

Linköping Studies in Science and Technology
Dissertation No. 945

Hybrid Built-In Self-Test and Test Generation Techniques for Digital Systems

Gert Jervan



Linköpings universitet
INSTITUTE OF TECHNOLOGY

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

Linköping 2005

To Liisu

Abstract

The technological development is enabling the production of increasingly complex electronic systems. All such systems must be verified and tested to guarantee their correct behavior. As the complexity grows, testing has become one of the most significant factors that contribute to the total development cost. In recent years, we have also witnessed the inadequacy of the established testing methods, most of which are based on low-level representations of the hardware circuits. Therefore, more work has to be done at abstraction levels higher than the classical gate and register-transfer levels. At the same time, the automatic test equipment based solutions have failed to deliver the required test quality. As a result, alternative testing methods have been studied, which has led to the development of built-in self-test (BIST) techniques.

In this thesis, we present a novel hybrid BIST technique that addresses several areas where classical BIST methods have shortcomings. The technique makes use of both pseudorandom and deterministic testing methods, and is devised in particular for testing modern systems-on-chip. One of the main contributions of this thesis is a set of optimization methods to reduce the hybrid test cost while not sacrificing test quality. We have devel-

oped several optimization algorithms for different hybrid BIST architectures and design constraints. In addition, we have developed hybrid BIST scheduling methods for an abort-on-first-fail strategy, and proposed a method for energy reduction for hybrid BIST.

Devising an efficient BIST approach requires different design modifications, such as insertion of scan paths as well as test pattern generators and signature analyzers. These modifications require careful testability analysis of the original design. In the latter part of this thesis, we propose a novel hierarchical test generation algorithm that can be used not only for manufacturing tests but also for testability analysis. We have also investigated 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.

Experiments, based on benchmark examples and industrial designs, have been carried out to demonstrate the usefulness and efficiency of the proposed methodologies and techniques.

Acknowledgments

I came to ESLAB in the spring of 1998 to work on my master thesis. I was 100% sure that I would stay for 3 months and I had no plans for longer residence or PhD studies at Linköping. How wrong I was! As you see – it is 2005 and the thesis is in your hands.

The biggest “culprit” is undoubtedly Professor Zebo Peng. He encouraged me to take up the PhD studies at ESLAB and has been a great support since then. Zebo’s supervision style is something that I really admire. He has always given me plenty of freedom while making sure that the work progresses towards the right direction. His recommendations and guidance have made me to stand on my own feet and it is hard to underestimate this.

A special thank should go to Professor Petru Eles, who has brought new ideas into my research and has enriched days with interesting remarks, either about research, politics or sports.

Many thanks also to my first scientific supervisor Professor Raimund Ubar from Tallinn University of Technology. We have had very fruitful cooperation in the past and I am very hopeful that the same will continue in the future. This cooperation has produced several excellent results, some of which are presented

also in this thesis. The cooperation with Gunnar Carlsson from Ericsson has provided invaluable insight into the industrial practices.

During my years at IDA I have met many wonderful people and I am really grateful for those moments. A special thank to Gunilla for taking care of many practical issues. Not to mention you, ESLAB guys! It has been many wonderful years at Linköping and it would have been much more boring without you.

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

Finally, I would like to thank my family. My mother, father and sister have always been the greatest supporters. Encouragement and care from Liisu have been invaluable. Suur aitähh teile: Eeve, Toomas, Getli ja Mai-Liis!

A handwritten signature in cursive script, reading "Gert Jervan". The ink is dark and the signature is fluid, with a long, sweeping tail on the final letter.

Gert Jervan

Linköping/Tallinn 2005

Table of Contents

Part I Preliminaries.....	1
1. Introduction.....	3
1.1. Digital Systems Design and Manufacturing Flow	5
1.2. Motivation.....	9
1.3. Problem Formulation	10
1.4. Contributions	11
1.5. Thesis Overview	13
2. Testing and Design for Testability	15
2.1. Introduction to Hardware Testing.....	16
2.2. Failures and Fault Models	18
2.2.1. Stuck-At Fault Model	19
2.2.2. Other Structural Fault Models	21
2.2.3. High-Level Fault Models	21
2.3. Automatic Test Pattern Generation	23
2.4. Test Generation at Higher Levels of Abstraction	26
2.5. Test Application.....	27
2.5.1. Off-line Test Application	28
2.5.2. Abort-on-First-Fail Testing	29
2.6. Design for Testability	30

2.6.1. Scan-Path Insertion	31
2.6.2. Built-In Self-Test	33
2.7. Emerging Problems in System-on-Chip Testing	44
2.7.1. Core Internal Test Knowledge Transfer	48
2.7.2. Core Test Access Challenges	48
2.7.3. Chip-level Test Challenges	49
2.7.4. Core Test Architecture	49
2.7.5. Power Dissipation	53
2.8. Conclusions	55
Part II Hybrid Built-In Self-Test	57
3. Introduction and Related Work	59
3.1. Introduction	60
3.2. Problems with Classical BIST	61
3.3. BIST Improvement Techniques	64
3.3.1. Test Point Insertion	64
3.3.2. Weighted Random Testing	66
3.3.3. Test Pattern Compression	66
3.3.4. Mixed-Mode Schemes	67
3.4. Conclusions	70
4. Hybrid BIST Concept	73
4.1. Introduction	74
4.2. Basic Principle	75
4.3. Cost Calculation	77
4.4. Architectures	79
4.4.1. Core-Level Hybrid BIST Architecture	80
4.4.2. System-Level Hybrid BIST Architectures	82
4.5. Conclusions	87
5. Hybrid BIST Cost Minimization for Single Core Designs ..	89
5.1. Introduction	89
5.2. Test Cost Minimization Algorithms	92

5.2.1. ATPG Based Approach	92
5.2.2. Fault Table Based Approach	93
5.2.3. Tabu Search Based Cost Optimization	95
5.3. Experimental Results	98
5.4. Conclusions	106
6. Hybrid BIST Time Minimization for Systems-on-Chip	109
6.1. Introduction	109
6.2. Parallel Hybrid BIST Architecture	110
6.2.1. Basic Definitions and Problem Formulation	112
6.2.2. Test Set Generation Based on Cost Estimates ..	115
6.2.3. Test Length Minimization Under Memory Constraints	121
6.2.4. Experimental Results	123
6.3. Broadcasting Based Hybrid BIST Architecture	129
6.3.1. Straightforward Approach	131
6.3.2. Iterative Approach	139
6.4. Conclusions	147
7. Hybrid BIST Energy Minimization	151
7.1. Introduction	151
7.2. Hybrid BIST and Possibilities for Energy Reduction .	152
7.3. Basic Definitions and Problem Formulation	154
7.3.1. Parameter Estimation	155
7.4. Heuristic Algorithms for Hybrid BIST Energy Minimization	155
7.4.1. Local Gain Algorithm	156
7.4.2. Average Gain Algorithm	157
7.5. Experimental Results	159
7.6. Conclusions	162
8. Hybrid BIST in an Abort-on-First-Fail Environment	163
8.1. Introduction	163

8.2. AOFF Test Scheduling	164
8.2.1. Definitions and Problem Formulation	165
8.2.2. Proposed Heuristic for Test Scheduling	170
8.2.3. Experimental Results	173
8.3. Conclusions	176
Part III Hierarchical Test Generation.....	177
9. Introduction and Modeling	179
9.1. Modeling with Decision Diagrams.....	180
9.1.1. Introduction	181
9.2. Modeling Digital Systems by Binary Decision Diagrams.....	182
9.3. Modeling with a Single Decision Diagram on Higher Levels	185
9.3.1. Decision Diagrams at the Behavioral Level	188
9.3.2. SICStus Prolog Representation of Decision Diagrams.....	188
10. Hierarchical Test Generation with DDs	191
10.1. Test Generation Algorithm	193
10.2. Scanning Test	195
10.2.1. Scanning Test in the HTG Environment.....	197
10.3. Conformity Test.....	199
10.4. Experimental Results.....	203
10.5. Conclusions	208
Part IV Conclusions and Future Work	209
11. Conclusions.....	211
12. Future Work	215
References	219

PART I

PRELIMINARIES

Chapter 1

Introduction

Jack S. Kilby devised the first integrated circuit (IC) almost five decades ago in 1958. Since that day, the semiconductor industry has distinguished itself by the rapid pace of improvement in its products. The most frequently cited trend is related to the integration level and is usually expressed via Moore's Law (i.e., the number of components per chip doubles every 18 months) [124]. The minimum feature sizes used to fabricate integrated circuits have decreased exponentially. The most significant trend from the consumers' point of view is the decreasing cost-per-function, which has led to significant improvements of productivity and quality of life through the proliferation of computers, electronic communication, and consumer electronics.

Until recently, reliability of electronic devices was mainly a concern in safety critical systems. In these systems, such as automotive or medical applications, failures may lead to catastrophic results and any failure should obviously be avoided. However, due to several reasons, reliability is becoming increasingly important also in other application domains, such as con-

CHAPTER 1

sumer electronics, desktop computing, telecommunication systems and others. This is mainly because electronic systems are omnipresent in almost every modern system and any failure might lead to negative effects, in terms of financial loss or decreased comfort of life.

In order to achieve a desired level of reliability it is important to find errors before encountering their consequences. Due to the complexity of modern systems and multitude of problems related to error detection, these activities are usually carried out at various stages of the design and production flow and target different sub-problems. For example, one has to make sure that we have designed the correct system, as it has to satisfy certain properties or conditions, which may be either general or specific to the particular system, directly derived from the initial specification. In addition, we have to check whether we have designed our system correctly, i.e. we have to obtain confidence in the designed system's ability to deliver the service in accordance with an agreed-upon system specification. In general, these tasks are called *verification* [108]. Similarly, we also have to certify that the manufactured hardware system corresponds to its original specification and no faults have been introduced during the manufacturing phase. Such activity, commonly called *testing* [3], is characterized by execution of the system while supplying it with inputs, often using dedicated *automatic test equipment* (ATE). Testing is also used to guarantee that the system continues to work according to its specifications, as it can detect many field failures caused by aging, electromagnetic interference, environmental extremes, wear-out and others.

This thesis addresses the problem of hardware testing, in particular we will focus on issues related to testing of digital hardware.

1.1. Digital Systems Design and Manufacturing Flow

The development of a very large scale integrated (VLSI) system can typically be divided into three main phases: specification and synthesis, implementation, and manufacturing, as depicted in Figure 1.1. During the *specification and synthesis* phase, the functionality of the circuit is described. This can be done at different levels of abstraction [47]: behavioral, register-transfer (RT) or gate level, using VHDL, Verilog or any other hardware description language (HDL) [48]. The transformations between different abstraction levels are usually performed by synthesis algorithms. Typically, the following synthesis steps can be distinguished (from the highest abstraction level downwards) [120]:

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).

CHAPTER 1

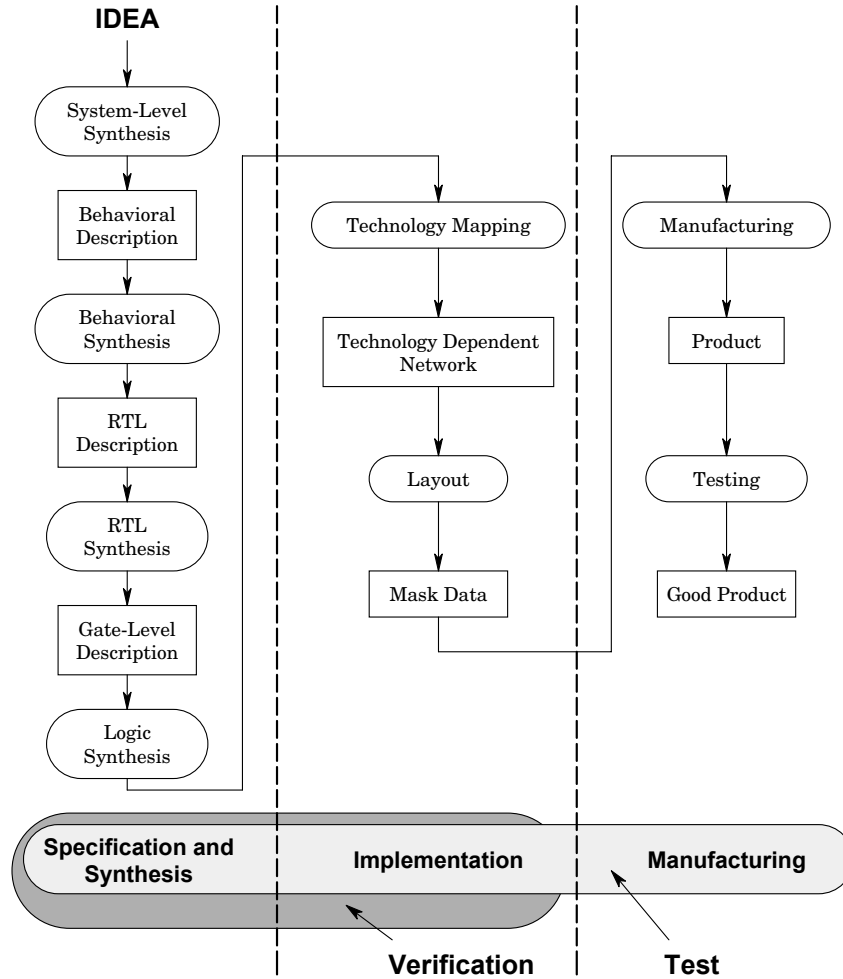


Figure 1.1. Design and production flow.

The output of the behavioral synthesis process is a description at the 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

INTRODUCTION

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 the 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.

During the *implementation* phase, the structural netlist of components, implementing the functions described in the specification, is generated and the design is transformed into layout masks. The transformation from the gate level to the physical level is known as technology mapping. The input of this step is a technology independent multi-level logic structure, a basic cell library, and a set of design constraints. During this phase appropriate library cells of a given target technology are selected for the network of abstract gates, produced as a result of logic synthesis, concluding thus the synthesis pipeline. The resulting layout gives designers possibility to extract design parameters, such as the load resistance and capacitance that are used for timing verification. Parameter extraction is becoming significantly important in modern deep submicron technologies.

At *manufacturing* stage the layout masks are used to produce the final circuitry in terms of a die on a wafer. The wafers are tested and all defective dies are identified. Good dies are pack-

CHAPTER 1

aged, tested and, finally, all good chips are shipped to the customers.

The latest advance in microelectronics technology has enabled the integration of an increasingly large number of transistors into a single die. The increased complexity together with reduced feature sizes means that errors are more likely to appear. For improving reliability, two types of activities are normally used: verification and testing (Figure 1.1). According to the current state of the art, for verification, designs are usually *simulated* on different abstraction levels, prior to their implementation in silicon [44], [140]. In some situations, verification is also performed after the first prototype of the chip is available. As for complex designs exhaustive simulation is practically infeasible, simulation based verification gives only a certain level of assurance about the design correctness [34]. One of the alternatives would be *formal verification* that uses mathematical reasoning for proving correctness of designs [63], [100]. This approach, with few exceptional methods, such as equivalence checking [76], has not become the mainstream, mainly due to the lack of appropriate tools.

Testing verifies that the manufactured integrated circuit corresponds to the intended function of the implementation. Its purpose is not to verify the correctness of the design; on the contrary, it verifies the correctness of the manufacturing process. It is performed on actual dies or chips, using test patterns that are generated to demonstrate that the final product is fault-free. In addition, testing can also be used during the latter stages of the product life cycle, in order to detect errors due to aging, environment or other factors.

In order to ease the complexity of the test pattern generation process specific hardware constructs, usually referred to as design-for-testability structures, are introduced into the circuits. Testability issues are currently becoming incorporated into the standard design-flows, although several testability techniques,

INTRODUCTION

like scan-chain insertion and self-test techniques are well investigated and ready to be used.

Testing is one of the major expenses in the integrated circuit (IC) design and manufacturing process, taking up to 35% of all costs. Test, diagnosis and repair costs of complex electronic systems reach often 40-50% of the total product realization cost and very soon the industry might face the challenge that the test of a transistor is more expensive than manufacturing it [153].

1.2. Motivation

As mentioned before, hardware testing is the process to check whether an integrated circuit is error-free. One of the reasons for errors are defects. As the produced circuits may contain different types of defects that are very complex, a model has to be defined 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. Most of the existing hardware testing techniques work at the abstraction levels where information about the final implementation architecture is already available. It has been proven mathematically that the generation of test patterns based on structural fault models is an NP-complete problem [80] and therefore different heuristics are usually used. Due to the increasing complexity of systems, these established low-level methods are not sufficient and more work has to be done at abstraction levels higher than the classical gate- and 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 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

CHAPTER 1

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.

In addition, various researches have shown that the switching activity, and consequently the dynamic power dissipation, during the test mode, may be several times higher than during the functional mode [32], [174]. Self-tests, regularly executed in portable devices, can hence consume significant amounts of energy and consequently reduce the lifetime of batteries [52]. Excessive switching activity during the test mode can also cause problems with circuit reliability [54]. The increased current levels can lead to serious silicon failure mechanisms (such as electromigration [115]) and may need expensive packages for removal of the excessive heat. Therefore, it is important to find ways for reducing power dissipation during testing.

Most DFT techniques require external test equipment for test application. *Built-in self-test* (BIST) technique, on the other hand, implements all test resources inside the chip. This technique does not suffer from the bandwidth limitations that exist for external testers and allows applying 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, it is important to address the weakness of the classical BIST techniques in order to utilize its potentials completely.

1.3. Problem Formulation

The previous section has presented the motivation for our work and given an indication of the current trends in the area of digital systems testing. The aim of the current thesis is twofold. First, we would like to propose a BIST strategy that can be used for reducing the testing effort for modern SOC designs and, sec-

INTRODUCTION

only, we are interested in performing test pattern generation as early as possible in the design process.

Since BIST structures are becoming more and more common in modern complex electronic systems, more emphasis should be put into minimization of costs caused by insertion of those structures. Our 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. This requires development of optimization methods that can take into account different design constraints imposed by the process technologies, such as tester memory, power dissipation, total energy and yield.

To deal with test pattern generation problem in early stages of the design flow we would like to develop a method that allows generation of test vectors starting directly from an implementation independent behavioral description. The developed method should have an important impact on the design flow, since it allows 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-level and low-level testability.

1.4. Contributions

The main contributions of this thesis are as follows:

- **A hybrid built-in self-test architecture and its optimization.** 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 apply a limited number of pseudorandom test vectors, which is then followed by the application of a stored deterministic test set,

CHAPTER 1

specially designed to shorten the pseudorandom test cycle and to target the random resistant faults. For supporting such a test strategy, we have developed several hybrid BIST architectures that target different test scenarios. As the test lengths of the two test sequences are 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 while taking into account different design constraints, such as tester memory limitations and power dissipation. In addition, we have also developed methods where the information about the quality of the manufacturing process can be incorporated into the optimization algorithms.

- **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. 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. In addition, it can identify possible hard-to-test modules of the system without waiting for the final implementation to be available. 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.

1.5. Thesis Overview

The rest of the thesis is structured as follows. Chapter 2 introduces the topic of digital systems test and design for testability. We cover typical failure mechanisms and methods for fault modeling and introduce concepts of automatic test pattern generation and different test application methods. Thereafter an overview of the most common design for test methods are given, followed by a discussion of emerging problems in the area of SOC testing.

Part II of the thesis is dedicated to the hybrid BIST techniques. In Chapter 3 we discuss the problems related to classical BIST schemes and give an overview of different methods devised for its improvement. Chapter 4 gives an overview of the proposed hybrid BIST approach, followed, in Chapter 5, by test cost minimization methods for single core designs. In Chapter 6 different algorithms for hybrid BIST time minimization for SOC designs are presented. In the first part of this chapter we concentrate on parallel hybrid BIST architectures while in the latter part of the chapter the test pattern broadcasting based architecture is covered. Chapter 7 introduces possibilities for hybrid BIST energy minimization and in Chapter 8 algorithm for hybrid BIST scheduling in an abort-on-first-fail environment is presented. In every chapter, the proposed algorithms are described together with experimental results to demonstrate the feasibility and usefulness of the algorithms.

The third part of this thesis covers the proposed hierarchical test generation algorithm. It starts with a detailed discussion of behavioral level decision diagrams used to capture a design at several levels of abstraction. Thereafter we describe selected fault models and present our test pattern generation algorithm. The chapter concludes with experimental results where we demonstrate the efficiency of our approach for generating manufacturing tests.

Part IV concludes this thesis and discusses possible directions for future work.

Chapter 2

Testing and Design for Testability

The aim of this chapter is to provide background for the thesis. It starts with an introduction to electronic systems testing. We will go through different fault types of complementary metal-oxide semiconductor (CMOS) integrated circuits and describe the ways to model them. Thereafter the chapter continues with the description of different testing and design-for-testability techniques. We give a short overview of the automatic test pattern generation (ATPG) algorithms and strategies and describe some systematic design modification techniques that are intended for improving testability, such as scan-chain insertion and built-in self-test (BIST).

The shift toward submicron technologies has enabled IC designers to integrate entire systems into a single chip. This new paradigm of system-on-chip (SOC) has introduced a magnitude of new testing problems and therefore at the end of this chapter emerging problems in SOC testing will also be described. We will

CHAPTER 2

in particular focus on power dissipation, test access and test scheduling problems.

2.1. Introduction to Hardware Testing

The testing activities for hardware systems can be classified according to many criteria. Generally speaking, we can distinguish two different types of testing: *parametric testing* and *functional testing*.

1. **Parametric Testing** measures electrical properties of pin electronics. This is done to ensure that components meet design specification for delays, voltages, power, etc. One of the parametric testing methodologies that has gained recently much attention is IDDq testing, a parametric technique for CMOS testing. IDDq testing monitors the current, IDD, a circuit draws when it is in a quiescent state. It is used to detect faults such as bridging faults, transistor stuck-open faults, gate oxide leaks, which increase the normally low IDD [84]. IDDq testing can detect some defects that are not detectable with other testing techniques and the results of IDDq testing can be used for reliability estimation.
2. **Functional Testing** aim at finding faults which cause a change in the functional behavior of the circuit. It is used in conjunction with the manufacturing process in order to ensure that only error-free chips are delivered to the customers. Some forms of functional testing can be used also for detecting faults that might occur during the chip lifetime, due to aging, environment and other factors.

Although highly important, this thesis will not cover aspects related to parametric testing and will focus solely on aspects related to functional testing of hardware circuits and systems. Therefore, also the word testing is used throughout this thesis to

TESTING AND DESIGN FOR TESTABILITY

denote functional testing of manufactured hardware systems, unless specified otherwise.

The purpose of hardware testing is to confirm that the function of each manufactured circuit corresponds to the function of the implementation [3]. During testing, the circuitry is exercised by applying the appropriate stimuli and its resulting responses are analyzed to determine whether it behaved correctly. If verification has assured that the design corresponds to its specification, then the incorrect behavior can be caused by *defects* introduced during the manufacturing process. There are many different types of defects, such as aging, misalignment, holes, contamination and others [130]. The diversity of defects leads to the very complex testing process, as the complexity of physical defects does not facilitate mathematical treatment of testing. Therefore, an efficient test solution requires an approach, where defects can be modeled by capturing the effect of the defect on the operation of the system at certain level of abstraction. This is called *fault modeling*. The most common alternative is to model faults at the logic level, such as *single stuck-at (SSA)* fault model. However, the increasing complexity of electronic systems necessitates the use of fault models that are derived from descriptions at higher abstraction levels, such as register-transfer (RT) and behavioral level.

After a fault model has been devised, efficient test stimuli can be generated by using an ATPG program that is applied to the *circuit under test* (CUT). However, this might not always be feasible, mainly because of the complexity of the testing process itself but also due to the complexity of the CUTs. Therefore, increasingly often designers introduce special structures, called design for testability structures, during the design phase of a digital system. The purpose of these structures is to make test pattern generation and test application easier and more efficient. Examples of typical DFT methods include scan-chain insertion and BIST.

CHAPTER 2

In the following, we are going to describe these basic concepts of digital hardware testing in more detail. We will give the background needed for better understanding of the thesis and introduce the state-of-the-art in the areas of the thesis contributions.

2.2. Failures and Fault Models

A typical 200-mm wafer in 0.25- μm CMOS technology can potentially contain a million printed geometries—the typically rectangular shapes that are the layout of transistors and the connections between them—in both x and y directions. This amounts to about 10^{12} possible geometries on each printed layer of a wafer. A few years back a chip typically had about six metal layers and a total number of lithography layers over 20 [130]. In 2004, we had already mass-market products produced in 90-nm technology, using 300-mm wafers with 7 interconnect layers [1]. Errors could arise in any geometry on any layer, so the possible number of defects is enormous.

Chip manufacturing is performed in multiple steps. Each of those steps, such as depositing, conducting, and insulating material, oxidation, photolithography, and etching [151], may introduce defects. Therefore, in an integrated circuit (IC) we can observe a wide range of possible defects. These include particles (small bits of material that might bridge lines, Figure 2.1), incorrect spacing (variations, which may short a circuit), incorrect implant value (due to machine error), misalignment (of layers), holes (exposed etched area), weak oxides (that might cause gate oxide breakdown), and contamination (unwanted foreign material) [130]. On circuit level, these defects appear as *failure modes*. Most common of them are shorts, opens and parameter degradations. However, at this low level of abstraction testing is still practically infeasible.

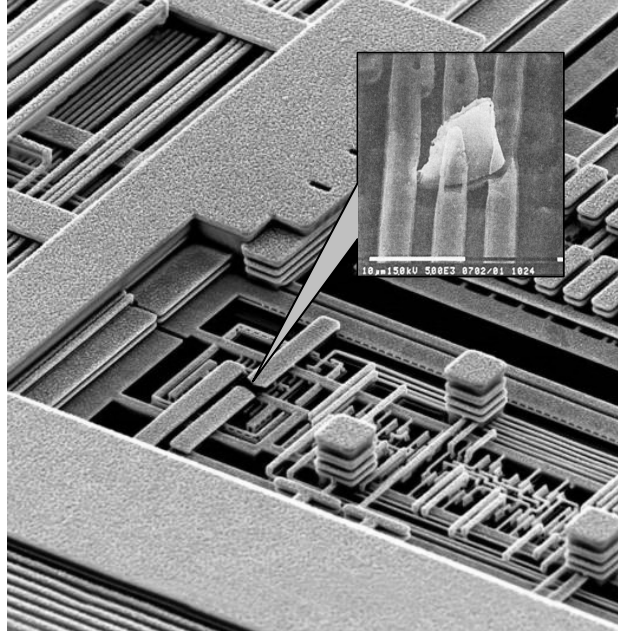


Figure 2.1. An example of a defect (© IBM)

At the logical level, the effects of failure modes appear as incorrect signal values and in order to devise efficient testing methods the effects of different failures should be captured in different *fault models*. The fault model does not necessarily have to capture the exact effect of the defect; rather it has to be useful in detecting the defects.

2.2.1. Stuck-At Fault Model

The earliest and most well-known fault model is the single stuck-at (SSA) fault model [38] (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). This means that with the SSA fault model it is assumed that the elementary components are fault-free and only their interconnects are affected [3]. This will reduce the number of faults to $2n$, where n is the num-

CHAPTER 2

ber of lines on which SSA faults can be defined. Experiments have shown that this fault model is useful (providing relatively high defect coverage, while being technology-independent) and can be used even for identifying the presence of multiple faults that 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. Nevertheless, 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 in 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.

The SSA fault 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.

Despite all its shortcomings, the stuck-at fault model has been the dominant fault model for several decades, and continues to be dominant even today, for both its simplicity and its demonstrated utility. Therefore, also in this thesis we are going to discuss testing in the context of testing for single stuck-at (SSA) fault model and the required fault coverage refers to stuck-at fault coverage.

2.2.2. Other Structural Fault Models

Although the SSA fault model is widely used both in academia and in industry, it is evident that the SSA fault model does not cover all possible defects. During recent years, several other fault models have gained popularity, such as *bridging faults*, *shorts* and *open faults*. However, these fault models still cannot address all the test issues with CMOS circuits. As a solution to this problem, two technologies have been proposed: Inductive fault analysis (IFA) [145] and inductive contamination analysis (ICA) [101]. 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).

2.2.3. High-Level Fault Models

When test issues are addressed at an abstraction level higher than the traditional gate-level, the first problem that must be addressed is the identification of a suitable high-level fault model. Most of the cited approaches rely on high-level fault models for behavioral HDL descriptions that have been developed by the current practice of software testing [14], and extend them to cope with hardware descriptions. Several authors have proposed alternative fault models. Nevertheless, a reference fault model playing, at the behavioral level, the same role the well-known SSA is playing at the gate level is still missing.

By working on system models that hide the detailed information gate-level netlists capture, the high-level fault models are

CHAPTER 2

not able to precisely foresee the gate-level fault coverage, which is normally used as the reference measure to quantify a circuit's testability. Nevertheless, they can be exploited to rank test sequences according to their testability value. The most common high-level fault models proposed in literature as metrics of the goodness of test sequences when working at higher levels of abstraction (RT level and behavioral level) include the following:

- *Statement coverage*: this is a well-known metric in the software testing field [14] intended to measure the percentage of statements composing a model that are executed by a set of given test patterns. Further improvements of this metric are the *Branch coverage* metric, which measures the percentage of branches of a model that are executed by the given test patterns, and the *Path coverage* metric which measures the percentage of paths that are traversed by the given test patterns, where a path is a sequence of branches that should be traversed for going from the start of the model description to its end.
- *Bit coverage*: in this model [42], [107] it is assumed 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 to the model outputs by a given test sequence.
- *Condition coverage*: the model is proposed in [42] and it is intended to represent 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 to the model outputs by a given test sequence. This model is used in [42] together with bit coverage for estimating the testability of complex circuits.
- *Mutation testing* [31] concentrates on selecting test vectors that are capable to distinguish a program from a set of

TESTING AND DESIGN FOR TESTABILITY

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 “ \times ”, we have to assure that $a + b \neq a \times b$.

All these 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 the relationship between the high-level fault models and the lower level ones. This has been done so far only experimentally (e.g. [90]) and there are no systematic methods currently available.

2.3. Automatic Test Pattern Generation

Digital systems are tested by applying appropriate stimuli and checking the responses. Generation of such stimuli together with calculation of their expected responses is called *test pattern generation*. Test patterns are in practice generated by an *automatic test pattern generation* (ATPG) tool and typically applied to the circuit using *automatic test equipment* (ATE). Due to several limitations of ATE, there exist approaches where the main functions of the external tester have been moved onto the chip. Such DFT practice is generally known as BIST.

With the evolution of test technology, various techniques have been developed for IC testing.

Exhaustive test: The most straightforward approach, where all possible input combinations are generated and applied to the CUT. Exhaustive test set is easy to generate and guarantees 100% fault coverage for combinatorial circuits. However, for an n -input combinatorial circuit the number of possible test vectors is 2^n and therefore this approach is practically infeasible for large

CHAPTER 2

circuits. As an example, it takes approx. 6 centuries to exhaustively test a 32-bit adder at a speed of 1 GHz ($2^{64} \approx 1,84 \times 10^{19}$ test patterns).

Pseudo-exhaustive test: The CUT is divided into smaller parts and every part is tested exhaustively [119]. This type of partitioning results in much smaller number of test vectors, but pseudo-exhaustive testing might still be infeasible with systems that are more complex and the hardware implementation of the pseudo-exhaustive test generator is difficult.

Pseudorandom test: A low-cost IC test solution, where test patterns are generated randomly. The process however is not truly random, as patterns are generated by a deterministic algorithm such that their statistical properties are similar to a randomly selected test set. The advantage of this approach is the ease of pattern generation, as the approach usually does not take into account the function or the structure of the circuit to be tested. The clear disadvantage of pseudorandom testing is the size of the generated test set (it might be several orders of magnitude larger than the same quality deterministic test set). And, due to the size, determining the quality of a test is problematic. Another difficulty is due to the so-called *random-pattern-resistant* or *hard-to-detect* faults that require a different approach than pseudorandom testing [37]. This problem will be discussed in conjunction with BIST later in this chapter.

There are several methods for pseudorandom test pattern generation. It is possible to use a software program, but more widespread methods are based on *linear feedback shift registers* (LFSR). An LFSR has a simple, regular structure and can be used for test pattern generation as well as for output response analysis. LFSRs are frequently used for on-chip test pattern generation in BIST environments and they will be discussed at a greater length later in the thesis.

Deterministic test: Deterministic tests are generated based on a given fault model and the structure of the CUT. This approach

TESTING AND DESIGN FOR TESTABILITY

is sometimes also referred to as *fault-oriented* or *structural* test generation approach. As a first step of the test generation process, the structure of the CUT will be analyzed and a list of all possible faults in the CUT will be generated. Thereafter, the tests are generated using an appropriate test pattern generation algorithm. The typical process of a structural test generation methodology is depicted in Figure 2.2.

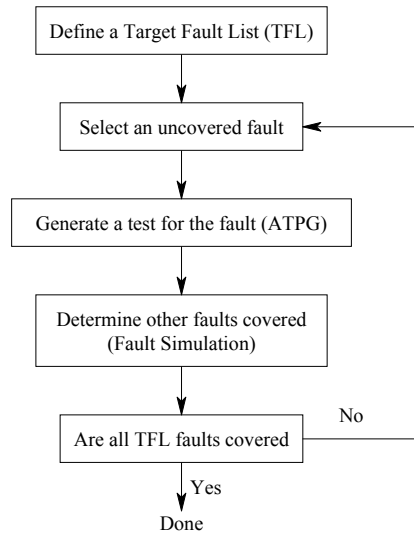


Figure 2.2. Structural test generation.

Deterministic test pattern generation belongs to a class of computationally difficult problems, referred to as NP-complete [80]. Several heuristics have been developed to handle test generation for relatively large combinational circuits in a reasonable time. These include the D-algorithm [139], the path-oriented decision-making (PODEM) algorithm [60], and the fan-oriented test generation (FAN) algorithm [45].

Test generation for sequential circuits is more difficult than for combinational circuits [73], [121]. There exist methods for test pattern generation for relatively small sequential circuits

CHAPTER 2

[27], [131], but for large sequential circuits test generation remains basically an unsolved problem, despite rapid increase of computational power. A possible solution can be found by moving to higher levels of abstraction and using more advanced test generation methods, like hierarchical test generation. Promising results in this domain have been reported in [136].

2.4. Test Generation at Higher Levels of Abstraction

While the design practice is quickly moving toward higher levels of abstraction, test issues are usually considered only when a detailed description of the design is available, typically at the gate level for test sequence generation and at RT-level for design for testability structure insertion.

Recently intensive research efforts have been devoted to devise solutions tackling test sequence generation in the early design phases, mainly at the RT level, and several approaches have been proposed. Most of them are able to generate test patterns of good quality, sometimes comparable or even better than those produced by gate-level ATPG tools. However, lacking general applicability, these approaches are still not widely accepted by the industry. The different approaches are based on different assumptions and on a wide spectrum of distinct algorithmic techniques. Some are based on extracting from a behavioral description the corresponding control machine [125] or the symbolic representation based on binary decision diagrams [41], while others also synthesize a structural description of the data path [40]. Some approaches rely on a direct examination of the HDL description [25], or exploit the knowledge of the gate-level implementation [141]. Some others combine static analysis with simulation [28]. In [97] the applicability of some classical software testing methods for hardware test generation has been investigated with not very encouraging results. The applicability of

a particular software testing technique, mutation testing [31], for hardware testing is discussed in [7], with results that are slightly better than those reported in [97]. However, it has been demonstrated that high-level test pattern generation methodology can successfully be used both for design validation and to enhance the test effectiveness of classic, gate-level test generation [144].

An alternative to these solutions are hierarchical test generation methods. 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 [126], tests generated at the lower level will be assembled at the higher abstraction level. The top-down strategy, introduced in [113], uses information, generated at the higher level, to derive tests for the lower level.

2.5. 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 the situation where testing is performed during normal operational mode and the latter when the circuit is not in normal operation but in so-called test mode. The primary interest of this thesis is off-line testing although some of the results can be applied in an on-line testing environment as well.

CHAPTER 2

2.5.1. Off-line Test Application

Off-line tests can be generated either by the system itself or outside the chip, using an ATPG, and applied by using Automatic Test Equipment (ATE). In Figure 2.3 a generic structure of the ATE is given [130]. It can be divided into 3 main modules: fixture, hardware and software. The module that holds the CUT and provides all necessary connections is usually referred to as a fixture. The fixture is connected to the hardware module that is a computer system with sufficient memory. The testing process is controlled by the tester software that guarantees correct format and timing of the test patterns.

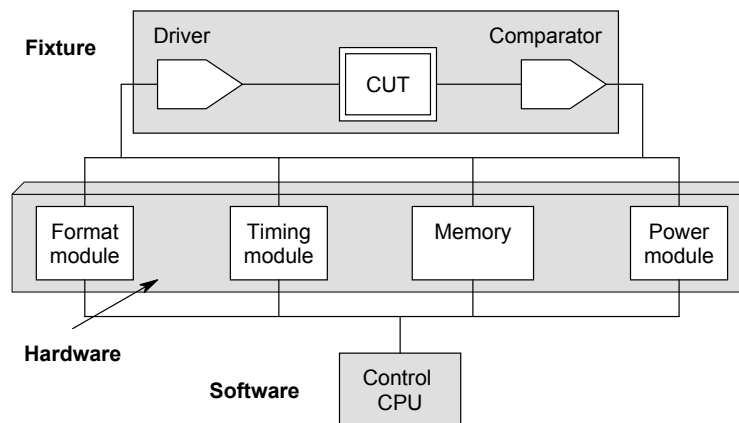


Figure 2.3. Block diagram of ATE.

The ATE memory size defines the amount of test patterns the ATE can apply in one test run, without memory reload. Such reloads are time consuming, thus making them undesired. Therefore, the test set should be devised so that all test patterns fit into the tester memory. However, with increased device density, the volume of test data is becoming increasingly large, thus setting difficult constraints for test engineers.

TESTING AND DESIGN FOR TESTABILITY

With the emerging of sub-micron and deep sub-micron technologies, the ATE approach is becoming increasingly problematic. There are several reasons for that:

- Very expensive test equipment: It is predicted that between 2009 and 2012 ICs will dissipate 100 to 120 W (at 0.6 V), run at frequencies between 3.5 and 10 GHz and have microprocessors with greater than 1 billion transistors. A tester for such a chip will bear 1 400 pins and have a price tag greater than 20 million USD [8], [153].
- Due to the increasing complexity and density of ICs, testing time is continuously increasing and time to market becomes unacceptably long.
- The test sizes and consequently memory requirements for ATEs are continuously increasing.
- The operating frequencies of ATEs should be higher or equal to the frequencies of CUT. This rules out testing cutting edge ICs as the frequency of existing ATEs is always one step behind the latest developments (it takes time until the latest technology reaches the ATE products). This increases inaccuracy of the testing process.

All those reasons have led to the investigation of different alternatives that could make testing of complex ICs more feasible. Several methods have been developed that reduce the significance of external testers and reduce the cost of the testing process, without compromising on quality. One of the alternatives is to partition the test function into on-chip and off-chip resources [74], [110]. By embedding different test activities on-chip makes it possible to use an ATE with significantly reduced requirements. Those methods are in general known as DFT techniques and are described in greater length later in this chapter.

