

The oldest form of testing relies on a functional approach, where the main idea is to exercise the DUT in its normal operational mode. The main task of functional testing is to verify that the circuit operates according to its specifications. For functional testing, the same set of test vectors that was used by the designer for verification during the design phase can be used. Functional testing can cover a relatively large percentage of faults in an IC but the disadvantage of this technique is the large size of the test sequences needed to achieve high test quality. Using this approach alone for testing complex digital circuits is therefore not practical.

Structural fault model based techniques are the most investigated testing techniques. The earliest and the most well-known structural fault model is the single stuck-at (SSA) fault model (also called single stuck line (SSL) fault model), which assumes that the defect will cause a line in the circuit to behave as if it is permanently stuck at a

logic value 0 (stuck-at-0) or 1 (stuck-at-1). The SSA model assumes that the design contains only one fault. However, with decreased device geometry and increased gate density on the chip, the likelihood is greater that more than one SSA fault can occur simultaneously and they may mask each other in such a way that the SSA test vectors cannot detect them. Therefore it may be necessary to assume explicitly multiple stuck-at faults as well.

The single stuck-at fault model became an industrial standard in 1959 [13]. Experiments have shown that this fault model can be very useful (providing relatively high defect coverage) and can be used even for identifying the presence of multiple faults which can mask each other's impact on the circuit behavior. The possibility to analyze the behavior of the circuit using Boolean algebra has contributed to research in this domain very much. There are several approaches to identify test vectors using purely Boolean-algebraic techniques, search algorithm based techniques or techniques based on the combination of the two. But there are also several problems related to the SSA fault model, which become more obvious with the growth of the size of an IC. The main problem lies on the fact that the computation process to identify tests can be extremely resource and time intensive and, additionally, the stuck-at fault model is not good at modeling certain failure modes of CMOS, the dominant IC manufacturing technology at the present time.

During recent years several other fault models (e.g. stuck-OPEN and bridging) have gained popularity but these fault models still cannot solve the problems with CMOS circuits. As a solution to these problems, two technologies have been proposed: Inductive fault analysis (IFA) [50] and, more recently, inductive contamination analysis (ICA) [38]. These techniques present a closer relationship between physical defects and fault models. The analysis of a fault is based on analyzing the given manufacturing process and layout of a particular circuit.

A completely different aspect of fault model based testing is testing for delay faults. An IC with delay faults operates correctly at

sufficiently low speed, but fails at rated speed. Delay faults can be classified into gate delay faults (the delay fault is assumed to be lumped at some gate output) and path delay faults (the delay fault is the result of accumulation of small delays as a signal propagates along one or more paths in a circuit).

All methods mentioned above rely on voltage measurement during testing; but there are also techniques which are based on current measurement. These techniques are commonly referred as IDDQ test techniques. The technique is based on measuring the quiescent current and can detect some of the faults which are not detectable with other testing techniques (except exhaustive functional testing). IDDQ testing can be also used for reliability estimation. The disadvantage of this technique is the very slow testing process, which makes testing very expensive.

2.3.2 Test Pattern Generation

Test pattern generation is the process of determining the stimuli necessary to test a digital system. The simplest approach for combinational circuits is exhaustive testing where all possible input patterns will be applied, which means applying 2^n test patterns (where n is the number of inputs). Such large number of test patterns means that exhaustive testing is possible only with small combinational circuits. As an example, a circuit with 100 inputs needs already $2^{100} \approx 10^{30}$ test patterns and is therefore practically infeasible. An alternative for exhaustive testing is pseudorandom testing, where test patterns are generated in pseudorandom manner. The cost of this type of test is considerably reduced but pseudorandom patterns cannot detect all possible faults and for so called random pattern-resistant faults we still need some type of deterministic tests.

To overcome those problems, several structural test generation techniques have been developed. In this case we assume that the elementary components are fault-free and only their interconnects are affected [1]. This will reduce the number of test patterns to 2n in

the case of the single stuck-at fault model. The typical cycle of a structural test generation methodology is depicted in Figure 5.

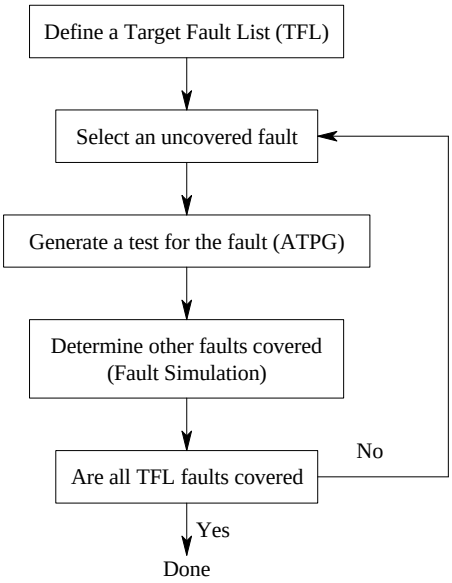


Figure 5. Structural test flow

There has been a lot of research in the area of test pattern generation and the current status is that test pattern generation for combinational circuits as well as for sequential circuits without global feedback is a solved problem and there exist commercial tools for it. Test pattern generation for complex sequential circuits remains still an unsolved problem (due to high complexity involving multiple time frames and other factors) and there is some skepticism about the possibility to have efficient commercial solutions available in the nearest future.

2.3.3 Test Application

As previously mentioned, hardware testing involves test pattern generation, discussed above, and test application. Test application

can be performed either on-line or off-line. The former denotes a situation where testing is performed during normal operational mode and the latter when the circuit is not in normal operation. The primary interest of this thesis is off-line testing although some of the results can be applied also for on-line testing as well.

Off-line tests can be generated either by the system itself or outside the chip and applied by using Automatic Test Equipment (ATE). With the emerging of sub-micron and deep sub-micron technologies, the ATE approach is becoming increasingly expensive, the quality of the tests and therefore also the quality of the device deteriorates, and time to market becomes unacceptably long. Therefore several methods have been developed to reduce the significance of external testers and to reduce the cost of the testing process, without compromising on quality. Those methods are known as design for testability (DFT) techniques. In the following, different DFT techniques are described.

2.3.4 Design for Testability

Test generation and application can be more efficient when testability is already considered and enhanced during the design phase. The aim of such an enhancement is to improve controllability and observability with minimal area and performance overhead. Controllability and observability together with predictability are the most important factors that determine the complexity of deriving a test set for a circuit. Controllability is the ability to establish a specific signal value at each node in a circuit by setting values on the circuit's inputs. Observability, on the other hand, is the ability to determine the signal value at any node in a circuit by controlling the circuit's inputs and observing its outputs. DFT techniques, used to improve a circuit's controllability and observability, can be divided into two major categories:

- DFT techniques which are specific to one particular design (ad hoc techniques) and cannot be generalized to cover different

types of designs. Typical examples are test point insertion and design partitioning techniques.

- Systematic DFT techniques are techniques that are reusable and well defined (can be even standardized).

In the following sections some systematic DFT techniques are discussed.

2.3.5 Scan-Design

To cope with the problems caused by global feedback and complex sequential circuits, several different DFT techniques have been proposed. One of them is internal scan. The general idea behind internal scan is to break the feedback paths and to improve the controllability and observability of the memory elements by introducing an over-laid shift register called scan path. Despite the increase in fault coverage, there are some disadvantages with using scan techniques:

- Increase in silicon area,
- Larger number of pins needed,
- Increased power consumption,
- Increase in test application time,
- Decreased clock frequency.

There are two different types of scan-based techniques:

1. Full scan
2. Partial scan

In case of partial scan only a subset of the memory elements will be included in the scan path. The main reason for using partial scan is to decrease the cost and increase the speed of testing.

In the case of complex chips or printed circuit boards (PCB) it is often useful for the purposes of testing and fault isolation to isolate one module from the others. This can be achieved by using boundary scan.

Boundary scan is well defined and standardized (IEEE 1149.1 standard). Boundary scan targets manufacturing defects around the boundary of a device and the interconnects between devices. These are the regions most likely to be damaged during board assembly.

2.3.6 Built-In Self-Test

As discussed earlier, the traditional form of off-line testing requires the use of ATEs. One of the problems, while using ATEs, is the growing disparity between the external bandwidth (ATE speed) and the internal one (internal frequency of the circuit under test). And as the importance of delay faults is increasing with newer technologies, and the cost of test pattern generation as well as the volume of test data keep increasing with circuit size, alternative solutions are needed. One such solution is built-in self-test (BIST).

The main idea behind a BIST approach is to eliminate the need for the external tester by integrating active test infrastructure onto the chip. A typical BIST architecture consists of a test pattern generator (TPG), usually implemented as a linear feedback shift register (LFSR), a test response analyzer (TRA), implemented as a multiple input shift register (MISR), and a BIST control unit (BCU), all implemented on the chip (Figure 6). This approach allows applying at-speed tests and eliminates the need for an external tester. Furthermore, the BIST approach is also one of the most appropriate techniques for testing complex SoCs, as every core in the system can be tested independently from the rest of the system. Equipping the cores with BIST features is especially preferable if the modules are not easily accessible externally, and it helps to protect intellectual property (IP) as less information about the core has to be disclosed.

There are two widely used BIST schemes: test-per-clock and test-per-scan. The test-per-scan scheme assumes that the design already has an existing scan architecture. During the testing phase the TPG fills the scan chains which will apply their contents to the circuit under test (CUT) [12]. All scan outputs are connected to the multiple input signature register (MISR), which will perform signature

compaction. There are possibilities to speed up the test process by using multiple scan chains or by using a partial scan solution. An example of such an architecture is Self-Test Using MISR and Parallel Shift Register Sequence Generator (STUMPS) [5].

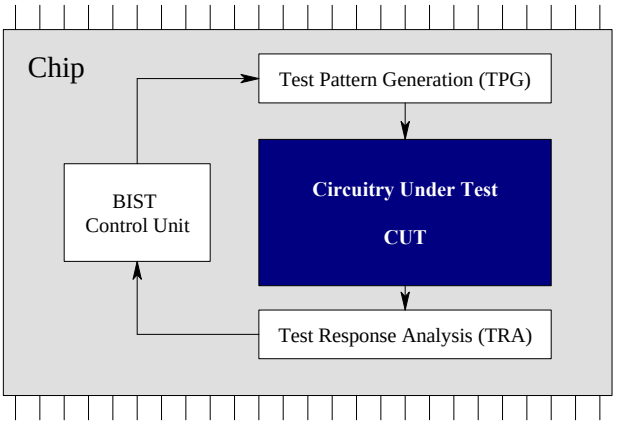


Figure 6. A typical BIST architecture

The test-per-clock scheme uses special registers that perform pattern generation and response evaluation. This approach allows to generate and to apply a new test pattern in each clock cycle. One of the first proposed test-per-clock architectures was the Built-In Logic Block Observer (BILBO), proposed in [40], which is a register that can operate both as a test pattern generator and a signature analyzer.

As the BIST approach does not require any external test equipment it can be used not only for production test, but also for field and maintenance test, to diagnose faults in field-replaceable units. Since the BIST technique is always implemented on the chip, using the same technology as the CUT, it scales very well with emerging technologies and can become one of the most important test technologies of the future.

2.4 Constraint Logic Programming

Most digital systems can be conceptually interpreted as a set of constraints, which is a mathematical formalization of relationships that hold in the system [44]. In the context of test generation, there are two types of constraints: the system constraints and the test constraints. The system constraints describe the relationships between the system variables, which capture the system functionality and requirements. The test constraints describe the relationships between the system variables in order to generate tests for the system. Constraint solving can be viewed as a procedure to find a solution to satisfy the desired test constraints for a system, if such a solution exists.

The easiest way for constraint solving is to enumerate all the possible values for the constraints and test if there exists a solution. Unfortunately enumeration methods are impractical in most cases. The problem of enumeration methods is that they only use the constraints in a passive manner, to test the result of applying values, rather than using them to construct values that will lead to a solution. There are lots of constraint solving strategies that make use of the types and number of constraints in order to speed up the solving process.

The backtracking strategy is a basic and important approach in constraint solving. Most constraint solvers such as CHIP [10], SICStus [51], etc., use the backtracking strategy as a basic method for constraint satisfaction. The search for a solution always involves a decision process. Whenever there are alternatives to solve a problem, one of them is chosen. If the selected decision leads to an inconsistency, backtracking is used in order to allow a systematic exploration of the complete space of possible solutions and recovery from the incorrect decision. Recovery involves restoring the state of the computation to the state existing before the incorrect decision.

For example, there are two possible solutions for the problem in Figure 7. We first choose one of them, D1, as a decision and try it. In this case, D11 and D12 are alternative decisions for finding a

solution with decision D1. We can select either D11 or D12 to try to find a solution. If decision D11 leads to an inconsistency, it means that the decision {D1,D11} cannot find a solution for the given problem. So the system will recover from D11 and try another alternative decision, D12. If the decision D12 also fails, it will cause the first decision D1 to fail. The system cannot find a solution with the selected decision D1. So we will go back to try another decision, D2, by backtracking. As shown in Figure 7, the selected decision D21 succeeds. It will lead to a solution for the given problem and the decision {D2,D21} is a correct decision for the problem. It is obvious that the ordering of the variables has an impact on the searching time.

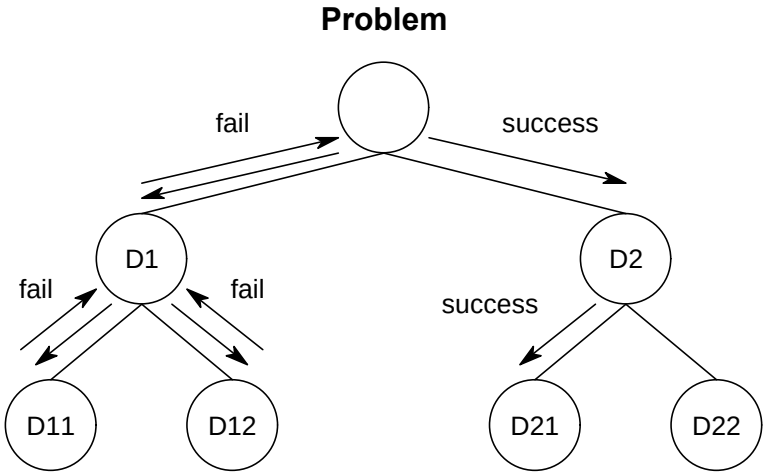


Figure 7: The backtracking strategy

Depending on the complexity of the problem, the search space, while using the previous search strategy, can become huge, and finding a solution is practically infeasible. Therefore, several heuristics have been developed that explore only part of the search space. Such is, for example, a search strategy which only spends a certain number of search cycles (credits) in each branch. If this credit is exhausted it goes back in the tree and chooses an alternative sub-

tree high-up in the (unexplored) tree to further explore. By controlling the amount of credit which is provided, we can control the search quite well. However, this approach may not be able to find the (best) solution, as it explores the search space only partially.

2.5 Conclusions

In this chapter we have presented some concepts that are important for understanding this thesis. We started with an introduction of a generic design flow for digital systems together with design representations at different levels of abstraction. We have given an overview of some basic hardware testing and DFT methodologies and, finally, introduced the concept of constraint solving.

As we were able to see, the design activities are moving toward higher levels of abstraction and there are well-established methods and tools to support this process. On the other hand, most of the test and DFT activities are still performed at the gate-level and this is becoming one of the limiting factors in the digital systems development cycle. Therefore, there is a strong demand for tools and methods that can handle test problems at a high level of abstraction.

Chapter 3
Hierarchical Test Generation at the Behavioral Level

As it has been shown in the introductory chapter, most of the test and DFT related activities are usually performed at the low abstraction levels. At the same time, design related activities have moved several levels up which has produced a large efficiency gap between design and test related tools.

In this chapter we propose an approach to reduce this gap. We present a novel hierarchical test generation approach, based on the analysis of a behavioral specification, which is able to produce test sequences during the early synthesis steps. Experimental results show that this approach can reduce the test generation effort, while keeping the same high quality in terms of fault coverage.

3.1 Introduction

In the past years, the introduction of new electronic design automation tools allowed designers to work on higher-levels of abstraction, leaving the synthesis of lower levels to automatic synthesis tools. At those early stages of the design flow only the behavior of the system is known, and very little information about

implementation is available. Even the partitioning between hardware and software components may not yet be decided. At the present time, such design environments, also known as hardware/software co-design environments, support an interesting set of facilities, allowing the designer to select the optimal solution for his/her design in terms of performance, cost (silicon area) and power consumption. Despite this trend, test-related activities are still performed at the gate-level, mainly because test is usually considered to be tightly linked to implementation details, which are absent at the higher levels. As a result, test constraints are taken into consideration much later in the design process, with some significantly negative consequences. First, in some cases the designer realizes very late (normally when a gate-level description of hardware modules is available) that the system has some critical points from the test point of view: this requires restarting from the earlier design phases, with very negative impact on time-to-market and cost. Secondly, since the area, performance, and power evaluations given by the co-design environments do not take care of test requirements, they can lead to significant approximations: as an example, by neglecting BIST structures one can make significant mistakes in the evaluation of the required silicon area.

Early information about testability of individual modules makes it also simpler to choose the best possible test strategy for every module and to perform test resource partitioning. For example, it can be highly beneficial to migrate some of the test activities from hardware to the software or vice versa. In such a way, the search space that designers explore for identifying the best architecture of a system is enriched with a new, test-related, dimension. Moreover, addressing testability issues starting from high-level descriptions can be highly beneficial, since it may allow the generation of good test sequences with high efficiency, and reduce the cost of design-for-testability structures.

In this thesis we propose an approach where tests are generated based on high-level hierarchical descriptions. We use as an input to our test pattern generator a behavioral level description of a system.

We extract behavioral level DDs, which will be used as a mathematical platform for test generation and improve our test generation environment by including some limited knowledge about the final architecture.

One of the main objectives of this thesis is to show how hierarchical test generation can be used at higher levels of abstraction and thus make it possible to reason about testability at very early stages of the design flow. We also want to demonstrate how DDs can be used for test generation at the behavioral level.

In this way, significant advantages could be achieved in terms of design cost (especially by reducing the time for designing a testable system) and design quality.

In the following section we will present some related work. We will turn our attention to some high-level fault models and hierarchical test generation, which are essential from this thesis' point of view. Thereafter we will discuss about the behavioral level decision diagrams and introduce the fault models we use. The following section will describe our hierarchical test generation approach together with some experimental results and finally the conclusions will be drawn.