2.5.2. Abort-on-First-Fail Testing

In a production test environment, where a large number of chips have to be tested, an *abort-on-first-fail* (AOFF) approach is usu-

CHAPTER 2

ally utilized. It means that the test process is stopped as soon as a fault is detected. This approach leads to reduced test times and consequently to reduced production costs, as faulty chips can be eliminated before completing the entire test flow. In such a test environment, the likelihood of a block to fail during the test should be considered for test scheduling in order to improve test efficiency [78], [85], [104], [111], [122]. In [104], for example, it was proposed to reduce the average test completion time by applying tests with short test times first. In [78] and [85], it was proposed to use defect probabilities of individual cores for efficient scheduling in an AOFF environment. Such probabilities can be extracted from the statistical analysis of the manufacturing process.

In general, these approaches reduce average test time in large-scale manufacturing test environments. However, it should be noted here, that this approach has especially high significance during the early phases of the production, when the yield is low and the defects are more likely to appear.

2.6. Design for Testability

Test generation and application can be more efficient when testability is already considered and enhanced during the design phase. The generic aim of such an enhancement is to improve controllability and observability with small 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 that 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, that are significant in the context of this thesis, will be discussed.

2.6.1. Scan-Path Insertion

To cope with the problems caused by global feedback and complex sequential circuits, several DFT techniques have been proposed. One of them is *scan-path* insertion [169]. The general idea behind scan-path is to break the feedback paths and to improve the controllability and observability of the sequential elements by introducing an over-laid shift register called scan path (or scan chain). Despite the increase in fault coverage and reduced ATPG complexity, there are some disadvantages with using scan techniques, like increase in silicon area, additional pins, increased power consumption, increase in test application time and decreased clock frequency. We can distinguish two different types of scan-based techniques — *partial scan* and *full scan*, which are illustrated in Figure 2.4.

In case of partial scan (Figure 2.4a), only a subset of the sequential elements will be included in the scan path. This leads to moderate increase in terms of silicon area while requiring more complex ATPG. The full scan approach (Figure 2.4b), in contrast, connects all sequential elements into one or multiple scan chains. The main advantage of this approach is that this reduces the ATPG problem for sequential circuits to the more computationally tractable problem of ATPG for combinatorial circuits.

CHAPTER 2

The scan-path insertion is illustrated in Figure 2.5 [128]. The original circuit is given in Figure 2.5a and the modified circuit with inserted scan-path in Figure 2.5b. Here, in the test mode, all sequential elements will be disconnected from the circuit and configured as a shift register. In large circuits the sequential elements can be divided between multiple scan-paths.

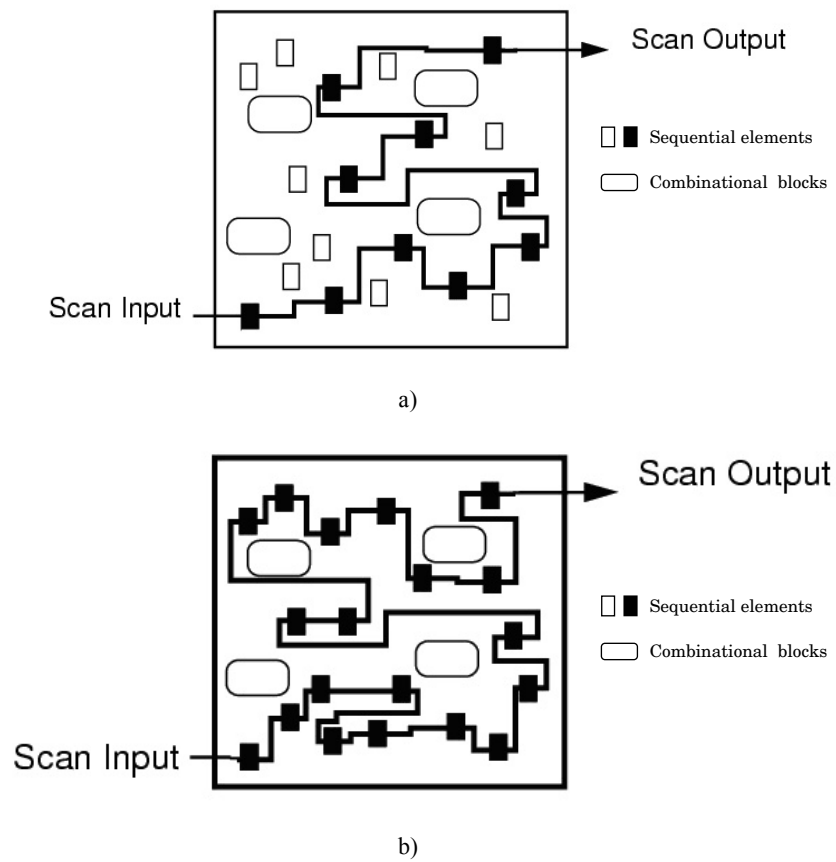


Figure 2.4. a) Partial scan b) Full scan.

TESTING AND DESIGN FOR TESTABILITY

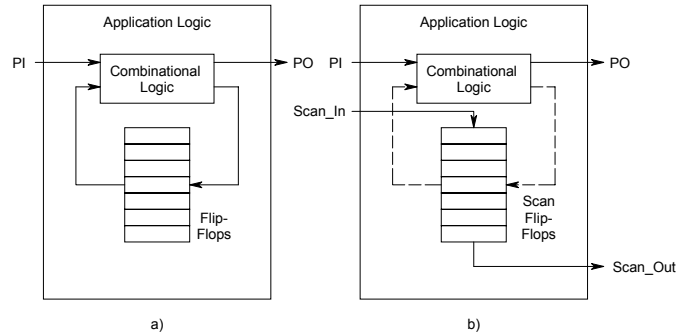


Figure 2.5. a) Original design b) Design with scan-path.

When a design does not contain any scan-paths, test patterns can be applied to the CUT at every clock cycle and the approach is called *test-per-clock*. The introduction of the scan-path requires test pattern application in so-called scan cycles. In such a *test-per-scan* approach, the test patterns are shifted into a scan chain before the pattern at the primary inputs can be applied. Thereafter the test responses are captured in the scan flip-flops and shifted out while a new test is being shifted in. The length of such a cycle is defined by the length of the scan-path and therefore such a test-per-scan approach is much slower than test-per-clock testing. It also makes at-speed testing impossible. The obvious advantage, on the other hand, is the reduced ATPG complexity. It offers also high fault coverage and enables efficient fault diagnosis by providing the direct access to many internal nodes of the CUT.

2.6.2. Built-In Self-Test

The main idea behind a BIST approach is to eliminate or reduce the need for an external tester by integrating active test infrastructure onto the chip. The test patterns are not any more generated externally, as it is done with ATE, but internally, using special BIST circuitry. BIST techniques can be divided into *off-line* and *on-line* techniques. On-line BIST is performed during

CHAPTER 2

normal functional operation of the chip, either when the system is in idle state or not. Off-line BIST is performed when the system is not in its normal operational mode but in special test mode. A prime interest of this thesis is off-line BIST that will be discussed further below. Every further reference to the BIST technique is in the context of off-line BIST.

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 (Figure 2.6). Examples of TPG are a ROM with stored patterns, a counter, or a LFSR. A typical TRA is a comparator with stored responses or an LFSR used as a *signature analyzer*. A BCU is needed to activate the test and analyze the responses. This approach eliminates virtually 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.

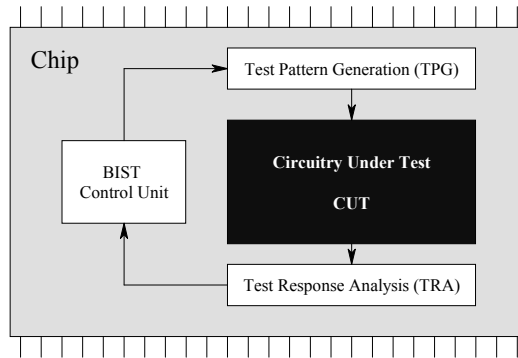


Figure 2.6. A typical BIST architecture.

In the following, the basic principles of BIST will be discussed. We are going to describe test pattern generation with LFSRs,

problems related to such an approach and describe some more known BIST architectures.

Test Pattern Generation with LFSRs

Typical BIST schemes rely on either exhaustive, pseudorandom testing, or pseudorandom testing and the most relevant approaches use LFSRs for test pattern generation [5], [12], [172]. This is mainly due to the simple and fairly regular structure of the LFSR. Although the LFSR generated tests are much longer than deterministic tests, they are much easier to generate and have good pseudorandom properties.

In Figure 2.7 a generic structure of the n -stage *standard* LFSR (also known as type 1 LFSR or *external-XOR* LFSR) and in Figure 2.8 a generic structure of the n -stage *modular* LFSR (also known as type 2 LFSR or *internal-XOR* LFSR) is given. An LFSR is a shift register, composed from memory elements (latches or flip-flops) and exclusive OR (XOR) gates, with feedback from different stages. It is fully autonomous, i.e. it does not have any input beside the clock. C_i in Figure 2.7 and Figure 2.8 denotes a binary constant and if $C_i = 1$ then there is a feedback from/to the i th D flip-flop; otherwise, the output of this flip-flop is not tapped and the corresponding XOR gate can be removed. The outputs of the flip-flops (Y_1, Y_2, \dots, Y_N) form the test pattern. The number of unique test patterns is equal to the number of states of the circuit, which is determined, by the number and locations of the individual feedback tabs. The configuration of the feedback tabs can be expressed with a polynomial, called *characteristic* or *feedback polynomial*. For an LFSR in Figure 2.8 the characteristic polynomial $P(x)$ is

$$P(x) = 1 + c_1x + c_2x^2 + \dots + c_nx^n.$$

An LFSR goes through a cyclic or periodic sequence of states and produces periodic output. The maximum length of this period is $2^n - 1$, where n is the number of stages, and the characteristic polynomials that cause an LFSR to generate maximum-length

CHAPTER 2

sequences are called *primitive polynomials* [62]. A necessary condition for a polynomial to be primitive is that the polynomial is irreducible, i.e. it cannot be factored.

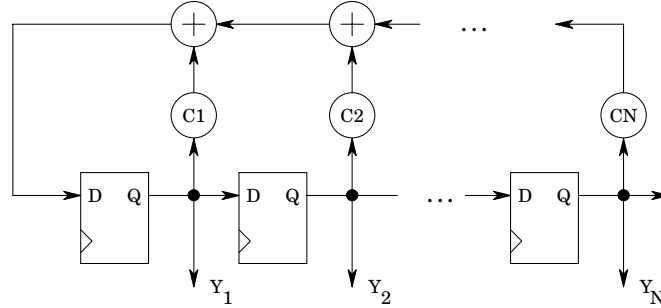


Figure 2.7. Generic standard LFSR.

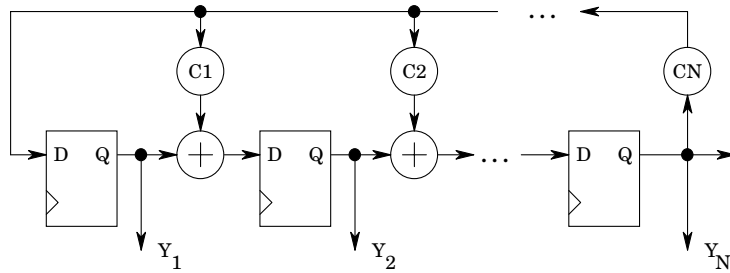


Figure 2.8. Generic modular LFSR.

The test vectors generated by an LFSR appear to be randomly ordered. They satisfy most of the properties of random numbers even though we can predict them deterministically from the LFSR's present state and its characteristic polynomial. Thus, these vectors are called *pseudorandom* vectors and such LFSRs can be called *pseudorandom pattern generator* (PRPG).

TESTING AND DESIGN FOR TESTABILITY

Test Response Analysis with LFSRs.

As with any other testing method, also with BIST, the response of the circuit has to be evaluated. This requires knowledge about the behavior of the fault-free CUT. For a given test sequence this can be obtained by simulating the known-good CUT. It is however infeasible to compare all response values on chip, as the number of test patterns in a test sequence can be impractically long. Therefore a better solution is to compact the responses of a CUT into a relatively short binary sequence, called a *signature*. Comparison of faulty and fault-free signatures can reveal the presence of faults. As such a compaction is not lossless, the signatures of faulty and fault-free CUT can be the same, although the response sequences of the two are different. This is called *aliasing*. The compression can be performed in two dimensions: time and space. *Time compression* compresses long sequences to a shorter signature and *space compression* reduces a large number of outputs to a smaller number of signals to be observed.

There are several compaction testing techniques, like parity testing, one counting, transition counting, syndrome calculation and signature analysis. In the following one of the most common techniques — *signature analysis* — is briefly described.

Signature analysis is a compression technique based on the concept of cyclic redundancy checking and implemented in hardware using LFSRs [46]. The responses are fed into the LFSR and at the end of the test application, the content of the LFSR is used as a signature. The simplest form of signature analysis is based on serial-input signature register. This kind of “serial” signature analysis, based on SLFSR, is illustrated in Figure 2.9. Here the LFSR is modified to accept an external input in order to act as a polynomial divider.

CHAPTER 2

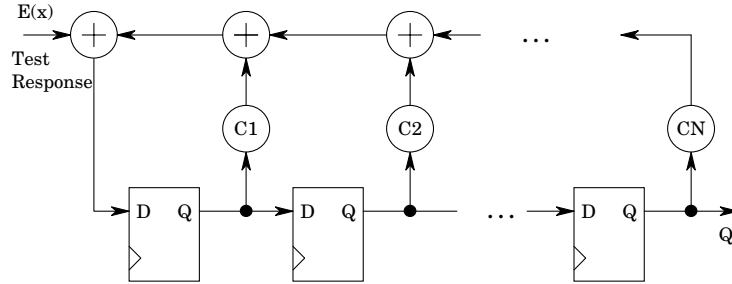


Figure 2.9. SLFSR based signature analysis

An extension of the serial-input signature register is the *multiple-input signature register* (MISR), where output signals are connected to the LFSR in parallel. There are several ways to connect the inputs (CUT outputs) to both types (standard and modular) of LFSRs to form an MISR. One of the possible alternatives is depicted in Figure 2.10. Here a number of XOR gates are added to the flip-flops. The CUT outputs are then connected to these gates.

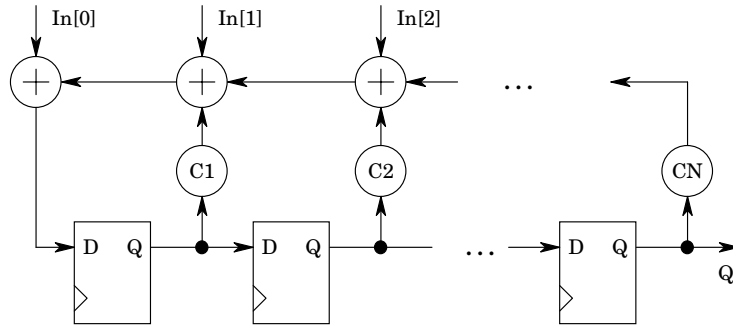


Figure 2.10. Multiple-input signature register.

Classification of BIST Architectures

BIST Architectures can be divided, based on test application methods, into two main categories: *parallel BIST* (a.k.a. *in-situ*

TESTING AND DESIGN FOR TESTABILITY

BIST) and *serial BIST* (a.k.a. *scan BIST*). A parallel BIST scheme uses special registers, which work in four modes. In the system mode they operate just as D-type flip-flops. In the pattern generation mode they perform autonomous state transitions, and the states are the test patterns. In the response evaluation mode the responses of the CUT are compressed, and in the shift mode the registers work as a scan path. In this approach, one test pattern is applied at every clock cycle. Hence, such architectures are called test-per-clock BIST architectures. Examples of such architectures are built-in logic block observer (BILBO) and circular self-test path (CSTP). In contrast, serial BIST architectures assume that test patterns are applied via the scan chain. Such test-per-scan approach requires $SCL+1$ clock cycles to shift in and to apply a test pattern and the same amount of clock cycles to shift out the test response, where SCL is the length of the longest scan chain, making it thus much slower than the test-per-clock approach. Although slower, this approach has several advantages, similar to the general scan-path based testing:

- It takes advantage of the traditional scan-path design, making it thus compatible with any commercial tool flow that supports scan chains, and requires a very small amount of additional design modifications.
- It can be implemented at the chip level even when the chip design uses modules that do not have any BIST circuitry, provided that they have been made testable using scan.
- Due to the scan path it requires simpler ATPG and has improved observability.
- Its overall hardware overhead is smaller than in test-per-clock architectures, as it requires simpler test pattern generators for pseudorandom testing.
- In most cases, the BIST control of a test-per-scan scheme is simpler than the BIST control of a test-per-clock scheme.

The main advantage of parallel BIST is that it supports testing at the normal clock rate of the circuit, i.e., *at speed testing*.

CHAPTER 2

This enables detection of faults that appear only at normal operational speed, such as transient faults in the power/ground lines caused by the switching of circuit lines. With a test-per-clock approach also a larger number of test patterns can be applied in a given test time, consequently a higher number of random pattern resistant faults could be detected. Therefore, test-per-scan architectures might require more complex TPGs, thus eliminating any advantage of the area overhead of serial BIST.

In the following, several BIST architectures will be described. We will describe architectures based on both paradigms, test-per-clock or *parallel BIST* and test-per-scan or *serial BIST*. Additional BIST architectures can be found in [3] and [12].

Parallel BIST Architectures

One of the first parallel BIST architectures was *built-in evaluation and self-test* (BEST). It is a simple architecture, where CUT inputs are driven by the PRPG and test responses are captured by a MISR, similar to Figure 2.6. This approach requires extensive fault simulation to determine an acceptable balance between fault coverage and test length, and might be ineffective for some circuits.

More widespread are *built-in logic block observer* (BILBO) and *circular self-test path* (CSTP) architectures. A BILBO register is a register that can operate both as a test pattern generator and as a signature analyzer [106] (Figure 2.11). In the test mode, the BILBO is configured as an LFSR or a MISR (Figure 2.12). A simple form of BILBO BIST architecture consists of partitioning a circuit into a set of registers and blocks of combinational logic, where the normal registers are replaced by BILBO registers.

TESTING AND DESIGN FOR TESTABILITY

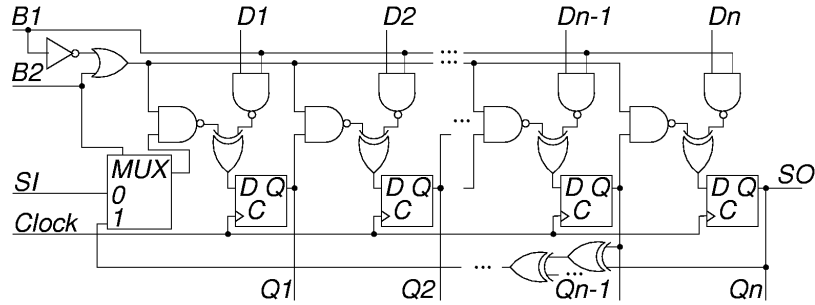


Figure 2.11. n-bit BILBO register.

The synthesis of a test-per-clock scheme is implemented in the easiest way by a circular BIST or a circular self-test path (CSTP) [105] (Figure 2.13). The scheme has two modes, the system mode and the test mode, where the flip-flops form the LFSR. Two arbitrary flip-flops may be the scan-in and scan-out inputs. In the test mode, the system performs signature analysis and pattern generation concurrently, and only a single control line is required for the basic cells of this scheme. The disadvantage of this scheme is low fault coverage for some circuits.

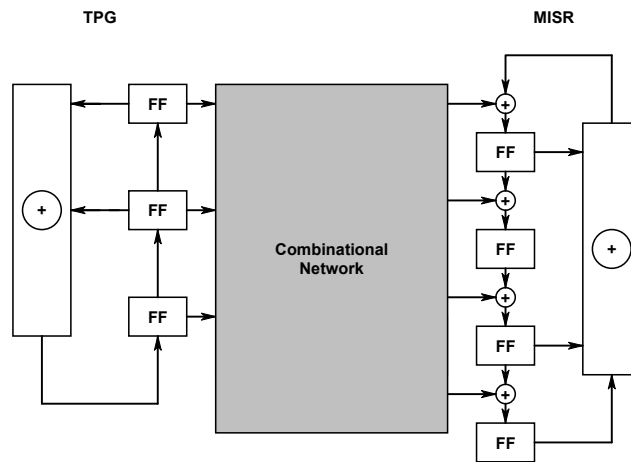


Figure 2.12. BIST Design with BILBO registers.

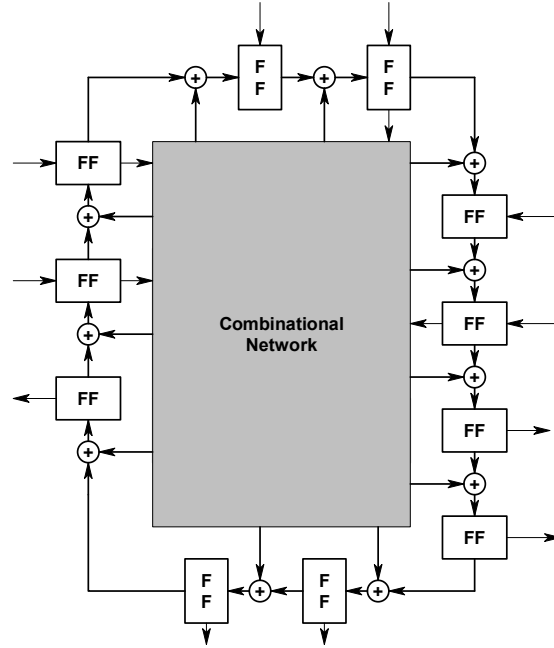


Figure 2.13. Circular self-test path.

Serial BIST Architectures

We describe two main types of scan-based BIST architectures. In the first type, all primary inputs are connected to the taps of a pseudorandom generator and all primary outputs are connected to a MISR. All or a subset of state flip-flops are configured as scan chains, while the primary input flip-flop (scan-in signal) is connected to another LFSR and the primary output flip-flop (scan-out signal) is connected to a SLFSR. Examples of this architecture are *random test socket* (RTS) [11] and *partial scan BIST* (PS-BIST) [116].

More efficient DFT approaches are modifications of these methodologies, such as *Self-Test Using MISR and Parallel SRSG* (STUMPS) [11] and *LSSD on-chip self-test* (LOCST) [112]. The

TESTING AND DESIGN FOR TESTABILITY

acronym SRSG (Shift Register Sequence Generator) may be considered as equivalent to PRPG, mentioned above.

The STUMPS architecture is shown in Figure 2.14. The basic assumption is that the memory elements of the CUT are included into the scan path. Often, a scan path is split into several scan chains. The multiplicity of scan chains speeds up test application, because the length of one test cycle is determined by the length of the scan path. At the same time, it equals only to the length of the longest scan chain for a CUT with multiple scan chains. However, there is always a trade-off: the more scan chains a core has the more scan inputs are required for it and thus longer LFSRs are needed.

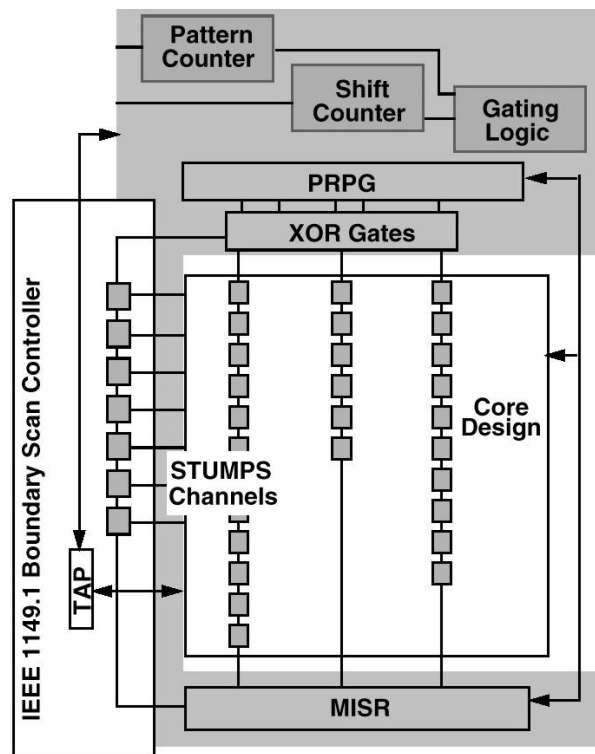


Figure 2.14. Self-Test Using MISR and Parallel SRSG
(© Mentor Graphics).

CHAPTER 2

The general idea of the STUMPS approach is the following. PRPG, MISR and scan registers are clocked simultaneously. All scan registers are loaded from PRPG. This takes SCL clock cycles, where SCL is the length of the longest scan chain. After test application, the data captured by scan registers is scanned out, and the results are analyzed by a MISR.

The sequences obtained from adjacent bits of a parallel LFSR are not linearly independent; the neighboring scan chains contain test patterns that are highly correlated [24]. This can affect fault coverage negatively since the patterns seen by the CUT do not really form a random sequence.

2.7. Emerging Problems in System-on-Chip Testing

The latest advances in microelectronics manufacturing technology have enabled integration of an increasingly large number of transistors. This shift toward very deep submicron technologies facilitates implementation of an entire system on a single chip. Such systems are usually composed from a large number of different functional blocks, usually referred as cores. This kind of design style allows designers to reuse previous designs, which will lead therefore to shorter time-to-market, and reduced cost. Such a *system-on-chip* (SOC) approach is very attractive from the designers' perspective. Testing of such systems, on the other hand, is problematic and time consuming, mainly due to the resulting IC's complexity and the high integration density [127].

A typical SOC consists of many complex blocks (embedded cores, RAM, ROM, user-defined logic (UDL), analog blocks etc.) as depicted in Figure 2.15. Until recently such designs were implemented using several ICs mounted together into a printed circuit board (PCB) (also called as systems on board (SOB)). Using the SOC's instead of the PCBs gives a possibility to produce chips

TESTING AND DESIGN FOR TESTABILITY

with better performance, lower power consumption and smaller geometrical dimensions [175].

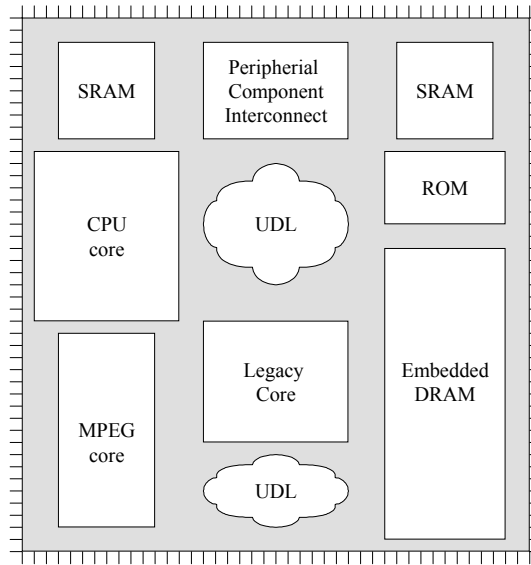


Figure 2.15. System-on-chip.

To cope with the difficulties to build such complex chips, a new design paradigm, called design reuse, has emerged. The main idea here is to use predefined and preverified reusable blocks to build up the chip. These reusable building blocks, so-called embedded cores, have several advantages. For example, in addition to reuse, embedded cores enable also import of an external design expertise, like high-end CPUs and DRAMs [118].

Embedded cores can be divided into three categories: soft, firm and hard [65]:

- *Soft cores*
 - May be available as synthesizable, technology-independent, HDL descriptions (RTL or higher).
 - Do not provide any protection for intellectual property.

CHAPTER 2

- Are flexible and process independent, thus allow modifications, such as optimization to the desired levels of performance or area, and DFT insertion.
- Firm cores
 - Are usually available as netlist of library cells (usually optimized for a target technology).
 - There is still a possibility to make some design modifications.
- Hard cores
 - Are optimized for area and performance. Mapped into a specific technology. Only functional specification together with a layout is available (The core will be treated as a “black box”).
 - Provide maximum protection of intellectual property.
 - No possibility to make any design modifications.

Design reuse based techniques are advancing very rapidly, but there are still several unsolved problems. The most critical ones include manufacturing test and design debug [176].

There are two separate issues in the manufacturing testing: go/no-go testing at the end of the manufacturing line and defect analysis during the diagnosis. As the complexity, performance and density of the ICs increase, the test community has to find effective methods to cope with the growing problems and challenges.

The core-based design process has several similarities with the traditional SOB design process, but the manufacturing test process is conceptually different (Figure 2.16).

In case of the SOB, all building blocks (ICs) are manufactured and tested before assembly, therefore assumed fault-free. During the system integration, the elements will be mounted and the board will be tested. As components are assumed to be fault-free then only the interconnects between the ICs should be tested. To

TESTING AND DESIGN FOR TESTABILITY

solve this problem the IEEE 1149.1 (also referred as JTAG or boundary-scan) standard has been developed [81].

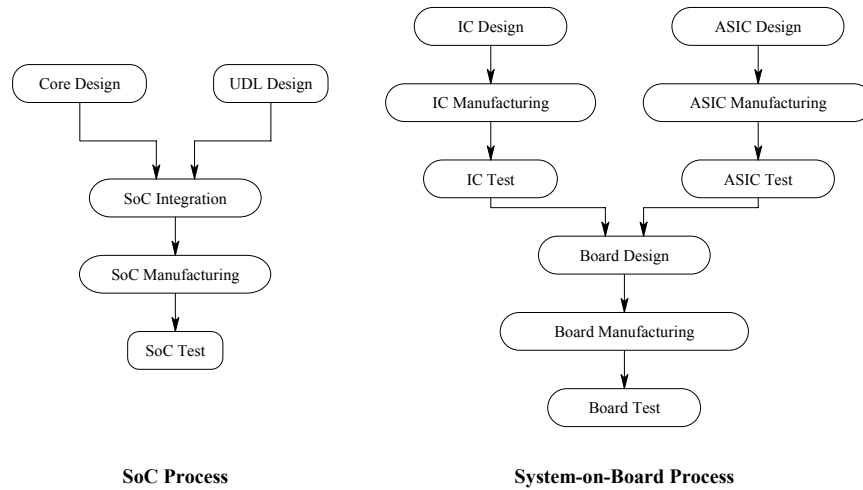


Figure 2.16. SOC versus SOB design development.

Cores are distributed from the core providers to the core users in form of a module description. There is no manufacturing testing done by the core provider, because he/she has only the functional description of the core and nothing is manufactured yet. The manufacturing test can be done only after the core is embedded into a system and finally manufactured. Therefore the system integrator has to deal not only with interconnect testing but also with core testing. In most of the cases the system integrator sees a core as a black box (this is especially true in case of the hard or encrypted cores) and has very little knowledge about the structural content of the core or does not have it at all. Therefore, the core tests should be developed by the core provider and provided together with the core. This task is even more complicated, because the core provider usually does not know anything about the final environment where the core will be implemented. This situation means that there is more than one

CHAPTER 2

person, who deals with test development. The test development process is distributed between different groups or even companies, and there are additional challenges due to the test knowledge transfer.

2.7.1. Core Internal Test Knowledge Transfer

In order to perform core test two key elements are needed: the set of test patterns and the internal design-for-testability (DFT) structures. This information should be transferred from the core provider to the core user together with validation of provided test patterns. Additionally, information about the test modes and corresponding test protocols, information about diagnosis and silicon debug should be included. All this information should be adequately described, ported and ready for plug and play. Thereby in addition to the design reuse, we should talk also about a test reuse. However, to support properly such activity some commonly used format (i.e. standard) for core's test knowledge transfer should be used. Such standard is currently under development by IEEE and referred to as *P1450.6 Core Test Language* (CTL) [26].

2.7.2. Core Test Access Challenges

Typically testing is done by applying the test vectors to the input pins of the IC and observing the output. This is used for PCBs, as physical access to the IC pins is easily achievable. A core is usually embedded deep into the system IC and direct physical access to its periphery is not available. Therefore, the system integrator should provide an infrastructure between core terminals and the IC pins to fulfill the core's test requirements and device the mechanisms for testing the user-defined logic (UDL), surrounding the core.

According to [118] the test access to the cores has two aspects. First, the core user has to ensure the existence of the access path in the on-chip hardware and secondly, the core tests, given by

the core provider, have to be transported from the core terminals to the IC pins.

2.7.3. Chip-level Test Challenges

One of the major test challenges for the core users is the integration and coordination of the on-chip tests. This composite test should cover cores, UDL and interconnects, and requires adequate test scheduling. Test scheduling is necessary to meet the requirements for the test application time, test power dissipation, area overhead, resource conflict avoidance, and so on [174]. Test scheduling will be discussed in greater length later in the thesis.

At the same time we should not forget, that SOC's are manufactured using very deep submicron technologies and therefore share all testing challenges of such chips, such as defect/fault coverage, overall test cost and time-to-market.

2.7.4. Core Test Architecture

Large SOC's are usually tested modularly, i.e. the various modules are tested as stand-alone units [61]. The main reasons behind that are the opportunities for test reuse and possibility to divide test problems into smaller sub-problems ("divide-and-conquer"). To Facilitate modular test of SOC's, the following components are required (Figure 2.17):

- *Test pattern source and sink* are used for test stimuli generation and response analysis.
- *Test access mechanism* is for test data transportation.
- *Core test wrapper* forms the interface between the embedded core and the environment.

All three elements can be implemented in various ways. Some basic principles will be given below.

CHAPTER 2

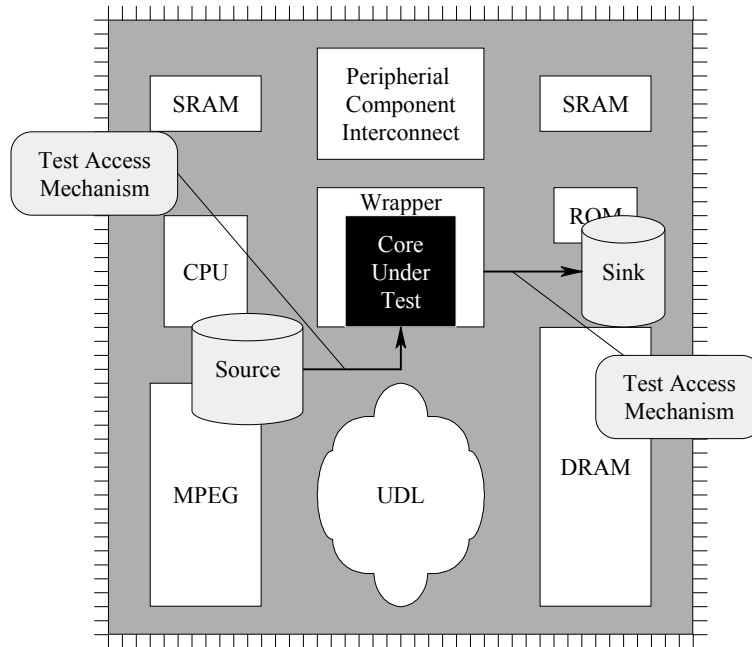


Figure 2.17. SOC test architecture.

Test Pattern Source and Sink

The test pattern source is for test pattern generation, whereas the test pattern sink compares the received response with the expected response. There are several ways to implement the test pattern source and sink. One possibility is to implement both on-chip by built-in self-test mechanisms, as depicted in Figure 2.17. Other possibilities include off-chip source/sink or combination of the previous ones. In case of the off-chip source/sink, external ATE is needed. On the other hand, on-chip test structures occupy additional silicon area, so there is no clear answer what kind of implementation to choose. The final implementation decision is influenced by three different factors: type of circuitry in the core, predefined test set provided by the core provider and requirements for test time, quality and cost.

TESTING AND DESIGN FOR TESTABILITY

Test Access Mechanism

The test access mechanism (TAM) is used to transfer the test data from the test pattern source to the core under test and from the core under test to the test pattern sink. It is always implemented on-chip. The key parameters of the TAM are the width and the length. A wider TAM provides faster data transfer rate but requires more wiring area and is therefore more expensive. The actual width of the TAM depends on the type of the source/sink and requirements to the time and cost.

Several alternatives for TAM implementation have been proposed. For example, the TAM can be based on existing chip infrastructure or be formed by dedicated test access hardware. A test access mechanism can either go through other cores or pass around them. It can be shared across multiple cores or every core can have independent access mechanism. A TAM might contain some intelligent test control functions or may just transport the signals. Well-known TAM techniques are based on Macro Test, transparency of cores, reusing the system bus, multiplexed access, a dedicated test bus, Boundary Scan, test scheduler, TestRail and Advanced Microcontroller Bus Architecture (AMBA). Detailed description of these methods and appropriate references can be found, among others, in [96].

Core Test Wrapper

The core test wrapper is the communication mechanism between the embedded core and the rest of the chip. It connects the core terminals to the test access mechanism and it should provide core test data access and core test isolation. The wrapper should contain three mandatory modes of operation:

- *Normal operation.* In this mode, the core is connected to its environment and the wrapper is transparent.
- *Core test mode (core internal test),* in which the core is connected to the TAM and therefore the test stimuli can be ap-

CHAPTER 2

plied to the core's inputs and the responses can be observed at the core's outputs.

- *Interconnect test mode (core external test)*, in which the TAM is connected to the UDL and interconnect wires. In this mode test stimuli can be applied to the core's outputs and responses can be observed at the next core's inputs

A core test wrapper should also provide core isolation (a.k.a. bypass mode), when needed (for example in case of testing neighboring cores or UDL).

A considerable amount of research has been done in the core test wrapper area. Examples of wrappers are TestShell [117], and a very similar wrapper called TestCollar [165]. The IEEE P1500 Standard for Embedded Core Test standardizes a core test wrapper [30], [132], that is very similar to the TestShell and TestCollar (Figure 2.18).

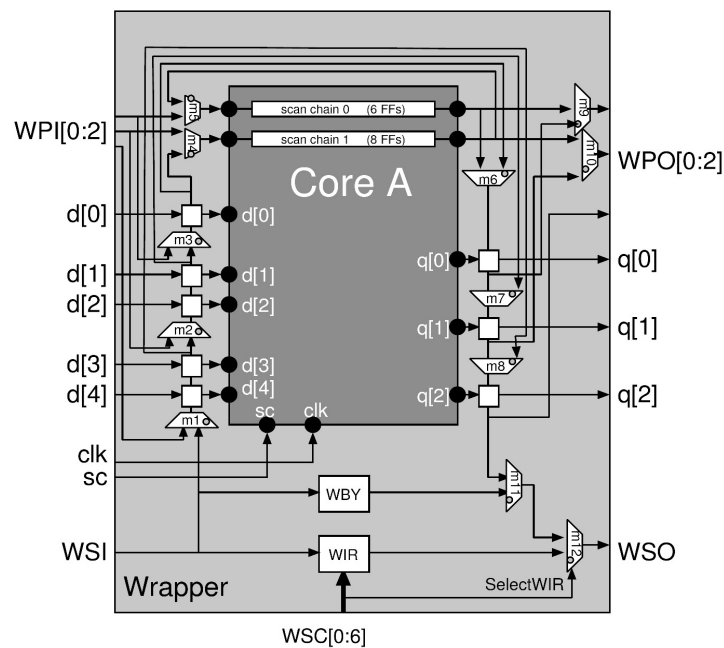


Figure 2.18. An example of an IEEE P1500 Core Test Wrapper.