3.2 Related Work

Recently, several researches have proved that testing can be addressed even when the circuit structure is not yet known, and that suitable techniques can be devised to exploit the information existing in a high-level description of a system for evaluating its testability and for reducing the cost for testing in the following design steps. Up to now, these new techniques have been experimented mainly with RT-level descriptions, and very few work has been done at the behavioral level. In the following we are looking into the high-level fault models and test generation algorithms proposed in the literature.

3.2.1 High-Level Fault Models

The ultimate task of a test generator is to generate such input sequences, which can distinguish an erroneous behavior of the system from the correct behavior. As it was discussed before, there can be several reasons behind an erroneous behavior: manufacturing defects, environmental or aging related failures, bugs in the specification and many others. For test generation purposes we need to describe the erroneous behavior through some type of mathematical formalism, which is called fault model. One fault model does not has to cover all possible defects, instead it usually targets a selected set of possible faults which enables higher accuracy of the model. As it was discussed earlier, there exists a wide spectrum of fault models over different levels of abstraction. At one end there are fault models describing electrical anomalies in deep submicron designs, at another end we have code coverage metrics for system level specifications. As an example, the dominant fault model for digital designs at the gate level has been for several decades a single stuck-at (SSA) fault model, as it can represent a large number physical defects, while being technology independent and simple. Although the SSA model has several limitations, it can be used successfully as the reference measure to quantify a circuit's testability and has therefore become an industrial standard. Therefore, also in this thesis the evaluation and comparison of generated test sequences is done at the gate level based on the SSA model.

One of the main problems while addressing test issues at an abstraction level higher than the traditional gate level is the identification of a suitable high-level fault model. At this level we have very little or no knowledge about the final implementation and therefore we cannot establish direct relationship between manufacturing defects and the fault model.

The most common high-level fault models proposed in literature as metrics of the goodness of sequences when working at high level of abstraction are mostly taken from the software testing area:

- Path coverage [6] measures the percentage of all possible control flow paths through the program executed by a given sequence. A related metric is statement coverage which is intended to measure the percentage of statements that are activated by a given test pattern. A further metric is branch coverage, which measures the percentage of branches of a model that are activated by a given test pattern.
- Bit coverage was proposed in [16]. The authors assume that each bit in every variable, signal or port in the model can be stuck to zero or one. The bit coverage measures the percentage of stuck-at bits that are propagated on the model outputs by a given test sequence.
- Condition coverage [16] is related to faults located in the logic implementing the control unit of a complex system. The authors assume that each condition can be stuck-at true or stuck-at false. Then, the condition coverage is defined as the percentage of stuck-at conditions that are propagated on the model outputs by a given test sequence.
- Mutation testing [11] concentrates on selecting test vectors that are capable to distinguish a program from a set of faulty versions or mutants. A mutant is generated by injecting a single fault into the program. For example, if we have the expression:

  ```
  X := (a + b) - c;
  ```

  To rule out the fault that the first "+" is changed to "–", b must not be 0 (because a + 0 = a – 0 and this fault cannot be detected). Additionally, to rule out the fault that instead of "+" there is "x", we have to assure that a + b ≠ a x b.

All those fault models target faults in the circuit's behavior, not in its structure. For targeting errors in the final implementation it is very important to establish a relationship between the high-level fault models and the lower level ones. This has been done so far only experimentally (e.g. [37]) and there are no systematic methods currently available. To overcome this problem, at least partially, we

propose to use a hierarchical fault model and hierarchical test generation.

3.2.2 Hierarchical Test Generation

The main idea of the hierarchical test generation (HTG) technique is to use information from different abstraction levels while generating tests. One of the main principles is to use a modular design style, which allows to divide a larger problem into several smaller problems and to solve them separately. This approach allows generating test vectors for the lower level modules based on different techniques suitable for the respective entities.

In hierarchical testing, two different strategies are known: top-down and bottom-up. In the bottom-up approach [47], tests generated at the lower level will be assembled at the higher abstraction level. The top-down strategy, introduced in [42], uses information, generated at the higher level, to derive tests for the lower level.

Previously mentioned as well as more recent approaches [48] have been successfully used for hardware test generation at the gate, logical and register-transfer (RT) levels.

In this thesis, the input to the HTG is a behavioral description of the design and a technology dependent, gate level library of functional units. Figure 8 shows an example of such a hierarchical representation of a digital design. It demonstrates a behavioral specification, a fragment of a corresponding behavioral level decision diagram and a gate level netlist of one of the functional units.

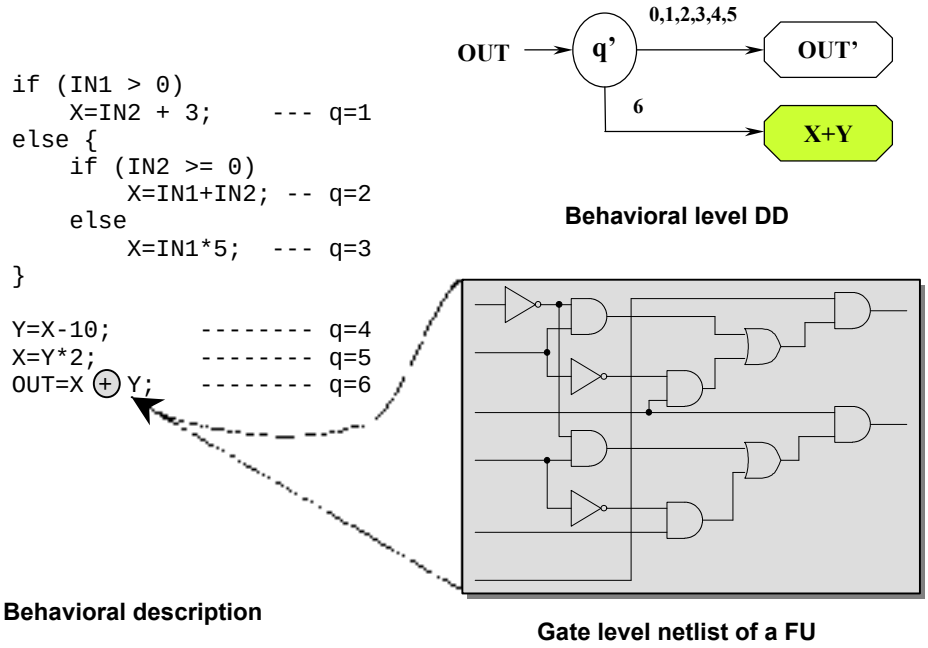


Figure 8. Hierarchical representation of a digital design

3.3 Decision Diagrams at the Behavioral Level

Our high-level hierarchical test generation approach starts from a behavioral specification, given in VHDL. At this level the design does not include any details about the final implementation, however we assume that a simple finite-state machine (FSM) has already been introduced and therefore the design is conceptually partitioned into the data path and control part. For this transformation we are using the CAMAD high-level synthesis system [15].

DD synthesis from a high-level description language consists of several steps, where data path and control part of the design will be converted into the DDs separately. In the following, an overview of

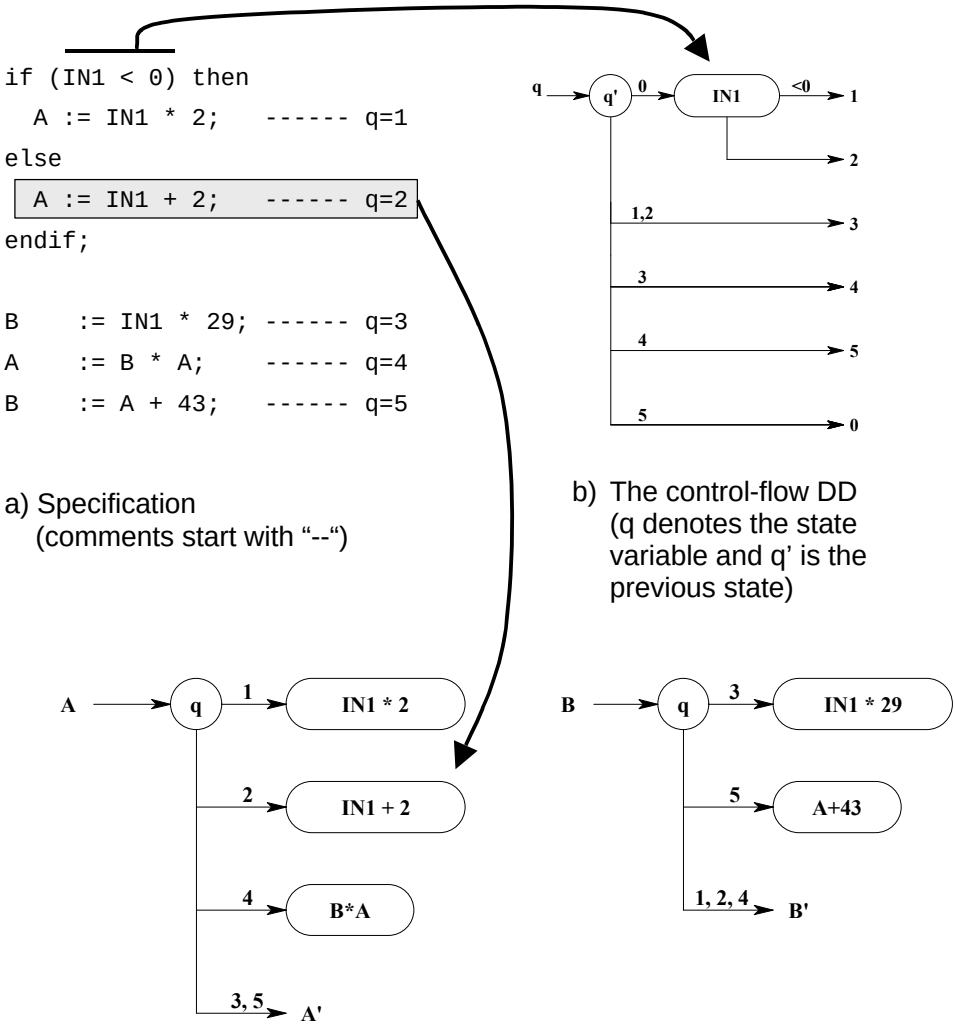
the DD synthesis process, starting from a VHDL description, will be given.

3.3.1 Decision Diagram Synthesis

In the general case, a DD is a directed, acyclic graph where non-terminal nodes represent logical conditions, terminal nodes represent operations, while branches hold the subset of condition values for which the successor node corresponding to the branch will be chosen. The variables in nonterminal nodes can be either Boolean (describing flags, logical conditions etc.) or integer (describing instruction words, control fields, etc.) The terminal nodes are labeled by constants, variables (Boolean or integer) or by expressions for calculating integer values.

At the behavioral level, for every internal variable and primary output of the design a data-flow DD will be generated. Such a data-flow DD has so many branches, as many times the variable appears on the left-hand side of the assignment. Further, an additional DD, which describes the control-flow, has to be generated. The control-flow DD describes the succession of statements and branch activation conditions.

Figure 9 depicts an example of DD, describing the behavior of a simple function. For example, variable *A* will be equal to *IN1+2*, if the system is in the state *q=2* (Figure 9c). If this state is to be activated, condition *IN1³0* should be true (Figure 9b). The DDs, extracted from a specification, will be used as a computational model in the HTG environment.



c) The data-flow DD

Figure 9. A decision diagram example

3.3.2 SICStus Prolog representation of Decision Diagrams

As described earlier, at the behavioral level there exist two types of DDs: control-flow DD and data-flow DDs. The control-flow DD carries two types of information: state transition information and path activation information. The state transition information captures the state transitions that are given in the FSM corresponding to the specified system. The path activation information holds conditions associated to state transitions.

For each internal or primary output variable corresponds one data-flow DD. In a certain system state, the value of a variable is determined by the terminal node in the data graph. In this case, the relationship between the terminal node and the variable can be viewed as a functional constraint on the variable at the state.

To generate a test pattern for a fault we have to excite the fault (justification) and to sensitize the fault effect at the primary outputs (propagation). For example, if we want to test the statement that is highlighted in Figure 9a, we have to bring the system to the state $q=2$. This can be guaranteed only when $q'=0$ and $IN1 \geq 0$. Those requirements can be seen as justification constraints.

For observing the fault effect at primary outputs, we have to distinguish between the faulty and the correct behavior of a variable under test (Variable "A" in our example). This requires, that $B \neq 0$ (from the statement `A:=B*A`) and consequently $IN1*29 \neq 0$ (from the statement `B:=IN1*29`), otherwise the variable "A" will have always value 0 and the fault cannot be detected. Those conditions can be seen as propagation constraints.

By solving the extracted constraints we will have a test pattern (combination of input values) which can excite the fault and propagate the fault effect to the primary outputs. For solving these constraints we employ a commercial constraint solver SICStus [51] and have developed a framework for representing a DD model in the form of constraints. First, we translate the control-flow DD into a set of state transition predicates and path activation constraints are extracted along the activated path. Then all the data-flow DDs are

parsed as functional constraints at different states by using predicates. Finally, a DD model is represented as a single Prolog module. See [54] for technical details about the translation process.

3.4 Hierarchical Test Generation Algorithm

This section presents our high-level hierarchical test generation algorithm. At first we introduce fault models used in our approach. Thereafter the corresponding tests are discussed and finally the whole test generation environment is presented.

3.4.1 Fault Modeling at the Behavioral Level

In this thesis we propose to use a hierarchical fault model where at the higher level we target errors in the system behavior and at the lower level our aim is to detect failures related to the chosen implementation style. In our approach we have chosen for the high level the branch coverage metric, while the low-level faults are modeled by using a SSA fault model. Those two fault models are complimentary to each other and the aim is to generate such test sequences, which can be used for manufacturing test of the final circuit.

As the fault model we are using is hierarchical, the HTG algorithm has to generate two types of tests. The first set is generated from the pure behavioral description based on the code coverage metric [32]. This test set targets errors in branch selection (nonterminal nodes of the control-flow DD). During the second test generation phase the functional blocks (e.g., adders, multipliers and ALUs) composing the behavioral model are identified (terminal nodes of the data-flow DD), and suitable test vectors are generated for the individual blocks. During the block-level test generation phase each block is considered as an isolated and fully controllable and observable entity; and a gate level test generation tool is used for this purpose. The test vectors generated for the basic blocks are then justified and their fault effects propagated in the behavioral model of the circuit under test.

In this way we can incorporate accurate structural information into the high-level test pattern generation environment while keeping the propagation and justification task still on a high abstraction level.

3.4.2 Test Pattern Generation

The test generation task is performed in the following way (Figure 10). Tests are generated sequentially for each nonterminal node of the control-flow DD. Symbolic path activation is performed and functional constraints are extracted. Solving the constraints gives us the path activation conditions to reach a particular segment of the specification. In order to test the operations, presented in the terminal nodes of the data-flow DD, different approaches can be used. In our approach we employ a gate level test pattern generator. In this way we can incorporate accurate structural information into the high-level test pattern generation environment while keeping the propagation and justification task still on a high abstraction level.

If the constraint solver is not able to find a solution, a new test case should be generated, if possible. This cycle should be continued until a solution is found or a timeout occurs.

In the following, the test pattern generation algorithm is described in more detail.

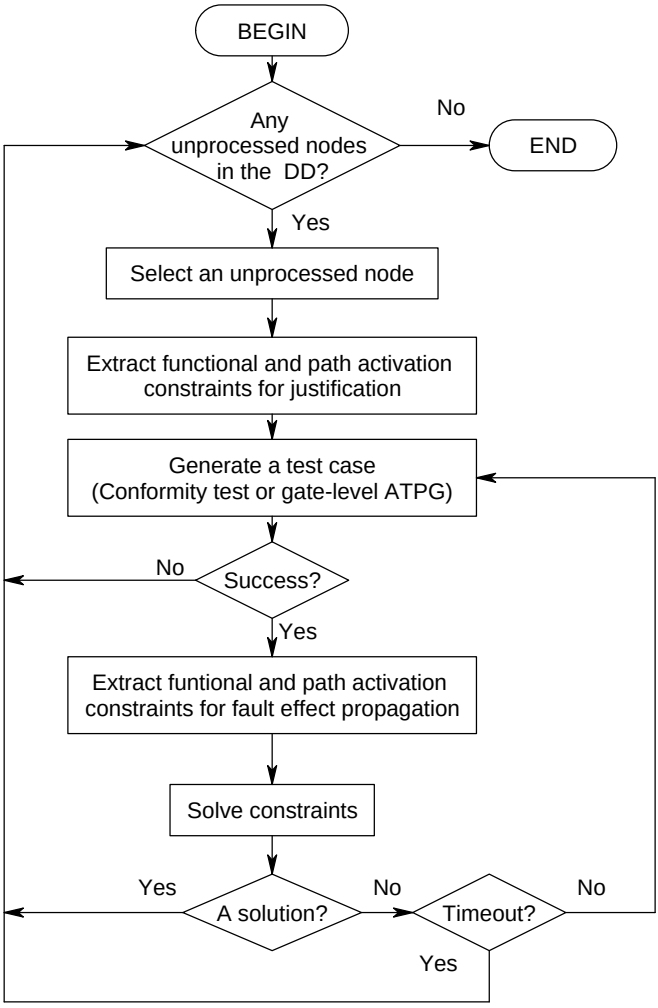


Figure 10. The general flow for hierarchical test generation algorithm

3.4.3 Conformity Test

For the nonterminal nodes of the control-flow DD, conformity tests will be applied. The conformity tests target errors in branch activation. For example, in order to test nonterminal node *IN1*

(Figure 11), one of the output branches of this node should be activated. Activation of the output branch means activation of a certain set of program statements. In our example, activation of the branch *IN1<0* will activate the branches in the data-flow DD where *q=1 (A:=X)*. For observability the values of the variables calculated in all the other branches of *IN1* have to be distinguished from the value of the variables calculated by the activated branch. In our example, node *IN1* is tested, in the case of *IN1<0*, if X^1Y. The path from the root node of the control-flow DD to the node *IN1* has to be activated to ensure the execution of this particular specification segment and the conditions generated here should be justified to the primary inputs of the module. This process will be repeated for each output branch of the node. In the general case there will be *n(n-1)* tests, for every node, where *n* is the number of output branches.