2.7.5. Power Dissipation

The integration of an increasingly large number of transistors into a single die has imposed a major production challenge, due to the increased density of such chips, reduced feature sizes, and consequently, increased power dissipation. At the same time the number of portable, battery operated devices (such as laptops, PDA-s, mobile phones) is rapidly increasing. These devices require advanced methods for reducing power consumption in order to prolong the life of the batteries and thus increase the length of the operating periods of the system. There are several well-investigated techniques for handling power dissipation during the normal operation. At the same time, various researches have shown that the switching activity, and consequently the power dissipation, during the testing phase may be several times higher than in normal operation mode [32], [174]. This increased switching activity causes increased heat dissipation and may therefore reduce reliability of the circuits, affect overall yield, and increase production cost. The self-tests, regularly executed in portable devices, can hence consume significant amounts of energy and consequently reduce the lifetime of the batteries [52]. Excessive switching activity during the test mode can also cause problems with circuit reliability [54]. And the increased current levels can lead to serious silicon failure mechanisms (such as electromigration [115]) and may need expensive packages for removal of the excessive heat. Therefore, it is important to find ways for handling circuit power dissipation during the testing.

There are different components contributing to the power consumption in case of standard CMOS technology: dynamic power dissipation caused by the switching activity, and static power dissipation caused mainly by leakage. The leaks contribute usually only marginally to the total power consumption and can therefore be neglected [133]. The main contributing factor is dynamic power dissipation caused by switching of the gate outputs. This activity accounts for more than 90% of the total power dis-

CHAPTER 2

sipation for current technology, even though the importance of static power dissipation will increase with the scaling down of feature sizes [22]. For every gate the dynamic power, P_d , required to charge and discharge the circuit nodes can be calculated as follows [33], [129]:

$$P_d = 0.5 \times C_{load} \times (V_{DD}^2 / T_{cyc}) \times N_G \quad (2.1)$$

where C_{load} is the load capacitance, V_{DD} is the supply voltage, T_{cyc} is the global clock period, and N_G is the switching activity, i.e., the number of gate output transitions per clock cycle.

While assuming that the V_{DD} as well as T_{cyc} remain constant during the test and that the load capacitance for each gate is equal to the number of fan-outs of this gate, we can define switching activity as a quantitative measure for power dissipation. Therefore, the most straightforward way to reduce the dynamic power dissipation of the circuit during test is to minimize the circuit's switching activity.

Several approaches have been proposed to handle the power issues during test application. They can be divided into three categories: energy, average power and peak power reduction techniques. Energy reduction techniques are targeting reduction of the total switching activity generated in the circuit during the test application and have thus impact on the battery lifetime [52], [53], [55], [56]. Average power dissipation is the amount of dissipated energy divided over the test time. The reduction of average power dissipation can improve the circuit's reliability by reducing temperature and current density. Some of the methods to reduce average power dissipation have been proposed in [21], [166]. The peak power corresponds to the maximum sustained power in a circuit. The peak power determines the thermal and electrical limits of components and the system packaging requirements. If the peak power exceeds certain limits, the correct functioning of the entire circuit is no longer guaranteed. The

methods for peak power reduction include those described in [13], [49], [138], [143], [168].

In a System-on-Chip testing environment, several test power related problems are handled at the core level, with methods described above. However, the high degree of parallelism in SOC's facilitates parallel testing to reduce the test application time. Consequently, this might also lead to excessive power dissipation. In such cases, the system-wide peak power values can be limited with intelligent test scheduling. It has been shown in [20] that test scheduling is equal to the open-shop scheduling problem, which is known to be NP-complete. Therefore, numerous heuristics have been developed, including those reported in [29], [77] and [83]. In [109] it was demonstrated that test scheduling can be successfully included into the generic SOC testing framework, where problems like test scheduling, test access mechanism design, test sets selection, and test resource placement, are considered simultaneously.

2.8. Conclusions

In this chapter the basic concepts of digital hardware testing were presented. We gave an overview of several emerging problems in the area and described problems with existing test methods.

The following two parts of the thesis will present the main contributions of our work. In the next section a novel hybrid BIST technique together with its cost optimization methods will be described. This will be followed by description of hybrid BIST time minimization techniques for different SOC test architectures. This thesis addresses also problems related to the power dissipation. We will propose a method for total switching energy minimization in our proposed hybrid BIST environment and heuristic for intelligent scheduling of hybrid test sequences in an abort-on-first-fail-environment.

CHAPTER 2

Third part of the thesis will concentrate on test generation methods in early stages of the design flow. We will propose a novel hierarchical test generation algorithm for generating test sequences when only limited information about the final implementation is available. We take into account information from different levels of abstraction and are therefore able to generate test sequences with higher fault coverage than those of a pure behavioral test generator.

PART II

HYBRID

BUILT-IN SELF-TEST

Chapter 3

Introduction and Related Work

The second part of this thesis focuses on a novel self-test approach called hybrid BIST. As it was mentioned in the introductory part, the classical BIST approach has several shortcomings in terms of test time and test quality, to mention a few. Therefore, we have worked with a method that tries to address these problems. Our hybrid BIST approach guarantees the highest possible test quality, while providing a possibility for trade-off between different parameters, such as test length, test memory requirements and others. It should be noted that the main contributions of this part are not related to the test generation nor to the test architectures, instead the main contribution is a set of optimization algorithms that can be used in conjunction with different hybrid BIST architectures.

In this chapter we are going to discuss some shortcomings of the classical, pseudorandom testing based BIST and we will describe different methods that have been devised in order to

CHAPTER 3

tackle these problems. The following chapter introduces the basic concept of the hybrid BIST technique, which is followed by a description of different hybrid BIST architectures. Chapter 5 presents different algorithms for hybrid BIST cost minimization for single core designs. In Chapter 6 methods for hybrid BIST time minimization, based on different architectural assumptions, will be described. Chapter 7 focuses on hybrid BIST energy minimization problems, and in Chapter 8 hybrid BIST time minimization in an abort-on-first-fail context will be presented.

3.1. Introduction

Typically, a SOC consists of microprocessor cores, digital logic blocks, analog devices, and memory structures. These different types of components were traditionally tested as separate chips by dedicated automatic test equipments of different types. Now they must be tested all together as a single chip either by a super tester, which is capable of handling the different types of cores and is very expensive, or by multiple testers, which is very time-consuming due to the additional time needed for moving from one tester to another.

Complexity of SOC testing can be reduced by introducing appropriate DFT mechanisms. At a core level, this task is usually accomplished by the core developer. Since the core developer has no idea about the overall SOC design and test strategy to be used, the inserted DFT mechanism may not be compatible with the overall design and test philosophy, leading to low test quality or high overhead. This problem needs to be solved in order to guarantee the high quality level of SOC products.

SOC testing requires also test access mechanisms to connect the core peripheries to the test sources and sinks, which are the SOC pins when testing by an external tester is assumed. The design of the test access mechanism must be considered together with the test-scheduling problem, in order to reduce the silicon

INTRODUCTION AND RELATED WORK

area used for test access and to minimize the total test application time, which includes the time to test the individual cores and user-defined logic as well as the time to test their interconnections. The issue of power dissipation in test mode should also be considered in order to prevent the chip being damaged by over-heating during test.

Many of the testing problems discussed above can be overcome by using a built-in self-test (BIST) strategy. For example, the test access cost can be substantially reduced by putting the test sources and sinks next to the cores to be tested. BIST can also be used to deal with the discrepancy between the speed of the SOC, which is increasing rapidly, and that of the tester, which will soon be too slow to match typical SOC clock frequencies. The introduction of BIST mechanisms in a SOC may also improve the diagnosis ability and field-test capability, which are essential for many applications where regular operation and maintenance test is needed.

Since the introduction of BIST mechanisms into a SOC is a complex task, we need to develop powerful automated design methods and tools to optimize the test function together with the other design criteria as well as to speed up the design process. However, the classical BIST approach has several shortcomings, as discussed below, and therefore, several methods have been developed for its improvement, which will be presented briefly in this chapter

3.2. Problems with Classical BIST

As described earlier, a classical 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. Different implementations of such BIST architectures have been available, and some of them have wide acceptance. Unfortunately, the classical BIST approaches suffer the problems of in-

CHAPTER 3

ducing additional delay to the circuitry and requiring a relatively long test application time.

In particular, one major problem with the classical BIST implementation is due to that the TPG is implemented by linear feedback shift registers (LFSR). The effectiveness of such TPG for a given circuit depends on the appropriate choice of the LFSRs as well as their length and configuration. The test patterns generated by an LFSR are pseudorandom by nature and have linear dependencies [62]. Such test patterns often do not guarantee a sufficiently high fault coverage (especially in the case of large and complex designs), and demand very long test application times. It is not uncommon to have a pseudorandom test sequence that is more than 10 times longer than the deterministic test sequence with similar efficiency [96]. The main reason behind this phenomenon is the presence of *random pattern resistant (RPR) faults* in the circuit under test. The RPR faults are the ones that are detected by very few test patterns, if not by only one. If this pattern is not in the generated pseudorandom test sequence, the fault will remain undetected.

In order to illustrate random pattern resistant faults let us use a simple 16-input AND-gate, depicted in Figure 3.1. The stuck-at-0 fault at the output of this gate is a good example of such faults. In order to detect this fault, all inputs of the gate must be set to 1 (this is the only test pattern that can activate this fault), and if uniformly distributed pseudorandom patterns are applied, the detection probability of this fault is 2^{-16} . This obviously leads to unacceptable test lengths.

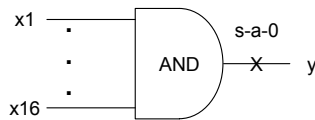


Figure 3.1. An example of a random pattern resistant fault.

Generally, pseudorandom test patterns can seldomly achieve 100% fault coverage. Figure 3.2 shows the fault coverage of

INTRODUCTION AND RELATED WORK

pseudorandom tests as a function of the test length, for some larger ISCAS'85 [18] 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 detect very few new faults, if any. Moreover, there may exist many faults that will never be detected with pseudorandom test vectors.

Therefore, several questions have to be answered while developing a LFSR-based self-test solution: What is the fault coverage achievable with pseudorandom patterns, compared to that of deterministic test methods? Will the required fault coverage be achieved by the number of pseudorandom patterns that can be generated in some acceptable interval of time? What are the characteristics of the LFSR that produce a test sequence with acceptable fault coverage? Such an analysis shows that in most cases a pseudorandom test leads to either unacceptably long test sequences or fault coverage figures that are not acceptable and much below those achievable by deterministic test sequences.

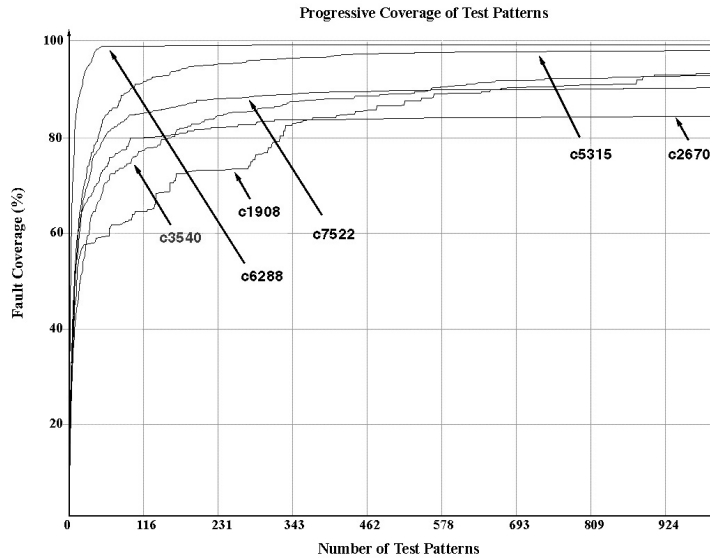


Figure 3.2. Pseudorandom test for some ISCAS'85 circuits.

CHAPTER 3

Therefore, several proposals have been made to combine pseudorandom test patterns, generated by LFSRs, with deterministic patterns, to form a hybrid BIST solution [6], [23], [69], [70], [71], [103], [154], [155], [173], [171]. The main concern of several such hybrid BIST approaches has been to improve the fault coverage by mixing pseudorandom vectors with deterministic ones, while the issue of test cost minimization has not been addressed directly. In the following sections, different classical BIST improvement techniques, including the hybrid BIST approaches, will be described.

3.3. BIST Improvement Techniques

The length of a test session is usually limited. If the fault coverage figure, after applying the specified number of test patterns, remains below the desired levels, some modifications to the test strategy and/or to the circuit under test have to be made. There are two alternatives. The first alternative is to improve the controllability and observability of the circuit, thus improving the detectability of hard-to-detect faults, for example, via *test point insertion*. Another possibility is to modify the TPG in order to generate test patterns that are more suitable for the given CUT. These two alternatives will be discussed in the following sections.

3.3.1. Test Point Insertion

Test point insertion is a DFT technique that is very widely used in commercial BIST tools. It can theoretically guarantee any level of fault coverage, provided a sufficient number of test points are used. The possible drawbacks are the area overhead and performance penalty. The area overhead is introduced by the additional logic and routing needed to introduce the test points. Performance degradation might come from the increased delays, if time-critical paths are affected.

INTRODUCTION AND RELATED WORK

There are two types of test points: *control points* and *observation points*. Control points are added to help control the value at a line and there are several types of them. Two of the most common ones are depicted in Figure 3.3:

- A *zero control point* can be obtained by adding an additional primary input together with an AND-gate (Figure 3.3c).
- A *one control point* can be obtained by adding an additional primary input together with an OR-gate (Figure 3.3d).

Although addition of control points alters also the observability of the remaining circuit, observability can explicitly be enhanced by adding dedicated observation points (Figure 3.3b) that are taps to the extra primary outputs. These points enable observation of the value at the line. In a BIST environment, these extra inputs and outputs introduced by test point insertion are connected to the TPGs and TRAs.

Test point insertion increases the efficiency of pseudorandom testing and can lead to complete or near-complete fault coverage.

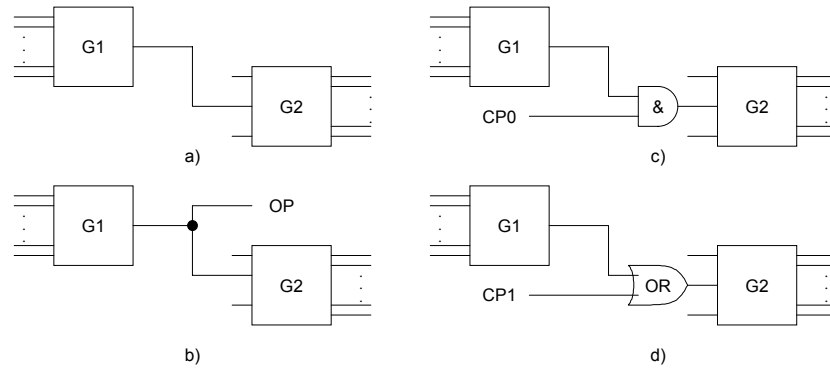


Figure 3.3. Test point insertion.
a) original circuit, b) an observation point,
c) a zero control point, d) a one control point.

CHAPTER 3

3.3.2. Weighted Random Testing

If the fault coverage for a particular CUT within a given test length remains below a desired level, a custom TPG may be designed to provide higher fault coverage for the CUT. Weighted random testing is one such approach.

As described earlier, LFSR based vectors are pseudorandom by nature. Therefore, the likelihood of zeros and ones in each binary position of these vectors is equal and random resistant faults are hard to detect. Weighted random testing uses an additional combinational circuit to modify the LFSR generated patterns so that the probabilities of zeros and ones are nonuniform. Such a weight circuit biases the test vectors so that tests targeting random resistant faults are more likely to occur. The particular probabilities applied are called a *weight set*.

For example, let us assume a 3-input CUT. A classical pseudorandom pattern generator, such as an LFSR, produces a test set, where the probabilities of zeros and ones are the same, i.e. logic 1 is applied to every input with probability 0.5. Therefore, we can say that the weight set of this particular example is $w = (0.5, 0.5, 0.5)$. When a weighted random testing is used then these weights can be, for example, $w = (1, 0, 0.5)$. In this case, logic 1 is applied to the first input with probability 1, to the second with probability 0 and to the third with probability 0.5.

In general, a circuit may require several sets of weights, and, for each weight set, a number of random patterns will be applied. Thus, the major objective of the weight generation process is to reduce the number of weight sets, and the number of test patterns to apply for each set. Several techniques have been proposed in the literature, including those reported in [12], [15], [98] and [170].

3.3.3. Test Pattern Compression

Another way to improve the quality of a self-test is to use deterministic test patterns instead of pseudorandom ones. A straight-

INTRODUCTION AND RELATED WORK

forward way involves the use of a ROM to store the precomputed test set. This test scheme is very efficient in terms of test application time and can provide high fault coverage. There have been also attempts to generate compact test sets for this purpose [79], [134]. However, this is not considered practical because of the silicon area required to store the entire test set in a ROM.

A more practical alternative is to encode the precomputed test set and store (or generate) only the compressed (encoded) test set, which can then be decoded during test application. This decompression logic together with storage requirements for the encoded test set are usually less than the storage requirements for the original deterministic test set. This method is usually called *store and generate* [4], [35], [36]. It has been shown that by applying efficient statistical encoding techniques, such as Huffman or Comma encoding, the storage requirements for testing sequential non-scan circuits can be reduced as well [82]. The encoding can be very efficient if intelligent X assignment in partially specified test sets (vectors with don't care signals) is used [99].

The reported results are promising but the main disadvantage of these approaches is the need for additional hardware to perform the decoding process. The encoded test set is in average 40-60% smaller than the original set, but due to the nondeterministic nature of the encoding process, there are no guarantees about the size of the final encoded test set. This means that the deterministic set can still be too big to be stored entirely in a ROM inside the system.

3.3.4. Mixed-Mode Schemes

Several considerations are central for efficient self-test pattern generator design. First, it is expected to guarantee very high fault coverage. Second, the TPG should be inexpensive to implement in hardware. Finally, it should minimize test application time and test data storage requirements. Therefore, mixed-mode test generation schemes have been developed for efficient self-

CHAPTER 3

testing. A mixed-mode scheme uses pseudorandom patterns to cover easy-to-detect faults and, subsequently, deterministic patterns to target the remaining hard-to-detect faults. The main strength of these approaches lays in the possibility to have a trade-off between test data storage and test application time by varying the ratio of pseudorandom and deterministic test patterns.

As described above, several methods have been developed, where complete deterministic test sets are stored in the ROM, either directly or by using some encoding mechanism. Mixed-mode schemes, on the other hand, store only a limited amount of information, thus reducing the test data storage requirements and consequently the test cost. In the following, some well-known mixed-mode test generation approaches are described.

LFSR Reseeding

When a sequence of test patterns is generated by an LFSR, many of these patterns do not detect any additional faults, thus non-useful patterns are applied to the CUT. The test application time can hence be reduced if non-useful patterns can be replaced with useful ones that occur much later in the sequence. This would increase the frequency with which useful vectors are applied to the circuit and hence reduce the test application time.

One of the mixed-mode approaches is based on LFSR reseeding. In this approach the quality of the test sequence is improved by generating only a limited number of test patterns from one LFSR seed (initial state) and during the test generation process the LFSR is reseeded with new seeds. This idea was first proposed by B. Koenemann in 1991 [103]. These new seeds are used to generate pseudorandom sequences and to encode the deterministic test patterns, in order to reduce the number of non-useful patterns. In this approach, only a set of LFSR seeds have to be stored instead of the complete set of patterns and as a result, less storage is needed (Figure 3.4).

INTRODUCTION AND RELATED WORK

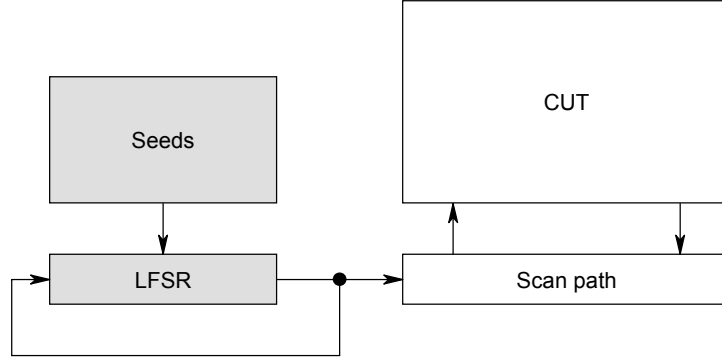


Figure 3.4. LFSR reseeding.

Several heuristic approaches have been proposed to identify multiple seeds, and the number of vectors applied starting with each seed, to minimize the overall test application time under a given constraint on the maximum number of seeds [69], [70], [71], [173]. If a small LFSR is used, it may not always be possible to find a seed that will generate a required deterministic test pattern, hence the fault coverage may remain low. Therefore, a different reseeding scenario based on Multiple-Polynomial LFSRs has been proposed in [70]. There, deterministic patterns are encoded with a number of bits specifying a seed and a polynomial identifier. During testing, not only the appropriate seed, but also the corresponding feedback polynomial, have to be loaded into the LFSR. Another alternative is to use variable-length seeds [173]. However, all these techniques generate test sets of excessive length.

Pattern Mapping

Another class of mixed-mode schemes embeds deterministic test patterns into LFSR sequences by mapping LFSR states to deterministic test patterns [6], [23], [154], [155], [171]. This can be achieved by adding extra circuitry to generate control signals that complement certain bits or fix them to either 0 or 1 [171]. A

CHAPTER 3

hardware used to implement the bit-flipping or bit-fixing sequence generation logic is the major cost of this approach, as it has to be customized for a given CUT and LFSR. An alternative approach transforms the LFSR generated patterns into a new set of test patterns with higher fault coverage. The transformation is carried out by a mapping logic, which decodes sets of ineffective patterns and maps them into vectors that detect the hard-to-test faults [23], [154]. The general architecture of a TPG for this approach is depicted in Figure 3.5. The outputs of an n -stage random TPG are input to a mapping logic and the outputs of the mapping logic drive the inputs of the CUT. Nevertheless, most of these variations of controlling the bits of the LFSR sequence have not yet solved the problems with random resistance.

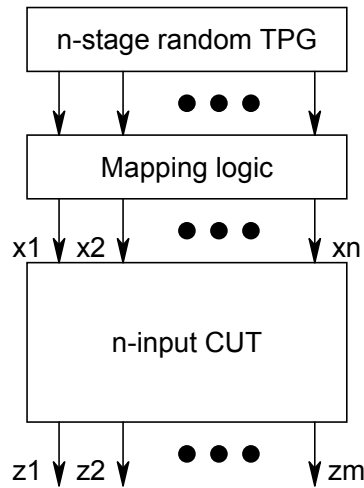


Figure 3.5. Pattern mapping.

3.4. Conclusions

In this chapter, we outlined the problems related to the classical LFSR-based BIST, namely problems stemming from random re-

INTRODUCTION AND RELATED WORK

sistance properties of the CUT. In addition, we gave an overview of different methods that have been developed to tackle those problems. The main objective of these methods has been test quality improvement in terms of fault coverage, while different aspects related to the test cost, like test length, area overhead and tester memory requirements, were largely omitted or handled in isolation.

In the following chapters an alternative approach, called hybrid BIST, will be described. In particular, different test optimization algorithms based on the proposed hybrid BIST architecture will be presented.

Chapter 4

Hybrid BIST Concept

In this thesis, we propose a new mixed-mode BIST approach that is based on a combination of pseudorandom and deterministic test patterns. Similar ideas have been exploited in different contexts already earlier. However, there has been no systematic treatment of the test cost minimization problem in the above-mentioned framework. In addition, the issues related to defect probabilities as well as power and energy consumption have not been touched earlier.

This chapter is devoted to describing the basic concepts of the proposed approach. Additionally an introduction to the hybrid BIST cost calculation principles will be given and different test architectures that were assumed during the experimental work will be described.

4.1. Introduction

As described earlier, a typical self-test approach employs usually some form of pseudorandom test patterns. These test sequences are often very long and not sufficient to detect all faults. To avoid the test quality loss due to random pattern resistant faults and to speed up the testing process, we can apply additional deterministic test patterns targeting the random resistant and difficult to test faults. This can dramatically reduce the length of the initial pseudorandom sequence and achieve the maximum achievable fault coverage.

In the introductory part, we described several existing methods based on this concept. These methods successfully increased the quality of the test by explicitly targeting random pattern resistant (RPR) faults. At the same time, most of these methods tried to address some of the following parameters: area overhead, tester memory (ROM size) and test length. The described approaches were able to reduce one or many of these parameters via different heuristics but the results were very dependent of the CUT and the chosen test scenario. Therefore, none of the approaches would be applicable if the bounds of those parameters are specified in advance and have to be met. Yet, in a realistic test environment, the test process is usually constrained by several limitations, such as tester memory and test time. None of the existing approaches would be able to device a solution under such circumstances. At the same time, there is obvious and realistic need for test solutions that can guarantee high test quality and, at the same time, fit into the existing test flow.

Our hybrid BIST approach is based on the intelligent combination of pseudorandom and deterministic test sequences that would provide a high-quality test solution under imposed constraints [95]. It is important to note that the main contribution of this thesis is not to develop a new “ideal” LFSR-based test generation approach but to develop a set of optimization methods that can produce a required solution. Our approach does not im-

HYBRID BIST CONCEPT

pose restrictions on the way any of the test sets is generated, nor does it assume any particular way the deterministic test set is stored in the system or outside the system. If needed, our techniques can be used in conjunction with the previously proposed ideas regarding test set generation, test set compression and encoding.

4.2. Basic Principle

As mentioned earlier, our hybrid BIST approach is based on an intelligent combination of pseudorandom and deterministic test patterns. Such a hybrid BIST approach starts usually with a pseudorandom test sequence of length L (Figure 4.1). After application of the pseudorandom patterns, a stored test approach with length S will be used [88]. For the stored test approach, pre-computed test patterns are applied to the core under test in order to reach the desirable fault coverage level. For off-line generation of the deterministic test patterns, arbitrary software test generators may be used based on deterministic, random or genetic algorithms [86].

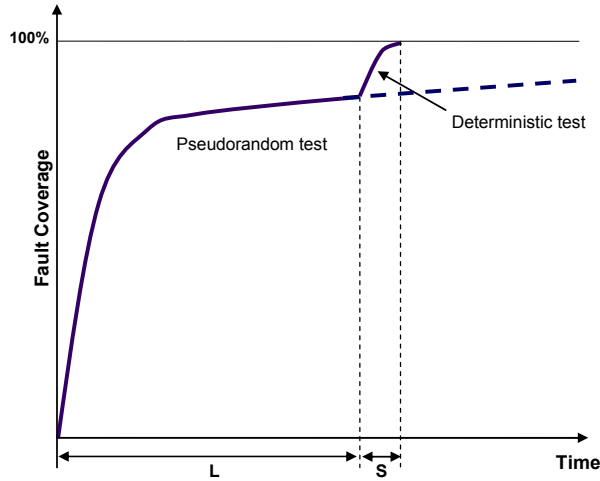


Figure 4.1. Hybrid BIST fault coverage curve.

CHAPTER 4

In a hybrid BIST technique the length of the pseudorandom test is an important design parameter, which determines the behavior of the whole test process. A shorter pseudorandom test sequence implies a larger deterministic test set. This requires additional memory space, but at the same time, shortens the overall test time. A longer pseudorandom test, on the other hand, will lead to larger test application time with reduced memory requirement. Therefore, it is crucial to determine the optimal length of the pseudorandom test in order to minimize the total testing cost.

This basic feature of hybrid BIST is illustrated in Table 4.1 with some selected ISCAS'89 benchmark designs [19] (full-scan versions). In this table, we have illustrated the increase of the fault coverage value after applying a small additional set of deterministic test patterns on top of the pseudorandom ones. As it can be seen, only a small number of deterministic test patterns are needed for that purpose.

Table 4.1. Illustration of the hybrid BIST concept.

Core	DET	FC _{DET} %	FC _{PR} %	H _{DET}	FC _{HYB} %	FC _{impr} %	DET _{red} %
s298	105	95,29%	85,61%	40	96,46%	10,85%	61,90%
s420	161	98,11%	69,59%	104	98,45%	28,86%	35,40%
s526	1308	89,86%	75,77%	105	95,57%	19,80%	91,97%
s641	462	97,16%	81,84%	121	99,15%	17,31%	73,81%
s838	19273	94,46%	57,69%	264	98,54%	40,85%	98,63%
s1423	9014	94,19%	86,82%	143	98,52%	11,70%	98,41%
s3271	6075	99,06%	77,50%	332	99,65%	22,15%	94,53%

<i>DET</i>	Number of deterministic patterns generated by the ATPG.
<i>FC_{DET}</i>	Fault coverage of the deterministic patterns.
<i>FC_{PR}</i>	Fault coverage of the PR patterns (1000 patterns).
<i>H_{DET}</i>	Number of additional deterministic patterns generated by the ATPG, after 1000 PR patterns.
<i>FC_{HYB}</i>	Final fault coverage (PR + deterministic).
<i>FC_{impr}</i>	Improvement of the fault coverage after adding deterministic patterns, compared to FC _{PR} .
<i>DET_{red}</i>	Reduction of the number of deterministic test patterns compared to the original deterministic test set (DET).

HYBRID BIST CONCEPT

4.3. Cost Calculation

Figure 4.2 illustrates graphically the total cost of a hybrid BIST solution consisting of pseudorandom test patterns and stored test patterns. The horizontal axis in Figure 4.2 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 biggest in this point. The figure illustrates the situation where 100% fault coverage is achievable with pseudorandom vectors alone, although this can demand a very long pseudorandom test sequence (in reality, in particular in the case of large and complex designs, 100% fault coverage might not be achievable at all).

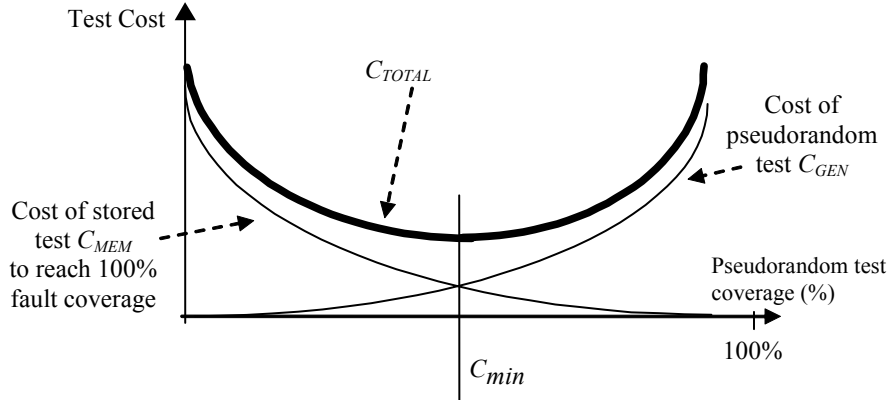


Figure 4.2. Cost calculation for hybrid BIST (under 100% assumption).

The total test cost of the hybrid BIST, C_{TOTAL} , can therefore be defined as:

$$C_{TOTAL} = C_{GEN} + C_{MEM} \approx \alpha L + \beta S \quad (4.1)$$

where C_{GEN} is the cost related to the effort for generating L pseudorandom test patterns (number of clock cycles) and C_{MEM} is re-

CHAPTER 4

lated to the memory cost for storing S pre-computed test patterns to improve the pseudorandom test set. α and β are constants to map the test length and memory space to the costs of the two components of the test solutions.

We should note that 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, results in a rather simplified cost model for the hybrid BIST technique. In this simplified model, neither the basic cost of silicon (or its equivalent) occupied by the LFSR-based generator, nor the effort needed for generating deterministic test patterns are taken into account. Similarly, all aspects related to test data transportation are omitted. However, these aspects can easily be added to the cost calculation formula after the desired hardware architecture and deterministic test pattern generation approaches are chosen. In the following chapters, we are going to provide the algorithms to find the best 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 invariant and will not influence the optimal selection of the hybrid BIST parameters.

On the other hand, the attempt to add “test time” to “memory space” (even in terms of their cost) seems rather controversial, as it is very hard to specify which one costs more in general (or even in particular cases) and how to estimate these costs. This was also the reason why the total cost of a BIST solution is not considered as the research objective in this thesis. The values of parameters α and β in the cost function are left to be determined by the designer and can be seen as one of the design decisions. If needed, it is possible to separate these two different costs (time and space), and consider, for example, one of them as a design constraint.

Figure 4.2 illustrates also how the cost of pseudorandom test is increasing when striving to higher fault coverage (the C_{GEN}

HYBRID BIST CONCEPT

curve). In general, it can be very expensive to achieve high fault coverage with pseudorandom test patterns alone. The C_{MEM} curve, on the other hand, describes the cost we have to pay for storing additional pre-computed tests to achieve the required fault coverage level. The total cost C_{TOTAL} is the sum of the above two costs. The C_{TOTAL} curve is illustrated in Figure 4.2, where the minimum point is marked as C_{min} .

As mentioned earlier, in many situations 100% fault coverage is not achievable with only pseudorandom vectors. Therefore, we have to include this assumption to the total cost calculation. This situation is illustrated in Figure 4.3, where the horizontal axis indicates the number of pseudorandom patterns applied, instead of the fault coverage level. The curve of the total cost C_{TOTAL} is still the sum of two cost curves $C_{GEN} + C_{MEM}$ with the new assumption that the maximum fault coverage is achievable only by either the hybrid BIST or pure deterministic test.

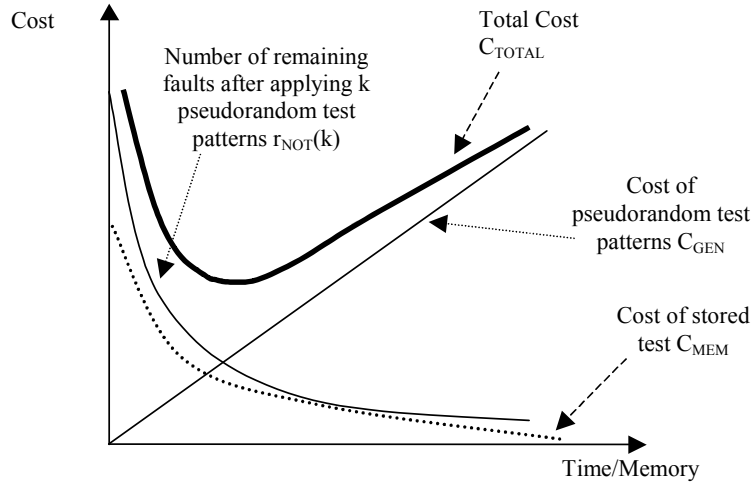


Figure 4.3. Cost calculation for hybrid BIST (under realistic assumptions).

4.4. Architectures

The previous sections have described the basic principles of hybrid BIST and introduced the test cost calculation formulas. In this section, some basic concepts of hybrid BIST architectures will be discussed. Although our optimization methods are not devised for a particular test architecture and different architectural assumptions can easily be incorporated into the algorithms, some basic assumptions have to be made.

Our optimization methods have been devised for single core designs as well as for system-on-chip designs. Consequently, we have to discuss about test architectures at two different levels of hierarchy: core-level test architectures and system-level test architectures.

4.4.1. Core-Level Hybrid BIST Architecture

We have divided cores into two large classes. To the first class belong the cores that are equipped with their own pseudorandom test pattern generator and only deterministic patterns have to be transported to the cores. The second class consists of cores with no pre-existing BIST structures. Such cores require an alternative approach, where pseudorandom and deterministic test patterns have to be transported to the core under test from external sources. For both classes we have studied test-per-clock as well as test-per-scan schemes.

At the core level, pseudorandom testing can be performed using many different scenarios, as described earlier. In our work we have assumed a core-level hybrid BIST architecture that is depicted in Figure 4.4, where the pseudorandom pattern generator (PRPG) and the Multiple Input Signature Analyzer (MISR) are implemented inside the core under test (CUT) using LFSRs or any other structure that provides pseudorandom test vectors with a required degree of randomness. The deterministic test

HYBRID BIST CONCEPT

patterns are precomputed off-line and stored outside the core, either in a ROM or in an ATE.

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. For combinatorial cores, where a test-per-clock scheme can be used, each primary input of the CUT has a multiplexer at the input that determines whether the test is coming from the PRPG or from the memory (Figure 4.4). The response is compacted into the MISR in both cases. The architecture can easily be modified with no or only minor modification of the optimization algorithms to be presented in the following chapters.

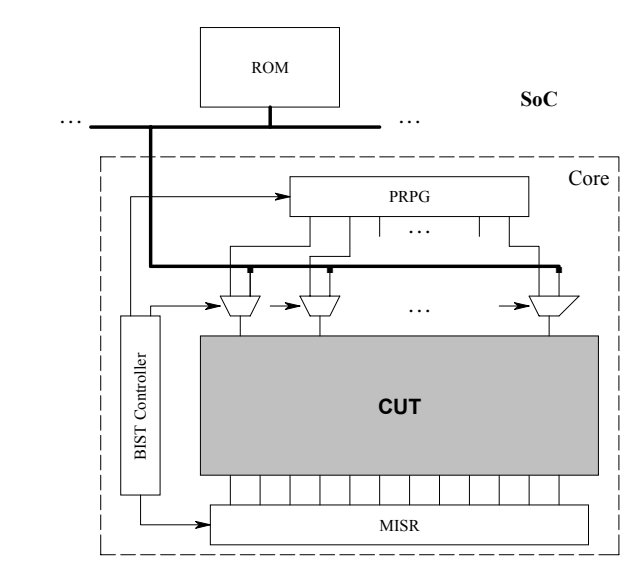


Figure 4.4. Hardware-based core-level hybrid BIST architecture.

As testing of sequential cores is very complex and development of efficient test pattern generation algorithm for sequential cores

CHAPTER 4

is outside the scope of this thesis, it is assumed here that every sequential core contains one or several scan paths (full scan). Therefore a test-per-scan scheme has to be used and, for every individual core, the “Self-Test Using MISR and Parallel Shift Register Sequence Generator” (STUMPS) [11] architecture is assumed. Both internally generated pseudorandom patterns and externally stored deterministic test patterns are therefore applied via scan chains.

In both situations, every core’s BIST logic is capable of producing a set of independent pseudorandom test patterns, i.e. the pseudorandom test sets for all the cores can be carried out simultaneously and independently.

4.4.2. System-Level Hybrid BIST Architectures

Parallel Hybrid BIST Architecture

We start with a system-level test architecture, where every core has its own dedicated BIST logic. The deterministic tests are applied from the external source (either on-chip memory or ATE), one core at a time; and in the current approach we have assumed for test data transportation an AMBA-like test bus [43]. AMBA (Advanced Microcontroller Bus Architecture) integrates an on-chip test access technique that reuses the basic bus infrastructure [67]. An example of a multi-core system, with such a test architecture is given in Figure 4.5.

Our optimization methods are not dependent of the location of the deterministic test patterns. These patterns can be applied either from the external ATE or from an on-chip memory (ROM). As we have assumed a bus-based test architecture, the time needed for test data transportation from the particular test source to a given CUT is always the same. The corresponding time overhead, related to the test data transportation, can easily be incorporated into the proposed algorithms.