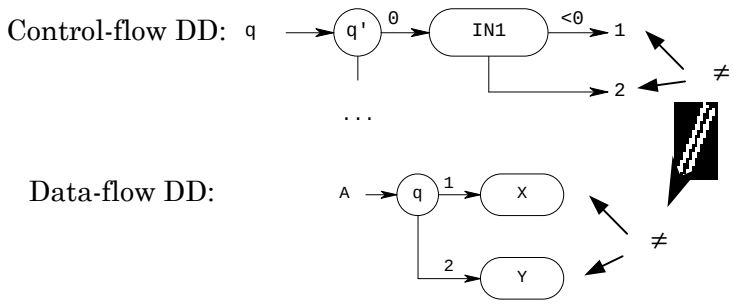


Figure 11. Conformity test

3.4.4 Testing Functional Units

Synthesis is the translation of a behavioral representation of a design into a structural one. One of the most important parameters guiding the synthesis process is the technology that will be used in the final implementation. After the technology is defined, the implementation details of the functional units (FUs), that will be used in the final design, can be found usually in the technology library. Our hierarchical test generation algorithm employs this

structural information for generating tests and estimating the testability of the final implementation when using one or another implementation of the FU from the same or even from completely different libraries. This reveals another advantage of our test pattern generation algorithm: we can derive information about the testability of a system, depending on what target technology it will be implemented in. We can generate tests for different possible implementations (different implementations of the same FU) and to select the solution that is the best from the testability point of view. This information can be used later in the synthesis while performing allocation and mapping.

Tests are generated in cooperation with low-level test pattern generators. The functional unit test generation is performed one by one for every FU given in the specification as depicted in Figure 12, where an example of generating low-level tests for an adder is given.

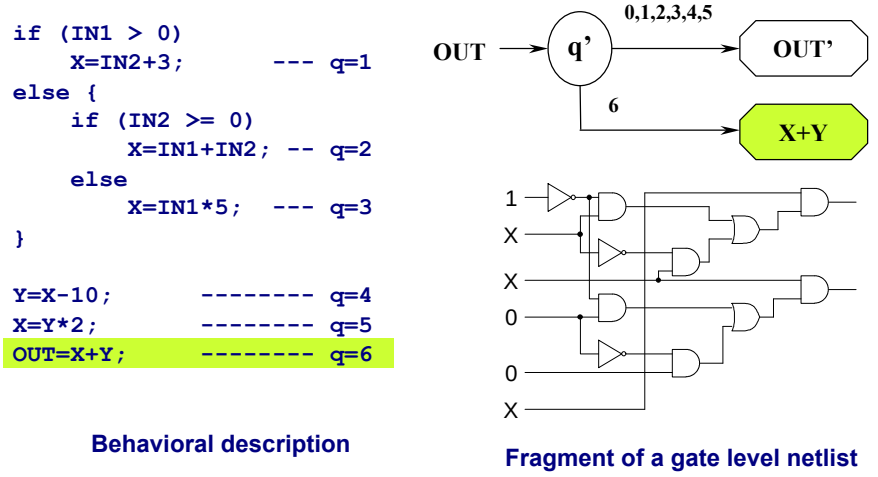


Figure 12. Testing functional units

We start by choosing a not tested operator from the specification and employ a gate level ATPG to generate a test pattern targeting structural faults in the corresponding FU. In our approach a PODEM like ATPG [31] is used but, in the general case, any gate level test

pattern generation algorithm can be applied. If necessary, pseudorandom patterns can be used for this purpose as well. The test patterns, which are generated by our current approach, can have some undefined bits (don't cares). As justification and propagation are performed at the behavioral level by using symbolic methods these undefined bits have to be set to a defined value. Selecting the exact values is an important procedure since not all possible values can be propagated through the environment, which potentially can lead to the degradation of fault coverage. A test vector that does not have any undefined bits is thereafter forwarded to the constraint solver, where together with the environmental constraints it forms a test case. Solving such a test case successfully means that the generated low-level test vector can be justified till the primary inputs and the fault effect is observable at the primary outputs. If the constraint solver cannot find an input combination that would satisfy the given constraints, another combination of values for the undefined bits has to be chosen and the constraint solver should be employed again. This process is continued until a solution is found or a timeout occurs. If there is no input combination that satisfies the generated test case, the given low-level test pattern will be abandoned and the gate level ATPG will be employed again to generate a new low-level test pattern. This task is continued until the low-level ATPG cannot generate any more patterns.

This can be illustrated with the following example (Figure 13). Let us assume that we want to test the FU which is in the statement *Y=X+IN2*. For this purpose the gate-level ATPG is employed and it returns a test vector *X=0X0X* and *IN2=1X11*. From the environment we know that variable *X* can hold only a very limited range of values. Either *X=1* or *X* has a value which is a multiple of 5 (0, 5, 10, 15, …). Therefore, if we replace the undefined bits so that *X=0001*, the justification process will be successful, but if *X=0100* (decimal value 4), the justification will fail.

We generate tests for every FU one by one and finally the fault coverage for every individual FU under the given environmental

constraints can be reported, which gives the possibility to rank all modules according to their testability.

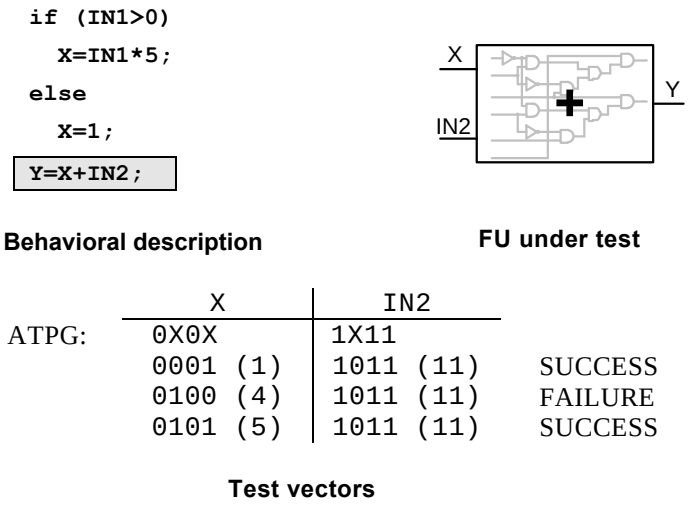


Figure 13. Selection of a test vector

The HTG environment is depicted in Figure 14. Our HTG environment accepts as input a behavioral VHDL specification. The VHDL code is translated into the DD model, which is used as a formal platform for test generation, and later into a Prolog model, which is used by the constraint solver. In our approach we use a commercial constraint solver SICStus [51]. The HTG algorithm generates test cases and forwards them in form of constraints to the constraint solver, which generates the final test vectors. Propagation and justification of the gate level test patterns are performed by the constraint solver as well.

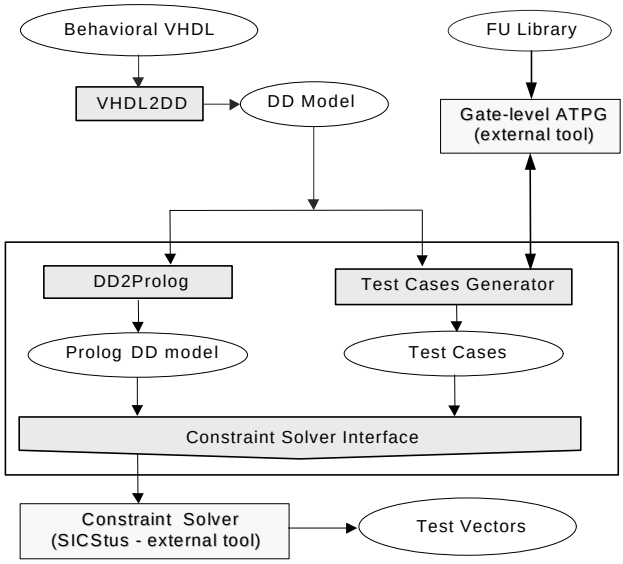


Figure 14. Our hierarchical test generation environment

3.5 Experimental Results

In this section we present our experimental results. We demonstrate that test sequences generated from high-level descriptions provide fault coverage figures comparable with figures obtained at the gate level, while the test generation time is reduced significantly. We will also demonstrate that our approach can successfully be used for testability evaluation.

We performed experiments on the DIFFEQ circuits taken from the High-Level Synthesis'91 benchmark suite. We have synthesized two gate level implementations of the same circuit: one optimized for speed (DIFFEQ 1) and the other optimized for area (DIFFEQ 2). Generated test patterns are applied to the gate level implementations of the circuit and the fault coverage is measured based on the SSA model. The results are reported in Table 1, where for every approach we have presented the obtained stuck-at fault

coverage (FC), number of generated test vectors (Len) and CPU time spent (CPU) for test generation.

We compare our results with pure high-level ATPG [37] and pure gate level ATPG (`testgen` from Synopsys). The pure high-level ATPG works at the behavioral level and generates tests based on different code coverage metrics. The gate-level ATPG, on the other hand, uses only gate-level information and can therefore be used only at the latter stages of the design cycle. The results show that the test sequences provided with our HTG approach can be successfully used for detecting stuck-at faults. These results also show that when moving test vector generation toward lower levels of abstractions, where more detailed information about the tested circuits are available, the obtained results in terms of fault coverage figures are improved. The fault coverage obtained by the hierarchical ATPG is higher than that of the pure high-level ATPG, while the fault coverage working at the gate level is the highest. However, all three different approaches can obtain very high and comparable fault coverage figures. On the other hand, moving test generation towards the higher levels of abstraction has positive effects on the test generation time and on the test length that are both significantly reduced.

We can also note, that our HTG approach can generate test sequences faster and with higher quality than pure high-level ATPG. This can be partially explained with the very simple test generation algorithm employed in the pure high-level ATPG approach reported here.

	Pure High-level ATPG			**Our Hierarchical ATPG**			Gate-level ATPG `testgen`		
	FC [%]	Len [#]	CPU [s]	FC [%]	Len [#]	CPU [s]	FC [%]	Len [#]	CPU [s]
DIFFEQ 1	97.25	553	954	98.05	199	468	99.62	1,177	4,792
DIFFEQ 2	94.57	553	954	96.46	199	468	96.75	923	4,475

Table 1. Results for the DIFFEQ benchmark circuit.

We have also investigated possibilities to apply our ATPG approach to an industrial design F4 [52], which is part of the F4/F5 layer of the ATM protocol, covering the main functionality as specified by standard references. The F4/F5 layer covers the Operation and Maintenance (OAM) functionality of the ATM switches. The F4 level handles the OAM functionality concerning virtual paths and the F5 level handles the OAM functionality concerning virtual channels. We have extracted two blocks from the specification: *F4_InputHandler_1* and *F4_OutputHandler_1*. Experimental results of these two examples are compared with those obtained using the commercial gate level ATPG tool from Mentor Graphics (FlexTest) and are presented in Table 2:

Design	VHDL Lines [#]	Stuck-at faults [#]	**Our Hierarchical ATPG**			Gate level ATPG FlexTest		
			Len [#]	CPU [s]	FC [%]	Len [#]	CPU [s]	FC [%]
F4_Input Handler_1	175	4872	62	228	64.22%	219	811	38.22%
F4_Output Handler_1	54	872	26	1.52	76.26%	170	5	81.30%

Table 2: ATPG results with F4 design

As it can be seen, HTG can produce results which are comparable with results obtained at the gate level, while having shorter test generation time and reduced test length. In case of the *F4_InputHandler_1* block, our HTG approach obtains even higher fault coverage figure than that of the gate-level ATPG. This illustrates very well the situation when a gate-level ATPG cannot produce high quality test vectors due to the higher complexity of descriptions at lower levels of abstraction, and a high-level ATPG tool can outperform a gate-level ATPG tool by producing test patterns with higher fault coverage.

In order to investigate the possibility of using the HTG approach for testability evaluation we have also performed a more thorough analysis using the DIFFEQ design. The results are presented in

Figure 15. We have associated with every FU a set of data. We use the instruction *y_var := y_var + t7;* in order to explain the significance of this data:

```
y_var := y_var + t7;
```

`-- Tested 389 faults`
: Total number of detected stuck-at faults in the FU, when implemented in the target technology.

`-- Untestable 0`
: Total number of untestable faults in the FU, when implemented in the target technology.

`-- Aborted 39`
: Total number of aborted faults (the faults that cannot be detected due to different reasons. For example, the generated gate-level test pattern could not be propagated and/or justified till primary inputs/outputs).

`-- Fault coverage: 90.887850`
: Final stuck-at fault coverage.

`-- 11 Vectors`
: Number of test vectors that were generated by a gate level ATPG and successfully justified till primary inputs and propagated till primary outputs.

As it can be seen, the fault coverage of functional units differs significantly, depending of the location and type of every individual FU. This information can be exploited at the latter design stages in order to improve the global testability of the design.

```
ENTITY diff IS
  PORT
      ( x_in     : IN integer;
        y_in     : IN integer;
        u_in     : IN integer;
        a_in     : IN integer;
        dx_in    : IN integer;
        x_out    : OUT integer;
        y_out    : OUT integer;
        u_out    : OUT integer
     ) ;
END diff ;

ARCHITECTURE behavior OF diff IS
BEGIN
  PROCESS
    variable x_var, y_var, u_var,
             a_var, dx_var  : integer;
    variable t1,t2,t3,t4,t5,
             t6,t7: integer ;
    BEGIN
        x_var := x_in;
        y_var := y_in;
        a_var := a_in;
        dx_var := dx_in;
        u_var := u_in;

    while x_var < a_var loop

    t1 := u_var * dx_var;
    -- Tested 5634 faults
    -- Untestable 0
    -- Aborted 14
    -- Fault coverage: 99.752125
    -- 52 Vectors

    t2 := x_var * 3;
    -- Tested 4911 faults
    -- Untestable 0
    -- Aborted 737
    -- Fault coverage: 86.951133
    -- 11 Vectors

    t3 := y_var * 3;
    -- Tested 4780 faults
    -- Untestable 0
    -- Aborted 868
    -- Fault coverage: 84.631728
    -- 10 Vectors

    t4 := t1 * t2;
    -- Tested 5621 faults
    -- Untestable 0
    -- Aborted 27
    -- Fault coverage: 99.521955
    -- 38 Vectors

    t5 := dx_var * t3;
    -- Tested 5616 faults
    -- Untestable 0
    -- Aborted 32
    -- Fault coverage: 99.433428
    -- 35 Vectors

    t6 := u_var - t4;
    -- Tested 368 faults
    -- Untestable 0
    -- Aborted 60
    -- Fault coverage: 85.981308
    -- 9 Vectors

    u_var := t6 - t5;
    -- Tested 424 faults
    -- Untestable 0
    -- Aborted 4
    -- Fault coverage: 99.065421
    -- 15 Vectors

    t7 := u_var * dx_var;
    -- Tested 1123 faults
    -- Untestable 0
    -- Aborted 4525
    -- Fault coverage: 19.883144
    -- 1 Vectors

    y_var := y_var + t7;
    -- Tested 389 faults
    -- Untestable 0
    -- Aborted 39
    -- Fault coverage: 90.887850
    -- 11 Vectors

    x_var := x_var + dx_var;
    -- Tested 414 faults
    -- Untestable 0
    -- Aborted 14
    -- Fault coverage: 96.728972
    -- 15 Vectors

  end loop ;

  x_out <= x_var;
  y_out <= y_var;
  u_out <= u_var;

  END PROCESS ;

END behavior;
```

Figure 15. DIFFEQ benchmark with testability figures for every individual functional unit

3.6 Conclusions

In this chapter we have proposed a novel high-level hierarchical test pattern generation approach. It uses, as input, a behavioral specification of the system, which is enriched with some information about the final architecture. This information can be derived from a technology library and, as a result, we can evaluate the testability of the final implementation. The test patterns are generated based on a hierarchical test generation technique that employs a dedicated constraint solver.

As our hierarchical test generation approach takes into account information from several abstraction levels it is able to generate test sequences with higher fault coverage than those produced using a pure behavioral test generator. Improvements in fault coverage are obtained by integrating structural information coming from lower levels of abstractions, while still mainly working at the behavioral level for test vector justification and propagation. We have also demonstrated the higher efficiency of our approach compared to the gate level ATPG in terms of required CPU time and the number of test vectors produced.

Chapter 4
A Hybrid BIST Architecture and its Optimization for SoC Testing

As demonstrated in the previous chapter it is feasible to reason about testability already in very early phases of the design cycle. Test sequences generated at the behavioral level can be used as a starting point for constructing manufacturing tests. Test generation results from the behavioral level can also be used for identification of hard to test modules of a design. After identification of such modules, suitable DFT technique has to be chosen and DFT modifications performed in order to guarantee a good testability of the final circuit.

In our approach we have selected a built-in self-test technique, as one of the mainstream DFT techniques for deep submicron designs. In this chapter we propose a hybrid self-test architecture, which is very attractive for modern SoCs. We also propose methods for calculating the total cost of such a self-test scheme and introduce a technique for finding the optimal test solution.

4.1 Introduction

Many systems are nowadays designed by embedding predesigned and preverified complex functional blocks, usually referred as cores, into one single die. The cores can be very different by their nature (from analog to memories, including all types of logic) and can be represented in several different ways (RTL code, netlist or layout) [59]. Such a design style allows designers to reuse previous designs and will lead therefore to a shorter time to market and a reduced cost. Such a System-on-Chip (SoC) approach is very attractive from the designers' perspective. Testing of SoC, on the other hand, shares all the problems related to testing modern deep submicron chips, and introduces also some additional challenges due to the protection of intellectual property as well as the increased complexity and higher density [34].

To test the individual cores on SoC, the test pattern source and sink have to be available together with an appropriate test access mechanism (TAM) [41], [62] as depicted in Figure 16. We can implement such a test architecture in several different ways. One widespread approach is to implement both source and sink off-chip and require therefore the use of external Automatic Test Equipment (ATE). But, as discussed earlier, the internal speed of SoC is constantly increasing and, thus, the demands for the ATE speed and memory size are continuously increasing too. However, the technology used in ATE is always one step behind the one used for advanced SoCs and, the ATE solution will soon become unacceptably expensive and inaccurate [28]. Therefore, in order to apply at-speed tests and to keep the test costs under control, on-chip self-test solutions are becoming more and more popular.