HYBRID BIST CONCEPT

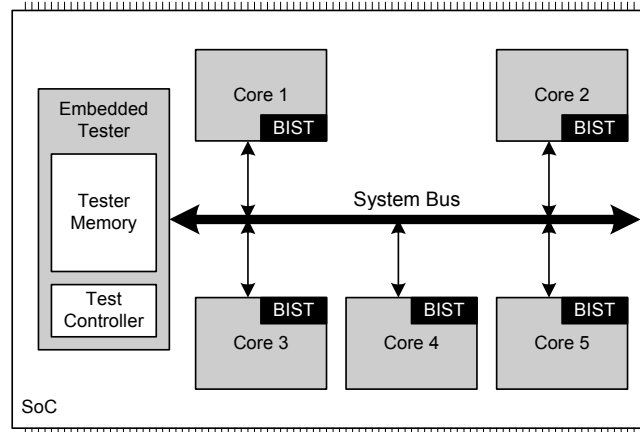


Figure 4.5. An example of a core-based system, with independent BIST resources.

Considering the assumed test architecture, only one deterministic test set can be applied at any given time, while any number of pseudorandom test sessions can take place in parallel. To enforce the assumption that only one deterministic test can be applied at a time, a simple ad-hoc scheduling can be used.

The above type of architecture, however, may not always be feasible as not all cores may be equipped with self-test structures. It may also introduce a significant area overhead and performance degradation, as some cores may require excessively large self-test structures (LFSRs).

Hybrid BIST Architecture with Test Pattern Broadcasting

In order to avoid redesign of the cores, a single pseudorandom test pattern generator for the whole system is an alternative. It can be implemented as a dedicated hardware block or in software. In this thesis we propose a novel solution, where only a single set of pseudorandom test patterns that is broadcasted to all cores simultaneously will be used. This common pseudorandom test set is followed by additional deterministic vectors applied to every individual core, if needed. These deterministic test

CHAPTER 4

vectors are generated during the development process and are stored in the system. This architecture together with the appropriate test access mechanism is depicted in Figure 4.6.

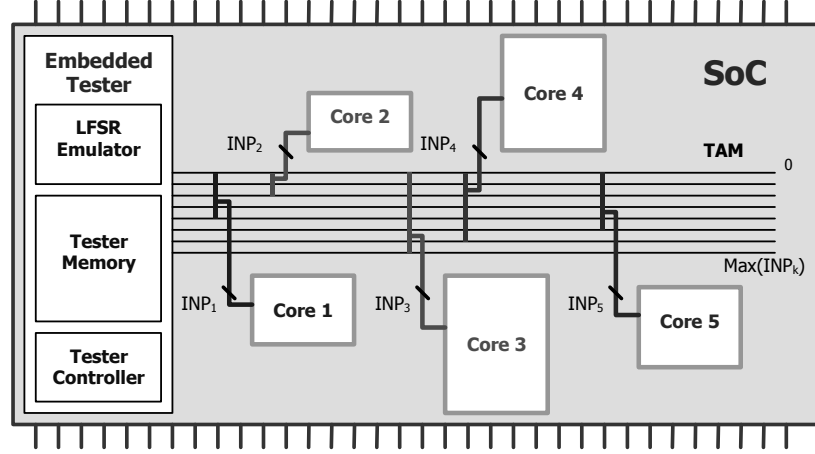


Figure 4.6. Hybrid BIST architecture with test pattern broadcasting.

For the test architecture depicted in Figure 4.6, testing of all cores is carried out in parallel, i.e. all pseudorandom patterns as well as each deterministic test sequence TD_k are applied to all cores in the system. The deterministic test sequence TD_k is a deterministic test sequence generated only by analyzing the core C_k . For the rest of the cores this sequence can be considered as a pseudorandom sequence. The width of the hybrid test sequence TH is equal to $MAXINP = \max\{INP_k\}$, $k=1, 2, \dots, n$, where INP_k is the number of inputs of the core C_k . For each deterministic test set TD_k , where $INP_k < MAXINP$, the not specified bits will be completed with pseudorandom data, so that the resulting test set TD_k^* can be applied in parallel to the other cores in the system as well.

For test response analysis a MISR-based solution is assumed.

HYBRID BIST CONCEPT

Software Based Hybrid BIST Architecture

Classical BIST architectures can be expensive in the case of large and complex designs, because of the long LFSRs. Long LFSRs can also influence the system performance by introducing additional delays. Short LFSRs on the other hand cannot be used because they are not able to produce the required level of randomness. To make the BIST 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 SOC, 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 [72]. However, some preliminary results regarding such an approach for PCBs have been reported in [9] and show that a software-based approach is feasible.

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

In the 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.

CHAPTER 4

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. This approach, however, may lead to a more complex test controller, as every core requires pseudorandom patterns with different characteristics (polynomial, initial state and length, for example). The general concept of the software based pseudorandom TPG is depicted in Figure 4.7.

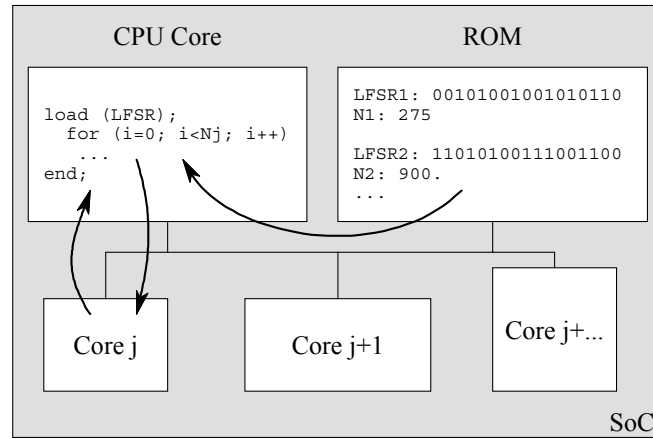


Figure 4.7. LFSR emulation.

As the LFSR is implemented in software, there are no hardware constraints for the actual implementation. This allows developing for each particular core an efficient pseudorandom scheme without concerning about the hardware cost except the cost for the ROM. As has been shown by experiments, the selection of the best possible pseudorandom scheme is an important factor for such an approach [72].

As discussed in [72], the program to emulate the LFSR can be very simple and therefore the memory requirements for storing

HYBRID BIST CONCEPT

the pseudorandom TPG program together with the LFSR parameters are relatively small. This, however, does not have any influence on the cost calculation and optimization algorithms, to be proposed. These algorithms are general, and can be applied to the hardware-based as well as to the software-based hybrid BIST optimization.

4.5. Conclusions

This chapter described the basic concepts of our hybrid BIST methodology. We explained the cost calculation principles and gave an overview of the architectures we have considered in our experiments. In the first part of this chapter the problems with classical, LFSR-based BIST were described. That was followed by an overview of different methods for BIST improvement that have been proposed in the literature.

The basic idea of our approach, to combine pseudorandom and deterministic test patterns, is not itself an uncommon one. However, the main contribution of this thesis is a set of optimization methods for test cost optimization. In the following chapter, the total test cost minimization for single core designs will be described. This will be followed by the time minimization algorithms for multi-core designs. Finally, methods for energy minimization and hybrid BIST time minimization in an abort-on-first-fail environment will be described.

Chapter 5

Hybrid BIST

Cost Minimization for Single Core Designs

5.1. Introduction

In this chapter different methods for total test cost minimization, while testing every core in isolation, will be described. The methods are based on the test cost calculation formulas, introduced in the previous chapter. The proposed cost model is rather simplified but our goal is not to develop a complete cost function for the whole BIST solution. The goal is to find the tradeoff between the length of pseudorandom test sequence and the number of deterministic patterns. For making such a tradeoff the basic implementation costs are invariant and will not influence the optimal solution of the hybrid BIST.

CHAPTER 5

The main goal of this chapter is to develop a method to find the global minimum of the Total Cost curve. Creating the curve $C_{GEN} = \alpha L$ is not difficult. For this purpose, only a simulation of the behavior of the LSFR used for pseudorandom test pattern generation is needed. A fault simulation should be carried out for the complete test sequence generated by the LFSR. As a result of such a simulation, we find for each clock cycle the list of faults which are covered at this clock cycle.

As an example, in Table 5.1 a fragment of the results of BIST simulation for the ISCAS'85 circuit c880 [18] is given, where

- k denotes 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 remaining faults after applying the sequence of patterns generated by the k clock signals, and
- $FC(k)$ is the fault coverage reached by the sequence of patterns generated by the k clock signals

Table 5.1. Pseudorandom test results.

k	$r_{DET}(k)$	$r_{NOT}(k)$	$FC(k)$	k	$r_{DET}(k)$	$r_{NOT}(k)$	$FC(k)$
0	155	839	15.59%	148	13	132	86.72%
1	76	763	23.24%	200	18	114	88.53%
2	65	698	29.78%	322	13	101	89.84%
3	90	608	38.83%	411	31	70	92.96%
4	44	564	43.26%	707	24	46	95.37%
5	39	525	47.18%	954	18	28	97.18%
10	104	421	57.65%	1535	4	24	97.58%
15	66	355	64.28%	1560	8	16	98.39%
20	44	311	68.71%	2153	11	5	99.50%
28	42	269	72.94%	3449	2	3	99.70%
50	51	218	78.07%	4519	2	1	99.89%
70	57	161	83.80%	4520	1	0	100.00%
100	16	145	85.41%				

In the list of BIST simulation results not all clock cycles will be presented. We are only interested in the clock numbers at

HYBRID BIST COST MINIMIZATION

which at least one new fault will be covered, and thus the total fault coverage for the pseudorandom test sequence up to this clock number increases. Let us call such clock numbers and the corresponding pseudorandom test patterns *efficient clocks* and *efficient patterns*. The rows in Table 5.1 correspond to the efficient, not to all, clock cycles for the circuit c880.

If we decide to switch from pseudorandom mode to the deterministic mode after the clock number k , then $L = k$.

More difficult is to find the values for $C_{MEM} = \beta S$, 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 the $r_{NOT}(k)$ not yet detected faults (these patterns should be pre-computed and used as stored test patterns in the hybrid BIST). As an example, the data for the circuit c880 are depicted in Table 5.2. The calculation of the data in the column $t(k)$ of Table 5.2 is the most expensive procedure. In the following section, the difficulties and possible ways to solve this problem are discussed.

Table 5.2. ATPG results.

k	$t(k)$	k	$t(k)$
0	104	148	46
1	104	200	41
2	100	322	35
3	101	411	26
4	99	707	17
5	99	954	12
10	95	1535	11
15	92	1560	7
20	87	2153	3
28	81	3449	2
50	74	4519	1
70	58	4520	0
100	52		

5.2. Test Cost Minimization Algorithms

There are two approaches to find the deterministic test set $t(k)$: ATPG based and fault table based. Let us introduce the following notations:

- i – the current number of the entry in the tables for PRG and ATPG (Table 5.1 and Table 5.2);
- $k(i)$ – the number of the clock cycle of the efficient clock at entry i ;
- $R_{DET}(i)$ - the set of new faults detected (covered) by the pseudorandom test pattern which is generated at the efficient clock signal number $k(i)$;
- $R_{NOT}(i)$ - the set of not yet covered faults after applying the pseudorandom test pattern number $k(i)$;
- $T(i)$ - the set of test patterns needed and found by the ATPG to cover the faults in $R_{NOT}(i)$;
- N – the number of all efficient patterns in the sequence created by the pseudorandom test;

5.2.1. ATPG Based Approach

Algorithm 5.1: 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 $q = N-1, N-2, \dots, 1$:

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.

The above 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

HYBRID BIST COST MINIMIZATION

reach the maximal achievable fault coverage. The number of deterministic test vectors at all efficient clocks is then used to create the curve $C_{MEM}(\beta S)$. The algorithm is straightforward, however, very time consuming because of repetitive use of ATPG.

Since usage of ATPG is a very time consuming 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.

5.2.2. Fault Table Based Approach

The fault table FT for a general case is defined as follows: given a set of test patterns T and a set of faults R , $FT = [\varepsilon_{ij}]^1$ where $\varepsilon_{ij} = 1$ if the test $t_i \in T$ detects the fault $r_j \in R$, and $\varepsilon_{ij} = 0$ 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 that 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, the deterministic test set to be applied can be significantly reduced as compared to the complete set T . 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 .

¹ FT is a $i \times j$ matrix, where i is the number of tests and j is the number of faults.

CHAPTER 5

The described procedure of updating the fault table FT can be carried out iteratively for all possible breakpoints $i = 1, 2, \dots, N$ of the pseudorandom test sequence by the following algorithm [160].

Algorithm 5.2: Fault Table based approach for finding test sets $T(i)$

1. Calculate the whole test T for the whole set of faults R by an ATPG to reach as high fault coverage C as possible
2. Create for T and R the fault table $FT = [\varepsilon_{ij}]$
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 $R(t_j) = \emptyset$
8. Optimize the test set T_i by a 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 $i = 1, 2, \dots, N$ to reach the same fault coverage reached by T .

As the result of Algorithm 5.2, the numbers of precomputed deterministic test patterns $S_i = |T_i|$ to be stored and the subsets of these patterns T_i for each $i = 1, 2, \dots, N$ are calculated. Based on this data the cost of stored test patterns for each i can be calculated by the formula $C_{MEM} = \beta S_i$. From the curve of the total cost

HYBRID BIST COST MINIMIZATION

$C_{TOTAL}(i) = \alpha L + \beta S$ the value of the minimum cost of the hybrid BIST $\min\{C_{TOTAL}(i)\}$ can be easily found.

As will be shown in section 5.3, for very large circuits, both algorithms presented will lead to very time-consuming runs. It would be desirable to find the global minimum of the total cost curve by as few sampled calculations as possible. Therefore, we introduce here an approach, based on a Tabu search heuristic, to speed up the calculations.

5.2.3. Tabu Search Based Cost Optimization

Tabu search ([57], [58], [59]) is a form of local neighborhood search. Each solution $SO \in \Omega$, where Ω is the search space (the set of all feasible solutions), has an associated set of neighbors $N(SO) \subseteq \Omega$. A solution $SO' \in N(SO)$ 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 [58]. 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 Ω , which becomes the first current solution SO . A solution in our hybrid BIST cost minimization problem is defined as the switching moment from the

CHAPTER 5

pseudorandom test mode to the stored test mode. The search space Ω 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 $V^* \subset N(SO)$ is generated. These solutions can be generated by using either the Algorithm 5.1 or Algorithm 5.2. In our current approach, Algorithm 5.2 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 ($|V^*| = n \ll |N(SO)|$). 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 $SO^* \in V^*$, is chosen for consideration as the next solution. The move to SO^* is considered, even if SO^* is worse than SO , that is, $Cost(SO^*) > Cost(SO)$. This feature enables escaping from local optima. The cost of a solution is calculated according to Equation (4.1) for calculating the total cost of hybrid BIST C_{TOTAL} , presented in section 4.3. 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. The Tabu list 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 [142]. 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.

HYBRID BIST COST MINIMIZATION

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 [88] 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 5.1). Based on the value of $|R_{NOT}(i)|$ it is possible to estimate the expected number of test patterns needed for covering the faults in $R_{NOT}(i)$. The starting point for the Tabu search procedure can be found by 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 [88] the following approximation is proposed: one remaining fault results in 0,45 test patterns needed to cover it. In this way, a simplified cost prediction function was derived: $C'_{TOTAL}(k) = C_{GEN}(k) + 0,45\beta R_{NOT}(k)$. The value k^* , where $C'_{TOTAL}(k^*) = \min(C'_{TOTAL}(k))$ was used as the initial solution for Tabu search.

To explain the algorithm, let us 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 5.3: Tabu Search

Begin

Start with initial solution $SO \in \Omega$;

BestSolution:=SO;

Add the initial solution SO to Tabu list T , $T=\{SO\}$;

While number of empty iterations $< E$ **Do**

Generate the sample of neighbor solutions $V^* \subset N(SO)$;

Find best $Cost(SO^* \subset V^*)$;

M: If (solution SO^* is not in T) **Or**

(aspiration criterion is satisfied) **Then**

$SO^{trial} := SO^*$;

Update tabu list T ;

CHAPTER 5

```

        Increment the iteration number;
Else
        Find the next best  $Cost(SO^* \prec V^*)$ ;
        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.

```

5.3. Experimental Results

Experiments were carried out on the ISCAS'85 benchmark circuits for comparing Algorithm 5.1 and Algorithm 5.2, 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 [86], [156] for deterministic test pattern generation, fault simulation, and test set compaction. The results are presented in Table 5.3 and illustrated by several diagrams [88], [89], [160], [161].

For calculating the total cost of hybrid BIST we used the formula $C_{TOTAL} = \alpha L + \beta S$. For simplicity, we assume here that $\alpha = 1$, and $\beta = B$ 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.

HYBRID BIST COST MINIMIZATION

In the columns of Table 5.3 the following data is depicted: ISCAS'85 benchmark circuit name, L - length of the pseudorandom test sequence, FC - fault coverage, S - number of test patterns generated by deterministic ATPG to be stored, C_T - total cost of the hybrid BIST, T_1 and T_2 - the time (sec) needed for calculating the cost curve by Algorithm 5.1 and Algorithm 5.2, T_3 - the time (sec) to find the minimal 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 initial pseudorandom sequence with length L was obtained by executing the LFSR until the same fault coverage as for the ATPG-based solution was reached or no new faults were detected after predetermined amount of time (the number denotes the last efficient pattern). The fault coverage of the final hybrid test set is the same as for the pure deterministic test set.

The results given in Table 5.3 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 efficient way.

The efficiency of the search depends significantly 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 composed and given in Figure 5.1 and Figure 5.2. 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

CHAPTER 5

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.

Table 5.3. Experimental Results.

Circuit	Pseudorandom test		Deterministic test		Hybrid test		
	L	FC	S	FC	L	S	C_T
C432	780	93.0	80	93.0	91	21	196
C499	2036	99.3	132	99.3	78	60	438
C880	5589	100.0	77	100.0	121	48	505
C1355	1522	99.5	126	99.5	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

Circuit	Calculation cost				Acc (%)
	T_1	T_2	T_3	T_s	
C432	1632	21	2.85	11	100.0
C499	74	3	0.50	19	100.0
C880	17	2	0.26	15	99.7
C1355	133	5	0.83	18	99.5
C1908	2132	25	3.83	28	100.0
C2670	230	13	0.99	9	99.1
C3540	22601	122	7.37	16	100.0
C5315	2593	38	1.81	12	97.2
C6288	200	6	1.70	15	100.0
C7552	15004	129	3.70	8	99.7

HYBRID BIST COST MINIMIZATION

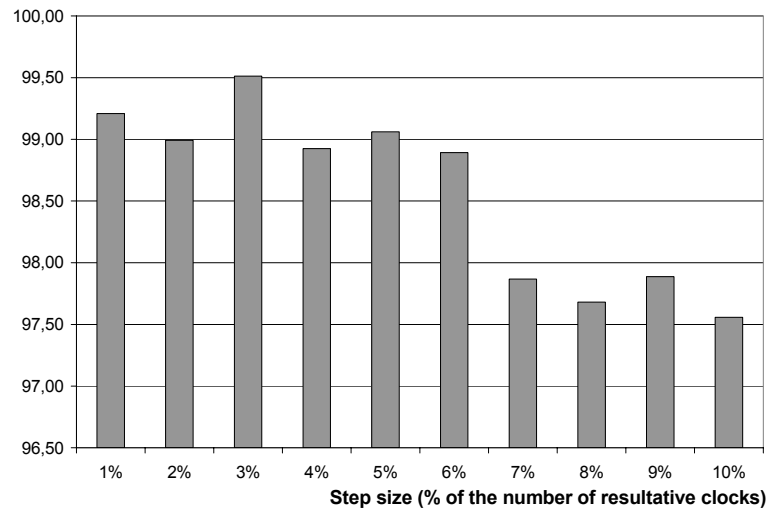


Figure 5.1. Dependency of estimation accuracy from neighborhood step size.

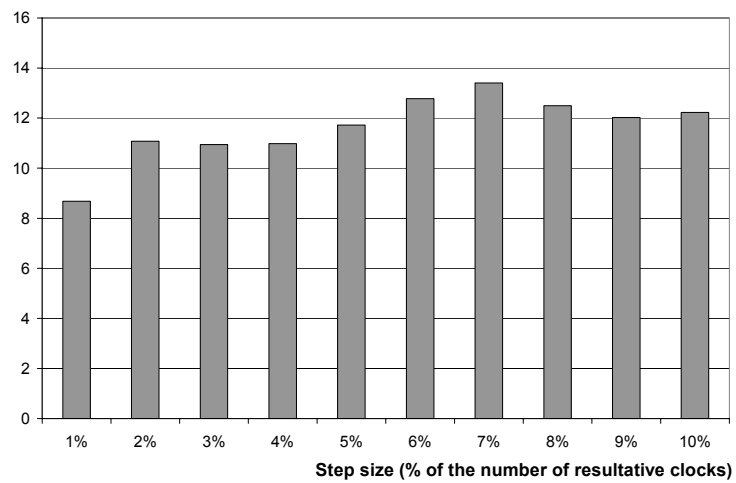
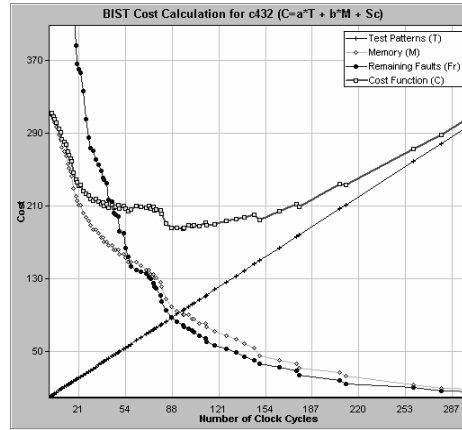


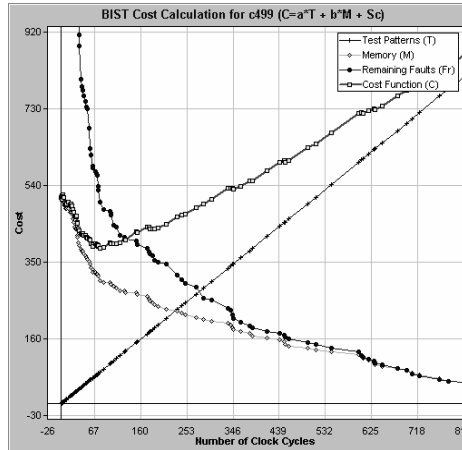
Figure 5.2. Dependency of average speedup from neighborhood size.

CHAPTER 5

In Figure 5.3, the curves of the cost $C_{GEN}=L$ (denoted on Figure 5.3 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).



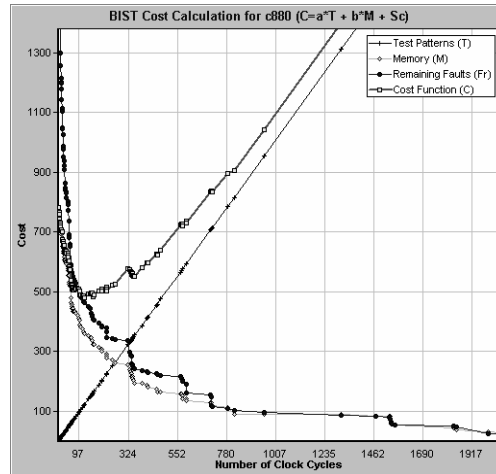
C432



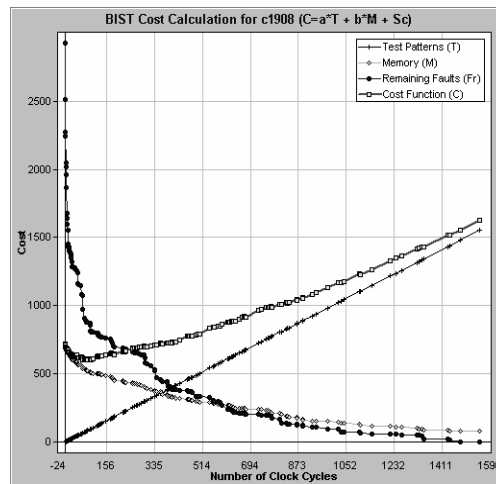
C499

Figure 5.3. Cost curves of hybrid BIST for some ISCAS'85 benchmark circuits.

HYBRID BIST COST MINIMIZATION



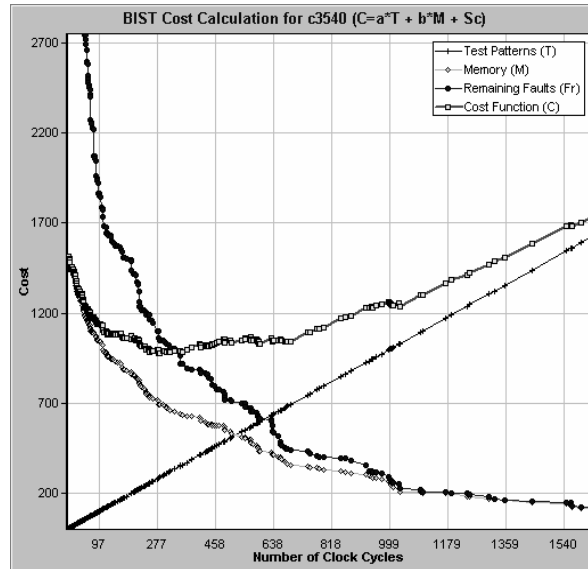
C880



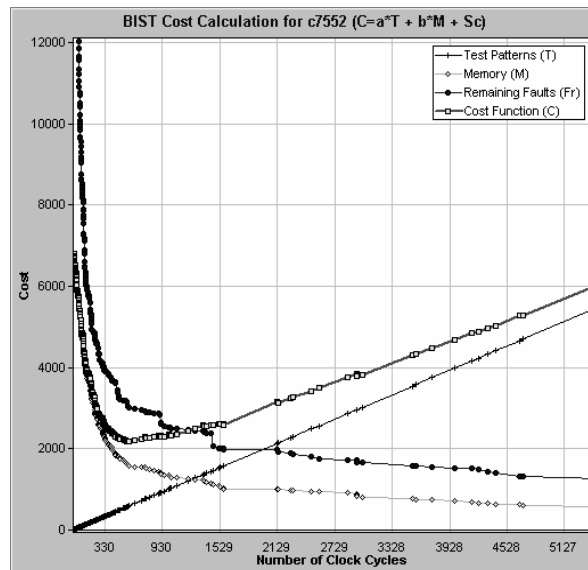
C1908

Figure 5.3. Cost curves of hybrid BIST for some ISCAS'85 benchmark circuits (cont.).

CHAPTER 5



C3540



C7552

Figure 5.3. Cost curves of hybrid BIST for some ISCAS'85 benchmark circuits (cont.).

HYBRID BIST COST MINIMIZATION

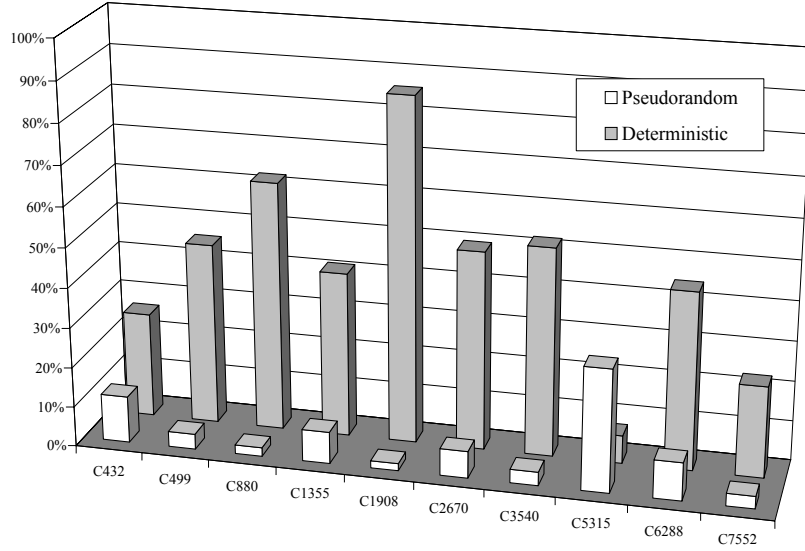


Figure 5.4. Percentage of test patterns in the optimized hybrid test sets compared to the original test sets.

In Figure 5.4 the amount of pseudorandom and deterministic test patterns in the optimal hybrid 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, while the maximum achievable fault coverage is guaranteed and achieved.

Figure 5.5 compares the costs of different approaches using for Hybrid BIST cost calculation Equation (4.1) 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 under this assumption of coefficient values (α, β) , 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 signifi-

CHAPTER 5

cant advantage compared to the pure pseudorandom or stored test approach in most of the cases.

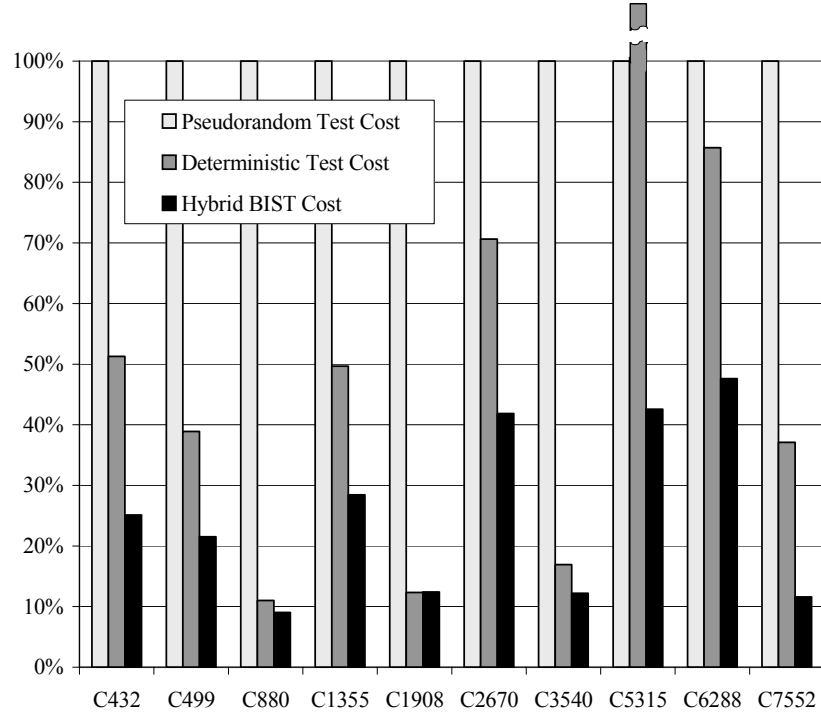


Figure 5.5. Cost comparison of different methods.
Cost of pseudorandom test is taken as 100%.

5.4. Conclusions

In this chapter a hybrid BIST cost optimization for single-core designs has been presented. 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

HYBRID BIST COST MINIMIZATION

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 an efficient 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 6

Hybrid BIST Time Minimization for Systems-on-Chip

6.1. Introduction

In the previous sections we have described the basic principles of hybrid BIST and discussed methods for test cost calculation and optimization for individual cores in isolation. In this chapter, we concentrate on hybrid BIST optimization for multi-core designs. As total cost minimization for multi-core systems is an extremely complex problem and is rarely used in reality, the main emphasis here is on test time minimization under memory constraints with different test architectures. The memory constraints can be seen as limitations of on-chip memory or ATE memory, where the deterministic test set will be stored, and therefore with high practical importance. We will concentrate on two large classes of

CHAPTER 6

test architectures. In one case we assume that every core is equipped with its own pseudorandom pattern generator and only deterministic patterns have to be transported to the cores. In the second case we assume test pattern broadcasting, where both pseudorandom and deterministic test patterns have to be transported to the cores under test. For both architectures we will describe test-per-clock as well as test-per-scan application schemes.

6.2. Parallel Hybrid BIST Architecture

We start with a test architecture where every core has its own dedicated BIST logic that is capable of producing a set of independent pseudorandom test patterns, i.e. the pseudorandom test sets for all the cores can be carried out simultaneously. At the system level, however, only one test access bus is assumed, thus the deterministic tests can only be carried out for one core at a time. Such architecture assumes that all patterns from the same test set (pseudorandom or deterministic) for the same CUT have the same test application time, thus simplifying test cost calculations. An example of a multi-core system, with such a test architecture is given in Figure 6.1.

In order to explain the test time minimization problem for multi-core systems, let us use an example design, consisting of 5 cores, each core as a different ISCAS benchmark (Figure 6.1). Using the hybrid BIST optimization methodology, described in Chapter 5, we can find the optimal combination between pseudorandom and deterministic test patterns for every individual core (Figure 6.2). Considering the assumed test architecture, only one deterministic test set can be applied at any given time, while any number of pseudorandom test sessions can take place in parallel. To enforce the assumption that only one deterministic test can be applied at a time, a simple ad-hoc scheduling method can be used. The result of this schedule defines the starting moments for every deterministic test session, the memory requirements,

HYBRID BIST TIME MINIMIZATION

and the total test length t for the whole system. This situation is illustrated in Figure 6.2.

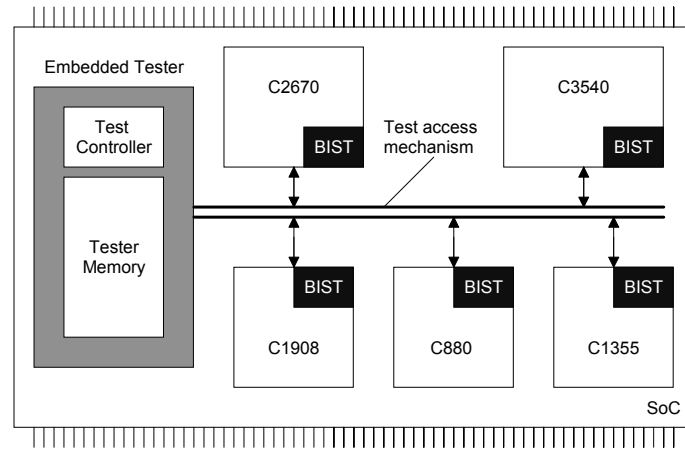


Figure 6.1. An example of a core-based system, with independent BIST resources.

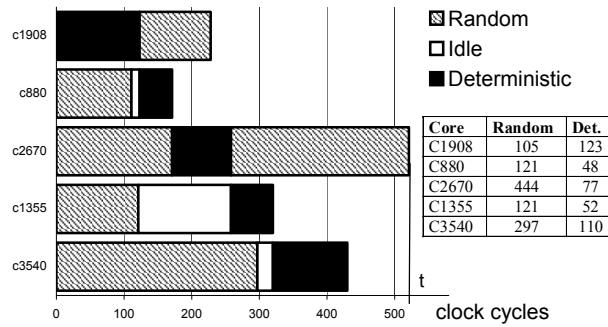


Figure 6.2. Ad-hoc test schedule for hybrid BIST of the core-based system example.

As it can be seen from Figure 6.2, the solution where every individual core has the best possible combination between pseudo-random and deterministic patterns usually does not lead to the

CHAPTER 6

best system-level test solution. In this example, we have illustrated three potential problems:

- The total test length of the system is determined by the single longest individual test set, while other tests may be substantially shorter;
- The resulting deterministic test sets do not take into account the memory requirements, imposed by the size of the on-chip memory or the external test equipment;
- The proposed test schedule may introduce idle periods, due to the scheduling conflicts between the deterministic tests of different cores;

There are several possibilities for improvement. For example, the ad-hoc solution in Figure 6.2 can easily be improved by using a better scheduling strategy. This, however, does not necessarily lead to a significantly better solution as the ratio between pseudorandom and deterministic test patterns for every individual core is not changed. Therefore, we have to explore different combinations between pseudorandom and deterministic test patterns for every individual core in order to find a solution where the total test length of the system is minimized and the memory constraints are satisfied. In the following sections, we will define this problem more precisely, and describe a fast iterative algorithm for calculating the optimal combination between different test sets for the whole system.

6.2.1. Basic Definitions and Problem Formulation

Let us assume that a system S consists of n cores C_1, C_2, \dots, C_n . For every core $C_k \in S$ a complete sequence of deterministic test patterns TD_k^F and a complete sequence of pseudorandom test patterns TP_k^F can be generated.

Definition 6.1: A *hybrid BIST* set $TH_k = \{TP_k, TD_k\}$ for a core C_k is a sequence of tests, constructed from a subset $TP_k \subseteq TP_k^F$ of the pseudorandom test sequence, and a deterministic test se-

HYBRID BIST TIME MINIMIZATION

quence $TD_k \subseteq TD_k^F$. The sequences TP_k and TD_k complement each other to achieve the maximum achievable fault coverage.

Definition 6.2: A pattern in a pseudorandom test sequence is called *efficient* if it detects at least one new fault that is not detected by the previous test patterns in the sequence. The ordered sequence of efficient patterns form an *efficient pseudorandom test sequence* $TPE_k = (P_1, P_2, \dots, P_n) \subseteq TP_k$. Each efficient pattern $P_j \in TPE_k$ is characterized by the length of the pseudorandom test sequence TP_k , from the start to the efficient pattern P_j , including P_j . An efficient pseudorandom test sequence TPE_k , which includes all efficient patterns of TP_k^F is called *full efficient pseudorandom test sequence* and denoted by TPE_k^F .

Definition 6.3: The cost of a hybrid test set TH_k for a core C_k is determined by the total length of its pseudorandom and deterministic test sequences, which can be characterized by their costs, $COST_{P,k}$ and $COST_{D,k}$ respectively:

$$COST_{T,k} = COST_{P,k} + COST_{D,k} = \sigma |TP_k| + \varphi_k |TD_k| \quad (6.1)$$

and by the cost of recourses needed for storing the deterministic test sequence TD_k in the memory:

$$COST_{M,k} = \gamma_k |TD_k|. \quad (6.2)$$

The parameters σ and φ_k ($k=1, 2, \dots, n$) can be introduced by the designer to align the application times of different test sequences. For example, when a test-per-clock BIST scheme is used, a new test pattern can be generated and applied in each clock cycle and in this case $\sigma = 1$. The parameter φ_k for a particular core C_k is equal to the total number of clock cycles needed for applying one deterministic test pattern from the memory. In a special case, when deterministic test patterns are applied by an external test equipment, application of deterministic test patterns may be up to one order of magnitude slower than applying BIST patterns. The coefficient γ_k is used to map the number of test patterns in the deterministic test sequence TD_k into the memory recourses, measured in bits.

CHAPTER 6

Definition 6.4: When assuming the test architecture described above, a hybrid test set $TH = \{TH_1, TH_2, \dots, TH_n\}$ for a system $S = \{C_1, C_2, \dots, C_n\}$ consists of hybrid tests TH_k for each individual core C_k , where the pseudorandom components of TH can be scheduled in parallel, whereas the deterministic components of TH must be scheduled in sequence due to the shared test resources.

Definition 6.5: $J = (j_1, j_2, \dots, j_n)$ is called the *characteristic vector* of a hybrid test set $TH = \{TH_1, TH_2, \dots, TH_n\}$, where $j_k = |TPE_k|$ is the length of the efficient pseudorandom test sequence $TPE_k \subseteq TP_k \subseteq TH_k$.

According to Definition 6.2, for each j_k corresponds a pseudorandom subsequence $TP_k(j_k) \subseteq TP_k^P$, and according to Definition 6.1, any pseudorandom test sequence $TP_k(j_k)$ should be complemented with a deterministic test sequence, denoted with $TD_k(j_k)$, that is generated in order to achieve the maximum achievable fault coverage. Based on this we can conclude that the characteristic vector J determines entirely the structure of the hybrid test set TH_k for all cores $C_k \in S$.

Definition 6.6: The test length of a hybrid test $TH = \{TH_1, TH_2, \dots, TH_n\}$ for a system $S = \{C_1, C_2, \dots, C_n\}$ is given by:

$$COST_T = \max_k \{ \max(\sigma |TP_k| + \varphi_k |TD_k|), \sum_k \varphi_k |TD_k| \}. \quad (6.3)$$

The total cost of resources needed for storing the patterns from all deterministic test sequences TD_k in the memory is given by:

$$COST_M = \sum_k COST_{M,k}. \quad (6.4)$$

Definition 6.7: Let us introduce a generic cost function $COST_{M,k} = f_k(COST_{T,k})$ for every core $C_k \in S$, and an integrated generic cost function $COST_M = f_k(COST_T)$ for the whole system S .

The functions $COST_{M,k} = f_k(COST_{T,k})$ will be created in the following way. Let us have a hybrid BIST set $TH_k(j) = \{TP_k(j), TD_k(j)\}$ for a core C_k with j efficient patterns in the pseudorandom

HYBRID BIST TIME MINIMIZATION

test sequence. By calculating the costs $COST_{T,k}$ and $COST_{M,k}$ for all possible hybrid test set structures $TH_k(j)$, i.e. for all values $j = 1, 2, \dots, |TPE_k^F|$, we can create the cost functions $COST_{T,k} = f_{T,k}(j)$, and $COST_{M,k} = f_{M,k}(j)$. By taking the inverse function $j = f_{T,k}^{-1}(COST_{T,k})$, and inserting it into the $f_{M,k}(j)$ we get the generic cost function $COST_{M,k} = f_{M,k}(f_{T,k}^{-1}(COST_{T,k})) = f_k(COST_{T,k})$ where the memory costs are directly related to the lengths of all possible hybrid test solutions.

The integrated generic cost function $COST_M = f(COST_T)$ for the whole system is the sum of all cost functions $COST_{M,k} = f_k(COST_{T,k})$ of individual cores $C_k \in S$.

From the function $COST_M = f(COST_T)$ the value of $COST_T$ for every given value of $COST_M$ can be found. The value of $COST_T$ determines the lower bound of the length of the hybrid test set for the whole system. To find the component j_k of the characteristic vector J , i.e. to find the structure of the hybrid test set for all cores, the equation $f_{T,k}(j) = COST_T$ should be solved.

The objective here is to find a shortest possible ($\min(COST_T)$) hybrid test sequence TH_{OPT} when the memory constraints are not violated i.e., $COST_M \leq COST_{M,LIMIT}$.

6.2.2. Test Set Generation Based on Cost Estimates

By knowing the generic cost function $COST_M = f(COST_T)$, the total test length $COST_T$ at any given memory constraint $COST_M \leq COST_{M,LIMIT}$ can be found in a straightforward way. However, the procedure to calculate the cost functions $COST_{D,k}(j)$ and $COST_{M,k}(j)$ is very time consuming, since it assumes that the deterministic test set TD_k for each $j = 1, 2, \dots, |TPE_k^F|$ has to be available. This assumes that after every efficient pattern $P_j \in TPE_k \subseteq TP_k$, $j = 1, 2, \dots, |TPE_k^F|$ a set of not yet detected faults $F_{NOT,k}(j)$ should be calculated. This can be done by repetitive use of the automatic test pattern generator or by systematically analyzing and compressing the fault tables for each j (see Chapter 5). Both algorithms are time-consuming and therefore not feasi-

CHAPTER 6

ble for larger designs. To overcome the complexity explosion problem we propose an iterative algorithm, where costs $COST_{M,k}$ and $COST_{D,k}$ for the deterministic test sets TD_k can be found based on estimates. The estimation method is based on fault coverage figures and does not require accurate calculations of the deterministic test sets for not yet detected faults $F_{NOT,k}(j)$.

In the following we will use $FD_k(i)$ and $FPE_k(i)$ to denote the fault coverage figures of the test sequences $TD_k(i)$ and $TPE_k(i)$, respectively, where i is the length of the test sequence.

Procedure 6.1: Estimation of the length of the deterministic test set TD_k .

1. Calculate, by fault simulation, the fault coverage functions $FD_k(i)$, $i = 1, 2, \dots, |TD_k^F|$, and $FPE_k(i)$, $i = 1, 2, \dots, |TPE_k^F|$. The patterns in TD_k^F are ordered in such a way that each pattern put into the sequence contributes with maximum increase in fault coverage.
2. For each $i^* \leq |TPE_k^F|$, find the fault coverage value F^* that can be reached by a sequence of patterns $(P_1, P_2, \dots, P_{i^*}) \subseteq TPE_k$ (see Figure 6.3).
3. By solving the equation $FD_k(i) = F^*$, find the maximum integer value j^* that satisfies the condition $FD_k(j^*) \leq F^*$. The value of j^* is the length of the deterministic sequence TD_k that can achieve the same fault coverage F^* .
4. Calculate the value of $|TD_k^E(i^*)| = |TD_k^F| - j^*$ which is the number of test patterns needed from the TD_k^F to reach to the maximum achievable fault coverage.

The value $|TD_k^E(i^*)| = |TD_k^F| - j^*$, calculated by Procedure 6.1, can be used to estimate the length of the deterministic test sequence TD_k in the hybrid test set $TH_k = \{TP_k, TD_k\}$ with i^* efficient test patterns in TP_k , ($|TPE_k| = i^*$).

By finding $|TD_k^E(j)|$ for all $j = 1, 2, \dots, |TPE_k^F|$ we get the cost function estimate $COST_{D,k}^E(j)$. Using $COST_{D,k}^E(j)$, other cost function estimates $COST_{M,k}^E(j)$, $COST_{T,k}^E(j)$ and $COST_{M,k}^E =$

HYBRID BIST TIME MINIMIZATION

$f_k^E(COST_{T,k}^E)$ can be created according to the Definition 6.3 and Definition 6.7.

Finally, by adding cost estimates $COST_{M,k}^E = f_k^E(COST_{T,k}^E)$ of all cores, we get the hybrid BIST cost function estimate $COST_M^E = f^E(COST_T^E)$ for the whole system.

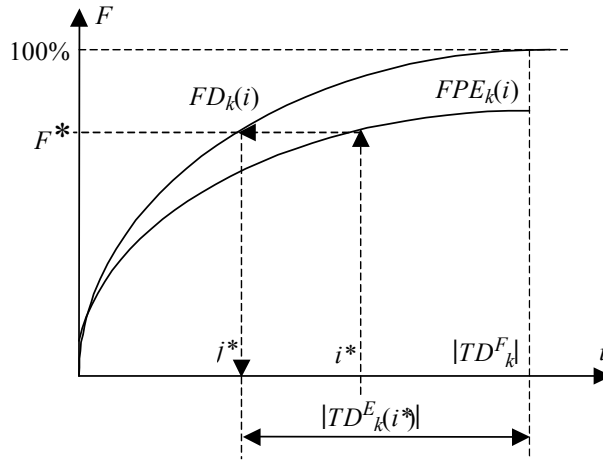


Figure 6.3. Estimation of the length of the deterministic test sequence.

This estimation mechanism is illustrated on Figure 6.4. It depicts fault simulation results of both, pseudorandom (TP) and deterministic (TD), test sets for a given core. The length of the pseudorandom sequence has to be only so long as potentially interesting. By knowing the length of the complete deterministic test set and fault coverage figures for every individual pattern we can estimate the size of the additional deterministic test set for any length of the pseudorandom test sequence, as illustrated in the Figure 6.4. Here we can see that for a given core 60 deterministic test cycles are needed to obtain the same fault coverage as 524 pseudorandom test cycles and it requires additional 30 deterministic test cycles to reach 100% fault coverage. Based on this information we assume, that if we will apply those 30 de-

CHAPTER 6

terministic test cycles on top of the 524 pseudorandom cycles, we can obtain close to the maximum fault coverage. This assumption is the basis for our cost estimation procedure. Obviously, this cannot be used as a final solution, but as we will demonstrate, it can be used as a good starting point for test time minimization algorithms.

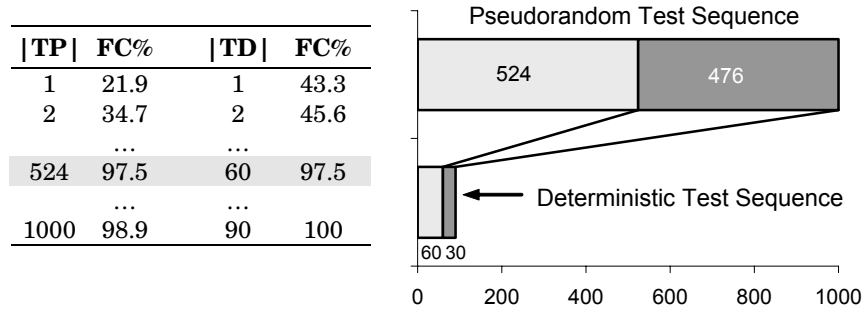


Figure 6.4. Estimation of the length of the deterministic test sequence (core s1423).

In order to demonstrate the feasibility of the proposed estimation methodology, we performed experiments with all designs from the ISCAS85 benchmark family. Some of these results are illustrated in Figure 6.5. More results can be found in [93]. In these charts we have depicted the memory requirement (the size of the deterministic test set) for every pseudorandom test length. Obviously – the longer the pseudorandom test sequence is, the smaller is the memory requirement. We have compared the proposed estimate against the real memory cost. This has been obtained by the repetitive use of the ATPG (see Chapter 5). As it can be seen from the results, the proposed estimation methodology gives very good estimate, mainly in the situations, when the hybrid test set contains smaller amount of pseudorandom test patterns.

HYBRID BIST TIME MINIMIZATION

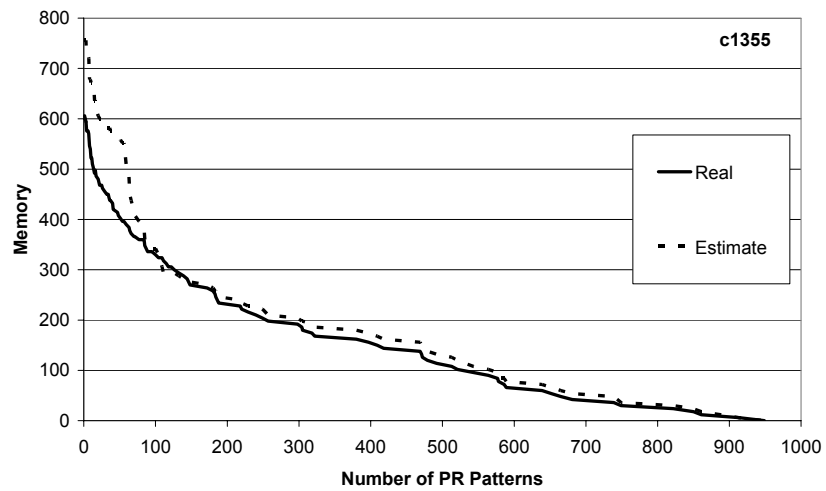
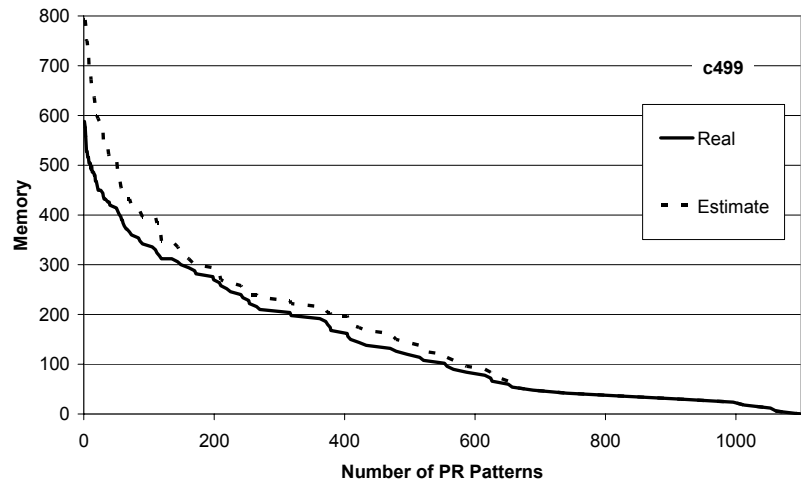


Figure 6.5. Estimation accuracy.

CHAPTER 6

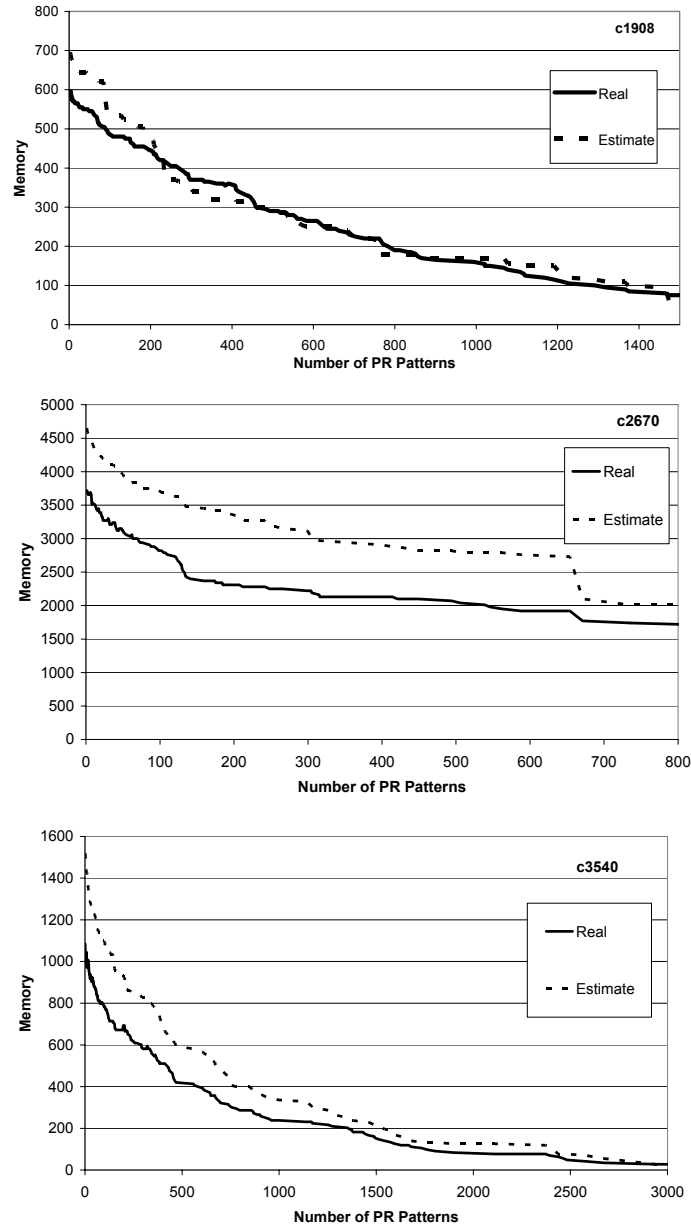


Figure 6.5. Estimation accuracy (cont.).

HYBRID BIST TIME MINIMIZATION

6.2.3. Test Length Minimization Under Memory Constraints

As described above, the exact calculations for finding the cost of the deterministic test set $COST_{M,k} = f_k(COST_{T,k})$ are very time-consuming. Therefore, we will use the cost estimates, calculated by Procedure 6.1 in the previous section, instead. Using estimates can give us a close to minimal solution for the test length of the hybrid test at given memory constraints. After obtaining this solution, the cost estimates can be improved and another, better solution can be calculated. This iterative procedure will be continued until we reach the final solution.

Procedure 6.2: Test length minimization.

1. Given the memory constraint $COST_{M,LIMIT}$, find the estimated total test length $COST_T^{E*}$ as a solution to the equation $f^E(COST_T^E) = COST_{M,LIMIT}$.
2. Based on $COST_T^{E*}$, find a candidate solution $J^* = (j_p^*, j_q^*, \dots, j_n^*)$ where each j_k^* is the maximum integer value that satisfies the equation $COST_{T,k}^E(j_k^*) \leq COST_T^{E*}$.
3. To calculate the exact value of $COST_M^*$ for the candidate solution J^* , find the set of not yet detected faults $F_{NOT,k}(j_k^*)$ and generate the corresponding deterministic test set TD_k^* by using an ATPG algorithm.
4. If $COST_M^* = COST_{M,LIMIT}$, go to the Step 9.
5. If the difference $|COST_M^* - COST_{M,LIMIT}|$ is bigger than that in the earlier iteration make a correction $\Delta t = \Delta t / 2$, and go to Step 7.
6. Calculate a new test length $COST_T^{E,N}$ from the equation $f_k^E(COST_T^E) = COST_M^*$, and find the difference $\Delta t = COST_T^{E,*} - COST_T^{E,N}$.
7. Calculate a new cost estimate $COST_T^{E,*} = COST_T^{E,*} + \Delta t$ for the next iteration.
8. If the value of $COST_T^{E,*}$ is the same as in an earlier iteration, go to Step 9, otherwise go to Step 2.

CHAPTER 6

9. **END:** The vector $J^* = (j^*_p, j^*_s, \dots, j^*_n)$ is the solution.

To illustrate the above procedure, in Figure 6.6 and Figure 6.7 an example of the iterative search for the shortest length of the hybrid test is given. Figure 6.6 represents all the basic cost curves $COST^E_{D,k}(j)$, $COST^E_{P,k}(j)$, and $COST^E_{T,k}(j)$, as functions of the length j of TPE_k where j_{min} denotes the optimal solution for a single core hybrid BIST optimization problem [88].

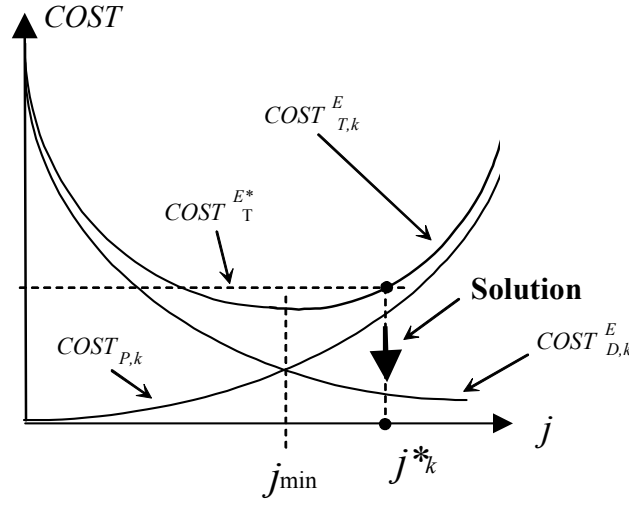


Figure 6.6. Cost curves for a given core C_k .

Figure 6.7 represents the estimated generic cost function $COST^E_M = f^E(COST^E_T)$ for the whole system. At first (Step 1), the estimated $COST^{E^*}_T$ for the given memory constraints is found (point 1 on Figure 6.7). Then (Step 2), based on $COST^{E^*}_T$ the length j^*_k of TPE_k for the core C_k in Figure 6.6 is found. This procedure (Step 2) is repeated for all the cores to find the characteristic vector J^* of the system as the first iterative solution. After that the real memory cost $COST^{E^*}_M$ is calculated (Step 3, point 1* in Figure 6.7). As we see in Figure 6.7 the value of $COST^{E^*}_M$ in point 1* violates the memory constraints. The difference Δt_1 is determined by the curve of the estimated cost (Step 6). After cor-

HYBRID BIST TIME MINIMIZATION

rection, a new value of $COST_T^{E*}$ is found (point 2 on Figure 6.7). Based on $COST_T^{E*}$, a new J^* is found (Step 2), and a new $COST_M^{E*}$ is calculated (Step 3, point 2* in Figure 6.7). An additional iteration via points 3 and 3* can be followed in Figure 6.7.

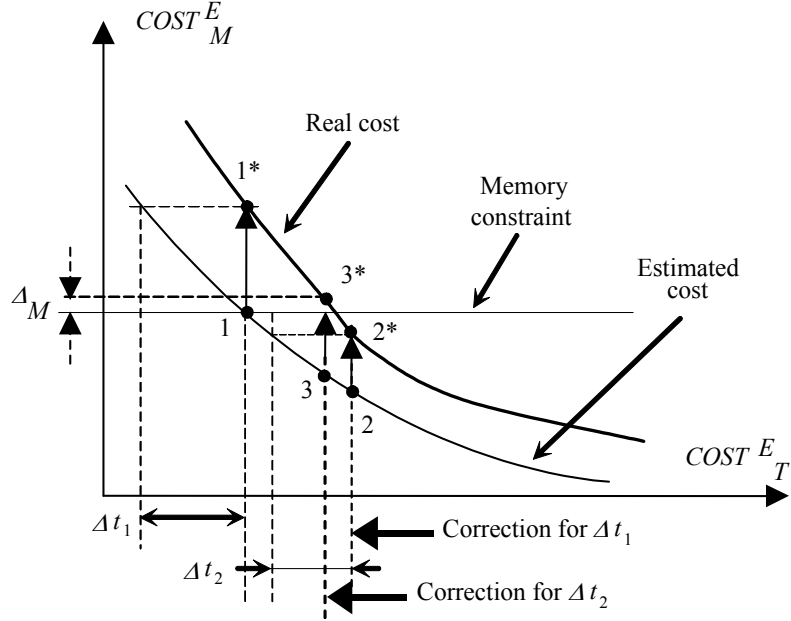


Figure 6.7. Minimization of the test length.

It is easy to see that Procedure 6.2 always converges. By each iteration we get closer to the memory constraints level, and also closer to the minimal test length at given constraints. However, the solution may be only near-optimal, since we only evaluate solutions derived from the estimated cost functions.

6.2.4. Experimental Results

We have performed experiments with several systems composed from different ISCAS benchmarks as cores. The results are presented in Table 6.1.

CHAPTER 6

In Table 6.1 we compare our approach where the test length is found based on estimates, with an approach where deterministic test sets have been found by manipulating the fault tables for every possible switching point between pseudorandom and deterministic test patterns. As it can be seen from the results, our approach can give significant speedup (more than one order of magnitude), while retaining acceptable accuracy (biggest deviation is less than 9% from the fault table based solution, in average 2.4%).

Table 6.1. Experimental results with combinatorial cores.

System	Number of cores	Memory Constraint (bits)	Fault table based approach		Our approach	
			Total Test Length (clocks)	CPU Time ¹ (seconds)	Total Test Length (clocks)	CPU Time (seconds)
S1	6	20 000	222	3772.84	223	199.78
		10 000	487		487	57.08
		7 000	552		599	114.16
S2	7	14 000	207	3433.10	209	167.3
		5 500	540		542	133.84
		2 500	1017		1040	200.76
S3	5	7 000	552	10143.14	586	174.84
		3 500	3309		3413	291.40
		2 000	8549		8 556	407.96

¹ CPU time for calculating all possible hybrid BIST solutions.

In Figure 6.8 we present the estimated cost curves for the individual cores and the estimated and real cost curves for one of the systems with 7 cores (different ISCAS85 benchmarks). We also show in this picture a test solution point for this system under given memory constraint that has been found based on our algorithm. In this example we have used a memory constraint $M_{LIMIT} = 5500$ bits. The final test length for this memory constraint is 542 clock cycles and that gives us a test schedule depicted in Figure 6.9. In Figure 6.10 we show another test schedule for the same system, when the memory constraints are different ($M_{LIMIT} = 14\,000$ bits).

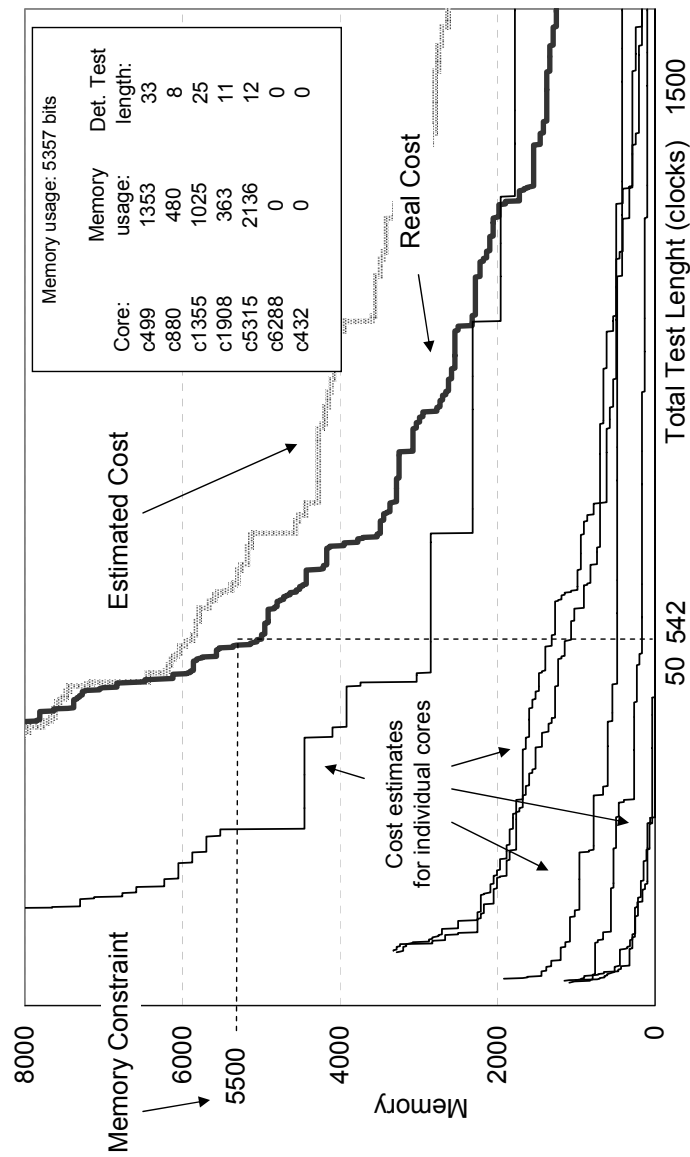


Figure 6.8. The final test solution for the system S2 ($M_{LMT} = 5\,500$).

CHAPTER 6

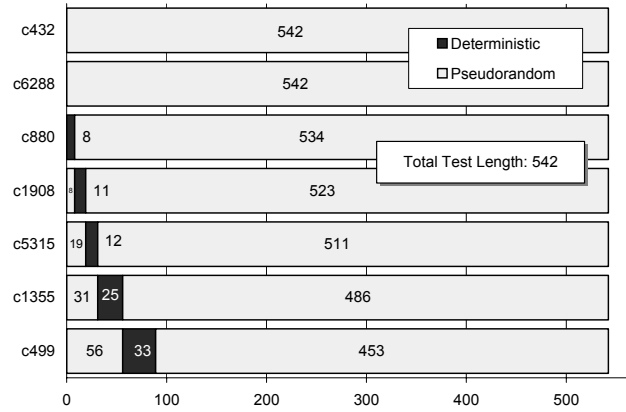


Figure 6.9. Test Schedule for the system S2 ($M_{LIMIT} = 5\,500$).

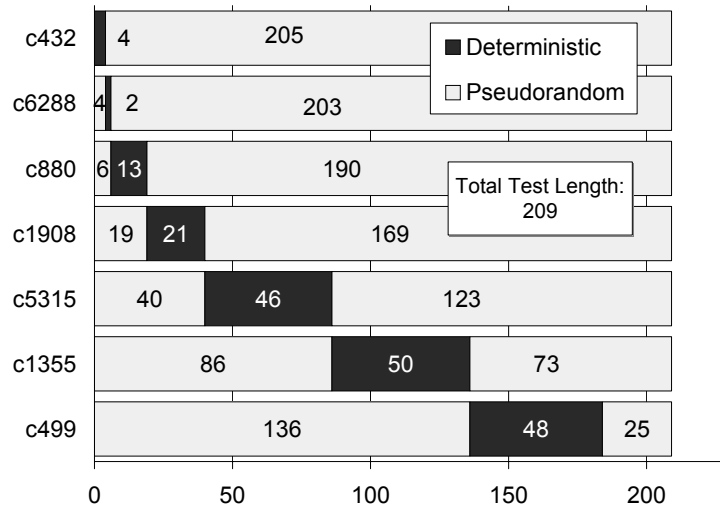


Figure 6.10. Test Schedule for the system S2 ($M_{LIMIT} = 14\,000$).

This approach can easily be extended to systems with full-scan sequential cores. The main difference lies in the fact that in case of a test-per-scan scheme, the test application is done via scan chains and one test cycle takes longer than one clock cycle. This is valid for both pseudorandom and deterministic test. As every

HYBRID BIST TIME MINIMIZATION

core contains scan chains with different lengths the analysis procedure has to account for this and switching from one core to another has to respect the local, core-level test cycles. In the following, the experimental results with systems where every individual core is equipped with Self-Test Using MISR and Parallel Shift Register Sequence Generator (STUMPS) [12] are presented [91].

While every core has its own STUMPS architecture, at the system level we assume the same architecture as described earlier: Every core's BIST logic is capable of producing a set of independent pseudorandom test patterns, i.e. the pseudorandom test sets for all the cores can be carried out simultaneously. The deterministic tests, on the other hand, can only be carried out for one core at a time, which means only one test access bus at the system level is needed. An example of a multi-core system with such a test architecture is given in Figure 6.11.

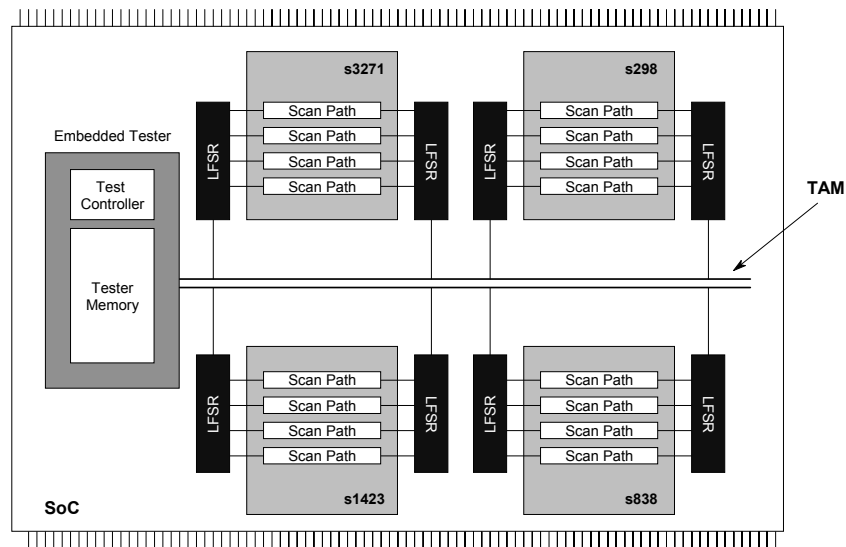


Figure 6.11. A core-based system example with the STUMPS test architecture.

CHAPTER 6

Experiments have been performed with several systems composed of different ISCAS'89 benchmarks as cores. All cores have been redesigned to include full scan path (one or several). The STUMPS architecture was simulated in software and for deterministic test pattern generation a commercial ATPG tool was used. The results are presented in Table 6.2.

Table 6.2. Experimental results with STUMPS architecture.

SOC	Number of Cores	Memory Constraint (bits)	Exhaustive Approach		Our Approach	
			Total Test Length (clocks)	CPU Time (sec.) ¹	Total Test Length (clocks)	CPU Time (sec.)
J	6	25 000	5750	57540	5775	270
		22 000	7100		7150	216
		19 000	9050		9050	335
K	6	22 000	5225	53640	5275	168
		17 000	7075		7075	150
		13 000	9475		9475	427
L	6	15 000	3564	58740	3570	164
		13 500	4848		4863	294
		12 200	9350		9350	464

¹ CPU time for calculating all possible hybrid BIST solutions.

In Table 6.2 we compare our approach where the test length is found based on estimates, with an exact approach where deterministic test sets have been found by a brute force method (repetitive use of test pattern generator) for every possible switching point between pseudorandom and deterministic test patterns. As it can be seen from the results, our approach can give significant speedup (several orders of magnitude), while retaining very high accuracy.

6.3. Broadcasting Based Hybrid BIST Architecture

In the previous section we analyzed systems where every core has its own dedicated BIST logic that is capable of producing a set of independent pseudorandom test patterns. This approach can be extended for multi-core systems where both combinatorial cores and sequential cores with full scan are used. This however may lead to high area overhead and may require redesign of the cores, as not all cores may be equipped from the beginning with self-test structures. Therefore, we have proposed a novel self-test architecture that is based on test pattern broadcasting [162]. In this approach, only a single pseudorandom test pattern generator is used and all test patterns are broadcasted simultaneously for all cores in the system. These patterns will be complemented with dedicated deterministic patterns for every individual core, if needed. These deterministic test vectors are generated during the development process and are stored in the system.

The deterministic test sequence is assembled, in general, from deterministic test sequences for each individual core $TD = \{TD_1, TD_2, \dots, TD_n\}$. Testing of all cores is carried out in parallel, i.e. all pseudorandom patterns as well as each deterministic test sequence TD_k is applied to all cores in the system. The deterministic test sequence TD_k is a deterministic test sequence generated only by analyzing the core $C_k \in S$. For the rest of the cores $C_j \in S$, $1 \leq j \neq k \leq n$ this sequence can be considered as a pseudorandom sequence. This form of parallel testing is usually referred to as test pattern broadcasting [114]. The width of the hybrid test sequence TH is equal to $MAXINP = \max\{INP_k\}$, $k=1, 2, \dots, n$, where INP_k is the number of inputs of the core C_k . For each deterministic test set TD_k , where $INP_k < MAXINP$, the not specified bits will be completed with pseudorandom data, so that the resulting test set TD_k^* can be applied in parallel to the other cores in the system as well. An example of such a hybrid test set is presented in Figure 6.12.

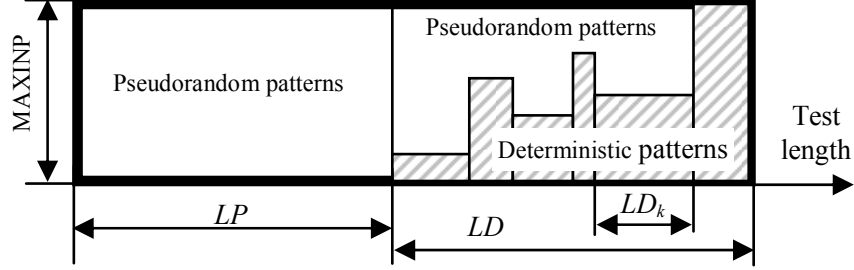


Figure 6.12. Hybrid test set for test pattern broadcasting.

In Figure 6.12, we denote with LP the length of the pseudorandom test set, with LD the length of the entire deterministic test set, and with LD_k the length of the deterministic test set of core C_k . Since some of the cores may be 100% testable by using only the pseudorandom test sequence and the deterministic test sequences of other cores, the deterministic test sequence TD_k for such a core C_k is not needed and $LD_k = 0$.

The memory size for storing the deterministic part of the hybrid test set is given by the following formula:

$$COST_M = \sum_{k=1}^n (LD_k * INP_k) \quad (6.5)$$

The main problem is to minimize the total length

$$LH = LP + \sum_{k=1}^n LD_k \quad (6.6)$$

of the hybrid test set $TH = \{TP, TD\}$ under given memory constraint $COST_M \leq COST_{M,LIMIT}$.

The problem of minimizing the hybrid BIST length at the given memory constraints for parallel multi-core testing is extremely complex. The main reasons of this complexity are the following:

- The deterministic test patterns of one core are used as pseudorandom test patterns for all other cores; unfortunately there will be $n*n$ relationships for n cores to analyze

HYBRID BIST TIME MINIMIZATION

to find the optimal combination; on the other hand the deterministic test sets are not readily available (see Algorithm 6.5, later in this section) and are calculated only during the analysis process;

- For a single core an optimal combination of pseudorandom and deterministic patterns can be found by rather straightforward algorithms; but as the optimal time moment for switching from pseudorandom to deterministic testing will be different for different cores the existing methods cannot be used and the parallel testing case is considerably more complex.
- For each core the best initial state of the LFSR can be found experimentally, but to find the best LFSR for testing all cores in parallel is a very complex and time consuming task.

To overcome the high complexity of the problem we will propose at first a straightforward algorithm for calculating TP and TD , where we neglect the optimal solutions for individual cores in favor of finding a near-optimal solution for the whole system. Thereafter we will propose a more sophisticated algorithm that uses test cost estimates to find the appropriate initial pseudorandom sequence $TP_{INITIAL}$, and based on the fault coverage of every individual core C_k , achieved by TP , finds an optimized (compacted) deterministic test sequence TD_k , thus reducing significantly the length of the final hybrid test set. In the following, the algorithms will be described in detail.

6.3.1. Straightforward Approach

To cope with the high complexity of the problem, we solve the test time minimization problem in three consecutive steps: first, we find an as good as possible initial state for the LFSR for all cores; second, we generate a deterministic test sequence if the 100% fault coverage cannot be reached by a pure pseudorandom test sequence for all cores or the required pseudorandom test sequence would be prohibitly long; and third, we update the test

CHAPTER 6

sequence by finding the quasi-optimal time moment for switching from parallel pseudorandom testing to parallel deterministic testing at the given memory constraint.

Finding the Initial State for the LFSR.

To find the best initial state for the parallel pseudorandom test generator, we carry out several experiments, with randomly chosen initial states, for all n cores. Within each experiment j we calculate for each core C_k the weighted length $LP_{k,j} * INP_k$ of the test sequence which achieves the maximal achievable fault coverage for the core C_k . Then, for all the experiments, we calculate the average weighted length

$$L_j = \frac{1}{n} \sum_{k=1}^n LP_{k,j} * INP_k \quad (2)$$

The best pseudorandom sequence is the one that gives the shortest L_j , $j = 1, 2, \dots, m$ (m is the total number of experiments). Let us call this initial pseudorandom test TP^0 .

Generation of the Initial Deterministic Test Set.

Suppose there are $k \leq n$ cores where maximal achievable fault coverage cannot be achieved with TP^0 because of the practical constraints to the pseudorandom test length. Let us denote this subset of cores with $S' \subseteq S$. Let us denote with FP_i^0 the fault coverage of the core C_i , achieved by TP^0 . Let us order the cores in S' as C_1, C_2, \dots, C_k , so that for each $i < j$, $1 \leq i, j \leq k$, we have $FP_i^0 \leq FP_j^0$. The deterministic patterns can be generated by using the following algorithm:

Algorithm 6.1:

1. Start with core C_i in S' , $i=1$.
2. Generate a deterministic test set TD'_i to complement TP^0 to increase the fault coverage FP_i^0 of the core C_i to 100%.

HYBRID BIST TIME MINIMIZATION

3. Fill the unused bits of TD'_i with pseudorandom data by continuing the pseudorandom test TP^0 . Denote this updated test by TD_i .
4. Broadcast the test TD_i for other cores in S' , fault simulate it for the cores in S' , and update the fault coverage FP_j^0 for other cores in S' .
5. Take the next core C_i in S' for $i = i + 1$.
6. If $i > k$, END.
7. If $FP_i^0 = 100\%$, go to Step 5 else go to Step 2.

By using Algorithm 6.1 an initial hybrid BIST sequence $TH^0 = \{TP^0, TD^0\}$ can be generated. This sequence guarantees 100% fault coverage for all cores in the system.