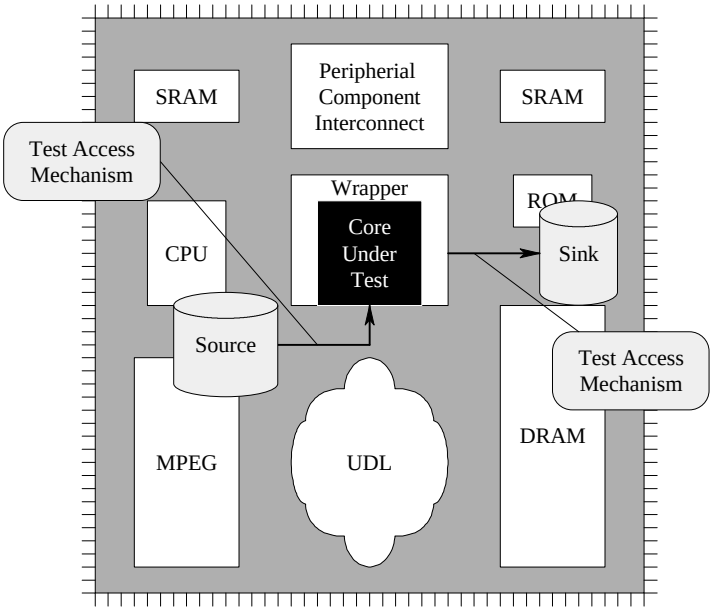


Figure 16. Testing a system-on-chip

A typical BIST architecture consists of a test pattern generator (TPG), a test response analyzer (TRA) and a BIST control unit (BCU), all implemented on the chip. This approach allows applying at-speed tests and eliminates the need for an external tester. Different BIST approaches have been available for a while and have got wide acceptance especially for memory test. For logic BIST (LBIST) there is still no industry-wide acceptance. One of the main reasons is the hardware overhead required to implement a BIST architecture. The BIST approach can also introduce additional delay to the circuitry and requires a relatively long test application time. At the same time, BIST is basically the only practical solution to perform at-speed test and can be used not only for manufacturing test but also for periodic field maintenance tests [17].

4.2 Related Work

The classical way to implement the TPG for LBIST is to use linear feedback shift registers (LFSR) [2], [5], [60]. The effectiveness of such TPG for a given circuit depends on the appropriate choice of the LFSRs as well as their length and configuration. This can yield relatively high fault coverage, but only for a combinational part of the circuitry. The problem with such approaches is that the test patterns generated by the LFSR are pseudorandom by nature [21]. Therefore the LFSR-based approach often does not guarantee sufficiently high fault coverage, especially in the case of large and complex designs, and demands very long test application times in addition to high area overheads. Several proposals have therefore been made to combine pseudorandom test patterns, generated by LFSRs, with deterministic patterns [9], [24], [25], [53], [55], [61] to form a hybrid BIST solution.

In [24] and [61] the authors have proposed methods to increase the quality of LFSR-based BIST schemes via reseeding while in [9] the authors have proposed using additional combinatorial logic to alter the generated pseudorandom sequences. The main concern of those approaches has been to improve the fault coverage, while the issue of cost minimization has not been addressed directly. An additional disadvantage of these approaches is the supplementary hardware needed to implement the hybrid BIST architecture.

To make the LBIST approach more attractive, we have to tackle the hardware overhead problem and to find solutions to reduce the additional delay and the long test application times. At the same time, fault coverage has to be kept at a high level. The simplest and most straightforward solution is to replace the hardware LFSR implementation by software, which is especially attractive to test SoCs, because of the availability of computing resources directly in the system (a typical SoC usually contains at least one processor core). The software-based approach, on the other hand, is criticized because of the large memory requirements, as we have to store the test program and some test patterns, which are required for

initialization and reconfiguration of the self-test cycle [25]. However, some preliminary results regarding such an approach for PCBs has been reported in [4] and shows that a software-based approach is feasible.

Similar work has been reported also in [25]. However, the approach presented there has no direct cost considerations and can therefore lead to very long test application times because of the large number of random test patterns used.

In our approach we propose to use a hybrid test set which contains a limited number of pseudorandom and deterministic vectors. The pseudorandom test vectors can be generated either by hardware or by software and is complemented by the stored deterministic test set which is specially designed to shorten the pseudorandom test cycle and to target the random resistant faults [33].

The main objective of our approach is to support the combination of pseudorandom and deterministic test vectors and to find the optimal balance between these two test sets to perform core test with minimum cost of time and memory, without losing test quality. In the following, two different algorithms to calculate the total cost of the hybrid BIST solution will be proposed. Additionally a method is proposed to estimate, with very low computational time, the time-moment that is close to the optimal moment to stop pseudorandom test generation and to apply deterministic patterns. This method is used to find a good starting point to search for the global optimum by sampled calculation of the real cost. The search is carried out by using a Tabu search method [18], [19], [20].

A similar problem has been addressed in [53], where an approach to minimize testing time has been presented. The authors have shown that hybrid BIST (or CBET in their terminology) can achieve shorter testing time than both pseudorandom and deterministic test. However, the proposed algorithm is not addressing total cost minimization (time and memory) and is therefore only a special case of our approach.

4.3 Hybrid BIST Architecture

A hardware-based hybrid BIST architecture is depicted in Figure 17, where the pseudorandom pattern generator (PRPG) and the Multiple Input Signature Analyzer (MISR) are implemented inside the core under test (CUT). The PRPG and MISR can be implemented by using LFSRs or any other structure able to provide pseudorandom test vectors with a required degree of randomness. The deterministic test patterns are precomputed off-line and stored inside the system.

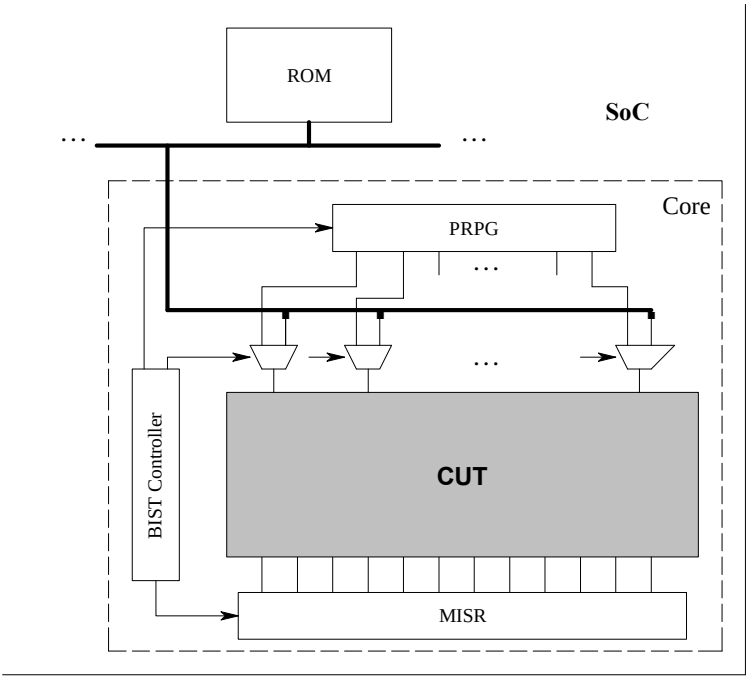


Figure 17. Hardware-based hybrid BIST architecture

Core test is performed in two consecutive stages. During the first stage pseudorandom test patterns are generated and applied. After a predetermined number of test cycles, additional test is performed with deterministic test patterns from the memory. Each primary

input of CUT has a MUX at the input that determines whether the test is coming from the PRPG or from the memory (Figure 17).

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

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

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

The pseudorandom TPG software is the same for all cores in the system and is stored as one single copy. All characteristics of the LFSR needed for emulation are specific to each core and are stored in the ROM. They will be loaded upon request. Such an approach is very effective in the case of multiple cores, because for each additional core only the BIST characteristics for this core have to be stored. The general concept of the software based pseudorandom TPG is depicted in Figure 18.

As the LFSR is implemented in software, there are no hardware constraints for the actual implementation except for the ROM. This allows to develop for each particular core the most efficient pseudorandom scheme without concerning about the hardware cost. As has been shown by experiments, the selection of the best possible pseudorandom scheme is an important factor for such an approach [25].

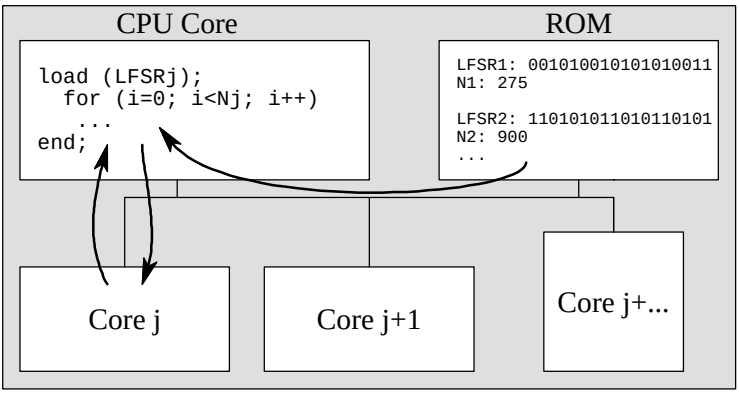


Figure 18. LFSR emulation

The quality of the pseudorandom test is of great importance. It is assumed that for the hybrid BIST the best pseudorandom sequence will be chosen. However, not always all parts of the system are testable by a pure pseudorandom sequence. It takes often a very long test application time to reach a good fault coverage level. In case of hybrid BIST, we can dramatically reduce the length of the initial pseudorandom sequence by complementing it with deterministic stored test patterns, and achieve the 100% fault coverage.

As discussed in [25], the program to emulate the LFSR can be very simple and therefore the memory requirements for storing the pseudorandom TPG program together with the LFSR parameters are relatively small.

In the ideal case, the LFSR-based test generator can be developed in such a way that a high fault coverage will be reached by a sufficiently short test sequence. In our approach, the task is not to develop a new "ideal" LFSR-based test generator for BIST with the length equal to the minimal test set with 100% fault coverage. In general, for complex circuits such a task is rather difficult and may be even impossible to solve, especially for sequential circuits. In this sense, the hybrid BIST considered here suggests a more general and simple solution, applicable also for sequential cores.

4.4 Test Cost Calculation for Hybrid BIST

As mentioned earlier, the test patterns generated by LFSRs are pseudorandom by nature. Such test sequences are usually very long and not sufficient to detect all the faults. Figure 19 shows the fault coverage of the pseudorandom test as a function of the test length for some larger ISCAS'85 [8] benchmark circuits. This figure illustrates an inherent property of pseudorandom test: the first few test vectors can detect a large number of faults while later test vectors can detect very few new faults, if any. There may also exist faults that will never be detected with pseudorandom test vectors, which is due to random pattern resistance.

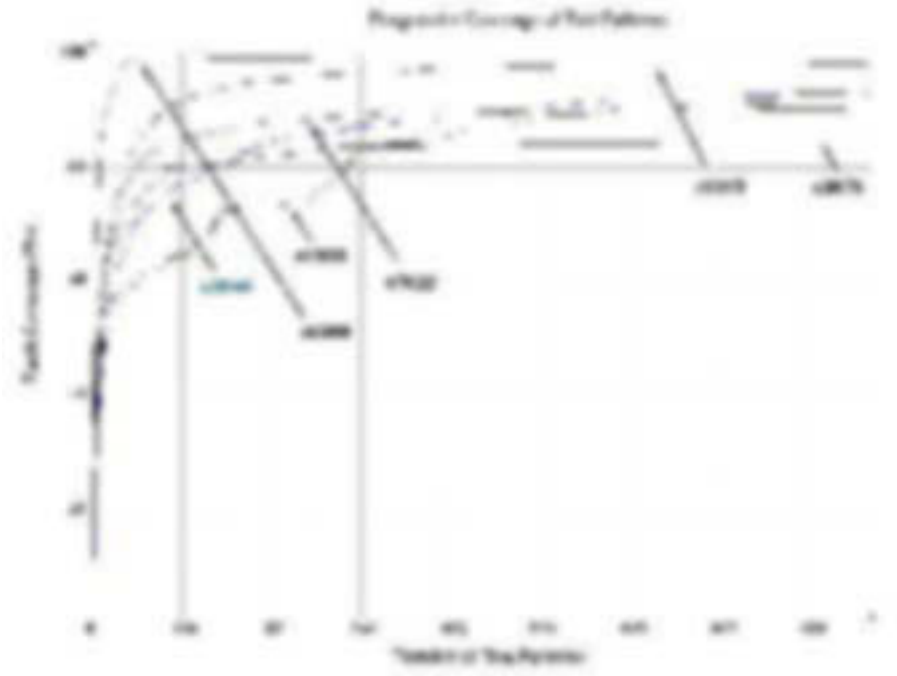


Figure 19. Pseudorandom test for some ISCAS'85 circuits

To avoid the test quality loss due to random pattern resistant faults and to speed up the testing process, we have to apply deterministic test patterns targeting the random resistant and

difficult to test faults. Such a hybrid BIST approach starts with a pseudorandom test sequence of length L. At the next stage, the stored test approach takes place: precomputed test patterns, stored in the system, are applied to the core under test to reach the desirable fault coverage. For off-line generation of the deterministic test patterns, arbitrary software test generators may be used, based on deterministic, random or genetic algorithms.

In a hybrid BIST technique the length of the pseudorandom test is an important parameter that determines the behavior of the whole test process [25]. It is assumed here that for the hybrid BIST the best polynomial for the pseudorandom sequence generation will be chosen. Removing the latter part of the pseudorandom sequence leads to lower fault coverage achievable by the pseudorandom test. The loss in fault coverage should be compensated by additional deterministic test patterns. In other words, a shorter pseudorandom test set implies a larger deterministic test set. This requires additional memory space, but at the same time, shortens the overall test process. A longer pseudorandom test, on the other hand, will lead to longer test application time with reduced memory requirements. Therefore it is crucial to determine the optimal length L_{OPT} of the pseudorandom test sequence, in order to minimize the total testing cost.

Figure 20 illustrates graphically the total cost of a hybrid BIST consisting of pseudorandom test patterns and stored test patterns, generated off-line. The horizontal axis in Figure 20 denotes the fault coverage achieved by the pseudorandom test sequence before switching from the pseudorandom test to the stored test. Zero fault coverage is the case when only stored test patterns are used and therefore the cost of stored test is the largest in this point. The figure illustrates the situation where 100% fault coverage is achievable with pseudorandom vectors alone although it can demand a very long pseudorandom test sequence. In the case of large and complex designs 100% fault coverage is not always achievable.

We can define the total test cost of a hybrid BIST, C_{TOTAL}, as:

$$C_{TOTAL} = C_{GEN} + C_{MEM} \quad (1)$$

where C_{GEN} is the cost related to the effort for generating the pseudorandom test patterns (proportional to the number of clock cycles) and C_{MEM} is related to the memory cost for storing the precomputed test patterns to improve the pseudorandom test set. Please note that the proposed cost calculation formula is not complete. For example, in case of hardware-based hybrid BIST architecture the cost of the LFSR's implementation should be taken into account, but as those costs are constant, regardless of the size and ratio of selected test sets, we are not considering them here.

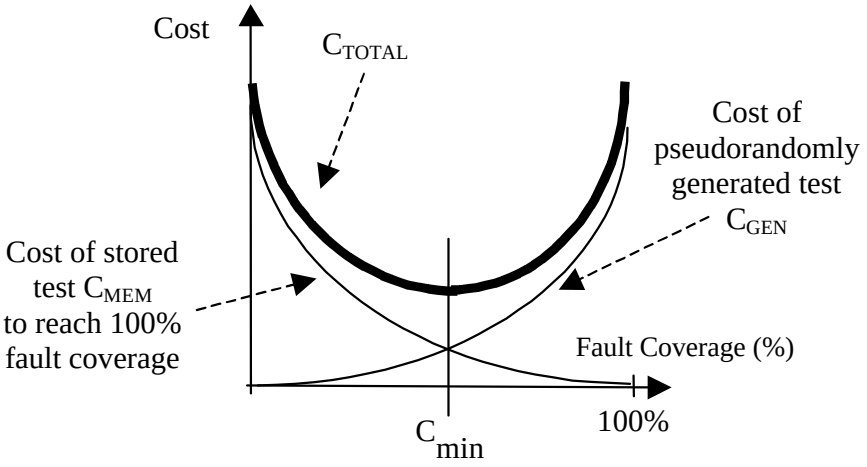


Figure 20. Cost calculation for hybrid BIST
(under 100% assumption)

Figure 20 illustrates also how the cost of the pseudorandom test is increasing when striving for higher fault coverage (the C_{GEN} curve). In general, it can be very expensive to achieve high fault coverage with pseudorandom test patterns only. The C_{MEM} curve describes the cost that we have to pay for storing additional precomputed tests

from the fault coverage level reached by pseudorandom testing to 100%. The total cost C_{TOTAL} is the sum of the above two costs. The C_{TOTAL} curve is illustrated in Figure 20, where the minimum point is marked as C_{min}. One of the main objectives of our approach is to find a fast method for calculating the curve C_{TOTAL} and to find the minimal cost point C_{min}.

As mentioned before, in many cases 100% fault coverage is not achievable with pseudorandom vectors only. Therefore we have to include this assumption to the total cost calculation and the new situation is illustrated in Figure 21, where the horizontal axis indicates the number of pseudorandom test patterns applied, instead of fault coverage level. The curve for the total cost C_{TOTAL} is still the sum of two cost curves $C_{GEN} + C_{MEM}$ with the new assumption that the maximum fault coverage reachable by only deterministic ATPG is achieved by the hybrid BIST.

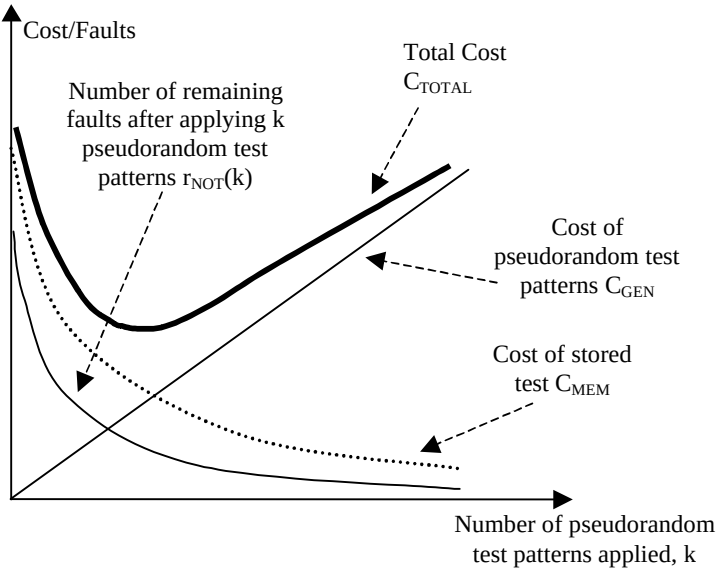


Figure 21. Cost calculation for hybrid BIST

We can also simplify the total cost of a hybrid BIST, C_{TOTAL}, as:

$$C_{TOTAL} = aL + bS \quad (2)$$

where L is the length of the pseudorandom test sequence, S is the number of deterministic patterns, and weights a and b reflect the correlation between the cost and the pseudorandom test time (number of clock cycles used) and between the cost and the memory size needed for storing the deterministic test sequence, respectively.

The idea of defining the test cost as a sum of two costs, the cost of time for the pseudorandom test generation, and the cost of memory associated with storing the TPG produced test, is a rather simplified cost model for the hybrid BIST. For example, neither the basic cost of memory (or its equivalent) occupied by the LFSR-based generator, nor the time needed for generating deterministic test patterns are taken into account. However, the goal of this thesis is not to develop a complete cost function for the whole BIST solution. The goal is to find the tradeoff between the length of the pseudorandom test sequence and the number of deterministic patterns. For making such a tradeoff the basic implementation costs are not important (they are basically the same for every combination of pseudorandom and deterministic test patterns) and will not influence the optimal solution of the hybrid BIST.

On the other hand, the attempt to add "time" to "space" (even in terms of their cost) seems rather controversial as it is very hard to specify which one costs more in general (as well as even in a particular case) and how to estimate these costs. This was also the reason why modeling the total cost of a BIST implementation was not considered as the research objective of our work. The values of parameters a and b in the cost function are determined by the system designer according to the particular requirements and can be used to drive the final implementation towards different alternatives (for example slower, but more memory efficient solutions). If needed, it is possible to separate these two different costs (time and space), consider certain restrictions on each of them, etc. For guiding the

search towards an acceptable solution, nevertheless, a simplified guiding cost function is still needed.

Creating the curve $C_{GEN} = aL$ is not difficult. For this purpose, the cumulative fault coverage (like in Figure 19) for the pseudorandom sequence generated by an LSFR should be calculated using fault simulation. As the result we find for each clock cycle the list of faults which were covered at this time moment.

- As an example, in Table 3, a fragment of the results of BIST simulation for the ISCAS'85 circuit c880 [8] is demonstrated.

k	$r_{DET}(k)$	$r_{NOT}(k)$	$FC(k)$
1	155	839	15.593%
2	76	763	23.239%
3	65	698	29.778%
4	90	608	38.832%
5	44	564	43.259%
6	39	525	47.183%
11	104	421	57.645%
16	66	355	64.285%
21	44	311	68.712%
28	42	269	72.937%
51	51	218	78.068%
71	57	161	83.802%
101	16	145	85.412%
149	13	132	86.720%
201	18	114	88.531%
323	13	101	89.839%
412	31	70	92.957%
708	24	46	95.372%
955	18	28	97.183%
1536	4	24	97.585%
1561	8	16	98.390%
2154	11	5	99.496%
3450	2	3	99.698%
4520	2	1	99.899%
4521	1	0	100.000%

Table 3. Pseudorandom test results

Here

- k is the number of the clock cycle,
- $r_{DET}(k)$ is the number of new faults detected (covered) by the test pattern generated at the clock signal k,
- $r_{NOT}(k)$ is the number of faults not yet covered by the sequence of patterns generated by the k clock signals,

$FC(k)$ is the fault coverage reached by the sequence of patterns generated by the k clock signals

In the list of BIST simulation results not all clock cycles are presented. We are only interested in the clock numbers at which at least one new fault will be covered, and the total fault coverage for the pseudorandom test sequence up to this clock number increases. Let us call such clock numbers and the corresponding pseudorandom test patterns *efficient clocks* and *efficient patterns*. The rows in Table 3 correspond to selected efficient clocks for the circuit c880. If we decide to switch from pseudorandom mode to the deterministic mode after the clock number k, then $L = k$.

It is more difficult to find the values for $C_{MEM} = bS$, the cost for storing additional deterministic patterns in order to reach the given fault coverage level (100% in the ideal case). Let $t(k)$ be the number of test patterns needed to cover $r_{NOT}(k)$ not yet detected faults (these patterns should be precomputed and used as stored test patterns in the hybrid BIST). As an example, these data for the circuit c880 are depicted in Table 4. Calculation of the data in the column $t(k)$ of Table 4 is the most expensive procedure. In the following section the difficulties and possible ways to solve the problem are discussed.

k	*t(k)*
1	104
2	104
3	100
4	101
5	99
6	99
11	95
16	92
21	87
29	81
51	74
71	58
101	52
149	46
201	41
323	35
412	26
708	17
955	12
1536	11
1561	7
2154	3
3450	2
4520	1
4521	0

Table 4. ATPG results

4.5 Calculation of the Cost for Stored Test

There are two approaches to find *t(k)*: ATPG based and fault table based. Let us have the following notations:

- *i* – the current number of the entry in the table of pseudorandom test results (Table 3);
- *k(i)* – the number of the efficient clock cycle;
- $R_{DET}(i)$ - the set of new faults detected (covered) by the pseudorandom test pattern generated at *k(i)*;

- $R_{NOT}(i)$ - the set of not yet covered faults after applying the pseudorandom pattern number $k(i)$;
- $T(i)$ - the set of test patterns needed and found by the ATPG to cover the faults in $R_{NOT}(i)$;
- N – the number of all efficient patterns in the sequence created by the pseudorandom test;
- FT – the fault table for a given set of test patterns T and for the given set of faults R: the table defines the subsets $R(t_j) \hat{I} R$ of detected faults for each pattern $t_j \hat{I} T$.

Algorithm 1: ATPG based approach for finding test sets *T(i)*

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

End.

This algorithm generates a new deterministic test set for the not yet detected faults at every efficient clock cycle. In this way we have the complete test set (consisting of pseudorandom and deterministic test vectors) for every efficient clock, which can reach to the maximal achievable fault coverage. The number of deterministic test vectors at all efficient clocks are then used to create the curve $C_{MEM}(bS)$. The algorithm is straightforward, however, very time consuming because of repetitive use of ATPG. The whole experiment of simulating the pseudorandom generation (PRG) behavior and of finding the numbers of test patterns to be stored for all possible switching points from PRG to stored test patterns for the whole set of ISCAS'85 benchmark circuits took approximately 8 hours.

Since usage of ATPG is a very costly procedure, we present in the following another algorithm based on iterative transformations of

fault tables. This algorithm allows a dramatic reduction of computation time for the hybrid BIST cost calculation.

The fault table FT for a general case is defined as follows: given a set of test patterns $T = \{t_i\}$ and a set of faults $R = \{r_j\}$, $FT = || e_{ij} ||$ where $e_{ij} = 1$ if the test $t_i \in T$ detects the $r_j \in R$, and $e_{ij} = 0$ otherwise. We denote by $R(t_i) \subset R$ the subset of faults detected by the test pattern $t_i \in T$.

We start the procedure for a given circuit by generating a test set T which gives the 100% (or as high as possible) fault coverage. This test set can be served as a stored test if no on-line generated pseudorandom test sequence will be used. By fault simulation of the test set T for the given set of faults R of the circuit, we create the fault table FT. Suppose now, that we use a pseudorandom test sequence T^L with a length L which detects a subset of faults $R^L \subset R$. It is obvious that when switching from the pseudorandom test mode with a test set T^L to the precomputed stored test mode with a T, the test set T can be significantly reduced. At first, by the fault subtraction operation $R(t_i) - R^L$ we can update all the contributions of the test patterns t_i in FT (i.e. to calculate for all t_i the remaining faults they can detect after performing the pseudorandom test). After that we can use any procedure of static test compaction to minimize the test set T.

The described procedure of updating the fault table FT can be carried out iteratively for all possible breakpoints $i = 1, 2, \ldots, N$ of the pseudorandom test sequence by the following algorithm [34].

Algorithm 2: Fault Table based approach for finding test sets *T(i)*

1. Calculate the whole test $T = \{t_j\}$ for the whole set of faults $R = \{r_j\}$ by any ATPG to reach as high fault coverage C as possible
2. Create for T and R the fault table $FT = \{R(t_j)\}$
3. Take $i = 1$; Rename: $T_i = T$, $R_i = R$, $FT_i = FT$
4. Take $i = i + 1$
5. Calculate by fault simulation the fault set $R_{DET,i}$

6. Update the fault table: $\forall j$, $t_j \in T_i$: $R(t_j)$ - $R_{DET,i}$
7. Remove from the test set T_i all the test patterns $t_j \in T_i$ where $R(t_j) = \varnothing$
8. Optimize the test set T_i by any test compaction algorithm; fix the value of $S_i = |T_i|$ as the length of the stored test for $L = i$;
9. If $i < L$, go to 4;

End.

It is easy to understand that for each value $L = i$ (the length of the pseudorandom test sequence) the procedure guarantees the constant fault coverage C of the hybrid BIST. The statement comes from the fact that the subset T_i of stored test patterns is complementing the pseudorandom test sequence for each $i = 1, 2, \ldots, N$ to reach the same fault coverage reached by T.

As the result of algorithm 2, the numbers of precomputed deterministic test patterns $S_i = |T_i|$ to be stored and the subsets of these patterns T_i for each i =1, 2, …, N are calculated. On the basis of this data the cost of stored test patterns for each i can be calculated by the formula $C_{MEM} = bS_i$. From the curve of the total cost $C_{TOTAL}(i) = aL + bS$ the value of the minimum cost of the hybrid BIST $min\{C_{TOTAL}(i)\}$ can be easily found.

In case of very large circuits both algorithms presented will lead to very expensive and time-consuming runs. It would be desirable to find the global optimum of the total cost curve by as few sampled calculations of the total cost for selected values of k as possible. Therefore we introduce here an approach, based on a Tabu search heuristic, to speed up the calculations.

4.6 Tabu Search Based Cost Optimization

Tabu search was introduced by Fred Glover ([18], [19] and [20]) as a general iterative heuristic for solving combinatorial optimization problems. Some initial ideas related to this technique were also proposed by Hansen [23] in his steepest ascent descent heuristic.

Tabu search is a form of local neighborhood search. Each solution $SO \hat{I} W$, where W is the search space (the set of all feasible solutions), has an associated set of neighbors $N(SO) \acute{I} W$. A solution $SO' \hat{I} N(SO)$ can be reached from SO by an operation called a move. At each step, the local neighborhood of the current solution is explored and the best solution is selected as a new current solution. Unlike local search which stops when no improved new solution is found in the current neighborhood, Tabu search continues the search from the best solution in the neighborhood even if this solution is worse than the current one. To prevent cycling, visited solutions are kept in a list called Tabu list. Tabu moves (moves stored in the current Tabu list) are not allowed. However, the Tabu status of a move is overridden when a certain criterion (aspiration criterion) is satisfied. One example of an aspiration criterion is when the cost of the selected solution is better than the best seen so far, which is an indication that the search is actually not cycling back, but rather moving to a new solution not encountered before [19]. Moves are only kept in the Tabu list for a given number of iterations (the so called "Tabu tenure").

The procedure of the Tabu search starts from an initial feasible solution in the search space W, which becomes the first current solution SO. A solution is defined as the switching moment from the pseudorandom test mode to the stored test mode. The search space W covers all possible switching moments. A neighborhood $N(SO)$ is defined for each SO. Based on the experimental results it was concluded that the most efficient step size for defining the neighborhood $N(SO)$ in our optimization problem was 3% of the number of efficient clocks. A larger step size, even if it can provide considerable speedup, will decrease the accuracy of the final result. In our algorithm a sample of neighbor solutions $V^* \grave{I} N(SO)$ is generated. Those solutions can be generated by using either the Algorithm 1 or 2. In our current approach Algorithm 2 was used. An extreme case is to generate the entire neighborhood, that is to take $V^* = N(SO)$. Since this is generally impractical (computationally expensive), a small sample of neighbors is generated, and called trial

solutions *(½V*½= n << ½N(SO)½)*. In case of ISCAS'85 benchmark circuits the best results were obtained, when the size of the sample of neighborhood solutions was 4. An increase of the size of *V** had no effect on the quality of results. From these trial solutions the best solution, say *SO* Î V**, is chosen for consideration as the next solution. The move to *SO** is considered, even if *SO** is worse than *SO*, that is, *Cost(SO*) > Cost(SO)*. It is this feature that enables escaping from local optima. The cost of a solution is calculated according to Equation 2 for calculating the total cost of hybrid BIST CTOTAL, presented in section 4.4. A move from *SO* to *SO** is made provided certain conditions are satisfied.

One of the parameters of the algorithm is the size of the Tabu list. A Tabu list *T* is maintained to prevent returning to previously visited solutions. The list contains information concerning forbidden moves. Generally, the Tabu list size is small. The size should also be determined by experimental runs, watching the occurrence of cycling when the size is too small, and the deterioration of solution quality when the size is too large [49]. Results have shown that the best average size for the ISCAS'85 benchmark family was 3. Larger sizes lead to a loss of result quality.

For finding a good initial feasible solution in order to make Tabu search more productive, a fast estimation method for an optimal L proposed in [33] is used. For this estimation, the number of not yet covered faults in $R_{NOT}(i)$ can be used. The value of $|R_{NOT}(i)|$ can be acquired directly from the PRG simulation results and is available for every significant time moment (see Table 3). Based on the value of $|R_{NOT}(i)|$ it is possible to estimate the expected number of test patterns needed for covering the faults in $R_{NOT}(i)$. The starting point for the Tabu search procedure can be found by considering a rough estimation of the total cost based on the value of $|R_{NOT}(i)|$. Based on statistical analysis of the costs calculated for ISCAS'85 benchmark circuits, in [33] the following approximation is proposed: 1 remaining fault = 0,45 test patterns needed to cover it. In this way, a simplified cost prediction function was derived: $C'_{TOTAL}(k) = C_{GEN}(k) +$

0,45b^{k}*R*$_{NOT}$*(k)*. The value k*, where *C'*$_{TOTAL}$*(k*) = min(C'*$_{TOTAL}$*(k))* was used for the initial solution for Tabu search.

To explain the algorithm, let's have the following additional notations: *E* - number of allowed empty iterations (i.e. iterations that do not result in finding a new best solution), defined for each circuit, and *SO*^{trial} - solution generated from the current solution as a result of the move.

Algorithm 3: Tabu Search

Begin
Start with initial solution SO ∈ Ω;
BestSolution:=SO;
Add the initial solution SO to Tabu list T, T={SO};
While number of empty iterations < E **Or**
there is no return to previously visited solution **Do**
Generate the sample of neighbor solutions V*⊂ N(SO);
Find best *Cost(SO* Ì V*)*;

M: **If** (solution SO* is not in T) or
(aspiration criterion is satisfied) **Then**
SO^{trial}:=SO*;
Update tabu list T;
Increment the iteration number;
Else
Find the next best *Cost(SO* Ì V*)*;
Go to M;
EndIf

If *Cost(SO*^{trial}*)* < *Cost(BestSolution)* **Then**
BestSolution := SO^{trial};
Else
Increment number of empty iterations
EndIf
SO:=SO^{trial};
EndWhile
End.

4.7 Experimental Results

Experiments were carried out on the ISCAS'85 benchmark circuits for comparing the algorithms 1 and 2, and for investigating the efficiency of the Tabu search method for optimizing the hybrid BIST technique. Experiments were carried out using the Turbo Tester toolset [31], [56] for deterministic test pattern generation, fault simulation, and test set compaction. The results are presented in Table 5 and illustrated by several diagrams.

For calculating the total cost of hybrid BIST we used the formula $C_{TOTAL} = aL + bS$. For simplicity, we assume here that $a = 1$, and $b = B$ where B is the number of bytes of an input test vector to be applied to the CUT. Hence, to carry out some experimental work for demonstrating the feasibility and efficiency of the following algorithms, we use, as the cost units the number of clocks used for pseudorandom test generation and the number of bytes in the memory needed for storing the precomputed deterministic test patterns.

In the columns of Table 5 the following data is depicted: ISCAS'85 benchmark circuit name, L - length of the pseudorandom test sequence, FC - fault coverage, S - number of test patterns generated by deterministic ATPG to be stored, C_T – total cost of the hybrid BIST, T_G - the time (sec) needed for ATPG to generate the deterministic test set, T_A - the time (sec) needed for carrying out manipulations on fault tables (subtracting faults, and compacting the test set), N - number of efficient patterns in the pseudorandom test sequence, T_1 and T_2 - the time (sec) needed for calculating the cost curve by Algorithms 1 and 2, T_3 – the time (sec) to find the optimal cost by using Tabu search. T_S – the number of iterations in Tabu search, Acc – accuracy of the Tabu search solution in percentage compared to the exact solution found from the full cost curve, The total time for performing the calculations for Algorithms 1 and 2 and for Tabu Search was calculated as follows:

$$T_1 = N * T_G$$

$$T_2 = T_G + N * T_A$$

$$T_3 = T_2 * (T_S / N) + D$$

where Δ is the time needed to perform the Tabu search calculations (was below 0.1 sec in the given experiments)

	Pseudorandom test		*Stored test*		*Hybrid test*		
Circuit	***L***	***FC***	***S***	***FC***	***L***	***S***	C_T
C432	780	93.0	80	93.0	91	21	196
C499	2036	99.3	132	99.3	78	60	438
C880	5589	100.0	77	100.0	121	48	505
C1355	1522	99.5	126	99.5	121	52	433
C1908	5803	99.5	143	99.5	105	123	720
C2670	6581	84.9	155	99.5	444	77	2754
C3540	8734	95.5	211	95.5	297	110	1067
C5315	2318	98.9	171	98.9	711	12	987
C6288	210	99.3	45	99.3	20	20	100
C7552	18704	93.7	267	97.1	583	61	2169

	Calculation cost							
Circuit	T_G	T_A	***N***	T_1	T_2	T_3	***Ts***	***Acc***
C432	20.1	0.01	81	1632	21	2.85	11	100.0
C499	0.7	0.02	114	74	3	0.50	19	100.0
C880	0.2	0.02	114	17	2	0.26	15	99.7
C1355	1.2	0.03	109	133	5	0.83	18	99.5
C1908	11.7	0.07	183	2132	25	3.83	28	100.0
C2670	1.9	0.09	118	230	13	0.99	9	99.1
C3540	85.3	0.14	265	22601	122	7.37	16	100.0
C5315	10.3	0.11	252	2593	38	1.81	12	97.2
C6288	3.8	0.04	53	200	6	1.70	15	100.0
C7552	53.8	0.27	279	15004	129	3.70	8	99.7

Table 5. Experimental results

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

The results given in Table 5 demonstrate the high efficiency (in number of required test vectors) of the hybrid BIST solution over pure pseudorandom or deterministic approaches. As expected, the optimal cost was found fastest with using the Tabu search algorithm, while the accuracy was not less than 97,2% for the whole family of ISCAS'85 benchmark circuits. In the following the experimental results will be explained further.

For the Tabu search method the investigation was carried out to find the best initial solution, the step defining *N(SO)*, the size of V* and the size of the Tabu list for using the Tabu strategy in a most efficent way.