Definition 6.8: A pattern in a joint pseudorandom test sequence is called *efficient* if it detects at least one new fault for at least one core that is not detected by previous test patterns in the sequence nor by any pattern in the deterministic test sequence.

Optimization of the Test Sequence.

After using Algorithm 6.1 we have obtained a hybrid BIST sequence $TH^0 = \{TP^0, TD^0\}$ with length LH^0 , consisting of the pseudorandom part TP^0 with length LP^0 , and of the deterministic part TD^0 with length LD^0 .

In particular cases TD^0 may be an empty set.

Let us denote with $COST_M(TD^0)$ the memory cost of the deterministic test set TD^0 . We assume that the memory constraints are satisfied: $COST_M(TD^0) < COST_{M,LIMIT}$. In a opposite case, if $COST_M(TD^0) > COST_{M,LIMIT}$, the length of the pseudorandom sequence has to be extended and the second step of Algorithm 6.1 has to be repeated.

If $COST_M(TD^0) = COST_{M,LIMIT}$ the procedure is finished.

With optimization of TH^0 we mean the minimization of the test length LH^0 at the given memory constraints $COST_{M,LIMIT}$.

CHAPTER 6

It is possible to minimize LH^0 by shortening the pseudorandom sequence, i.e. by moving step-by-step efficient patterns from the beginning of TP^0 to TD^0 and by removing all other patterns between the efficient ones from TP^0 , until the memory constraints will become violated, i.e., $COST_M(TD^0) > COST_{M,LIMIT}$.

We cannot remove patterns with the same goal from the other end of TP^0 because the pseudorandom sequence will be extended and merged with the deterministic part TD^0 to update the free bits of deterministic test patterns generated by Algorithm 6.1 (step 3). In other words, by removing pseudorandom patterns from the end of the TP^0 would break the continuity of the pseudorandom test generation process on the border between TP^0 and TD^0 .

To find the efficient test patterns in the beginning of the TP^0 we have to fault simulate the whole test sequence TH^0 for all the cores in the opposite way from the end to the beginning. As a result of the fault simulation we get for each pattern the increments of fault coverage in relation to each core $\Delta = \{\Delta_1, \Delta_2, \dots, \Delta_n\}$. According to Definition 6.8, we call the pattern *efficient* if

$$\exists k, k = 1, 2, \dots, n : \Delta_k \neq 0$$

The optimization procedure will be carried out by using the following algorithm.

Algorithm 6.2:

1. Start with the first pattern P_i from the beginning of TP^0 , set $i = 1$.
2. If P_i is efficient, move it from TP^0 to TD^0 .
3. Recalculate the memory cost
 $COST_M(TD^0) = COST_M(TD^0) + COST_M(P_i)$.
4. If $COST_M(TD^0) < COST_{M,LIMIT}$ go to Step 5,
 else if $COST_M(TD^0) > COST_{M,LIMIT}$ go to Step 7,
 else go to Step 8.
5. Take the next pattern P_i in TP^0 , $i = i + 1$.

HYBRID BIST TIME MINIMIZATION

6. If P_i is not efficient,
 remove it from TP^0 , and go to Step 5;
 else go to Step 2.
7. Remove P_i from TD^0 back to TP^0 . Go to 10.
8. Take the next pattern P_i in TP^0 , $i = i + 1$.
9. If P_i is not efficient, remove it from TP^0 , and go to Step 8.
10. END: take P_i as the new beginning of the pseudorandom test sequence TP^0 .

As the result of the Algorithm 6.2 we create a new hybrid BIST sequence $TH = \{TP, TD\}$ with total length LH and with lengths $LP \leq LP^0$ and $LD \geq LD^0$ for the new pseudorandom and deterministic parts correspondingly. Due to removal of all non-efficient patterns $LP - LP^0 \gg LD^0 - LD$. Hence, the total length of the new hybrid BIST sequence will be considerably shorter compared to its initial length, i.e., $LH < LH^0$.

The memory constraints, according to the Algorithm 6.2, remain satisfied: $COST_M(TD) < COST_{M,LIMIT}$.

The described procedure does not guarantee an optimal test length, however, it is rather straightforward (similar to a greedy algorithm) and fast and therefore suitable for use in the design exploration process. The method can be used to find a cheap practical solution as well as for a fast reference for comparison with more sophisticated optimization algorithms.

Experimental Results

We have performed experiments with three systems composed of different ISCAS benchmarks as cores. The systems are presented in Table 6.3 (the lists of used cores in each system).

To show the importance of the first step of the procedure, i.e. the significance of the quality of the initial state of the LFSR, a comparison of the best and worst initial states of the LFSR for all 3 experimental systems has been carried out. The lengths of a

CHAPTER 6

complete pseudorandom test sequence (100% fault coverage), starting from the best and worst initial state, are depicted in Table 6.4. In case of system S3, the pseudorandom sequence was unacceptably long. Therefore, the pseudorandom test generation was interrupted and an initial set of deterministic test patterns was generated in order to achieve 100% fault coverage.

Table 6.3. Systems used for the experiments.

System name	S1 6 cores	S2 7 cores	S3 5 cores
List of used cores	c5315	c432	c880
	c880	c499	c5315
	c432	c880	c3540
	c499	c1355	c1908
	c499	c1908	c880
	c5315	c5315	
		c6288	

Table 6.4. Quality of different pseudorandom sequences.

System Name	The best initial state for the pseudorandom test		The worst initial state for the pseudorandom test	
	Pseudorandom test length (clocks)	Deterministic test length (clocks)	Pseudorandom test length (clocks)	Deterministic test length (clocks)
S1	2 520	0	23 482	0
S2	7 060	0	23 482	0
S3	14 524	26	25 000	33

The experimental results for three different systems are presented in Table 6.5. The lengths of the pseudorandom test sequence, the number of additional deterministic test patterns and the total length of the hybrid test sequence is calculated for three different memory constraints and for the best and worst initial states of the LFSR for all three systems. The CPU time needed for the analysis is presented as well.

HYBRID BIST TIME MINIMIZATION

Table 6.5. Experimental results.

System Name	Number of cores	Memory Constraint (bits)	The best initial state for the pseudorandom sequence			
			PR test length (clocks)	DET test length (clocks)	Total test length (clocks)	CPU time (sec)
S1	6	20 000	85	181	266	187, 64
		10 000	232	105	337	
		5 000	520	55	575	
S2	7	20 000	92	222	314	718.49
		10 000	250	133	383	
		5 000	598	71	669	
S3	5	20 000	142	249	391	221,48
		10 000	465	161	626	
		5 000	1 778	88	1866	

System Name	Number of cores	Memory Constraint (bits)	The worst initial state for the pseudorandom sequence			
			PR test length (clocks)	DET test length (clocks)	Total test length (clocks)	CPU time (sec)
S1	6	20 000	2 990	138	3128	228.67
		10 000	4 446	73	4519	
		5 000	5 679	40	5719	
S2	7	20 000	3 015	151	3166	969.74
		10 000	4 469	82	4551	
		5 000	5 886	49	5935	
S3	5	20 000	3 016	200	3216	318.38
		10 000	4 521	121	4642	
		5 000	8 604	72	8676	

For the first two systems S1 and S2 the cost of the procedure is determined only by the CPU time for the pseudorandom test pattern generation and by subsequent simulation of the test patterns for all cores in the system. For the third system, S3, the CPU time includes also the time needed to generate the additional deterministic test patterns.

The full overview about all possible hybrid BIST solutions for the three systems is presented in Figure 6.13, representing the

CHAPTER 6

memory cost as a function of the total test length. Based on these curves for an arbitrary memory constraint the corresponding total testing time can be found. The three constraints presented in Table 6.5 are also highlighted in Figure 6.13. It can be seen that after some certain length the memory cost will increase very fast when reducing the length of the test sequence further. This can be explained by the fact that in the beginning of the pseudorandom sequence nearly all test patterns are efficient, and nearly each pattern that is excluded from the pseudorandom part should be included into the deterministic part.

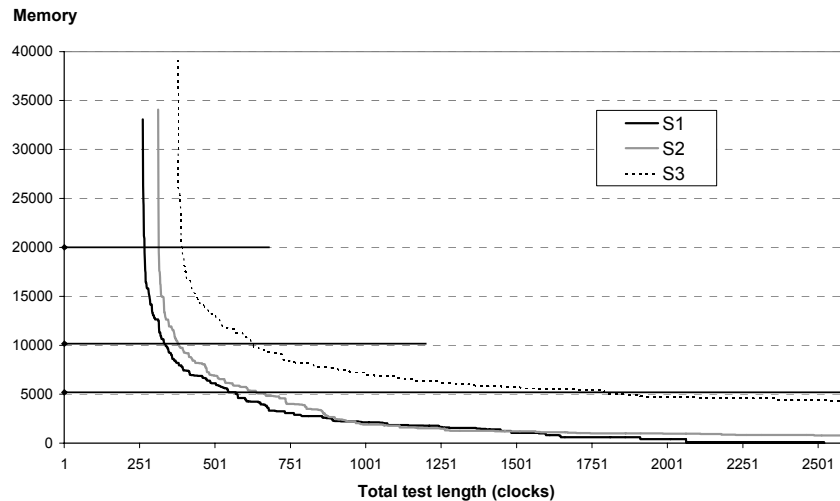


Figure 6.13. Memory usage as the function of the total test length for all three systems.

A comparison of the curves for the memory cost as a function of the total test length for the best and for the worst initial pseudorandom sequences is depicted for the system S2 in Figure 6.14. This illustrates the importance of choosing the best possible pseudorandom sequence for testing the system.

HYBRID BIST TIME MINIMIZATION

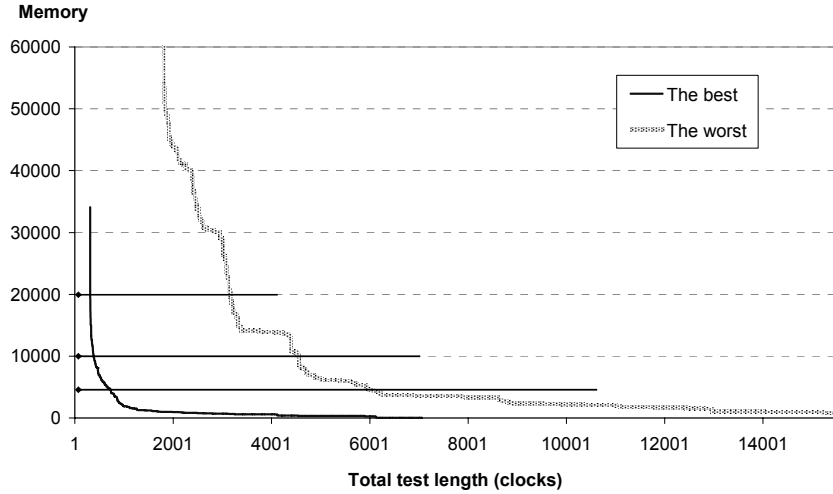


Figure 6.14. Memory usage as the function of the total test length for the best and the worst initial pseudorandom sequences.

6.3.2. Iterative Approach.

The disadvantage of the straightforward approach is that the deterministic test set TD is generated based on the initial test sequence $TP_{INITIAL}$ and is not minimized. Minimization of TD (test compaction) would be extremely difficult, since TD is assembled simultaneously for all cores in the system and individual deterministic tests for different cores TD_k are difficult to identify.

Therefore, we have developed a more advanced iterative algorithm that is based on analysis of different cost relationships as functions of the hybrid test structure to find the appropriate initial pseudorandom sequence and an optimized (compacted) deterministic test sequence [163]. The main tasks of the algorithm are as follows:

Algorithm 6.3: Test length minimization.

1. Find the best initial state for the LFSR that can generate the shortest common pseudorandom sequence $TP_{INITIAL}$, sufficient

CHAPTER 6

for testing simultaneously all the cores with maximum achievable fault coverage. Due to practical reasons the $TP_{INITIAL}$ might be unacceptably long and therefore an adequately long $TP'_{INITIAL}$ should be chosen and complemented with an initial deterministic test set $TD_{INITIAL}$ in order to achieve maximum achievable fault coverage and to satisfy the basic requirements for the test length.

2. Based on our estimation methodology (Section 6.2.2) find the length LD_k^E of the estimated deterministic test set TD_k^E and calculate the first iteration of the optimized test structure $TH^E = \{TP^*, TD^E\}$, so that the memory constraints are satisfied. TP^* denotes here a shortened pseudorandom sequence, found during the calculations.
3. Find the real total test length LH and the real memory cost $COST_M$ of the hybrid test sequence $TH = \{TP^*, TD\}$ for the selected pseudorandom sequence TP^* .
4. If the memory constraint is not satisfied, i.e., $COST_M > COST_{M,LIMIT}$, improve the estimation, choose a new pseudorandom sequence TP^* , and repeat step 3.
5. If the memory limit has not been reached, i.e., $COST_M < COST_{M,LIMIT}$, reduce the length of TH by moving efficient pseudorandom patterns from the pseudorandom test set to the deterministic test set.

The previously proposed straightforward algorithm (Algorithm 6.2), in fact, corresponds to the 5th step of this algorithm.

Test Set Generation Based on Cost Estimates

In this section we explain the first two steps of Algorithm 6.3. It is assumed that we have found the best configuration (polynomial and initial state) for the parallel pseudorandom test pattern generator. Let us call this an initial pseudorandom test sequence $TP_{INITIAL}$.

HYBRID BIST TIME MINIMIZATION

By knowing the structure of the hybrid test set TH the total hybrid test length LH at any given memory constraint $COST_M \leq COST_{M,LIMIT}$ could be found in a straightforward way. However, calculation of the exact hybrid test structure is a costly procedure, since it assumes that for each possible length of TP the deterministic test sets TD_k for each core should be calculated and compressed while following the broadcasting idea. This can be done either by repetitive use of the automatic test pattern generator or by systematically analyzing and compressing the fault tables. Both procedures are accurate but time-consuming and therefore not feasible for larger designs (see section 5.2).

To overcome the high complexity of the problem we propose an iterative algorithm, where the values of LD_k and $COST_{M,k}$ for the deterministic test sets TD_k can be found based on estimates. The estimation method, that is an extension of the method proposed for sequential hybrid BIST (see section 6.2.2), is based on the fault coverage figures of TD_k only, and does not require accurate calculations of the deterministic test sets for not yet detected faults.

The estimation method requires the following: a complete deterministic test set for every individual core, TD_k , together with fault simulation results of every individual test vector FD_k and fault simulation results of the pseudorandom sequence $TP_{INITIAL}$ for every individual core, FP_k . Let us denote with $TP_{INITIAL}(i)$ a pseudorandom sequence with length i .

The length of the deterministic test sequence $LD_k(i)$ and the corresponding memory cost $COST_{M,k}(i)$ for any length of the pseudorandom test sequence $i \leq LP$ can be estimated for every individual core with the following algorithm:

Algorithm 6.4:

For each $i = 1, 2, \dots, LD_k$:

1. Find fault coverage value $F(i)$ that can be reached by a sequence of pseudorandom patterns $TP_{INITIAL}(i)$.

CHAPTER 6

2. Find the highest integer value j , where $FD_k(j) \leq F(i)$. The value of j is the required length of the deterministic sequence TD_k to achieve fault coverage $F(i)$.
3. Calculate the estimated length of the deterministic test subsequence $TD_k^E(i)$ as $LD_k^E(i) = LD_k - j$. This is the estimated number of deterministic test patterns needed to complement the pseudorandom sequence $TP_{INITIAL}(i)$, so that the maximum achievable fault coverage can be achieved.

This algorithm enables us to estimate the memory requirements of the hybrid BIST solution for any length of the pseudorandom sequence for every individual core and by adding the memory requirements of all individual cores $C_k \in S$ for the entire system. In a similar manner, the length of the pseudorandom sequence LP for any memory constraint can be estimated and this defines uniquely the structure of the entire hybrid test set.

Minimization of the Hybrid Test Sequence

The memory cost estimation function helps us to find the length LP^* of the pseudorandom test sequence TP^* for the estimated hybrid test sequence $TH^E = \{TP^*; TD^E\}$. The real length LH of the estimated hybrid test sequence TH^E can be found with the following algorithm.

Algorithm 6.5:

1. Simulate the pseudorandom sequence TP^* for each core $C_k \in S$ and find a set of not detected faults $F_{NOT,k}$. Generate the corresponding deterministic test set TD'_k by using any ATPG tool. As a result, a preliminary real hybrid test set will be generated: $TH = \{TP^*; TD\}$.
2. Order the deterministic test set $TD' = (TD'_1, TD'_2, \dots, TD'_n)$ in such the way that for each $i < n$, $INP_i \leq INP_{i+1}$.
3. Perform the analysis of the test pattern broadcasting impact for $i = 2, 3, \dots, n$:

HYBRID BIST TIME MINIMIZATION

- Calculate a set of not detected faults $F_{NOT,i}$ for the test sequence $(TP^*; TD'_1, TD'_2, \dots, TD'_{i-1})$;
- Compress the test patterns in TD'_i with respect to $F_{NOT,k}$ by using any test compacting tool.

As a result of Algorithm 6.5, the real hybrid test sequence $TH = \{TP^*; TD\} = \{TP^*; TD_1, TD_2, \dots, TD_n\}$ will be generated. The length of the resulting sequence $LH \leq LH^E$ as deterministic test patterns of one core, while broadcasted to the other cores, may detect some additional faults. In general, $LD_k \leq LD_k^E$ for every $k = 2, 3, \dots, n$.

The length of the deterministic test sequence, generated with Algorithm 6.5, can be considered as a near-optimal solution for the given TAM structure, for all the cores. Ordering of the deterministic test sets, according to the step 2 in Algorithm 6.5 has the following result: the larger the number of inputs of core C_k the more patterns will be broadcasted to C_k from other cores, and hence the chances to reduce its own deterministic test set TD_k are bigger and larger amount of memory can be reduced.

After finding the real deterministic test sequence according to Algorithm 6.5, the following three situations may occur:

1. If $COST_M > COST_{M,LIMIT}$ a new iteration of the cost estimation should be carried out. The initial estimation of the pseudorandom test sequence length LP should be updated, and a new cost calculation, based on Algorithm 6.5, should be performed (see *Iterative Procedure*).
2. If $COST_M = COST_{M,LIMIT}$ the best possible solution for the given pseudorandom sequence TP^* is found. We have hence $TH = \{TP^*; TD_1, TD_2, \dots, TD_n\}$.
3. If $COST_M < COST_{M,LIMIT}$ the test length minimization process should be continued by moving efficient test patterns from the pseudorandom test set to the deterministic sequence.

CHAPTER 6

In the following, possible steps for further improvement are described in detail.

Iterative Procedure

Let us suppose that our first estimated solution, based on pseudorandom test sequence TP , with length LP , produces a test structure with total memory requirement “Real $COST_M$ ” higher than accepted (see Figure 6.15). A correction of the estimated solution should be made $LP_{NEW} = LP + \Delta LP$ and a new solution “New real $COST_M$ ” should be calculated based on Algorithm 6.5. These iterations should be repeated until the memory constraint $COST_M \leq COST_{M,LIMIT}$ is satisfied.

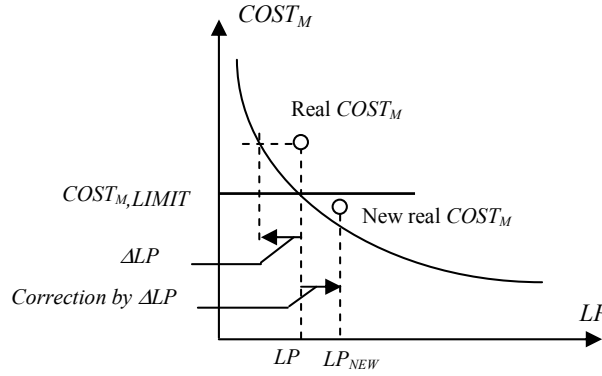


Figure 6.15. Iterative cost estimation.

It should be mentioned that Algorithm 6.5 is the most expensive procedure of the whole approach, due to repetitive use of ATPG and test compaction tools. Therefore, we cannot start with an arbitrary initial solution and an accurate estimation procedure minimizes the number of iterations considerably.

Total Test Length Reduction

Suppose that the real cost of the found solution is below the memory constraint $COST_M < COST_{M,LIMIT}$. There are two alternatives for further reduction of the test length:

HYBRID BIST TIME MINIMIZATION

1. Additional iterations by using Algorithm 6.5 to move the solution as close to the memory limit $COST_{M,LIMIT}$ as possible. As mentioned earlier, Algorithm 6.5 is an expensive procedure and therefore recommended to use as little as possible.
2. It is possible to minimize the length of the hybrid test sequence TH by shortening the pseudorandom sequence, i.e. by moving step-by-step efficient patterns from the beginning of TP to TD and by removing all other patterns between the efficient ones from TP , until the memory constraint $COST_M \leq COST_{M,LIMIT}$ gets violated. This procedure is based on the algorithm used for straightforward optimization of the parallel hybrid BIST. As a result, the final hybrid test sequence is created: $TH_F = \{TP_F; TD_F\} = \{TP_F; TD_1, TD_2, \dots, TD_n, \Delta TD\}$ where ΔTD is a set of efficient test patterns moved from TP to TD . This will lead to the situation where the length of the pseudorandom sequence has been reduced by ΔLP and the length of the deterministic test sequence has been increased by ΔLD . The total length LH_F of the resulting hybrid test $TH_F = \{TP_F; TD_F\}$ is shorter, $LH_F < LH$, because in general $\Delta LD \ll \Delta LP$ (not every pattern in the pseudorandom test set is efficient).

The final hybrid BIST test structure $TH_F = \{TP_F; TD_F\}$ with the total length LH_F is represented in Figure 6.16.

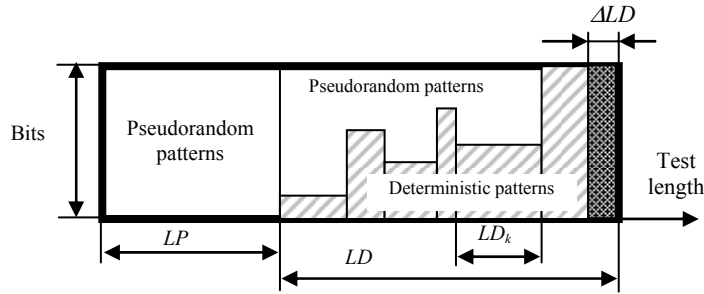


Figure 6.16. Final hybrid test structure.

CHAPTER 6

The accuracy of the solution (proximity of the total length LH_F to the global minimum LH_{MIN}) for the given initial pseudorandom sequence $TP_{INITIAL}$ can be estimated by the length of ΔLD , assuming that the deterministic test set was optimally compacted. Since efficient patterns, moved from TP to TD , were not taken into account during the compaction procedure for TD' (Algorithm 6.5) the new deterministic test sequence $TD_F = \{TD_1, TD_2, \dots, TD_n, \Delta TD\}$ is not optimal and should be compacted as well. However, since TD' was compacted optimally, the upper bound of the gain in test length cannot be higher than ΔLD . Hence, the difference between the exact minimum LH_{MIN} and the current solution LH_F for the given pseudorandom sequence $TP_{INITIAL}$ cannot be higher than $LH_F - LH_{MIN} = \Delta LH$.

Experimental Results

We have performed experiments with three systems composed from different ISCAS benchmarks as cores. In Table 6.6 the experimental results for these three systems under different memory constraints are presented. In column 3 the estimated length of the hybrid test structure, found by using Algorithm 6.4, is given. For the systems S1 and S2 only a single iteration was needed (the estimation was rather accurate), for the system S3 two iterations were needed. In columns 4 and 5 the real length of the hybrid test sequence, found by using Algorithm 6.5 with cost estimates from column 3, is given. In column 4 the total length of the pseudorandom and deterministic test sequences and the memory cost in bits is given without taking into account broadcasting effect (step 1 in Algorithm 6.5), and in column 5 together with broadcasting (steps 2 and 3 in Algorithm 6.5). In columns 6-8 the results of the final optimization are depicted: the final length of the pseudorandom sequence (column 6), the final length of the deterministic sequence (column 7), and the final length of the hybrid test set together with memory requirements in bits (column 8). In column 7 the first component represents the result of the Algorithm 6.5, and the second component represents the

HYBRID BIST TIME MINIMIZATION

last improvement, when efficient patterns were moved from the pseudorandom part to the deterministic part.

In Table 6.7 the results are compared with the straightforward approach (Section 6.3.1). The length of the pseudorandom test sequence (columns 3, 7), deterministic test sequence (columns 4, 8) and the hybrid test sequence (columns 5, 9) together with required CPU time (columns 6, 10) are compared. As it can be seen, the improved algorithm gives a noteworthy reduction of the test length while the analysis time is approximately the same.

In Figure 6.17 the estimated memory cost as the function of the total test length for different cores in system S2 together with the estimated total memory cost are depicted. For comparison, the real costs values for 4 different test lengths are shown as well. As it can be seen the accuracy of the estimation procedure is rather good.

6.4. Conclusions

This chapter presented algorithms for SOC test time minimization with several different implementations of the hybrid BIST architecture. The algorithms are based on the analysis of different cost relationships as functions of the hybrid BIST structure. We have proposed straightforward algorithms and more sophisticated iterative heuristics for this purpose. Our methods can minimize test time under given test memory constraint for test-per-clock and test-per-scan schemes. The optimization algorithms have been presented together with experimental results to demonstrate their efficiency.

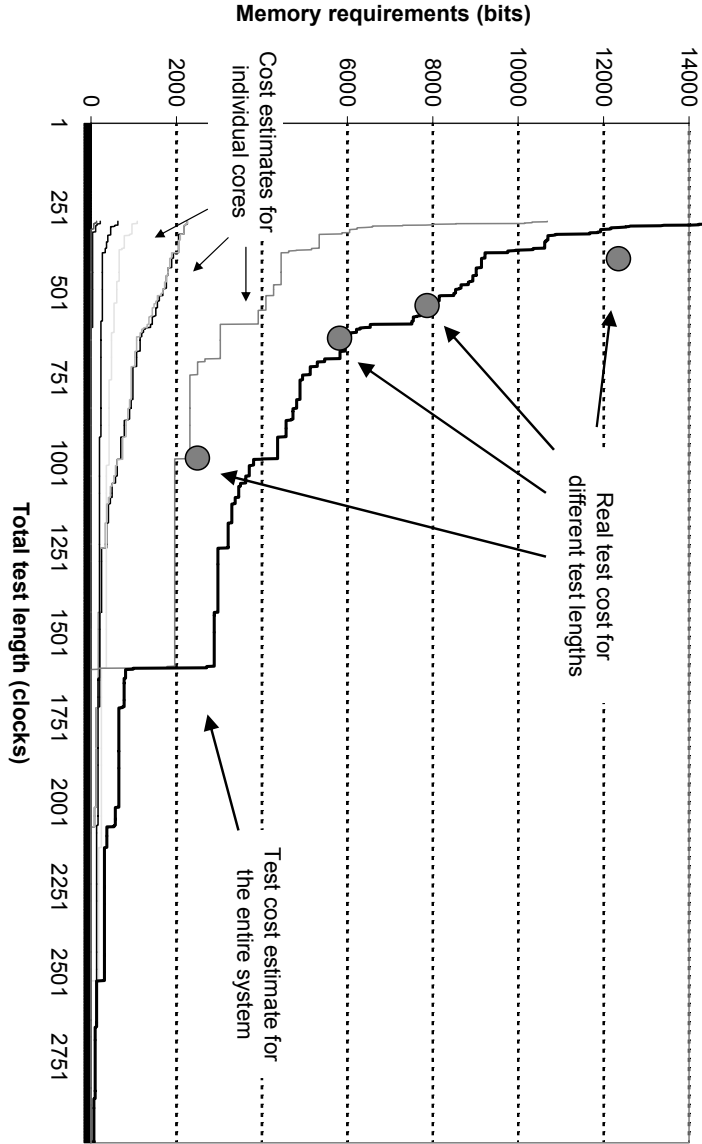


Figure 6.17. Comparison of estimated and real test costs.

HYBRID BIST TIME MINIMIZATION


System	Memory constraint (bits)	Estimated initial test length (clocks)	Calculated test structure		Final test structure		
			Initial (without broadcasting)	With broadcasting	PR length (clocks)	DET length (clocks)	Total length
S1 (6 cores)	10000	604	596 clocks 8660 bits	548 clocks 4500 bits	145	58+49 = 107	252 clocks 9891 bits
	10000	374	399 clocks 12345 bits	335 clocks 8294 bits	163	110+14 = 124	287 clocks 9852 bits
S2 (7 cores)	5000	741	740 clocks 4197 bits	717 clocks 3117 bits	469	51+18 = 69	538 clocks 4979 bits
	3000	1251	1245 clocks 1547 bits	1240 clocks 1342 bits	783	23+19 = 42	825 clocks 2995 bits
S3 (5 cores)	10000	1st iteration	412 clocks 13285 bits	379 clocks 10047 bits	 Need to make another iteration		
		2nd iteration	489 clocks 10952 bits	466 clocks 8756 bits			
					262	130+10 = 140	402 clocks 9919 bits

Table 6.6. Experimental data from three systems.

CHAPTER 6

System	Memory constraint (bits)	Straightforward approach				Our approach				Comparison	
		PR length (clocks)	DET length (clocks)	Total length (clocks)	CPU time (sec)	PR length (clocks)	DET length (clocks)	Total length (clocks)	CPU time (sec)	Total test length	CPU time
S1 (6 cores)	10000	232	105	337	187,64	145	58+49 = 107	252	289,73	-25.2%	+54.5%
	10000	250	133	383	718,49	163	110+14 = 124	287	1093,5	-25.1%	+52.2%
	5000	598	71	669		469	51+18 = 69	538	1124,4	-19.6%	+56.5%
S2 (7 cores)	3000	819	48	867	221,48	783	23+19 = 42	825	1109,4	-4.8%	+54.4%
	10000	465	161	626		262	130+10 = 140	402	334,28	-35.8%	+51.1%
S3 (5 cores)											

Table 6.7. Comparison with straightforward approach.

Chapter 7

Hybrid BIST Energy Minimization

7.1. Introduction

The current trend in consumer electronics can be described with mobility and portability. Such devices are mostly battery-driven and one of the most important design parameters is thus battery life-time. Therefore, such systems have to be designed with careful consideration of the energy dissipation.

There are numerous methods tackling power dissipation during the normal operation mode. However, such devices are frequently undergoing periodic self-tests and we have seen an advent of low-power and low-energy testing methods. These methods range from pattern reordering to clocking scheme modifications. At the same time, our hybrid BIST concept offers us a straightforward possibility for reducing the circuit's switching activity and consequently for energy optimization.

CHAPTER 7

In this chapter we propose two heuristics that try to minimize the total switching energy without exceeding the assumed test memory constraint. The solutions are obtained by modifying the ratio of pseudorandom and deterministic test patterns for every individual core such that the total energy dissipation is minimized.

7.2. Hybrid BIST and Possibilities for Energy Reduction

For portable systems, one of the most important test constraints is the total amount of on-chip test memory. In the previous chapters we introduced several methods, based on different test architectures, for test time minimization. Those algorithms were able to find a shortest possible test time under given test memory constraint for test-per-clock and test-per-scan schemes. Our algorithms, however, do not require explicit specification, whether the deterministic test set has to be stored in the ATE or on-chip memory (ROM). In the latter case hybrid BIST solutions can be used not only for manufacturing test but also for periodical field maintenance tests in portable devices.

In the following we have assumed a hybrid BIST test architecture where all cores have their own dedicated BIST logic that is capable of producing a set of independent pseudorandom test patterns. The deterministic tests, on the other hand, are applied from an on-chip memory, one core at a time (See Section 4.4.2).

If the objective is only test time minimization and power/energy is not taken into account then the shortest test schedule for such a test architecture, is the one where all cores are tested concurrently and have the same tests lengths, as was explained in Section 6.2 and illustrated in Figure 6.9. It is important to note that the exact composition of pseudorandom and deterministic test patterns defines not only the test length and test memory requirements but also the energy consumption. In gen-

HYBRID BIST ENERGY MINIMIZATION

eral, since a deterministic test pattern is more effective in detecting faults than a pseudorandom pattern, using more deterministic test patterns for a core will lead to a short test sequence, and consequently less energy on the average case. However, the total number of deterministic test patterns is constrained by the test memory requirements, and at the same time, the deterministic test patterns of different cores of a SOC have different energy and fault detection characteristics. Namely, some cores might require only few test patterns, while these patterns have large memory requirements, due to big number of inputs or excessively long scan chains. These few patterns might generate less switching activity than a long sequence of test patterns with smaller memory requirements. A careful trade-off between the deterministic pattern lengths of the cores must therefore be made in order to produce a globally optimal solution. Moreover, as deterministic test patterns are stored in the memory, energy dissipation during memory access and test data transportation has to be taken into account as well.

In a hybrid BIST approach the test set is composed of pseudorandom and deterministic test patterns, where the ratio of these patterns is defined by different design constraints, such as test memory and test time. From a energy perspective, different cores have different energy dissipation while applying the same amount of test patterns. Furthermore, the pseudorandom and deterministic test sequences for the same core have different power characteristics. Therefore, for total energy minimization it is important to find, for every individual core, such a ratio of the pseudorandom and deterministic test patterns that leads to the overall reduction of switching energy. At the same time the basic design constraints, such as test memory, should not be violated. In the following some basic definitions together with the problem formulation are given.

7.3. Basic Definitions and Problem Formulation

Let us note that according to Definition 6.5, $J = (j_1, j_2, \dots, j_n)$ is called the *characteristic vector* of a hybrid test set $TH = \{TH_1, TH_2, \dots, TH_n\}$, where $j_k \in J$ is the length of the pseudorandom test sequence $TP_k \subseteq TH_k$.

Definition 7.1: Let us denote with $M_k(j_k)$ and $E_k(j_k)$ respectively, the memory cost and energy cost of the hybrid BIST set $TH_k = \{TP_k, TD_k\}$ of the core $C_k \in S$ as functions of its pseudorandom test sequence with length j_k ($k=1, 2, \dots, n$).

Note that it is very time consuming to calculate the exact values of $M_k(j_k)$ and $E_k(j_k)$ for any arbitrary hybrid BIST set TH_k , since it requires exact calculation of the corresponding hybrid test set which is an expensive procedure (see Chapter 5.2). To overcome the problem we propose to use a power estimation method that is based only on a few critical point calculations.

Definition 7.2: Let us denote with $M(J)$ and $E(J)$ respectively the memory cost and energy cost of the corresponding hybrid BIST set TH with characteristic vector J . These costs can be calculated using the following formulas:

$$M(J) = \sum_{k=1}^n M_k(j_k) \quad E(J) = \sum_{k=1}^n E_k(j_k) \quad (3)$$

A hybrid BIST set $TH = \{TH_1, TH_2, \dots, TH_n\}$ for a system $S = \{C_1, C_2, \dots, C_n\}$ consists of hybrid BIST sets TH_k for each individual core C_k . In the proposed approach we assume that the pseudorandom components of the TH are going to be scheduled in parallel, while the deterministic components of the TH , have to be scheduled in sequence.

The objective can be thus formulated as finding a hybrid test set TH with a characteristic vector J for the system S , so that $E(J)$ is smallest possible and the memory constraint $M(J) \leq M_{LIMIT}$ is satisfied.

7.3.1. Parameter Estimation

For hybrid BIST energy minimization at a given memory constraint we have to obtain values of $M_k(j_k)$ and $E_k(j_k)$ for every core $C_k \in S$ and for any possible j_k value. This would give us a possibility to compare memory and energy values of the different alternatives. However, the procedure to calculate the cost functions $M(J)$ and $E(J)$ is very time consuming, since it assumes that the deterministic test sets TD_k for all possible values of the characteristic vector J are available. This means that for each possible pseudorandom test TP_k , a set of not yet detected faults $F_{NOT}(TP_k)$ should be calculated, and the needed deterministic test set TD_k has to be found. This is a time-consuming task and therefore not feasible for larger designs.

To overcome the complexity explosion problem we propose an iterative algorithm, where the costs $M(J)$ and $E(J)$ for the deterministic test sets TD_k are calculated based on estimates in a similar way as described in Section 6.2.2. The estimation method is based on fault coverage figures and does not require accurate calculations of the deterministic test sets for not yet detected faults. For memory estimation we need fault simulation results of complete pseudorandom and deterministic test sequences. For energy estimation the energy simulation results for the same test sets are used and we have used the number of signal switches as a quantitative measure for power dissipation.

7.4. Heuristic Algorithms for Hybrid BIST Energy Minimization

To minimize the energy consumption at the given memory constraint we have to create a hybrid test TH with characteristic vector J for the system S , so that $E(J)$ is minimal at the constraint $M(J) \leq M_{LIMIT}$.

To solve this complex combinatorial task we have proposed two fast heuristic algorithms: Local Gain Algorithm and Average

CHAPTER 7

Gain Algorithm [164]. Both are based on the estimation methodology described in Section 6.2.2 (Procedure 6.1).

7.4.1. Local Gain Algorithm

The main idea of this algorithm is to start with pure deterministic test sets $TH_k = \{TP_k = \emptyset, TD_k^F\}$ for all cores $C_k \in S$. Next, the deterministic test patterns are gradually substituted by corresponding sequences of pseudorandom patterns $PR_i \subseteq TP_k^F$ until the memory constraints are satisfied. For every deterministic test pattern substitution a core $C_k \in S$ with maximum memory-power ratio ($\Delta M_{k,i} / \Delta P_{k,i}$) is selected. Here $\Delta M_{k,i}$ corresponds to the estimated memory gain when deterministic test pattern $DP_i \in TD_k^F$ is removed from the memory, and $\Delta P_{k,i}$ corresponds to the estimated increase in energy consumption by the sequence of pseudorandom patterns $PR_i \subseteq TP_k^F$ that are substituting the deterministic test pattern $DP_i \in TD_k^F$. In other words, at any iteration of the algorithm we always select a core that provides the best local gain in terms of $\Delta M_{k,i} / \Delta P_{k,i}$ and substitute in the hybrid test set of this core one or several deterministic test pattern with appropriate number of pseudorandom patterns. The number of inserted pseudorandom test patterns is calculated so that the fault coverage of the core should not be reduced. However, in some certain situations this might not be possible (due to random resistant faults) and therefore different deterministic test pattern should be chosen for substitution.

Let us introduce the following additional notations: M – current memory cost, L – current pseudorandom test length and M_{LIMIT} – memory constraint. The algorithm starts with initial values: $L = 0$, and $M = M(TD_1^F) + M(TD_2^F) + \dots + M(TD_n^F)$ where $M(TD_k^F)$ is memory cost of the complete deterministic test set of core $C_k \in S$. Initially: $TH_k = \{TP_k = \emptyset, TD_k^F\}$.

Algorithm 7.1: Local Gain Algorithm

1. Select core $C_k \in S$ where $\Delta M_{k,i} / \Delta P_{k,i} = \max$;

HYBRID BIST ENERGY MINIMIZATION

2. Remove $DP_{k,i} \in TD_k$ from TD_k , estimate the needed PR_i and include PR_i into TP_k .
3. Update the current memory cost: $M = M - \Delta M_{k,i}$
4. If $M > M_{LIMIT}$ then go to 1
5. END.