For finding the Tabu list size, experiments were carried out with different sizes of the list. Results showed that the best average size for the ISCAS'85 benchmark family was 3. Smaller list sizes would cause cycling around local minimum, a larger size would result in deterioration of the solution quality. The size of the sample of neighborhood solutions V* giving the best results for all circuits, was 4. Smaller sizes would make the process of finding the minimum very long, resulting in very small speedup. A larger size of V* did not improve the results.

The efficiency of the search depends significally on the step size defining the neighborhood *N(SO)*. Based on the experimental results, the charts of dependancy of the overall estimation accuracy and of the overall speedup on step size were compozed and given in Figure 22 and Figure 23. Analyzing results depicted in those figures led to the conclusion that the most favorable step size can be considered as 3% of the number of efficient clocks, where the average estimation accuracy is the highest. Although a larger step size would result in a speedup, it was considered impractical because of the rapid decrease in the cost estimation accuracy.

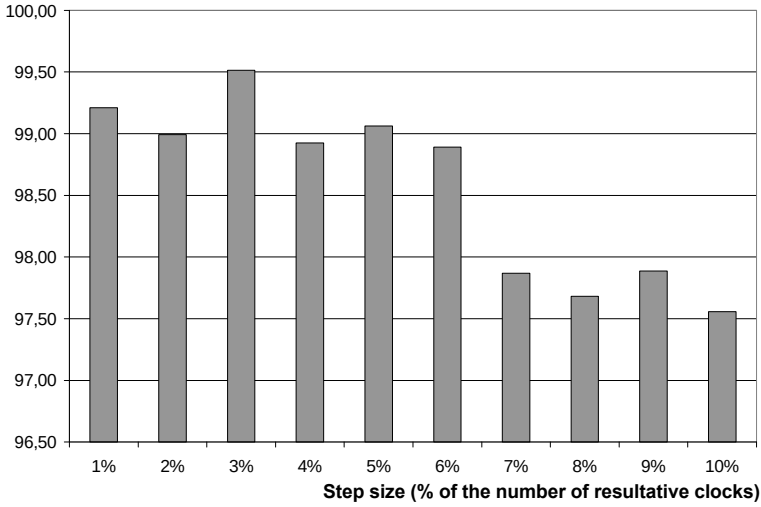


Figure 22. Dependency of estimation accuracy from neighborhood step size

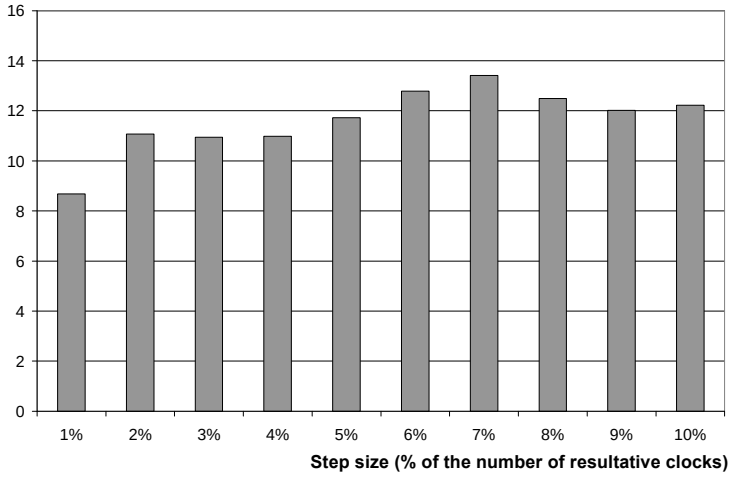
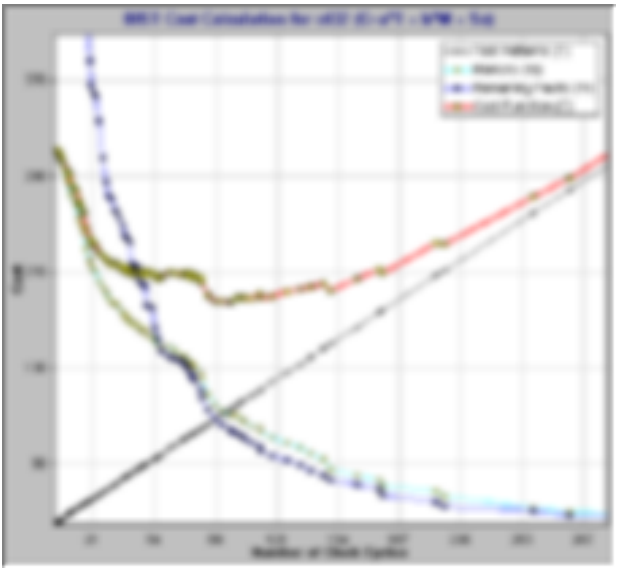


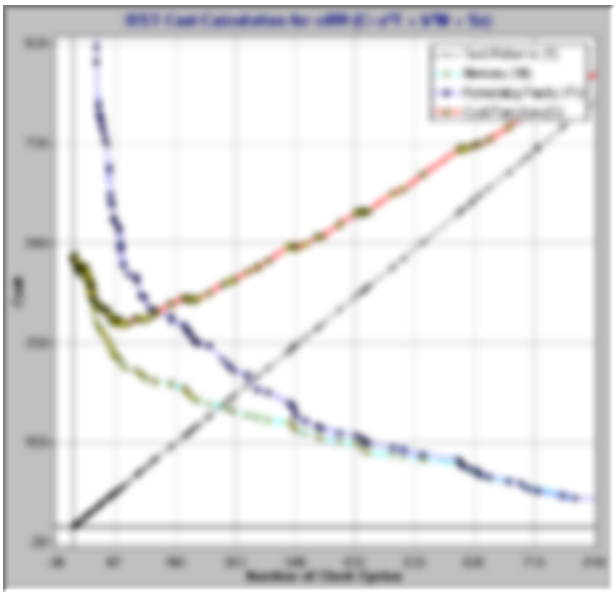
Figure 23. Dependency of average speedup from neighborhood size

In Figure 24, the curves of the cost $C_{GEN} = L$ (denoted on Figure 24 as T) for on-line pseudorandom test generation, the cost $C_{MEM} = B_k * S$ (denoted as M) for storing the test patterns, the number $|R_{NOT}|$ of not detected faults after applying the pseudorandom test sequence (denoted as F_r), and the total cost function C_{TOTAL} are depicted for selected benchmark circuits C432, C499, C880, C1908, C3540 and C7552 (Sc = 0 is used as a constant in the cost function formula).

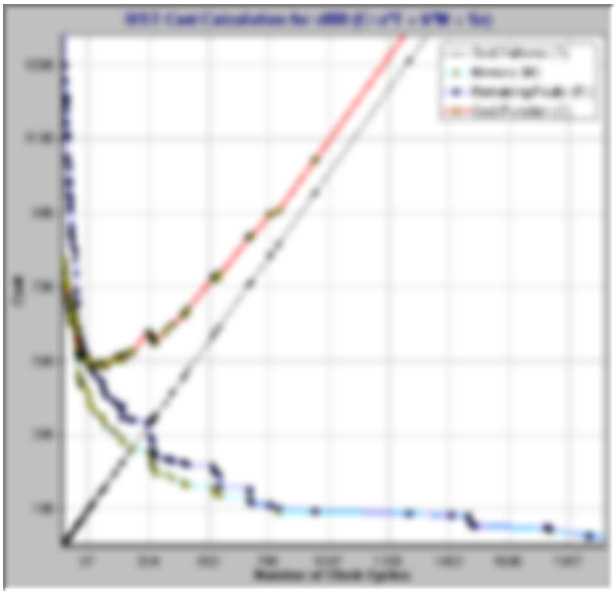


C432

Figure 24. Cost curves of hybrid BIST for some ISCAS'85 benchmark circuits

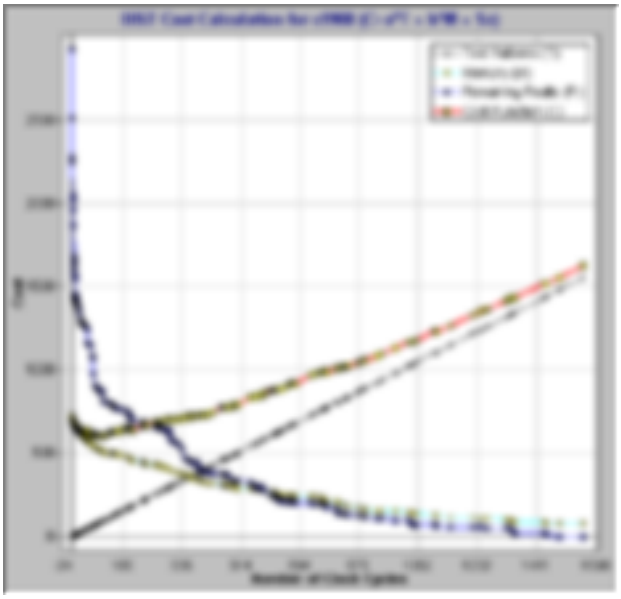


C499

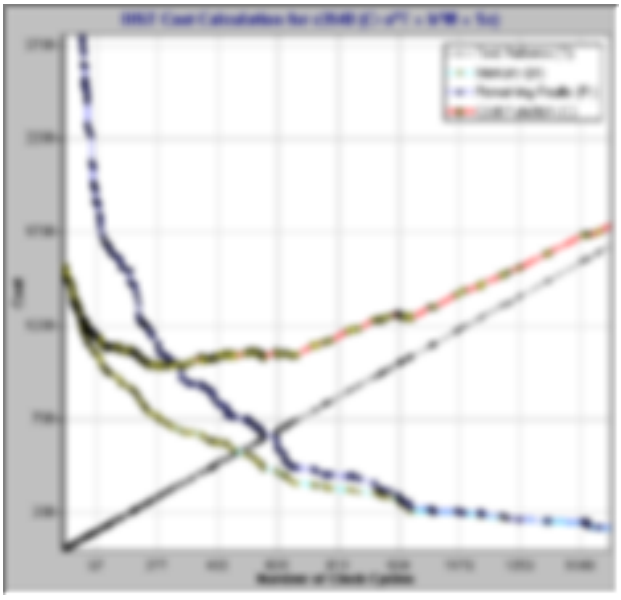


C880

Figure 24. Cost curves of hybrid BIST for some ISCAS'85 benchmark circuits (cont.)

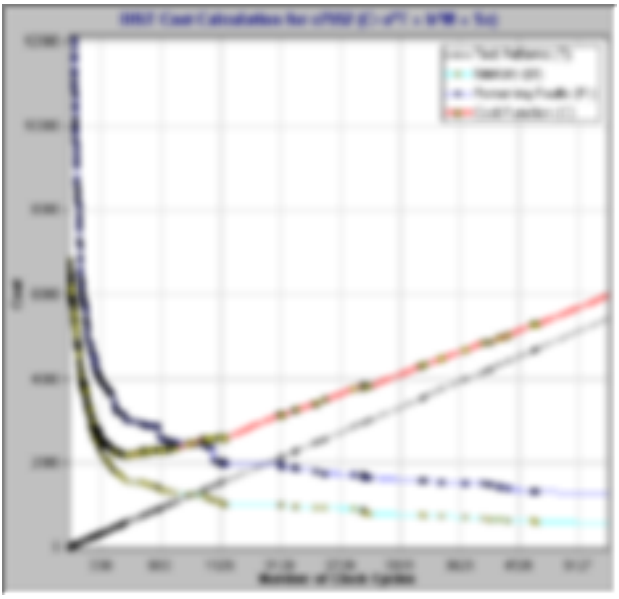


C1908



C3540

Figure 24. Cost curves of hybrid BIST for some ISCAS'85 benchmark circuits (cont.)



C7552

Figure 24. Cost curves of hybrid BIST for some ISCAS'85 benchmark circuits (cont.)

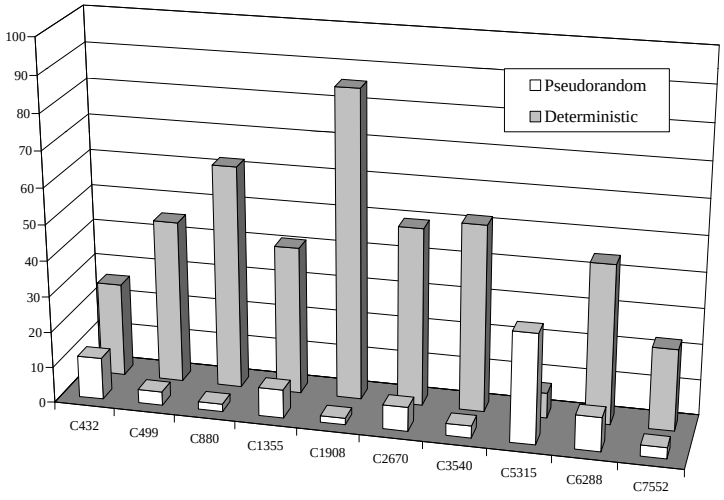


Figure 25. Percentage of test patterns in the optimized test sets compared to the original test sets

In Figure 25 the amount of pseudorandom and deterministic test patterns in the optimal BIST solution is compared to the sizes of pseudorandom and deterministic test sets when only either of these sets is used. In the typical cases less than half of the deterministic vectors and only a small fraction of pseudorandom vectors are needed, however the maximum achievable fault coverage is guaranteed and achieved.

Figure 26 compares the costs of different approaches using for Hybrid BIST cost calculation equation 2 with the parameters $\alpha = 1$, and $\beta = B$ where B is the number of bytes of the input test vector to be applied on the CUT. As pseudorandom test is usually the most expensive method, it has been selected as a reference and given value 100%. The other methods give considerable reduction in terms of cost and as it can be seen, the hybrid BIST approach has significant advantage compared to the pure pseudorandom or stored test approach in most of the cases.

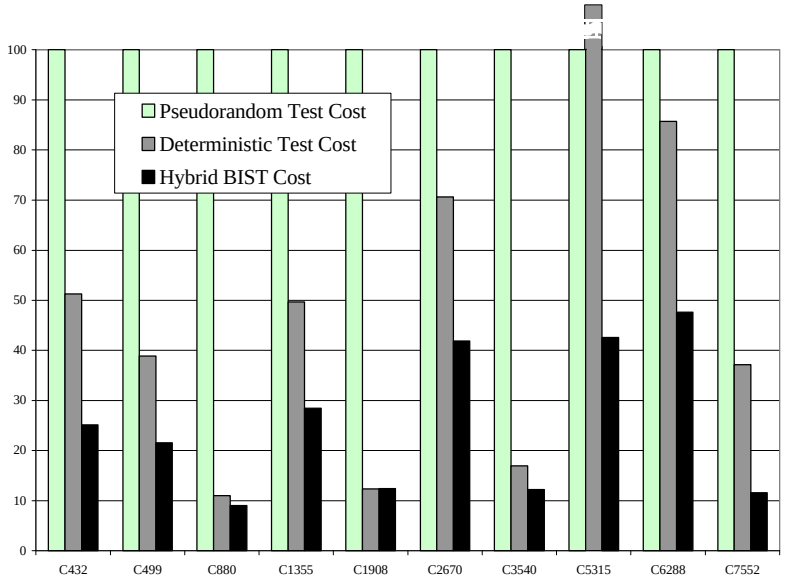


Figure 26. Cost comparison of different methods.
Cost of pseudorandom test is taken as 100%

4.8 Conclusions

In this chapter we described a hybrid BIST architecture for testing systems-on-chip. It supports the combination of pseudorandom test patterns with deterministic test patterns in a cost-effective way. The self-test architecture can be implemented either in classical way, by using LFSRs, or in software to reduce the area overhead and to take advantage of the SoC architecture. For selecting the optimal switching moment from the pseudorandom test mode to the stored test mode two algorithms were proposed for calculating the complete cost curve of the different hybrid BIST solutions. The first one is a straightforward method based on using traditional fault simulation and test pattern generation. The second one is based on fault table manipulations and uses test compaction. A Tabu search algorithm was also developed to reduce the number of calculations in search for the optimal solution for hybrid BIST. The experimental results demonstrate the feasibility of the approach and the efficiency of the fault table based cost calculation method combined with Tabu search for finding optimized cost-effective solutions for hybrid BIST.

Chapter 5
Conclusions and Future Work

The aim of this thesis is to propose a technique for test pattern generation at high abstraction level and to develop a built-in self-test methodology that requires minimal amount of overhead. In this section we summarize the thesis and outline possible directions for the future work.

5.1 Conclusions

The introduction of new EDA tools has allowed designers to work on higher abstraction levels, leaving the task of generating lower level designs to automatic synthesis tools. Despite this trend, test-related activities are still mainly performed at the gate level, and the risk of reiterating through the design flow due to test problems is high. Due to the increased complexity, the test generation process is also one of the most expensive and time-consuming steps of the entire design flow. Therefore, new methods for test pattern generation and testability analysis at the early stages of the design flow are highly beneficial. The design flow can be further improved by different design-for-testability techniques. In this way, significant improvement could be achieved in terms of design cost (especially by reducing the time for designing a testable system) and design quality

(by identifying the optimal solution in terms not only of area, time, and power constraints, but also of testing).

In this thesis we have proposed a novel high-level hierarchical test pattern generation algorithm. It works at the implementation independent behavioral level but also takes into account information from lower abstraction levels and is therefore able to generate test sequences with higher fault coverage than those test generation algorithms that are working purely on a behavioral level. We have demonstrated that the generated sequences can be successfully used for manufacturing test as well as for testability evaluation at the very early stages of the design cycle.

We have also proposed an architecture for self-testing systems-on-chip. It is a hybrid BIST architecture, which supports application of a hybrid test set. This architecture can be implemented either in hardware or software. In the case of software implementation, only small modifications of the existing system are required. The hybrid test set is composed of a limited number of pseudorandom test vectors and some additional deterministic test patterns that are specially designed to shorten the pseudorandom test cycle and to target random resistant faults. We have provided several methods for finding the optimal balance between those two test sets and for calculating the total cost of implementing a hybrid BIST solution.

5.2 Future Work

The following are some possible directions for future research:

High-level hierarchical test pattern generation:

- *Testability of hardware/software systems.* The testing of the hardware and software parts of a system are, at this moment, considered usually as separate problems and solved with very different methods. It would be very innovative to develop a test generation technique that is both applicable to the hardware and the software domains. As an example, the early generated test sequences could be

effective in testing hardware components against manufacturing defects, but could also be useful for debugging the code implementing the same component, if the designer decides to choose a software solution. Future work will investigate whether it is possible that, to some extent, the concept of testability is independent of the adopted implementation in hardware or software.