The algorithm is illustrated with the example given in Figure 7.1. We start from an all-deterministic solution. At every step we calculate the memory-power ratio for all cores if one deterministic test pattern (denoted as white boxes in Figure 7.1a) would be replaced with pseudorandom patterns. Thereafter the core with highest $\Delta M_k / \Delta P_k$ value is selected and a deterministic test pattern in this core's test set is replaced with a set of pseudorandom patterns. In our example Core 3 was selected (Figure 7.1b). At the end of every step we can calculate new memory (M) and power (P) values for the entire system. This procedure is repeated until $M \leq M_{LIMIT}$.

7.4.2. Average Gain Algorithm

A second heuristic proposed by us is called Average Gain Algorithm. The main idea of the Average Gain Algorithm is to guide the selection of cores based on the highest average ratio of $\Delta M_k / \Delta P_k$ over all iterations of the algorithm. Here ΔM_k denotes the estimated memory gain from the beginning of the algorithm, including the selected substitution, for the core $C_k \in S$, and ΔP_k denotes the estimated increase of energy consumption for the same core from the beginning of the algorithm, including the current selected substitution.

The algorithm starts again with initial values: $L = 0$, and $M = M(TD_1^F) + M(TD_2^F) + \dots + M(TD_n^F)$ where $M(TD_k^F)$ is the memory cost of the complete deterministic test set of the core $C_k \in S$. Initially: $TH_k = \{TP_k = \emptyset, TD_k^F\}$. For all cores $\Delta M_k = \Delta M_{k,1}$, and $\Delta P_k = \Delta P_{k,1}$, and for all cores $i = 1$.

CHAPTER 7

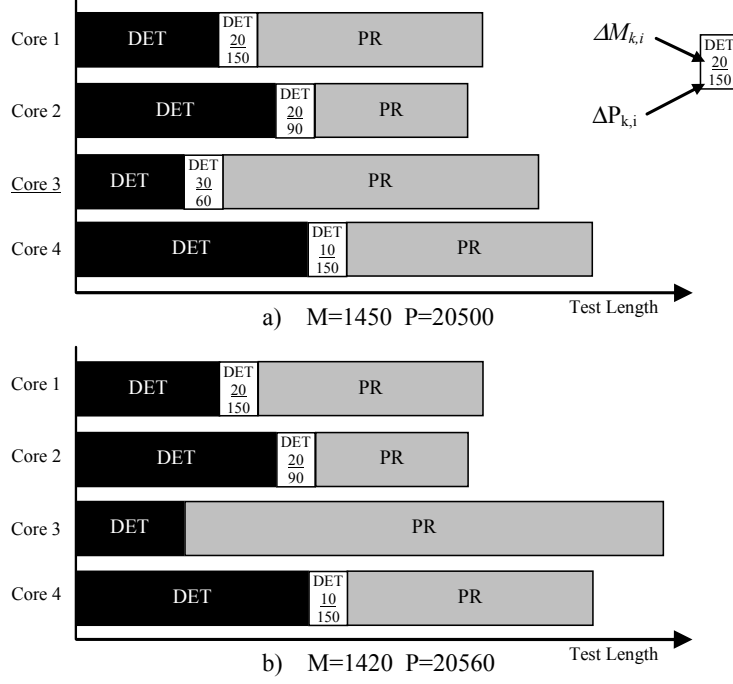


Figure 7.1. Local Gain Algorithm.

Algorithm 7.2: Average Gain Algorithm

1. Select the core $C_k \in S$ where $\Delta M_{k,i} / \Delta P_{k,i} = \max$;
2. Remove $DP_{k,i} \in TD_k$ from TD_k , and include PR_i into TP_k .
3. Update the current memory cost: $M = M - \Delta M_{k,i}$.
4. Update the total memory cost for the selected core: $\Delta M_k = \Delta M_k - \Delta M_{k,i+1}$.
5. Update the total power consumption for the selected core: $\Delta P_k = \Delta P_k + \Delta P_{k,i+1}$.
6. Update for the selected core $i = i + 1$.
7. If $M > M_{LIMIT}$ then go to 1
8. END

HYBRID BIST ENERGY MINIMIZATION

The main difference between Algorithm 7.1 and Algorithm 7.2 is that Algorithm 7.1 takes into account only the immediate effect of the test pattern substitution. Algorithm 7.2, on the other hand, takes into account the entire history of pattern substitutions.

Both Algorithm 7.1 and Algorithm 7.2 create hybrid BIST solution $TH_k = \{TP_k, TD_k\}$ where energy consumption is reduced with respect to the given memory constraint. After obtaining a solution, the cost estimates can be improved and another, better, solution can be calculated. For this purpose the previously proposed iterative procedure (see Section 6.2.3) can be used. This transfers a solution that was generated based on estimates to the solution that is calculated based on real test sets. The outcome of this algorithm is the final solution: amount of pseudorandom and deterministic test patterns for every individual core such that the system memory constraint is satisfied.

7.5. Experimental Results

We have performed experiments with different designs containing the ISCAS'89 benchmarks as cores. The complexity of these designs ranges from system with 6 cores to system with 20 cores. All cores were redesigned in order to include a scan chain. For simplicity we assumed that all flip-flops are connected into one single scan chain. For the BIST part a STUMPS architecture was used.

In Table 7.1 we have listed the results for every system with three different memory constraints. We have listed results from [92], which provide shortest possible test length without considering energy consumption. Our two algorithms (Local Gain Algorithm is denoted with A1 and Average Gain Algorithm with A2) are both considered with the iterative improvement mentioned at the end of the previous section. Finally, for comparison, results obtained with a simulated annealing (SA) based optimization are

CHAPTER 7

also reported. In every solution, the minimized test time solution from [92] has been taken as a baseline (100%) and every solution is compared against this result.

Simulated annealing is a generic probabilistic meta-algorithm for the global optimization problem, namely locating a good approximation to the global optimum of a given function in a large search space [102]. In the SA method, each point s of the search space is compared to a state of some physical system, and the function $E(s)$ to be minimized is interpreted as the internal energy of the system in that state. Therefore, the goal is to bring the system, from an arbitrary initial state, to a state with the minimum possible energy. At each step, the SA heuristic considers some neighbors of the current state s , and probabilistically decides between moving the system to state s' or staying put in state s . The probabilities are chosen so that the system ultimately tends to move to states of lower energy. Typically, this step is repeated until the system reaches a state, which is good enough for the application, or until a given computation budget has been exhausted.

As shown in Table 7.1, both proposed algorithms lead to reduced energy solutions (in some cases up to with 52% reduction of the total switching activity) compared to the solution where only the test time was minimized [92]. When compared to the simulated annealing algorithm our heuristics have significantly lower execution time, while maintaining acceptable accuracy.

To understand the impact of our algorithms on the test length we have also collected these figures and reported them in Table 7.1. As can be expected in all these solutions generated by our techniques the test time has increased compared to the technique which targets towards test length minimization [92]. Nevertheless, if the main objective is to reduce energy dissipation during the test mode (for example in portable devices) the relatively small increase of the test length is tolerable.

HYBRID BIST ENERGY MINIMIZATION

Table 7.1. Experimental results.

Alg.	M _{LIMIT}	Energy (switches)	Comp. to [92] (%)	Test Length (clocks)	Comp. to [92] (%)	CPU Time (sec)
System 1 – 6 cores						
[92]	1500	2588822	100.00%	24689	100.00%	8.41
A1		1281690	49.51%	31619	128.07%	11.09
A2		1281690	49.51%	31619	128.07%	6.64
SA		1240123	47.90%	31619	128.07%	5326.24
[92]	2500	635682	100.00%	6726	100.00%	24.61
A1		426617	67.11%	10559	156.99%	14.23
A2		446944	70.31%	10679	158.77%	4.84
SA		409576	64.43%	10529	156.54%	2944.26
[92]	3000	717026	100.00%	7522	100.00%	26.51
A1		265282	37.00%	8126	108.03%	36.31
A2		286883	40.01%	8129	108.07%	26.96
SA		241123	33.63%	8153	108.39%	1095.21
System 2 – 6 cores						
[92]	1700	6548659	100.00%	52145	100.00%	12.05
A1		5502763	84.03%	70331	134.88%	12.49
A2		5318781	81.22%	70331	134.88%	4.28
SA		4747498	72.50%	83865	160.83%	3805.23
[92]	3000	2315958	100.00%	19208	100.00%	20.21
A1		1998390	86.29%	23774	123.77%	7.66
A2		1861844	80.39%	24317	126.60%	18.79
SA		1845022	79.67%	28134	146.47%	5032.05
[92]	4700	893184	100.00%	8815	100.00%	21.47
A1		742462	83.13%	9537	108.19%	26.45
A2		746479	83.58%	9537	108.19%	55.09
SA		723817	81.04%	12596	142.89%	3654.02
System 3 – 20 cores						
[92]	5000	12830419	100.00%	40941	100.00%	47.49
A1		9242890	72.04%	70331	171.79%	51.43
A2		9839005	76.68%	70331	171.79%	40.49
SA		7367201	57.42%	60495	147.76%	29201.96
[92]	7000	6237211	100.00%	20253	100.00%	53.39
A1		4039622	64.77%	31619	156.12%	73.58
A2		4223263	67.71%	52145	257.47%	14.36
SA		3500894	56.13%	31919	157.60%	20750.03
[92]	10000	4686729	100.00%	15483	100.00%	45.37
A1		1719726	36.69%	17499	113.02%	115.53
A2		1815129	38.73%	17554	113.38%	90.52
SA		1606499	34.28%	17992	116.20%	14572.33

7.6. Conclusions

In this chapter we have proposed two heuristics for test energy reduction for hybrid BIST. Both algorithms modify the ratio between pseudorandom and deterministic test patterns. We have also proposed a fast estimation mechanism for the modification of this ratio together with an iterative procedure for transforming the estimated results to the real results. Experimental results have shown the efficiency of these heuristics for energy reduction under test memory constraints.

Chapter 8

Hybrid BIST in an Abort-on-First-Fail Environment

8.1. Introduction

Many different methods have been researched for test cost reduction. They range from test set reduction techniques to test scheduling heuristics. In the previous chapters we described some methods for hybrid test set generation, which serve the purpose of test cost reduction.

In a production test environment the test process is stopped as soon as a fault is detected. This approach, called *abort-on-first-fail* (AOFF), leads to reduced test costs as faulty dies can be eliminated before completing the entire test flow. Therefore, the likelihood of defects in different parts of the design, together with test set characteristics (such as efficiency and length) should be taken into account, because tests should be ordered such that defective ICs are failed early in the test process. It is

CHAPTER 8

important to note that any test scheduling algorithm will only reduce the time needed to test defective dies, since the time needed to test good dies will be independent of the order in which the test are applied. However, in a production test environment, where a large number of dies have to be tested, such an approach can produce significant gain in terms of total test time. The approach is especially relevant in low-yield situations.

In general, it is rather difficult to obtain information for defect probability analysis. In our work, we use probabilities of individual faults, which are usually derived from the statistical analysis of the production process or generated based on inductive fault analysis. In this chapter, a hybrid BIST scheduling algorithm in an AOFF environment will be described. A hybrid BIST approach gives us the opportunity not only to schedule the tests as black boxes, as it has been done in previous approaches, but also to select the best internal structure for individual test sets. This means choosing the best order of pseudorandom and deterministic test patterns for every individual core, so that the total test time of the entire system can be minimized. The proposed algorithm [68] assumes that all cores can be tested in parallel (hence the test power is not considered) and is limited to test-per-clock test architectures. The algorithm is based on a parallel hybrid BIST architecture, described in Section 4.4.2, where every core has its own dedicated BIST logic that is capable to produce a set of independent pseudorandom test patterns and the deterministic tests are carried out for one core at a time, using a single test access bus (Figure 4.5).

8.2. AOFF Test Scheduling

The main idea of our algorithm is to sort the tests so, that the *expected total test time* (ETTT) is the shortest. ETTT denotes here the expectation of the total test application time for testing a die in a production test environment, where some of the dies

might be faulty and therefore discarded before completing the entire test flow. The main purpose is to reduce the total test time, when large volumes of dies have to be tested.

8.2.1. Definitions and Problem Formulation

Suppose that the system S , consisting of cores C_1, C_2, \dots, C_n , has a test architecture depicted in Figure 4.5. Let us denote with DT_{ij} and PR_{ij} , respectively, the j -th deterministic pattern and j -th pseudorandom pattern in the respective sequences for core C_i . d_i and r_i are the total number of deterministic and pseudorandom test patterns for core C_i .

We assume that from the test scheduling point of view the deterministic test sequence is an undividable block, i.e. the deterministic test sequence cannot be divided into smaller subsequences. The pseudorandom test sequence, on the other hand, can be stopped and restarted in order to enable the application of the deterministic test sequence at a certain scheduled time moment. This assumption is a consequence of the assumed test architecture, where pseudorandom patterns are generated locally by every core, while deterministic test patterns are applied from one single test source and applied to one core at a time.

System Defect Probability

The system defect probability $DF(S)$ is defined as the probability of a system to be detected faulty during the production test. Similarly, the core defect probability $DF(C_i)$ is defined as the probability of a core to be detected faulty during the production test. Given the core defect probabilities, the system defect probability can be calculated as follows:

$$DF(S) = 1 - \prod_{i=1}^n (1 - DF(C_i)) \quad (8.1)$$

CHAPTER 8

In order to define the ETTT, the individual fault coverage of a test pattern has to be defined.

Individual Fault Coverage of a Test Pattern

To test a core C_i we have to apply d_i deterministic test patterns and r_i pseudorandom test patterns (in particular cases one of these sets may be empty). By calculating the number of faults $F(DT_{ij})$ or $F(PR_{ij})$ that are not yet detected by the previous patterns before the j -th pattern and detected just by the j -th pattern (either pseudorandom or deterministic), the independent fault coverage of each test pattern can be calculated as follows:

$$\begin{aligned} IFC(DT_{ij}) &= \frac{F(DT_{ij})}{F_i}, (1 \leq j \leq d_i) \\ IFC(PR_{ij}) &= \frac{F(PR_{ij})}{F_i}, (1 \leq j \leq r_i) \end{aligned} \tag{8.2}$$

where $IFC(DT_{ij})$ and $IFC(PR_{ij})$ are respectively the independent fault coverage of the j -th deterministic and j -th pseudorandom pattern for core C_i . F_i is the total number of non-redundant faults in core C_i .

Expected Total Test Time

Based on the given hybrid BIST test architecture we can generate and apply pseudorandom tests to all cores in parallel, thus reducing the total test time. However, a deterministic test sequence and a pseudorandom test sequence belonging to the same core cannot be scheduled in parallel due to the test conflict. This explains, for example, why the deterministic test pattern DT_{31} cannot be applied directly after the pseudorandom test sequence in Figure 8.1.

In this approach we treat deterministic test patterns and pseudorandom test sequences in a similar way, since pseudorandom test sequence can be treated as a single test pattern with a length corresponding to the length of the particular pseudoran-

HYBRID BIST IN AN ABORT-ON-FIRST-FAIL ENVIRONMENT

dom sequence. Therefore, in the following discussion a “test pattern” is used to denote both individual deterministic test patterns and pseudorandom test sequences, if not mentioned otherwise. It is important also to note that if a test-per-clock architecture is assumed then the length of one deterministic test pattern is one clock cycle. On the other hand, in a test-per-scan environment the length of a deterministic test pattern is defined by the length of the scan cycle. Correspondingly, the lengths of the pseudorandom sequences are also different.

This leads us to a set of all possible test termination points: after every individual deterministic test pattern and at the end of every pseudorandom test sequence, as illustrated in Figure 8.1. The possible test termination points are marked with black dotted lines.

Note that in test-per-clock architectures many of these points overlap and therefore are treated as one identical possible test termination point. Due to the differences in scan chain lengths the termination points at test-per-scan architectures are not periodical.

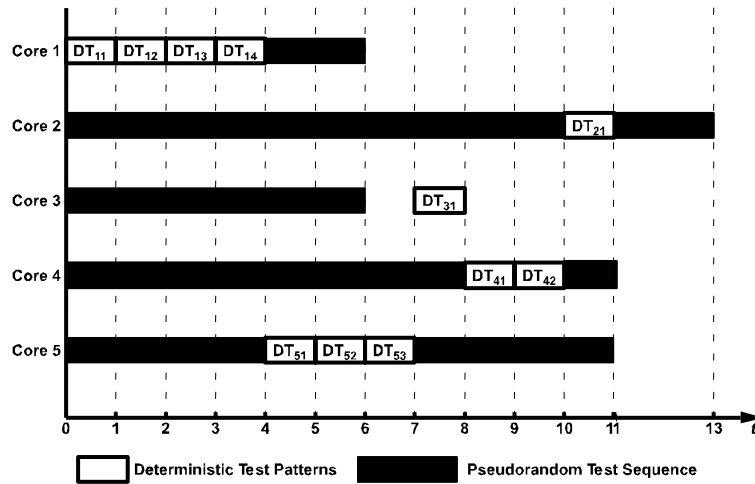


Figure 8.1. Hybrid BIST sessions for a system with 5 cores.

CHAPTER 8

We are interested in the ETTT as the expectation of the total test application time in the AOFF environment. In Equation (8.3) below we give a generic formula for ETTT calculation.

$$ETTT = \sum_{\forall x \in X} (t_x \times p(A_x)) + L \times p(T) \quad (8.3)$$

Equation (8.3) is presented as the sum of two literals. The first corresponds to the situation when a test is terminated prematurely and the second one corresponds to the case where all tests are passed to the completion. At every possible test termination point $x \in X$ we can calculate a test abortion probability $p(A_x)$ together with a test length t_x at this test termination point x . With A_x we denote the event that the test has been aborted at test termination point x . Similarly we can also calculate the probability $p(T)$ that no faults are detected and all tests (T) are exercised till their completion. The length of the complete test set is denoted with L .

At every test termination point $x \in X$ we can distinguish two different sets of tests – the tests that have failed and the tests that have passed. The failed set Y_x consists of all test patterns y that have finished exactly at this point. They are supposed to detect at least one fault, otherwise the test would not have been stopped at this point. The passed set Z_x consists of all test sequences that have successfully finished before this point x . They are all supposed to be passed otherwise the test would have been aborted before this point. This leads us to the following formula:

$$A_x = \left(\bigcup_{\forall y \in Y_x} F_y \right) \cap \left(\bigcap_{\forall z \in Z_x} P_z \right) \quad (8.4)$$

Here F_y is an event that the test pattern y detects at least one fault and P_z is an event that the test sequence z is passed. Thus the event A_x can be described as an event at test termination point x , such that any of the test patterns in the failed test set Y_x detect at least one fault, and all test sequences in the passed test set Z_x have passed. Please note that if a test pattern is included

to the failed set then all other patterns testing the same core (including both the pseudorandom patterns and deterministic patterns) should be removed from the passed set, since the probability of these patterns passing the test has been already considered due to the use of incremental fault coverages (see Equation (8.8) below).

We assume that defect occurrences in different cores are independent of each other. Thus, we can calculate the probability that the test is terminated at a possible termination point x as follows:

$$p(A_x) = p\left(\bigcup_{\forall y \in Y_x} F_y\right) \prod_{\forall z \in Z_x} p(P_z) = \left(1 - \prod_{\forall y \in Y_x} (1 - p(F_y))\right) \prod_{\forall z \in Z_x} p(P_z) \quad (8.5)$$

Similarly we can also calculate the probability $p(T)$ that no faults are detected and all tests are exercised till their completion, as follows:

$$T = \bigcap_{\forall z \in Z_e} P_z ; \quad p(T) = p\left(\bigcap_{\forall z \in Z_e} P_z\right) = \prod_{i=1}^n (1 - DP(C_i)) \quad (8.6)$$

where $DP(C)$ denotes the defect probability of core C .

This leads us to the refined version of Equation (8.3):

$$ETTT = \sum_{\forall x \in X} \left[t_x \times \left(1 - \prod_{\forall y \in Y_x} (1 - p(F_y))\right) \prod_{\forall z \in Z_x} p(P_z) \right] + L \times \prod_{i=1}^n (1 - DP(C_i)) \quad (8.7)$$

The probability of the event that at least one fault is detected by a test pattern $y \in Y_x$ at the test termination point x can be calculated as

$$p(F_y) = IFC(y) \times DP(C) \quad (8.8)$$

where the incremental fault coverage $IFC(y)$ of a single test pattern y is defined as a percentage of the faults only detected by y and not detected by any previous test pattern.

Similarly, the probability of the event that no faults are detected by a test sequence $z \in Z_x$ can be calculated as

CHAPTER 8

$$p(P_z) = 1 - \sum_{j=1}^n IFC(v_j) \times DP(C) \quad (8.9)$$

where n is the total number of test patterns in the test sequence $z \in Z_x$, and v_j is the j -th test pattern.

8.2.2. Proposed Heuristic for Test Scheduling

Based on the proposed cost function, defined in the previous section, we developed an iterative heuristic for ETTT minimization. As described earlier, the test scheduling problem in the hybrid AOFF test environment is essentially scheduling of deterministic test sequences, such that the ETTT of the system is minimal.

By changing the schedule of deterministic sequences, the set of passed test sequences Z_x and the set of failed test sequences Y_x , affiliated to every possible test termination point x , is also changed. Consequently, the individual fault coverage of each test pattern must be recalculated, since the passing probability of these patterns is changed. This will lead to the recalculation of the ETTT as described in the previous section.

It would be natural to order the tests in such a way, that the cores with high failure probability would be tested first. However, such a naïve schedule does not necessarily lead to the minimal expected total test time. In addition to the defect probabilities also the efficiency of test patterns and length of individual test sequences have to be taken into account. Due to the complexity of the problem we propose here an iterative heuristic that can be efficiently used.

In our heuristic we assume that we start from a subset of m already scheduled deterministic sequences, $m < n$. The objective is to increase this subset to $m+1$ scheduled deterministic sequences. This is obtained by selecting a deterministic sequence from the remaining unscheduled $n-m$ candidate sequences and inserting it into the existing sequence in such a way, that the resulting ETTT is as short as possible. This procedure is repeated

HYBRID BIST IN AN ABORT-ON-FIRST-FAIL ENVIRONMENT

for all cores $m=0, 1, \dots, n-1$. For the initial solution ($m=0$) the test sequence with the lowest ETTT is chosen.

At every iteration $(n-m)(m+1)$ different solutions have to be explored since there are $n-m$ candidate sequences and $m+1$ insertion points for each candidate. The heuristic is illustrated in Figure 8.2. Here we have illustrated a situation where two deterministic test sequences out of five are already scheduled ($m=2$, $n=5$). For every candidate schedule there are three different insertion points, indicated by arrows. During the iteration step, the ETTT for all candidate sequences for all possible insertion points is calculated and the candidate sequence will be finally inserted to the point with lowest ETTT.

The new situation is illustrated in Figure 8.3. In this example the deterministic test sequence of core 4 was chosen and inserted into the schedule after the core 1.

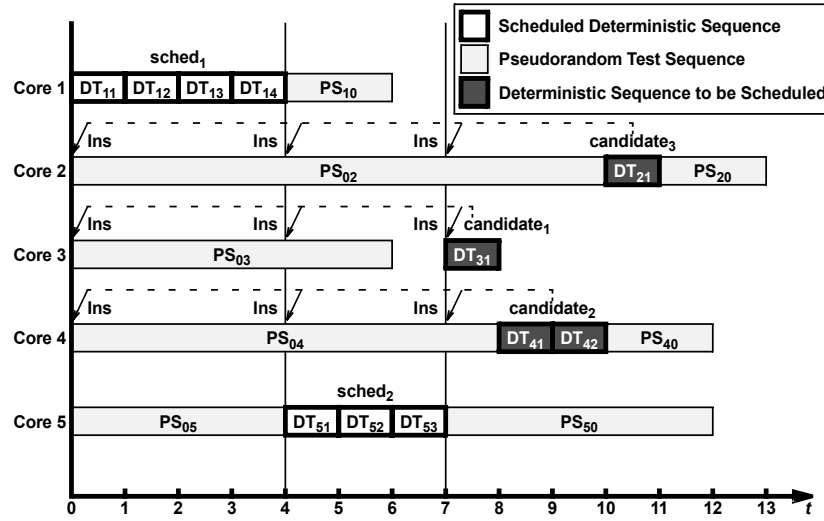


Figure 8.2. Initial solution for the iteration.

CHAPTER 8

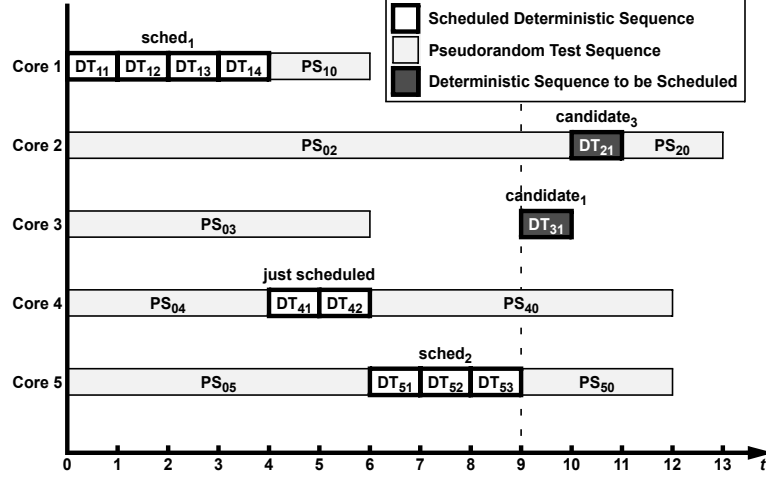


Figure 8.3. The new locally optimal order after the iteration.

In the following the pseudo-code of the algorithm is presented. As described earlier, the test schedule is obtained constructively by enlarging the subset of scheduled test sequences. During each iteration we add one additional sequence into the previously scheduled set of sequences and explore the solution space in order to find the optimal insertion point for the new sequence. The algorithm starts with the initialization phase. Thereafter comes the iterative scheduling heuristic, consisting of three main loops. The outer loop iteratively increases the number of already scheduled sequences. The middle loop iteratively goes through all candidate sequences in the current candidate set and the inner loop calculates for the chosen candidate sequence a new ETTT in every possible insertion point. A solution with lowest ETTT is chosen and the new schedule forms an initial solution for the next iteration. The heuristic stops when all test have been scheduled.

Algorithm 8.1. *Pseudo-code of the algorithm.*

Begin

Initialization

For (current_scheduled = 1 to n) **Do**

$m = \text{current_scheduled_number}(\text{Scheduled}[]);$

For (current_candidate = 1 to $n-m+1$) **Do**

For (current_insert_position = 0 to m) **Do**

Calculate new cost;

If (new_cost < min_cost)

min_cost = new_cost

chosen_candidate = current_candidate;

chosen_insert_position = current_insert_position;

EndIf;

EndFor;

Insert candidate to the chosen position;

EndFor;

EndFor;

Output Schedule;

End.

8.2.3. Experimental Results

We have performed experiments with 9 different designs, consisting of 5 to 50 cores (Table 8.1). In order to obtain diversification we have calculated for every experimental design 5 different hybrid test sets (different ratio of pseudorandom and deterministic test patterns) and the experimental results illustrate the average of five experiments. The defect probabilities for individual cores have been given randomly, while keeping the system defect probability at the value 0.6.

CHAPTER 8

Table 8.1. Experimental results.

Design Size	5		7		10		12		15	
	ETTT	CPU Time (s)	ETTT	CPU Time (s)	ETTT	CPU Time (s)	ETTT	CPU Time (s)	ETTT	CPU Time (s)
No Optimization	248.97	1.1	261.38	64.4	366.39	311.8	415.89	346.8	427.34	371.6
Our Heuristic	228.85	0.6	232.04	1.4	312.13	6.6	353.02	12.2	383.40	25.2
SA	228.85	1144.2	231.51	1278.5	311.68	3727.6	352.10	4266.8	381.46	5109.2
Exhaustive Search	228.70	1.2	231.51	80.0	311.68	112592.6	N/A	N/A	N/A	N/A

Design Size	17		20		30		50		
	ETTT	CPU Time (s)	ETTT	CPU Time (s)	ETTT	CPU Time (s)	ETTT	CPU Time (s)	
No Optimization	544.37	466.6	566.13	555.4	782.88	822.4	1369.54	1378.0	
Our Heuristic	494.57	43.6	517.02	85.4	738.74	380.4	1326.40	3185.0	
SA	493.93	6323.8	516.89	7504.4	736.51	11642.4	1324.44	21308.8	
Exhaustive Search	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

In order to illustrate the significance of test scheduling we have performed another set of experiments for comparison, where a random schedule is assumed. As it can be seen from Table 8.1, by employing our heuristic the ETTT can be reduced in a range of 5-15%, which is very relevant for large volume production testing.

As our heuristic can produce only a near optimal solution, experiments for estimating the accuracy of the solution were performed. For this purpose a simulated annealing algorithm and an exhaustive search has been used, where possible. As it can be seen from Table 8.1 our heuristic is able to produce results similar or very close to the results obtained with simulated annealing and exhaustive search, while having significantly lower computation times. These comparisons are also illustrated in Figure 8.4

HYBRID BIST IN AN ABORT-ON-FIRST-FAIL ENVIRONMENT

and Figure 8.5. In Figure 8.4 we have compared the ETTT values, calculated with different approaches, while in Figure 8.5 the CPU times with our heuristic and with simulated annealing are compared.

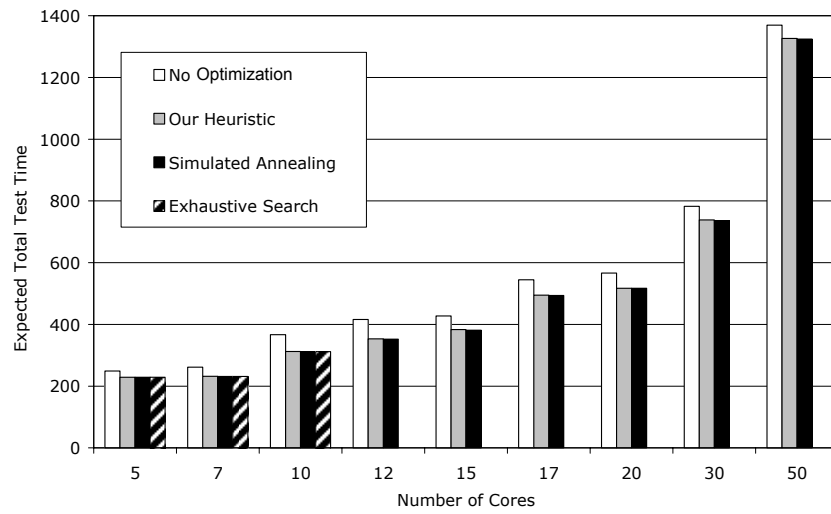


Figure 8.4. Comparison of expected total test times.

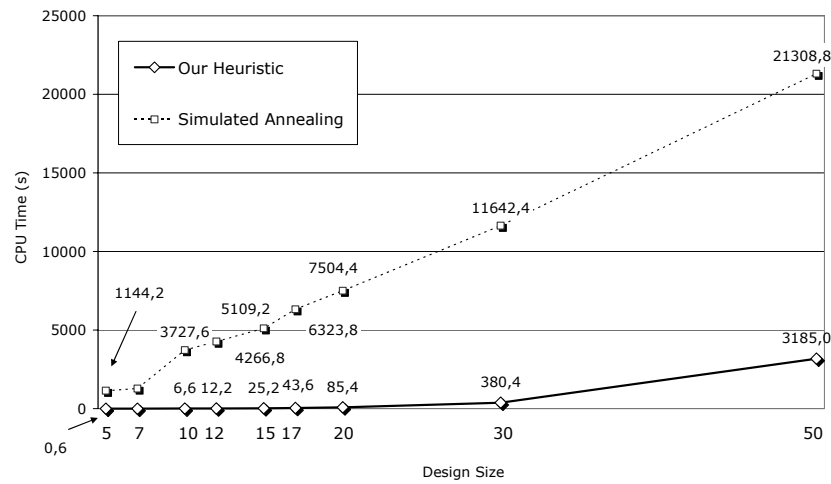


Figure 8.5. Comparison of CPU times.

CHAPTER 8

8.3. Conclusions

In this chapter we have proposed a methodology for hybrid BIST scheduling in an abort-on-first-fail environment, where the test is terminated as soon as a defect is detected. We have developed a methodology for expected total test time calculation, based on defect probabilities and proposed a heuristic for ETTT minimization. Experimental results have shown the efficiency of the proposed method.

PART III

HIERARCHICAL TEST

GENERATION

Chapter 9

Introduction and Modeling

As described in the previous chapters, the introduction of System-on-Chip (SOC) entails several challenges in respect to the design, test and manufacturing of such systems. To cope with the challenges faced by SOC designers, tools and techniques dealing with design at higher levels of abstraction have been developed. For example, behavioral-level synthesis tools and hardware/software co-design techniques are starting to play an important role in the initial phases of the design process. The main advantages of deploying such high-level design tools is the possibility to quickly evaluate the costs and benefits of different architecture alternatives, including both hardware and software components, starting from a high-level functional specification of the implemented system.

While the main design focus is quickly moving toward higher levels of abstraction, the test issues are usually considered only when a detailed description of the design is available, typically at the gate level for test sequence generation and at register transfer (RT) level for design for testability structure insertion.

CHAPTER 9

To address the problems associated with test generation and design-for-test, when performed at the later design stages, intensive research efforts have been devoted to devise solutions to test sequence generation and DFT in the early design phases, mainly at the RT level. For high-level test generation, several proposed approaches are able to generate test patterns of good quality, sometimes even better than those of gate-level ATPG tools. However, due to the lack of general applicability, most of these approaches are still not used in the industry.

This part of the thesis presents a high-level hierarchical test generation approach for improving the results obtained by a pure high-level test generator. The hierarchical test generator takes into account structural information from lower levels of abstraction while generating test sequences on the behavioral level [94]. We will start our discussion with the description of the modeling technique we use to model the design under test and the corresponding fault modeling techniques. In the next chapter the hierarchical test generation approach will be described.

9.1. Modeling with Decision Diagrams

Test generation for digital systems encompasses three main activities: selecting a description method, developing a fault model, and generating tests to detect the faults covered by the fault model. The efficiency of test generation (quality and speed) is highly depending on the description method and fault models which have been chosen. In order to generate tests at the high abstraction levels, we need a modeling technique that can capture designs at the levels in concern. Since the hierarchical test generation approach takes advantages of both high-level and low-level design information, we need a modeling technique which spans several levels of abstraction. This section will describe such a model, called Decision Diagrams.

9.1.1. Introduction

For high-level test generation, different high-level design and fault models have been introduced. The main idea of high-level modeling is to capture the high-level description of the system in a formal model, and to obtain different incorrect versions of the design by introducing a fault into the model. This approach is called model perturbation [64]. The models can be “perturbed” in certain ways, e.g. by truth-table modification, micro-operation modification, etc. In one way or the other, this idea is implemented in different high-level fault models for different classes of digital systems.

In the case of microprocessors, individual functional fault models and their corresponding test strategies have been developed for different function classes, such as register decoding, instruction decoding, control, data storage, data transfer, data manipulation, etc [16], [152]. The main disadvantage of this approach is that only microprocessors are handled and the results obtained cannot be extended to cover the general digital systems testing problem. When using register transfer languages (RTL-approach), a formal definition of an RTL statement is defined, and nine categories of functional faults for components of RTL statements are identified [146], [149]. Recently, a lot of attention has been devoted to generating tests directly from high level description languages [50], [51], [167]. Some attempts to develop special functional fault models for different data-flow network units like decoders, multiplexers, memories, PLAs, etc. are described in [2].

The drawback of traditional multi-level and hierarchical approaches to test generation lies in the need of different languages and models for different levels. For example, one might use logic expressions for combinational circuits; state transition diagrams for finite state machines (FSM); abstract execution graphs, system graphs, instruction set architecture (ISA) descriptions, flowcharts, hardware description languages, or Petri nets for system

CHAPTER 9

level description, etc. All these models need different manipulation algorithms and fault models which are difficult to merge into a coherent hierarchical test method. To address this problem, Decision Diagrams (DDs) can be used [17], [123], [136], [157], [158], [159]. Binary DDs (BDD) have found already very broad applications in logic design as well as in logic test [17], [123]. Structurally Synthesized BDDs (SSBDD) are able to represent gate-level structural faults directly in the graph [157], [158]. Recent research has shown that generalization of BDDs for higher levels provides a uniform model for both gate and RT level [136], [159], and even behavioural level test generation [87], [90].

In our approach, a method for describing digital systems and for modeling faults is based on decision diagrams. DDs serve as a basis for a general theory of test design for mixed-level representations of systems, similarly as we have the Boolean algebra for the plain logical level. DDs can be used to represent systems uniformly either at logic level, high-level or simultaneously at both levels. The fault model defined on DDs represents a generalization of the classical gate-level stuck-at fault model.

9.2. Modeling Digital Systems by Binary Decision Diagrams

Let us first consider binary decision diagrams in order to illustrate the basic notations. BDDs are a special case of DDs that are described later in this chapter for behavior level diagnostic modeling of digital systems. We will first describe logic level BDDs to prepare the readers for a better understanding of the generalization of BDDs for higher level system representation. We will use the graph-theoretical definitions instead of traditional logic oriented *ite* expressions [17], [123] because all the procedures defined further for DDs are based on the topological

reasoning rather than on graph symbolic manipulations as in the case of BDDs.

Definition 9.1: A BDD that represents a Boolean function $y = f(X)$, $X = (x_1, x_2, \dots, x_n)$, is a directed acyclic graph $G_y = (M, \Gamma, X)$, with a set of nodes M and a mapping Γ from M to M . $M = M^N \cup M^T$ consists of two types of nodes: nonterminal M^N and terminal M^T nodes. A terminal node $m^T \in M^T = \{m^{T,0}, m^{T,1}\}$ is labeled by a constant $e \in \{0, 1\}$ and is called a *leaf*; while all nonterminal nodes $m \in M^N$ are labeled by variables $x \in X$, and have exactly two successors. Let us denote the variable associated with node m as $x(m)$, then m^0 is the successor of m for the value $x(m) = 0$ and m^1 is the successor of m for $x(m) = 1$.

Definition 9.2: By the value of $x(m) = e$, $e \in \{0, 1\}$, we say the edge between nodes $m \in M$ and $m^e \in M$ is *activated*. Consider a situation where all the variables $x \in X$ are assigned by a Boolean vector $X^t \in \{0, 1\}^n$ to some value. The activated edges by X^t form an *activated path* $l(m_0, m^T) \subseteq M$ from the root node m_0 to one of the terminal nodes $m^T \in M^T$.

Definition 9.3: We say that a BDD $G_y = (M, \Gamma, X)$ represents a Boolean function $y = f(X)$, iff for all the possible vectors $X^t \in \{0, 1\}^n$ a path $l(m_0, m^T) \subseteq M$ is activated so that $y = f(X^t) = x(m^T)$.

Definition 9.4: Consider a BDD $G_y = (M, \Gamma, X)$ where X is the vector of literals of a function $y = P(X)$ represented in the equivalent parenthesis form [158], the nodes $m \in M^N$ are labeled by $x(m)$ where $x \in X$ and $|M| = |X|$. The BDD is called a *structurally synthesized BDD* (SSBDD), iff there exists a one-to-one correspondence between literals $x \in X$ and nodes $m \in M^N$ given by the set of labels $\{x(m) \mid x \in X, m \in M^N\}$, and for all the possible vectors $X^t \in \{0, 1\}^n$ a path $l(m_0, m^T)$ is activated, so that $y = f(X^t) = x(m^T)$.