- *High-level fault models.* The problem with existing high-level fault models is that their efficiency has been so far demonstrated only experimentally. Therefore, it would be highly beneficial to develop a theoretical framework concerning high-level testability. Such a theoretical foundation is crucial for generation of efficient test sequences, testability analysis and DFT insertion at the high level of abstraction. This may lead to the development of new fault models that are able to represent the physical defects or software bugs and to map them on high-level descriptions.

Hybrid BIST:

- *Hybrid BIST for sequential circuits.* In this thesis we have proposed a hybrid BIST approach for combinational circuits. A more complex problem is to propose an architecture and optimization mechanisms for sequential circuits. The difficulty of developing such architecture and mechanisms is not only due to the complex nature of sequential circuits, but also related to pseudorandom testability. In case of combinatorial circuits, pseudorandom patterns have relatively high fault detection capabilities. This is not valid for sequential circuits and alternative methods for reducing the test data amount has to be developed. One of the possibilities is to apply pseudorandom patterns only for a combinatorial section of

the design while the rest of the design is tested with deterministic patterns.

- *Self test methods for other fault models.* Most of the existing work in the area of BIST is targeting the classical SSA fault model. At the same time it has been demonstrated that the SSA fault model can only cover some failure modes in CMOS technology. Thus, the importance of other fault models (like transition and path delay) is increasing rapidly. Therefore, it would be very interesting to analyze the quality of hybrid test set in terms of defect detection capabilities and to develop a methodology to support the detection of other failures than the stuck-at ones.

References

[1] M. Abramovici, M. A. Breuer, A. D. Friedman, "Digital Systems Testing and Testable Design," *IEEE Press*, 1990.

[2] V. D. Agrawal, C. R. Kime, K. K. Saluja, "A Tutorial on Built-In Self-Test," *IEEE Design and Test of Computers*, pp. 73-82, March 1993, pp. 69-77, June 1993.

[3] S. B. Akers, "Binary Decision Diagrams," *IEEE Trans. on Computers,* Vol. 27, pp. 509-516, 1978.

[4] C. Angelbro, "P-Bist Test Method Conquer the Ericsson World," *Ericsson Telecom AB*, 1997.

[5] P. H. Bardell, W. H. McAnney, J. Savir, "Built-In Test for VLSI Pseudorandom Techniques," *John Wiley and Sons*, 1987.

[6] B. Beizer, "Software Testing Techniques," (2nd edition), *Van Nostrand Reinhold*, 1990.

[7] M. A. Breuer, A. D. Friedman, "Diagnosis and Reliable Design of Digital Systems," *Computer Science Press*, 1976.

[8] F. Brglez, H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," *IEEE Int. Symp. on Circuits and Systems*, pp. 663-698, June 1985.

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

[10] CHIP V5, System Documentation, *COSYTEC SA*, 1998.

[11] R. A. DeMillo, R. J. Lipton, F. G. Sayward, "Hints on Test Data Selection: Help for the Practical Programmer," *IEEE Computer*, Vol.11, No.4, April 1978.

[12] E. B. Eichelberg, E. Lidbloom, "Random Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test," *IBM Journal of Research and Development,* Vol. 27, No. 3, pp. 265-272, May 1983.

[13] R. D. Eldred, "Test Routines Based on Symbolic Logic Systems," *Journal of the ACM*, Vol. 6, No. 1, pp. 33-36, 1959.

[14] P. Eles, K. Kuchcinski, Z. Peng, M. Minea, "Compiling VHDL into a High-Level Synthesis Design Representation," *EURO-DAC*, pp. 604-609, 1992.

[15] P. Eles, K. Kuchcinski, Z. Peng, "System Synthesis with VHDL", *Kluwer Academic Publishers*, 1997.

[16] F. Ferrandi, G. Ferrara, D. Scuito, A. Fin, F. Fummi, "Functional Test Generation for Behaviorally Sequential Models," *Design, Automation and Test in Europe (DATE 2001)*, pp. 403-410, 2001.

[17] A. Flint, "Multichip Module Self-Test Provides Means to Test at Speed," *EE-Evaluation Engineering*, pp. 46-55, September 1995.

[18] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & Ops. Res.*, pp. 533-549, No. 5, 1986.

[19] F. Glover, E. Taillard, and D. de Werra, "A user's guide to tabu search," *Annals of Operations Research*, 41:3-28, 1993.

[20] F. Glover and M. Laguna. "Modern Heuristic Techniques for Combinatorial Problems", *Blackwell Scientic Publishing*, pp. 70-141, 1993.

[21] S. W. Golomb, "Shift Register Sequences," *Aegan Park Press*, 1982.

[22] A. Grochowski, D. Bhattacharya, T. R. Viswanathan, K. Laker, "Integrated Circuit Testing for Quality Assurance in Manufacturing: History, Current Status, and Future Trends,"

IEEE Trans. on Circuits and Systems – II, Vol. 44, pp. 610-633, August 1997.

[23] P. Hansen. The steepest ascent mildest descent heuristic for combinational programming. *Congress on Numerical Methods in Combinatorial Optimization*, 1986.

[24] S. Hellebrand, S. Tarnick, J. Rajski, B. Courtois, "Generation Of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," *IEEE Int. Test Conference (ITC'92)*, pp. 120-129, 1992.

[25] S. Hellebrand, H.-J. Wunderlich, A. Hertwig, "Mixed-Mode BIST Using Embedded Processors," *Journal of Electronic Testing: Theory and Applications*, pp. 127-138, No. 12, 1998.

[26] W. E. Howden, "Weak Mutation Testing and Completeness of Test Sets," *IEEE Transactions on Software Engineering*," Vol. SE-8, No.4, July 1982.

[27] O. H. Ibarra, S. Sahni, "Polynomially Complete Fault Detection Problems," *IEEE Transactions on Computers,* Vol. C-24, No. 3, pp. 242-249, March 1975.

[28] "The International Technology Roadmap for Semiconductors (ITRS 2001)," *Semiconductor Industry Association,* 2001.

[29] "IEEE Standard VHDL Language Reference Manual," *IEEE Std 1076-1987*, March 31, 1988

[30] "IEEE Standard VHDL Language Reference Manual," *ANSI/IEEE Std 1076-1993, (Revision of IEEE Std 1076-1987)*, June 6, 1994

[31] G. Jervan , A. Markus, P. Paomets, J. Raik, R. Ubar, "A CAD system for Teaching Digital Test*," 2nd European Workshop on Microelectronics Education*, pp. 287-290, 1998.

[32] G. Jervan, P. Eles, Z. Peng, "A Hierarchical Test Generation Technique for Embedded Systems," *Electronic Circuits and Systems Conference (ECS'99)*, pp. 21-24, 1999.

[33] G. Jervan, Z. Peng, R. Ubar, "Test Cost Minimization for Hybrid BIST," *IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems (DFT'00)*, pp.283-291, 2000.

[34] G. Jervan, Z. Peng, E. Orasson, R. Raidma, R. Ubar, "Fast Test Cost Calculation for Hybrid BIST in Digital Systems," *Euromicro Symposium on Digital Systems Design*, pp. 318-325, 2001.

[35] G. Jervan, Z. Peng, R. Ubar, H. Kruus, "Using Tabu Method for Optimizing the Cost of Hybrid BIST," *16th Conference on Design of Circuits and Integrated Systems (DCIS 2001)*, pp. 445-450, 2001.

[36] G. Jervan, Z. Peng, R. Ubar, H. Kruus, "A Hybrid BIST Architecture and its Optimization for SoC Testing," *IEEE 2002 3rd International Symposium on Quality Electronic Design (ISQED'02)*, pp. 273-279, 2002.

[37] G. Jervan, Z. Peng, O. Goloubeva, M. Sonza Reorda M. Violante, "High-Level and Hierarchical Test Sequence Generation," *IEEE International Workshop on High Level Design Validation and Test (HLDVT'02),* 2002 (to be published).

[38] J. Khare, W. Maly, N. Tiday, "Fault characterization of standard cell libraries using inductive contamination analysis (ICA)," *14th VLSI Test Symposium,* pp. 405-413, 1996.

[39] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol. 220, No. 4598, pp. 671-680, 1983.

[40] B. Könemann, J. Mucha, G. Zwiehoff, "Built-In Test for Complex Digital Integrated Circuits", *IEEE J. Solid-State Circuits*, Vol. SC-15, No. 3, pp. 315-319, June 1980.

[41] E. Larsson, Z. Peng, "An Integrated Framework for the Design and Optimization of SOC Test Solutions", *Journal of Electronic Testing; Theory and Applications (JETTA)*, vol. 18, no. 4, August 2002.

[42] J. Lee, J. H. Patel, "Architectural Level Test Generation for Microprocessors," *IEEE Trans. CAD,* vol. 13, no. 10, pp. 1288-1300, October 1994.

[43] E. J. Marinissen, Y. Zorian, "Challenges in Testing Core-Based System ICs," *IEEE Communications Magazine*, pp. 104-109, June 1999.

[44] K. Marriott, P. J. Stuckey, Programming with Constraints: Introduction, *MIT Press*, 1998.

[45] P. Michel, U. Lauther, P. Duzy, "The Synthesis Approach To Digital System Design," *Kluwer Academic Publishers*, 1992

[46] G. E. Moore, "Cramming More Components Onto Integrated Circuits", *Electronics*, April 19, 1965.

[47] B. T. Murray, J. P. Hayes, "Hierarchical Test Generation Using Precomputed Tests for Modules," *International Test Conference*, pp. 221-229, 1988.

[48] J. Raik, R. Ubar. "Fast Test Pattern Generation for Sequential Circuits Using Decision Diagram Representations." *Journal of Electronic Testing: Theory and Applications (JETTA)*, vol. 16, no. 3, pp. 213-226, June, 2000.

[49] S. M. Sait, H. Youssef, "Iterative Computer Algorithms with Application in Engineering. Solving Combinatorial Optimization Problems," *IEEE Computer Society Press*, Los Alamitos, CA, 1999.

[50] J. P. Shen, W. Maly, F. J. Ferguson, "Inductive Fault Analysis of MOS Integrated Circuits," *IEEE Design and Test of Computers,* Vol. 2, No. 6, pp. 13-26, December 1985.

[51] SICStus Prolog User's Manual, *Swedish Institute of Computer Science*, 2001.

[52] M. Sonza Reorda, M. Violante G. Jervan, Z. Peng, "COTEST Report D1: Report on benchmark identification and planning of experiments to be performed", *Politecnico di Torino*, 2002. http://www.ida.liu.se/~eslab/cotest.html

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

[54] Y. Sun, "Automatic Behavioral Test Generation By Using a Constraint Solver", Final Thesis, LiTH-IDA-Ex-02/13, *Linköping University*, 2001.

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

[56] Turbo Tester Reference Manual. Version 3.99.03. *Tallinn Technical University* 1999. http://www.pld.ttu.ee/tt

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

[58] R. Ubar, "Test Synthesis with Alternative Graphs," *IEEE Design and Test of Computers*, Vol. 13, No. 1, pp. 48-57, Spring 1996.

[59] "Virtual Socket Interface Architectural Document," *VSI Alliance*, Nov. 1996.

[60] V. N. Yarmolik, I. V. Kachan, "Self-Checking VLSI Design," *Elsevier Science Ltd*, 1993.

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

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

LINKÖPINGS UNIVERSITET

Avdelning, institution
Division, department

Institutionen för datavetenskap

Department of Computer and Information Science

Datum
Date

2002-10-15

Språk
Language

[ ] Svenska/Swedish
[X] Engelska/English
[ ] _____

Rapporttyp
Report category

[X] Licentiatavhandling
[ ] Examensarbete
[ ] C-uppsats
[ ] D-uppsats
[ ] Övrig rapport

_____

URL för elektronisk version

http://www.ida.liu.se/~eslab

ISBN 91-7373-442-X

ISRN LiU-Tek-Lic-2002:46

Serietitel och serienummer
Title of series, numbering

ISSN 0280-7971

Linköping Studies in Science and Technology

Thesis No. 973

Titel
Title

High-Level Test Generation and Built-In Self-Test Techniques for Digital Systems

Författare
Author

Gert Jervan

Sammanfattning
Abstract

The technological development is enabling production of increasingly complex electronic systems. All those systems must be verified and tested to guarantee correct behavior. As the complexity grows, testing is becoming one of the most significant factors that contribute to the final product cost. The established low-level methods for hardware testing are not any more sufficient and more work has to be done at abstraction levels higher than the classical gate and register-transfer levels. This thesis reports on one such work that deals in particular with high-level test generation and design for testability techniques.

The contribution of this thesis is twofold. First, we investigate the possibilities of generating test vectors at the early stages of the design cycle, starting directly from the behavioral description and with limited knowledge about the final implementation architecture. We have developed for this purpose a novel hierarchical test generation algorithm and demonstrated the usefulness of the generated tests not only for manufacturing test but also for testability analysis.

The second part of the thesis concentrates on design for testability. As testing of modern complex electronic systems is a very expensive procedure, special structures for simplifying this process can be inserted into the system during the design phase. We have proposed for this purpose a novel hybrid built-in self-test architecture, which makes use of both pseudorandom and deterministic test patterns, and is appropriate for modern system-on-chip designs. We have also developed methods for optimizing hybrid built-in self-test solutions and demonstrated the feasibility and efficiency of the proposed technique.

This work has been supported by the Swedish Foundation for Strategic Research (SSF) under the INTELECT program

Nyckelord
Keywords

High-Level Test, Test Generation, Testability Analysis, Design for Testability, Hybrid Built-In Self-Test

Department of Computer and Information Science
Linköpings universitet

Linköping Studies in Science and Technology
Faculty of Arts and Sciences - Licentiate Theses

No 17 **Vojin Plavsic:** Interleaved Processing of Non-Numerical Data Stored on a Cyclic Memory. (Available at: FOA, Box 1165, S-581 11 Linköping, Sweden. FOA Report B30062E)

No 28 **Arne Jönsson, Mikael Patel:** An Interactive Flowcharting Technique for Communicating and Realizing Algorithms, 1984.

No 29 **Johnny Eckerland:** Retargeting of an Incremental Code Generator, 1984.

No 48 **Henrik Nordin:** On the Use of Typical Cases for Knowledge-Based Consultation and Teaching, 1985.

No 52 **Zebo Peng:** Steps Towards the Formalization of Designing VLSI Systems, 1985.

No 60 **Johan Fagerström:** Simulation and Evaluation of Architecture based on Asynchronous Processes, 1985.

No 71 **Jalal Maleki:** ICONStraint, A Dependency Directed Constraint Maintenance System, 1987.

No 72 **Tony Larsson:** On the Specification and Verification of VLSI Systems, 1986.

No 73 **Ola Strömfors:** A Structure Editor for Documents and Programs, 1986.

No 74 **Christos Levcopoulos:** New Results about the Approximation Behavior of the Greedy Triangulation, 1986.

No 104 **Shamsul I. Chowdhury: S**tatistical Expert Systems - a Special Application Area for Knowledge-Based Computer Methodology, 1987.

No 108 **Rober Bilos:** Incremental Scanning and Token-Based Editing, 1987.

No 111 **Hans Block:** SPORT-SORT Sorting Algorithms and Sport Tournaments, 1987.

No 113 **Ralph Rönnquist:** Network and Lattice Based Approaches to the Representation of Knowledge, 1987.

No 118 **Mariam Kamkar, Nahid Shahmehri:** Affect-Chaining in Program Flow Analysis Applied to Queries of Programs, 1987.

No 126 **Dan Strömberg:** Transfer and Distribution of Application Programs, 1987.

No 127 **Kristian Sandahl:** Case Studies in Knowledge Acquisition, Migration and User Acceptance of Expert Systems, 1987.

No 139 **Christer Bäckström:** Reasoning about Interdependent Actions, 1988.

No 140 **Mats Wirén:** On Control Strategies and Incrementality in Unification-Based Chart Parsing, 1988.

No 146 **Johan Hultman:** A Software System for Defining and Controlling Actions in a Mechanical System, 1988.

No 150 **Tim Hansen:** Diagnosing Faults using Knowledge about Malfunctioning Behavior, 1988.

No 165 **Jonas Löwgren:** Supporting Design and Management of Expert System User Interfaces, 1989.

No 166 **Ola Petersson:** On Adaptive Sorting in Sequential and Parallel Models, 1989.

No 174 **Yngve Larsson:** Dynamic Configuration in a Distributed Environment, 1989.

No 177 **Peter Åberg:** Design of a Multiple View Presentation and Interaction Manager, 1989.

No 181 **Henrik Eriksson:** A Study in Domain-Oriented Tool Support for Knowledge Acquisition, 1989.

No 184 **Ivan Rankin:** The Deep Generation of Text in Expert Critiquing Systems, 1989.

No 187 **Simin Nadjm-Tehrani:** Contributions to the Declarative Approach to Debugging Prolog Programs, 1989.

No 189 **Magnus Merkel:** Temporal Information in Natural Language, 1989.

No 196 **Ulf Nilsson:** A Systematic Approach to Abstract Interpretation of Logic Programs, 1989.

No 197 **Staffan Bonnier:** Horn Clause Logic with External Procedures: Towards a Theoretical Framework, 1989.

No 203 **Christer Hansson:** A Prototype System for Logical Reasoning about Time and Action, 1990.

No 212 **Björn Fjellborg:** An Approach to Extraction of Pipeline Structures for VLSI High-Level Synthesis, 1990.

No 230 **Patrick Doherty:** A Three-Valued Approach to Non-Monotonic Reasoning, 1990.

No 237 **Tomas Sokolnicki:** Coaching Partial Plans: An Approach to Knowledge-Based Tutoring, 1990.

No 250 **Lars Strömberg:** Postmortem Debugging of Distributed Systems, 1990.

No 253 **Torbjörn Näslund:** SLDFA-Resolution - Computing Answers for Negative Queries, 1990.

No 260 **Peter D. Holmes:** Using Connectivity Graphs to Support Map-Related Reasoning, 1991.

No 283 **Olof Johansson:** Improving Implementation of Graphical User Interfaces for Object-Oriented Knowledge-Bases, 1991.

No 298 **Rolf G Larsson:** Aktivitetsbaserad kalkylering i ett nytt ekonomisystem, 1991.

No 318 **Lena Strömbäck:** Studies in Extended Unification-Based Formalism for Linguistic Description: An Algorithm for Feature Structures with Disjunction and a Proposal for Flexible Systems, 1992.

No 319 **Mikael Pettersson:** DML-A Language and System for the Generation of Efficient Compilers from Denotational Specification, 1992.

No 326 **Andreas Kågedal:** Logic Programming with External Procedures: an Implementation, 1992.

No 328 **Patrick Lambrix:** Aspects of Version Management of Composite Objects, 1992.

No 333 **Xinli Gu:** Testability Analysis and Improvement in High-Level Synthesis Systems, 1992.

No 335 **Torbjörn Näslund:** On the Role of Evaluations in Iterative Development of Managerial Support Sytems, 1992.

No 348 **Ulf Cederling:** Industrial Software Development - a Case Study, 1992.

No 352 **Magnus Morin:** Predictable Cyclic Computations in Autonomous Systems: A Computational Model and Implementation, 1992.

No 371 **Mehran Noghabai:** Evaluation of Strategic Investments in Information Technology, 1993.

No 378 **Mats Larsson:** A Transformational Approach to Formal Digital System Design, 1993.

No 380 **Johan Ringström:** Compiler Generation for Parallel Languages from Denotational Specifications, 1993.

No 381 **Michael Jansson:** Propagation of Change in an Intelligent Information System, 1993.

No 383 **Jonni Harrius:** An Architecture and a Knowledge Representation Model for Expert Critiquing Systems, 1993.

No 386 **Per Österling:** Symbolic Modelling of the Dynamic Environments of Autonomous Agents, 1993.

No 398 **Johan Boye:** Dependency-based Groudness Analysis of Functional Logic Programs, 1993.

No 402 **Lars Degerstedt:** Tabulated Resolution for Well Founded Semantics, 1993.
No 406 **Anna Moberg:** Satellitkontor - en studie av kommunikationsmönster vid arbete på distans, 1993.
No 414 **Peter Carlsson:** Separation av företagsledning och finansiering - fallstudier av företagsledarutköp ur ett agentteoretiskt perspektiv, 1994.
No 417 **Camilla Sjöström:** Revision och lagreglering - ett historiskt perspektiv, 1994.
No 436 **Cecilia Sjöberg:** Voices in Design: Argumentation in Participatory Development, 1994.
No 437 **Lars Viklund:** Contributions to a High-level Programming Environment for a Scientific Computing, 1994.
No 440 **Peter Loborg:** Error Recovery Support in Manufacturing Control Systems, 1994.
FHS 3/94 **Owen Eriksson:** Informationssystem med verksamhetskvalitet - utvärdering baserat på ett verksamhetsinriktat och samskapande perspektiv, 1994.
FHS 4/94 **Karin Pettersson:** Informationssystemstrukturering, ansvarsfördelning och användarinflytande - En komparativ studie med utgångspunkt i två informationssystemstrategier, 1994.
No 441 **Lars Poignant:** Informationsteknologi och företagsetablering - Effekter på produktivitet och region, 1994.
No 446 **Gustav Fahl:** Object Views of Relational Data in Multidatabase Systems, 1994.
No 450 **Henrik Nilsson:** A Declarative Approach to Debugging for Lazy Functional Languages, 1994.
No 451 **Jonas Lind:** Creditor - Firm Relations: an Interdisciplinary Analysis, 1994.
No 452 **Martin Sköld:** Active Rules based on Object Relational Queries - Efficient Change Monitoring Techniques, 1994.
No 455 **Pär Carlshamre:** A Collaborative Approach to Usability Engineering: Technical Communicators and System Developers in Usability-Oriented Systems Development, 1994.
FHS 5/94 **Stefan Cronholm:** Varför CASE-verktyg i systemutveckling? - En motiv- och konsekvensstudie avseende arbetssätt och arbetsformer, 1994.
No 462 **Mikael Lindvall:** A Study of Traceability in Object-Oriented Systems Development, 1994.
No 463 **Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av Sandviks förvärv av Bahco Verktyg, 1994.
No 464 **Hans Olsén:** Collage Induction: Proving Properties of Logic Programs by Program Synthesis, 1994.
No 469 **Lars Karlsson:** Specification and Synthesis of Plans Using the Features and Fluents Framework, 1995.
No 473 **Ulf Söderman:** On Conceptual Modelling of Mode Switching Systems, 1995.
No 475 **Choong-ho Yi:** Reasoning about Concurrent Actions in the Trajectory Semantics, 1995.
No 476 **Bo Lagerström:** Successiv resultatavräkning av pågående arbeten. - Fallstudier i tre byggföretag, 1995.
No 478 **Peter Jonsson:** Complexity of State-Variable Planning under Structural Restrictions, 1995.
FHS 7/95 **Anders Avdic:** Arbetsintegrerad systemutveckling med kalkylkprogram, 1995.
No 482 **Eva L Ragnemalm:** Towards Student Modelling through Collaborative Dialogue with a Learning Companion, 1995.
No 488 **Eva Toller:** Contributions to Parallel Multiparadigm Languages: Combining Object-Oriented and Rule-Based Programming, 1995.
No 489 **Erik Stoy:** A Petri Net Based Unified Representation for Hardware/Software Co-Design, 1995.
No 497 **Johan Herber:** Environment Support for Building Structured Mathematical Models, 1995.
No 498 **Stefan Svenberg:** Structure-Driven Derivation of Inter-Lingual Functor-Argument Trees for Multi-Lingual Generation, 1995.
No 503 **Hee-Cheol Kim:** Prediction and Postdiction under Uncertainty, 1995.
FHS 8/95 **Dan Fristedt:** Metoder i användning - mot förbättring av systemutveckling genom situationell metodkunskap och metodanalys, 1995.
FHS 9/95 **Malin Bergvall:** Systemförvaltning i praktiken - en kvalitativ studie avseende centrala begrepp, aktiviteter och ansvarsroller, 1995.
No 513 **Joachim Karlsson:** Towards a Strategy for Software Requirements Selection, 1995.
No 517 **Jakob Axelsson:** Schedulability-Driven Partitioning of Heterogeneous Real-Time Systems, 1995.
No 518 **Göran Forslund:** Toward Cooperative Advice-Giving Systems: The Expert Systems Experience, 1995.
No 522 **Jörgen Andersson:** Bilder av småföretagares ekonomistyrning, 1995.
No 538 **Staffan Flodin:** Efficient Management of Object-Oriented Queries with Late Binding, 1996.
No 545 **Vadim Engelson:** An Approach to Automatic Construction of Graphical User Interfaces for Applications in Scientific Computing, 1996.
No 546 **Magnus Werner :** Multidatabase Integration using Polymorphic Queries and Views, 1996.
FiF-a 1/96 **Mikael Lind:** Affärsprocessinriktad förändringsanalys - utveckling och tillämpning av synsätt och metod, 1996.
No 549 **Jonas Hallberg:** High-Level Synthesis under Local Timing Constraints, 1996.
No 550 **Kristina Larsen:** Förutsättningar och begränsningar för arbete på distans - erfarenheter från fyra svenska företag. 1996.
No 557 **Mikael Johansson:** Quality Functions for Requirements Engineering Methods, 1996.
No 558 **Patrik Nordling:** The Simulation of Rolling Bearing Dynamics on Parallel Computers, 1996.
No 561 **Anders Ekman:** Exploration of Polygonal Environments, 1996.
No 563 **Niclas Andersson:** Compilation of Mathematical Models to Parallel Code, 1996.
No 567 **Johan Jenvald:** Simulation and Data Collection in Battle Training, 1996.
No 575 **Niclas Ohlsson:** Software Quality Engineering by Early Identification of Fault-Prone Modules, 1996.
No 576 **Mikael Ericsson:** Commenting Systems as Design Support—A Wizard-of-Oz Study, 1996.
No 587 **Jörgen Lindström:** Chefers användning av kommunikationsteknik, 1996.
No 589 **Esa Falkenroth:** Data Management in Control Applications - A Proposal Based on Active Database Systems, 1996.
No 591 **Niclas Wahllöf:** A Default Extension to Description Logics and its Applications, 1996.
No 595 **Annika Larsson:** Ekonomisk Styrning och Organisatorisk Passion - ett interaktivt perspektiv, 1997.
No 597 **Ling Lin:** A Value-based Indexing Technique for Time Sequences, 1997.

No 598 **Rego Granlund:** C^3Fire - A Microworld Supporting Emergency Management Training, 1997.
No 599 **Peter Ingels:** A Robust Text Processing Technique Applied to Lexical Error Recovery, 1997.
No 607 **Per-Arne Persson:** Toward a Grounded Theory for Support of Command and Control in Military Coalitions, 1997.
No 609 **Jonas S Karlsson:** A Scalable Data Structure for a Parallel Data Server, 1997.
FiF-a 4 **Carita Åbom:** Videomötesteknik i olika affärssituationer - möjligheter och hinder, 1997.
FiF-a 6 **Tommy Wedlund**: Att skapa en företagsanpassad systemutvecklingsmodell - genom rekonstruktion, värdering och vidareutveckling i T50-bolag inom ABB, 1997.
No 615 **Silvia Coradeschi**: A Decision-Mechanism for Reactive and Coordinated Agents, 1997.
No 623 **Jan Ollinen:** Det flexibla kontorets utveckling på Digital - Ett stöd för multiflex? 1997.
No 626 **David Byers:** Towards Estimating Software Testability Using Static Analysis, 1997.
No 627 **Fredrik Eklund:** Declarative Error Diagnosis of GAPLog Programs, 1997.
No 629 **Gunilla Ivefors:** Krigsspel coh Informationsteknik inför en oförutsägbar framtid, 1997**.**
No 631 **Jens-Olof Lindh:** Analysing Traffic Safety from a Case-Based Reasoning Perspective, 1997
No 639 **Jukka Mäki-Turja:**. Smalltalk - a suitable Real-Time Language, 1997.
No 640 **Juha Takkinen**: CAFE: Towards a Conceptual Model for Information Management in Electronic Mail, 1997.
No 643 **Man Lin**: Formal Analysis of Reactive Rule-based Programs, 1997.
No 653 **Mats Gustafsson**: Bringing Role-Based Access Control to Distributed Systems, 1997.
FiF-a 13 **Boris Karlsson:** Metodanalys för förståelse och utveckling av systemutvecklingsverksamhet. Analys och värdering av systemutvecklingsmodeller och dess användning, 1997.
No 674 **Marcus Bjäreland:** Two Aspects of Automating Logics of Action and Change - Regression and Tractability, 1998.
No 676 **Jan Håkegård**: Hiera rchical Test Architecture and Board-Level Test Controller Synthesis, 1998.
No 668 **Per-Ove Zetterlund**: Normering av svensk redovisning - En studie av tillkomsten av Redovisningsrådets rekommendation om koncernredovisning (RR01:91), 1998.
No 675 **Jimmy Tjäder**: Projektledaren & planen - en studie av projektledning i tre installations- och systemutvecklingsprojekt, 1998.
FiF-a 14 **Ulf Melin**: Informationssystem vid ökad affärs- och processorientering - egenskaper, strategier och utveckling, 1998.
No 695 **Tim Heyer**: COMPASS: Introduction of Formal Methods in Code Development and Inspection, 1998.
No 700 **Patrik Hägglund:** Programming Languages for Computer Algebra, 1998.
FiF-a 16 **Marie-Therese Christiansson:** Inter-organistorisk verksamhetsutveckling - metoder som stöd vid utveckling av partnerskap och informationssystem, 1998.
No 712 **Christina Wennestam:** Information om immateriella resurser. Investeringar i forskning och utveckling samt i personal inom skogsindustrin, 1998.
No 719 **Joakim Gustafsson:** Extending Temporal Action Logic for Ramification and Concurrency, 1998.
No 723 **Henrik André-Jönsson:** Indexing time-series data using text indexing methods, 1999.
No 725 **Erik Larsson:** High-Level Testability Analysis and Enhancement Techniques, 1998.
No 730 **Carl-Johan Westin:** Informationsförsörjning: en fråga om ansvar - aktiviteter och uppdrag i fem stora svenska organisationers operativa informationsförsörjning, 1998.
No 731 **Åse Jansson:** Miljöhänsyn - en del i företags styrning, 1998.
No 733 **Thomas Padron-McCarthy:** Performance-Polymorphic Declarative Queries, 1998.
No 734 **Anders Bäckström:** Värdeskapande kreditgivning - Kreditriskhantering ur ett agentteoretiskt perspektiv, 1998.
FiF-a 21 **Ulf Seigerroth:** Integration av förändringsmetoder - en modell för välgrundad metodintegration, 1999.
FiF-a 22 **Fredrik Öberg:** Object-Oriented Frameworks - A New Strategy for Case Tool Development, 1998.
No 737 **Jonas Mellin:** Predictable Event Monitoring, 1998.
No 738 **Joakim Eriksson:** Specifying and Managing Rules in an Active Real-Time Database System, 1998.
FiF-a 25 **Bengt E W Andersson:** Samverkande informationssystem mellan aktörer i offentliga åtaganden - En teori om aktörsarenor i samverkan om utbyte av information, 1998.
No 742 **Pawel Pietrzak:** Static Incorrectness Diagnosis of CLP (FD), 1999.
No 748 **Tobias Ritzau:** Real-Time Reference Counting in RT-Java, 1999.
No 751 **Anders Ferntoft:** Elektronisk affärskommunikation - kontaktkostnader och kontaktprocesser mellan kunder och leverantörer på producentmarknader,1999.
No 752 **Jo Skåmedal:** Arbete på distans och arbetsformens påverkan på resor och resmönster, 1999.
No 753 **Johan Alvehus:** Mötets metaforer. En studie av berättelser om möten, 1999.
No 754 **Magnus Lindahl:** Bankens villkor i låneavtal vid kreditgivning till högt belånade företagsförvärv: En studie ur ett agentteoretiskt perspektiv, 2000.
No 766 **Martin V. Howard:** Designing dynamic visualizations of temporal data, 1999.
No 769 **Jesper Andersson:** Towards Reactive Software Architectures, 1999.
No 775 **Anders Henriksson:** Unique kernel diagnosis, 1999.
FiF-a 30 **Pär J. Ågerfalk:** Pragmatization of Information Systems - A Theoretical and Methodological Outline, 1999.
No 787 **Charlotte Björkegren:** Learning for the next project - Bearers and barriers in knowledge transfer within an organisation, 1999.
No 788 **Håkan Nilsson:** Informationsteknik som drivkraft i granskningsprocessen - En studie av fyra revisionsbyråer, 2000.
No 790 **Erik Berglund:** Use-Oriented Documentation in Software Development, 1999.
No 791 **Klas Gäre:** Verksamhetsförändringar i samband med IS-införande, 1999.
No 800 **Anders Subotic:** Software Quality Inspection, 1999.
No 807 **Svein Bergum**: Managerial communication in telework, 2000.

No 809 **Flavius Gruian:** Energy-Aware Design of Digital Systems, 2000.
FiF-a 32 **Karin Hedström:** Kunskapsanvändning och kunskapsutveckling hos verksamhetskonsulter - Erfarenheter från ett FOU-samarbete, 2000.
No 808 **Linda Askenäs:** Affärssystemet - En studie om teknikens aktiva och passiva roll i en organisation, 2000.
No 820 **Jean Paul Meynard:** Control of industrial robots through high-level task programming, 2000.
No 823 **Lars Hult:** Publika Gränsytor - ett designexempel, 2000.
No 832 **Paul Pop:** Scheduling and Communication Synthesis for Distributed Real-Time Systems, 2000.
FiF-a 34 **Göran Hultgren:** Nätverksinriktad Förändringsanalys - perspektiv och metoder som stöd för förståelse och utveckling av affärsrelationer och informationssystem, 2000.
No 842 **Magnus Kald:** The role of management control systems in strategic business units, 2000.
No 844 **Mikael Cäker:** Vad kostar kunden? Modeller för intern redovisning, 2000.
FiF-a 37 **Ewa Braf**: Organisationers kunskapsverksamheter - en kritisk studie av ”knowledge management”, 2000.
FiF-a 40 **Henrik Lindberg:** Webbaserade affärsprocesser - Möjligheter och begränsningar, 2000.
FiF-a 41 **Benneth Christiansson:** Att komponentbasera informationssystem - Vad säger teori och praktik?, 2000.
No. 854 **Ola Pettersson:** Deliberation in a Mobile Robot, 2000.
No 863 **Dan Lawesson**: Towards Behavioral Model Fault Isolation for Object Oriented Control Systems, 2000.
No 881 **Johan Moe:** Execution Tracing of Large Distributed Systems, 2001.
No 882 **Yuxiao Zhao:** XML-based Frameworks for Internet Commerce and an Implementation of B2B e-procurement, 2001.
No 890 **Annika Flycht-Eriksson:** Domain Knowledge Management inInformation-providing Dialogue systems, 2001.
Fif-a 47 **Per-Arne Segerkvis**t: Webbaserade imaginära organisationers samverkansformer, 2001.
No 894 **Stefan Svarén:** Styrning av investeringar i divisionaliserade företag - Ett koncernperspektiv, 2001.
No 906 **Lin Han**: Secure and Scalable E-Service Software Delivery, 2001.
No 917 **Emma Hansson:** Optionsprogram för anställda - en studie av svenska börsföretag, 2001.
No 916 **Susanne Odar:** IT som stöd för strategiska beslut, en studie av datorimplementerade modeller av verksamhet som stöd för beslut om anskaffning av JAS 1982, 2002.
Fif-a-49 **Stefan Holgersson:** IT-system och filtrering av verksamhetskunskap - kvalitetsproblem vid analyser och beslutsfattande som bygger på uppgifter hämtade från polisens IT-system, 2001.
Fif-a-51 **Per Oscarsson:**Informationssäkerhet i verksamheter - begrepp och modeller som stöd för förståelse av informationssäkerhet och dess hantering, 2001.
No 919 **Luis Alejandro Cortes:** A Petri Net Based Modeling and Verification Technique for Real-Time Embedded Systems, 2001.
No 915 **Niklas Sandell:** Redovisning i skuggan av en bankkris - Värdering av fastigheter. 2001.
No 931 **Fredrik Elg:** Ett dynamiskt perspektiv på individuella skillnader av heuristisk kompetens, intelligens, mentala modeller, mål och konfidens i kontroll av mikrovärlden Moro, 2002.
No 933 **Peter Aronsson:** Automatic Parallelization of Simulation Code from Equation Based Simulation Languages, 2002.
No 938 **Bourhane Kadmiry**: Fuzzy Control of Unmanned Helicopter, 2002.
No 942 **Patrik Haslum**: Prediction as a Knowledge Representation Problem: A Case Study in Model Design, 2002.
No 956 **Robert Sevenius:** On the instruments of governance - A law & economics study of capital instruments in limited liability companies, 2002.
FiF-a 58 **Johan Petersson:** Lokala elektroniska marknadsplatser - informationssystem för platsbundna affärer, 2002.
No 964 **Peter Bunus:** Debugging and Structural Analysis of Declarative Equation-Based Languages, 2002.
No 973 **Gert Jervan:** High-Level Test Generation and Built-In Self-Test Techniques for Digital Systems, 2002.