Unlike the traditional BDDs [17], [123], SSBDDs [158] support structural representation of gate-level circuits in terms of signal paths. By superposition of DDs [158], we can create SSBDDs with one-to-one correspondence between graph nodes and signal

CHAPTER 9

paths in the circuit. The whole circuit can then be represented as a network of tree-like subcircuits (macros), each of them represented by a SSBDD. Using SSBDDs, it is possible to ascend from the gate-level to a higher macro level without losing accuracy of representing gate-level signal paths.

Our intention is to make use of the SSBDDs to capture both the structural and functional properties of a given circuit in order to generate high-quality test patterns.

Figure 9.1 shows a representation of a tree-like combinational circuit by a SSBDD. For simplicity, values of variables on edges of the SSBDD are omitted (by convention, an edge going to the right corresponds to 1, and an edge going down corresponds to 0). Also, terminal nodes with constants 0 and 1 are omitted: leaving the graph to the right corresponds to $y = 1$, and down, to $y = 0$. The SSBDD graph contains 7 nodes, and each of them represents a signal path in the given subcircuit (denoted as a macro in Figure 9.1). By bold lines an activated path in the graph corresponding to the input pattern $x_1x_2x_3x_4x_5x_6 = 110100$ is highlighted. The value of the function $y = 1$ for this pattern is determined by the value of the variable $x_5 = 1$ in the terminal node of the path.

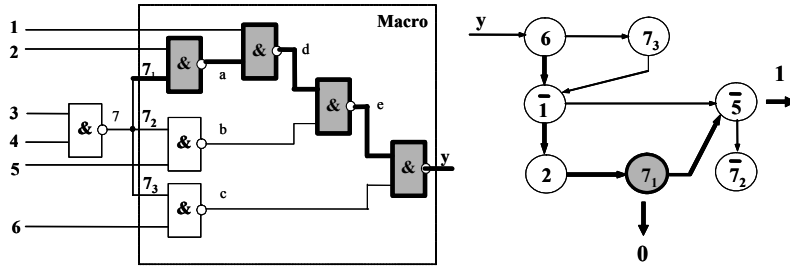


Figure 9.1. A gate level circuit and its corresponding SSBDD.

The path activation properties can efficiently be used in test generation, for example.

Procedure 9.1: Test generation. To generate a test for a node $m \in M^N$ in G_y , the following paths have to be activated:

MODELING WITH DECISION DIAGRAMS

- (1) $l(m_\emptyset, m)$,
- (2) $l(m^1, m^{T,1})$, and
- (3) $l(m^0, m^{T,0})$.

To generate a test pattern for the path from $x_{7,1}$ to y in the circuit by using SSBDD means to generate a test pattern for the corresponding node $x_{7,1}$ in the graph. To test the node $x_{7,1}$, according to Procedure 1, the following paths should be activated; $(6, -1, 2, 7_1)$, $(-1, m^{T,1})$, and $(-1, m^{T,0})$, which produces the test pattern: $x_1x_2x_3x_4x_5x_6 = 11xx00$. For example, to test a physical defect of a bridge between the lines 6 and 7, which is activated on the line 7, additional constraint $W = -x_6 \wedge x_7 = 1$ has to be used, which updates the test vector to 111x00.

9.3. Modeling with a Single Decision Diagram on Higher Levels

Consider now a digital system $S = (Z, F)$ as a network of components (or processes), where Z is the set of variables (Boolean, Boolean vectors or integers) that represent connections between components, as well as inputs and outputs of the network. Denote by $X \subset Z$ and $Y \subset Z$, correspondingly, the subsets of input and output variables. $V(z)$ denotes the set of possible values for $z \in Z$, which are finite.

Let F be the set of digital functions on Z : $z_k = f_k(z_{k,1}, z_{k,2}, \dots, z_{k,p}) = f_k(Z_k)$ where $z_k \in Z$, $f_k \in F$, and $Z_k \subset Z$. Some of the functions $f_k \in F$, for the state variables $z \in Z_{STATE} \subset Z$, are next state functions.

Definition 9.5: A *decision diagram* is a directed acyclic graph $G = (M, \Gamma, Z)$ where M is a set of nodes, Γ is a relation in M , and $\Gamma(m) \subset M$ denotes the set of successor nodes of $m \in M$. The nodes $m \in M$ are marked by labels $z(m)$. The labels can be variables $z \in Z$, algebraic expressions $f_m(Z(m))$ of $Z(m) \subseteq Z$, or constants.

CHAPTER 9

For non-terminal nodes $m \in M^N$, where $I(m) \neq \emptyset$, an onto function exists between the values of $z(m)$ and the successors $m^e \in I(m)$ of m . By m^e we denote the successor of m for the value $z(m) = e$. The edge (m, m^e) which connects nodes m and m^e is called *activated* iff there exists an assignment $z(m) = e$. Activated edges, which connect m_i and m_j , make up an *activated path* $l(m_i, m_j) \subseteq M$. An activated path $l(m^0, m^T) \subseteq M$ from the initial node m^0 to a terminal node m^T is called a *full activated path*.

Definition 9.6: A decision diagram $G_{z,k}$ represents a high-level function $z_k = f_k(z_{k,1}, z_{k,2}, \dots, z_{k,p}) = f_k(Z_k)$, $z_k \in Z$ iff for each value $v(Z_k) = v(z_{k,1}) \times v(z_{k,2}) \times \dots \times v(z_{k,p})$, a full path in $G_{z,k}$ to a terminal node $m^T \in M^T$ in $G_{z,k}$ is activated, so that $z_k = z(m^T)$ is valid.

Depending on the class of the system (or its representation level), we may have various DDs, where nodes have different interpretations and relationships to the system structure. In RTL descriptions, we usually partition the system into control and data parts. Nonterminal nodes in DDs correspond to the control path, and they are labelled by state and output variables of the control part serving as addresses or control words. Terminal nodes in DDs correspond to the data path, and they are labelled by the data words or functions of data words, which correspond to buses, registers, or data manipulation blocks.

When using DDs for describing complex digital systems, we have to, first, represent the system by a suitable set of interconnected components (combinational or sequential subcircuits). Then, we have to describe these components by their corresponding functions which can be represented by DDs.

Figure 9.2 depicts an example of a DD describing the behavior of a digital system together with its possible RTL implementation. The variables R_1 , R_2 and R_3 represent registers, IN represents the input bus, the integer variables y_1, y_2, y_3, y_4 represent the control signals, M_1, M_2, M_3 are multiplexers, and the functions $R_1 + R_2$ and $R_1 * R_2$ represent the adder and multiplier, correspondingly. Each node in DD represents a subcircuit of the sys-

MODELING WITH DECISION DIAGRAMS

tem (e.g. the nodes y_1, y_2, y_3, y_4 represent multiplexers and decoders,). The whole DD describes the behavior of the input logic of the register R_2 . To test a node means to test the corresponding subcircuit.

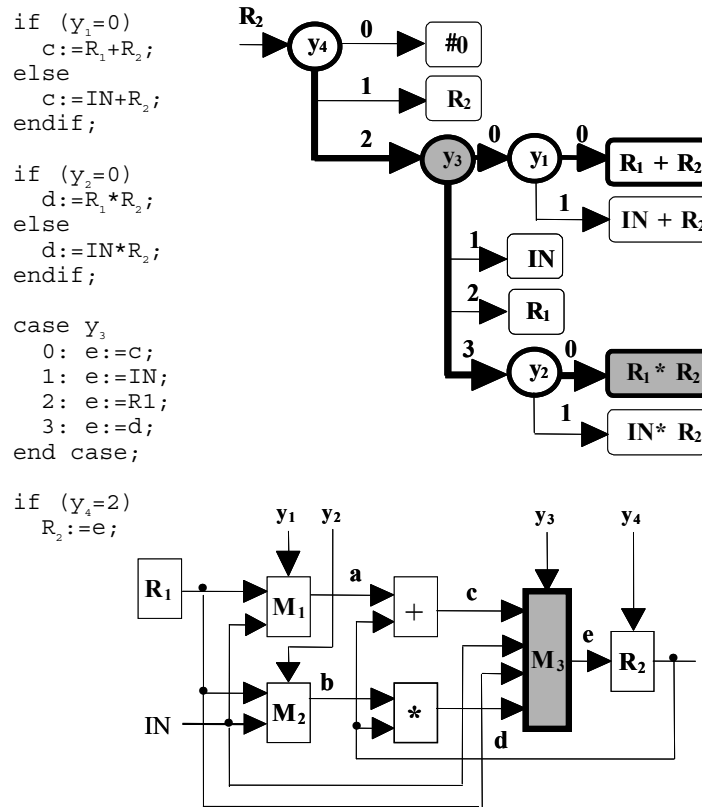


Figure 9.2. Representing a data path by a decision diagram.

For test pattern simulation, a path is traced in the graph, guided by the values of input variables until a terminal node is reached, similarly as in the case of SSBDDs. In Figure 9.2 the result of simulating the vector $y_1, y_2, y_3, y_4, R_1, R_2, IN = 0, 0, 3, 2, 10, 6, 12$ is $R_2 = R_1 * R_2 = 60$ (bold arrows mark the activated path). Instead of simulating by a traditional approach all the

CHAPTER 9

components in the circuit, in the DD only 3 control variables are visited during simulation, and only a single data manipulation $R_2 = R_1 * R_2$ is carried out.

9.3.1. Decision Diagrams at the Behavioral Level

Our 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 [39].

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. As described earlier, the entire system can be represented with a single DD. However, in our case, the design is already partitioned into the control part and data path, and therefore both parts will be converted into separate DDs.

Figure 9.3 depicts an example of a 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 9.3c). If this state is to be activated, condition $IN1 \geq 0$ should be true (Figure 9.3b). The DDs, extracted from a specification, will be used as a computational model in the hierarchical test generation environment.

9.3.2. SICStus Prolog Representation of Decision Diagrams

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.

MODELING WITH DECISION DIAGRAMS

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 9.3a, we have to bring the system to the state $q=2$. This can be guaranteed only when $q'=0$ and $IN1 \geq 0$. These requirements can be seen as justification constraints.

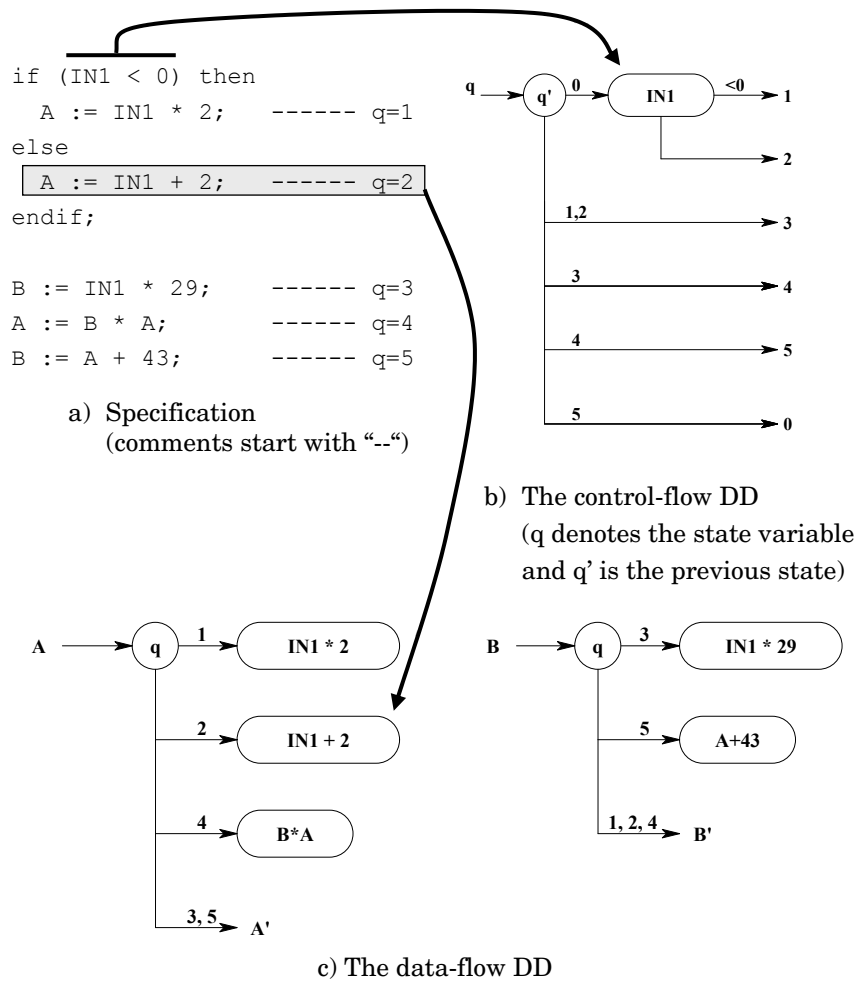


Figure 9.3. A decision diagram example.

CHAPTER 9

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. These 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 [147] 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 [150] for technical details about the translation process.

Chapter 10

Hierarchical Test Generation with DDs

One possible approach to deal with test generation complexity is to raise up the level of design abstractions at which the basic test generation procedure is performed. In the following we will describe an approach that performs the test generation procedure using the high-level behavioral description captured by DDs, but at the same time, takes into account some detailed information of the basic components at the lower levels.

At the behavioral level we can represent digital system with a single DD or to partition the system into the control-flow DD and data-flow DDs. For illustrative purposes we will use hereby the latter approach. 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.

CHAPTER 10

Depending on the partition of a system into a network of subsystems we can represent the whole DD-model as a set of DDs, so that for every output of a subsystem a DD will be associated with it.

A test for a system represented by DDs can be created in two parts [157]:

- A *scanning test*, which makes sure that the different functional blocks are working correctly; and
- A *conformity test*, which makes sure that the different working modes chosen by control signals are properly carried out.

In [90] it has been shown that in some cases there exists a gap between the fault coverage figures attained by test sequences generated purely on a high-level and those by the gate-level ones. This gap can be reduced by integrating structural information to the test generation process by employing the hierarchical test generation (HTG) approach to be discussed here.

The main idea of a HTG technique [126] 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 us to divide a larger problem into several smaller subproblems 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.

The HTG algorithm of interest to us generates conformity tests from pure behavioral descriptions. This test set targets errors in branch selection (nonterminal nodes of the DDs). 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 DDs), 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 propagation and justification task still on a high abstraction level.

10.1. Test Generation Algorithm

The test generation task is performed in the following way (Figure 10.1). 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.

The HTG environment is depicted in Figure 10.2. 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 [147]. 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.

CHAPTER 10

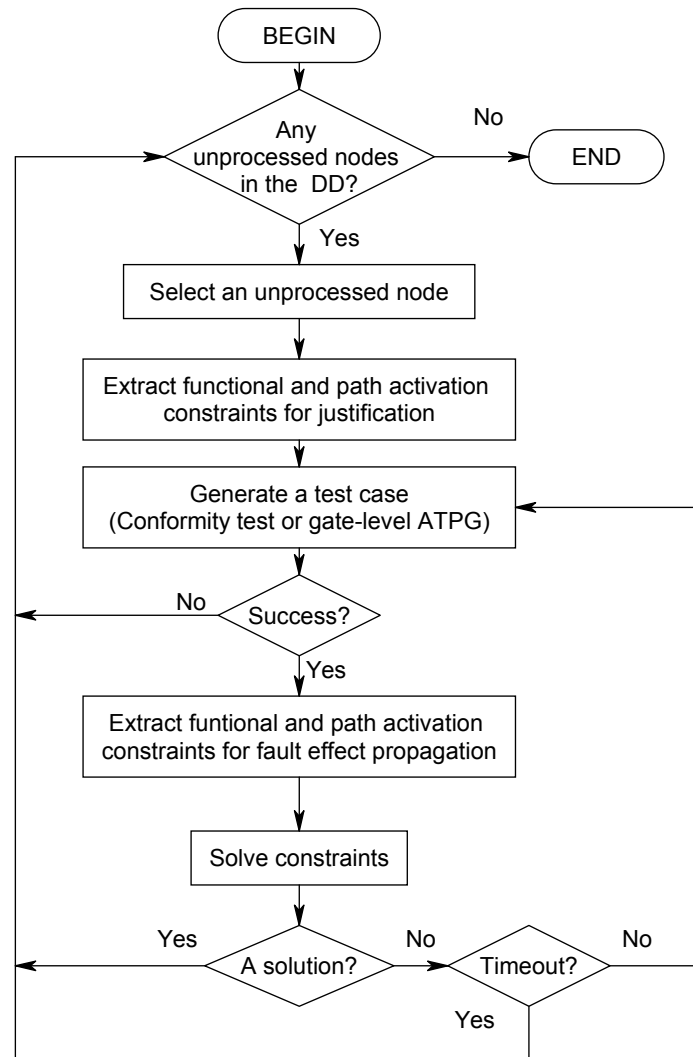


Figure 10.1. The general flow for hierarchical test generation algorithm.

HIERARCHICAL TEST GENERATION WITH DDS

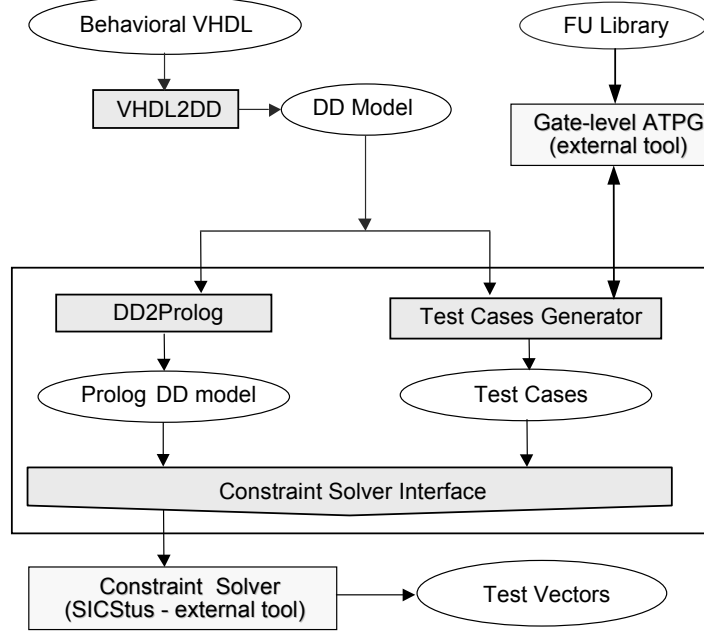


Figure 10.2. Our hierarchical test generation environment.

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

10.2. Scanning Test

Consider a terminal node $m^T \in M^T$ in $G_{z,k}$, labeled by a functional expression $f_m(Z(m^T))$. To generate a test for the node m^T means to generate a test for the function $f_m(Z(m^T))$.

For generating a test for $f_m(Z(m^T))$ we have to solve two tasks:

1. to activate a path $l(m_\phi, m^T) \subseteq M$, from the root node m_ϕ of the DD up to m^T by choosing proper values $z(m)^*$ for all the control variables $z(m)$ in the nodes $m \in l(m_\phi, m^T) \setminus m^T$; and
2. to find the proper sets of data values $D = (D^1, D^2, \dots, D^p)$ for the variables $Z(m^T)$ to test the function $f_m(Z(m^T))$.

CHAPTER 10

For executing these two tasks, we can use the following test program:

Algorithm 10.1:

```

FOR all the values of  $t = 1, 2, \dots, p$ 
BEGIN
    Load the data registers  $Z(m^T)$  with  $D^t$ ;
    Carry out the tested working mode at the control values
         $z(m)^*$  for all  $z(m)$ ,  $m \in l(m_\varphi, m^T) \setminus m^T$ ;
    Read the value of  $z_k$  and compare it to the reference value
         $f_m(D^t)$ .
END.

```

The task of the scanning test is to detect the faults in registers, buses and data manipulation blocks. In terms of DDs the terminal nodes are tested by the scanning test.

Example 10.1: We illustrate how a test can be generated for testing the multiplier in Figure 9.2. In the DD of Figure 9.2 we have two terminal nodes with the multiplier function. Let us choose the node $R_1 * R_2$ for testing. By activating the path to this node (shown by bold in Figure 9.2) we generate a control word $y_2, y_3, y_4 = 0, 3, 2$. To find the proper values of R_1 and R_2 we need to descend to the lower level (e.g., gate level) and generate test patterns by a low level ATPG for the implementation of the multiplier. Let us have a test set of n test patterns $(D_{11}, D_{21}; D_{12}, D_{22}; \dots, D_{1p}, D_{2p})$ generated for the multiplier with inputs R_1 and R_2 .

Based on the above information, the following test program can be used:

Algorithm 10.2:

```

FOR all the values of  $i = 1, 2, \dots, p$ 
BEGIN
    Load the data registers  $R1 = D_{1i}, R2 = D_{2i}$ ;

```


HIERARCHICAL TEST GENERATION WITH DDS

Carry out the tested working mode at the control values
 $y_2, y_3, y_4 = 0, 3, 2$;

Read the value of $R2$ and compare to the reference $D_{1i} * D_{2i}$.

END.

10.2.1. Scanning Test in the HTG Environment

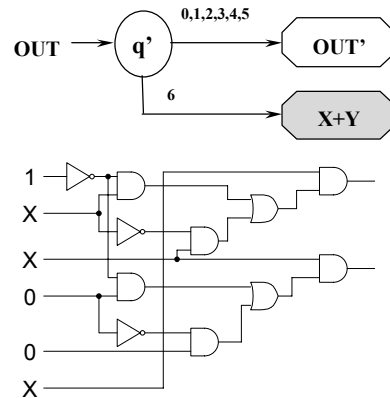
One of the most important parameters guiding the design synthesis process is the technology and module library that will be used in the final implementation. By defining the technology and module library, we can have information about the implementation of functional units that will be used in the final design. The hierarchical test generation algorithm can employ this structural information for generating tests. Tests are generated by cooperation of high-level and low-level test pattern generators. It is usually performed one by one for every arithmetic operator given in the specification (Figure 10.3).

```

if (IN1 > 0)
    X=IN2+3;      --- q=1
else {
    if (IN2 >= 0)
        X=IN1+IN2; -- q=2
    else
        X=IN1*5;   --- q=3
}

Y=X-10;          ----- q=4
X=Y*2;           ----- q=5
OUT=X+Y;         ----- q=6
    
```

Behavioral description



Fragment of a gate level netlist

Figure 10.3. Testing functional units

In the HTG environment we describe here, the algorithm starts by choosing an operator not yet tested from the specification, and uses a gate level ATPG to generate a test pattern tar-

CHAPTER 10

getting structural faults in the corresponding functional unit. In this approach a PODEM like ATPG 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 this 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 given value. Selecting the exact values is an important procedure since not all possible values can be propagated through the environment and it can therefore lead to the degradation of fault coverage.

A test vector that does not have any undefined bits is thereafter forwarded to a constraint solver, where together with the environmental constraints it forms a test case. Solving such a test case 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 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 10.4). 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

HIERARCHICAL TEST GENERATION WITH DDS

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.

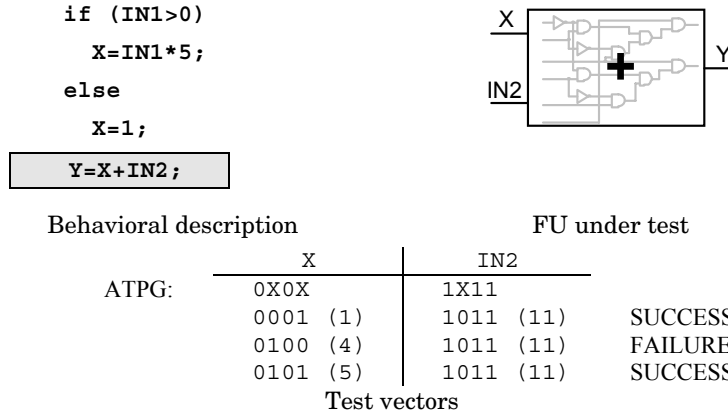


Figure 10.4. Selection of a test vector.

10.3. Conformity Test

Consider a nonterminal node m labeled by a control variable $z(m)$ in a given DD $G_{z,k}$, representing a digital system with a function $z_k = f_k(Z_k)$. Let $Z = (Z_c, Z_d)$, where Z_c is the vector of control variables and Z_d is the vector of data variables. To generate a test for the node m means to generate a test for the control variable $z(m) \in Z_c$. Suppose that the variable $z(m)$ may have $n = |z(m)|$ different values. For testing $z(m)$, we have to activate and exercise all the proper working modes controlled at least once by each value of $z(m)$. At the same time, for each of such a working mode, a current state of the system should be generated, so that

CHAPTER 10

every possible faulty change of $z(m)$ should produce a faulty next state different compared to the expected next state for the given working mode.

Let us denote by m^e the successor node of the node m for the value $z(m) = e$, where $e = 1, 2, \dots, n$. For generating a test for m we have to solve the following tasks on the DD:

1. to activate a path $l(m_\phi, m) \setminus m \subseteq M$ from the root node of the DD up to the node m by choosing proper values $z(m')^*$ for all the control variables $z(m') \in Z_C$ in the nodes $m' \in l(m_\phi, m) \setminus m$;
2. to activate for all neighbors m^e of m nonoverlapping paths $l(m^e, m^{e,T})$ from m^e up to the nonoverlapping terminal nodes $m^{e,T}$ by choosing proper values $z(m')^*$ for all the control variables $z(m') \in Z_C$ in the nodes of $m' \in l(m^e, m^{e,T})$; and
3. to find the proper set of data (the values z^* of the variables $z \in Z_D$), by solving the inequality $z(m^{T,1}) \neq z(m^{T,2}) \neq \dots \neq z(m^{T,n})$ where $n = |v(z(m))|$.

Consider the resulting test as a set of symbolic test patterns $T = \{(z(m) = e, Z_C^*, Z_D^*, z(m^{T,e})) \mid e \in v(z(m))\}$, where e is the symbolic value of the tested variable $z(m)$; Z_C^* is the constant vector of the other control signals corresponding to the set of variables $Z_C \subseteq Z$, and generated by the first two steps of the algorithm; Z_D^* is the constant vector of the data values corresponding to the set of variables $Z_D \subseteq Z$, and generated by the third step of the algorithm; and, finally, $z(m^{T,e})$ is the expected output value of the system corresponding to the value e of the tested control variable $z(m)$. The final conformity test of the control variable $z(m)$ created from the symbolic test pattern T consists of the following program:

FOR each value of $e = 1, 2, \dots, |z(m)|$

BEGIN

Load the data registers with Z_D^* ;

Carry out the tested working mode at the control signals

$z(m) = e$, and Z_C^* ;

HIERARCHICAL TEST GENERATION WITH DDS

Read the value of z_k , and compare with the reference value $z(m^{T,e})$.

END.

The task of the conformity test is to detect the control faults and the faults in multiplexers. In terms of DDs the nonterminal nodes are tested by the conformity test.

For example, in order to test nonterminal node $IN1$ in Figure 10.5, 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 $X \neq Y$. 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.

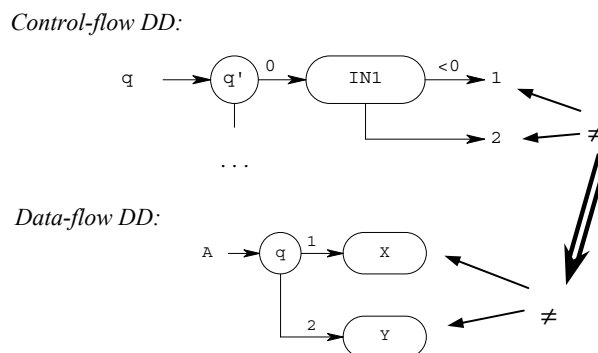


Figure 10.5. Conformity test example

CHAPTER 10

Example 10.2: Let us consider how to generate a test program for testing the node m labeled by y_3 in Figure 9.2. First, we activate the path $l(m_o, m) \setminus m$, which results in $y_3 = 2$. Then we activate 4 paths $l(m, m^{eT})$ for each value $e = 1, 2, 3, 4$ of y_3 , which results in $y_1 = 0$ and $y_2 = 0$. Two of the 4 paths for values $y_3 = 1$ and $y_3 = 2$ are “automatically” activated since the successors of the node y_3 for these values are terminal nodes. The test data $R1 = D_1, R2 = D_2, IN = D_3$ are found by solving the inequality:

$$R1 + R2 \neq IN \neq R1 \neq R1 * R2 \quad (10.1)$$

From the procedure described above, the following conformity test for the control variable y_3 is generated:

Algorithm 10.3:

FOR $e = 1, 2, 3$, and 4

BEGIN

Load the data registers $R1 = D_1, R2 = D_2$;

Carry out the tested working mode at

$y_3 = e, y_1 = 0, y_2 = 0, y_4 = 2$ and $IN = D_3$;

Read the value of $R2$, and compare it to the reference value $z(m^{T,e})$.

END.

In the case when the control values are data dependent then the algorithms become more complicated, since the data found for nonterminal nodes by activating the paths in the DD should be consistent with data found in processing the terminal nodes.

In the general case, a digital system cannot be represented by a single DD. In this case a system will be represented as a network of components or subsystems where each subsystem is modelled by its own DD. The test sequences generated for a subsystem with its DD by the procedures described above are to be treated as local test sequences. To generate the whole test sequence in a global sense, the classical *fault propagating* and *line*

justification tasks should be solved on the system level. For solving these tasks, DDs can also be used.

To justify a value D for a variable z_k represented by a DD $G_{z,k}$, a path should be activated in $G_{z,k}$ from the root node to a terminal node m^T labelled by a register, bus or input variable z , and the value D is assigned to z . If z corresponds to an input or any other directly controllable point, the line justification task is finished. Otherwise, if z is a register or a bus represented by its own DD G_z , the line justification tasks will be iteratively solved for z using the graph G_z .

To propagate the fault from the point represented by a variable z through a subsystem which is represented by a DD $G_{z,k}$, a test generation procedure described above should be carried out in $G_{z,k}$ for the node m labeled by z . The test generated for the node m is propagating any erroneous value of $z(m)$ to the output variable z_k of the subsystem.

10.4. 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 10.1, where for every approach we have presented the ob-

CHAPTER 10

tained stuck-at fault coverage (FC), number of generated test vectors (Len) and CPU time spent (CPU) for test generation.

Table 10.1. Results for the DIFFEQ benchmark circuit.

	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

We compare our results with pure high-level ATPG [90] 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.

HIERARCHICAL TEST GENERATION WITH DDS

We have also investigated possibilities to apply our ATPG approach to an industrial design F4 [148], 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 10.2:

Table 10.2. ATPG results with F4 design.

Design	VHDL Lines [#]	Stuck-at faults [#]	Our Hierarchical ATPG			Gate level ATPG Flex-Test		
			Len [#]	CPU [s]	FC [%]	Len [#]	CPU [s]	FC [%]
<i>F4_InputHandler_1</i>	175	4872	62	228	64.22%	219	811	38.22%
<i>F4_OutputHandler_1</i>	54	872	26	1.52	76.26%	170	5	81.30%

As it can be seen, HTG can produce results that 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 10.6. We have annotated the VHDL behav-

CHAPTER 10

ioral description of the design with the test generation results. We use the instruction $y_var := y_var + t7$; in order to explain the attached information:

```
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 propa-
    gated and/or justified till primary inputs/outputs).
-- Fault coverage: 90.89
    Final stuck-at fault coverage.
-- 11 Vectors
    Number of test vectors that were generated by a gate
    level ATPG and successfully justified till primary in-
    puts and propagated till primary outputs.
```

As it can be seen, fault coverage of functional units differs significantly, depending of the location and type of every individual FU. This information can be successfully exploited at the latter stage of the DFT flow, when selecting modules for DFT modifications.

HIERARCHICAL TEST GENERATION WITH DDS

```

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: 99.75
        -- Fault efficiency: 99.75
        -- 52 Vectors

        t2 := x_var * 3;
        -- Tested 4911 faults
        -- Untestable 0
        -- Aborted 737
        -- Fault coverage: 86.95
        -- Fault efficiency: 86.95
        -- 11 Vectors

        t3 := y_var * 3;
        -- Tested 4780 faults
        -- Untestable 0
        -- Aborted 868
        -- Fault coverage: 84.63
        -- Fault efficiency: 84.63
        -- 10 Vectors

        t4 := t1 * t2;
        -- Tested 5621 faults
        -- Untestable 0
        -- Aborted 27
        -- Fault coverage: 99.52
        -- Fault efficiency: 99.52
        -- 38 Vectors

        t5 := dx_var * t3;
        -- Tested 5616 faults
        -- Untestable 0
        -- Aborted 32
        -- Fault coverage: 99.43
        -- Fault efficiency: 99.43
        -- 35 Vectors

        t6 := u_var - t4;
        -- Tested 368 faults
        -- Untestable 0
        -- Aborted 60
        -- Fault coverage: 85.98
        -- Fault efficiency: 85.98
        -- 9 Vectors

        u_var := t6 - t5;
        -- Tested 424 faults
        -- Untestable 0
        -- Aborted 4
        -- Fault coverage: 99.06
        -- Fault efficiency: 99.06
        -- 15 Vectors

        t7 := u_var * dx_var;
        -- Tested 1123 faults
        -- Untestable 0
        -- Aborted 4525
        -- Fault coverage: 19.88
        -- Fault efficiency: 19.88
        -- 1 Vectors

        y_var := y_var + t7;
        -- Tested 389 faults
        -- Untestable 0
        -- Aborted 39
        -- Fault coverage: 90.88
        -- Fault efficiency: 90.88
        -- 11 Vectors

        x_var := x_var + dx_var;
        -- Tested 414 faults
        -- Untestable 0
        -- Aborted 14
        -- Fault coverage: 96.72
        -- Fault efficiency: 96.72
        -- 15 Vectors

      end loop ;

      x_out <= x_var;
      y_out <= y_var;
      u_out <= u_var;
    END PROCESS ;
  END behavior;

```

Figure 10.6. DIFFEQ benchmark with testability figures for every individual functional unit.

10.5. Conclusions

This part of the thesis described a modeling technique, the Decision Diagrams, which is used to capture a digital design at several levels of abstraction. We illustrate first how DDs can be used to capture a gate-level design, with respect of both functional and structural information. The use of DDs to capture designs at the register-transfer and behavioral levels were then described.

With the help of the Decision Diagrams, a hierarchical test generation approach could be developed to generate efficiently test patterns based on information from several abstraction levels. The described hierarchical test pattern generation technique generates test sequences with higher fault coverage than those of a pure behavioral test generator. This improvement in fault coverage has been obtained by integrating structural information coming from lower-level design. The algorithms maintain efficiency in terms of execution speed by mainly working at the behavioral level for test vector justification and propagation. In the particular hierarchical test generation implementation, a constraint solving algorithm is used to solve the vector justification and propagation problems.

PART IV

CONCLUSIONS AND
FUTURE WORK

Chapter 11

Conclusions

The aim of this thesis is to develop a built-in self-test methodology with the corresponding optimization methods and to propose a technique for test pattern generation at high abstraction level. In this chapter we summarize the thesis and underline the main contributions. Possible directions for the future work will be given in the next chapter.

Integrated circuits have been one of the most rapidly developing research domains during the last twenty years. Such circuits have evolved from few-hundred-transistor controllers to the modern systems-on-chip with hundreds of millions of transistors. 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 meth-

CHAPTER 11

ods 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).

The contribution of this thesis is twofold:

Hybrid BIST technique and its optimization methods. In the second part of the thesis an approach for improving classical BIST, called hybrid BIST, was described. The method is based on a hybrid test set that is composed of a limited number of pseudo-random test vectors and some additional deterministic test patterns that are specially designed to shorten the pseudorandom test cycle and to target random resistant faults.

The main contribution of the thesis is a set of algorithms for hybrid BIST optimization. We have analyzed hybrid BIST in both environments: single-core designs and multi-core designs. For single core designs, algorithms for total test cost calculation and test cost minimization were devised. For multi-core systems, algorithms for test time minimization, based on different test architectures were proposed. Due to the complexity of optimizing several SOC test parameters simultaneously, we have devised a solution, where one of the parameters is constrained (test memory) and we try to minimize the second one (test time). This approach has high significance, for example, in handheld devices where available memory is usually very limited. In addition, we have developed several algorithms for hybrid BIST energy reduction and hybrid BIST test scheduling in an abort-on-first-fail test environment.

High-level hierarchical test pattern generation. In the third part of the thesis a novel high-level hierarchical test pattern generation algorithm was proposed. It works at an imple-

CONCLUSIONS

mentation 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.

Chapter 12

Future Work

The thesis covers several aspects related to hybrid BIST and high-level test generation. In both domains we foresee several directions for future research.

Hybrid BIST:

- *Complex optimization.* In this thesis we have proposed a method for total test cost minimization only for single-core designs. The proposed algorithms for multi-core designs can minimize only one of the test parameters. Therefore, a possible future development is an optimization algorithm that can minimize several test parameters concurrently. Consequently a method that can lead to a highly efficient test solution in the general case could be developed. This task is, however, very dependent on the selected test strategy and the defined priorities. For this reason, this work should be carried out in close cooperation with chip manufacturers.

CHAPTER 12

- *Hybrid BIST for sequential circuits.* In this thesis we have proposed a hybrid BIST approach for combinational circuits and sequential circuits with full scan. A more complex problem is to propose an architecture and optimization mechanisms for sequential circuits without any scan (or with partial scan). The difficulty of developing such an architecture and optimization 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 have 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 sets in terms of defect detection capabilities and to develop a methodology to support the detection of other failures than the stuck-at ones.
- *Power constrained test scheduling.* In this thesis we have proposed different heuristics for test time minimization and test scheduling. Any of these approaches, however, did not take into account power consumption of the tests. Scheduling too many tests concurrently might unfortunately simply burn the circuit. Therefore, the current work should be extended by incorporating also the peak power constraint into the test scheduling process.

FUTURE WORK

High-level hierarchical test pattern generation:

- *Testability of hardware/software systems.* The testing of the hardware and software parts of a system is, 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 should also 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.

References

- [1] “90 Nanometer: The World's Most Advanced Chip-Making Process,” *Intel Research*, 2003.
<http://www.intel.com/techtrends/research/>
- [2] J. A. Abraham, “Fault Modeling in VLSI. VLSI Testing,” *North-Holland*, pp 1-27, 1986.
- [3] M. Abramovici, M. A. Breuer, A. D. Friedman, “Digital Systems Testing and Testable Design,” *IEEE Press*, 1990.
- [4] V. K. Agarwal, E. Cerny, “Store and Generate Built-In Testing Approach,” *International Symposium on Fault-Tolerant Computing*, pp. 35-40, 1981.
- [5] 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.
- [6] B. Akers, W. Jansz, “Test Set Embedding in Built-In Self-Test Environment,” *IEEE International Test Conference*, pp. 257-263, 1989.

- [7] G. al-Hayek, C. Robach, "An Enhancement Process for High-Level Hardware Testing Using Software Methods," *IEEE European Test Workshop*, pp. 215-219, 1998.
- [8] L. Ali, R. Sidek, I. Aris, B. S. Suparjo, M. A. M. Ali, "Challenges and Directions for Testing IC," *Integration, the VLSI Journal*, Vol. 37, No. 1, pp. 17-28, February 2004.
- [9] C. Angelbro, "P-Bist Test Method Conquer the Ericsson World," *Ericsson Telecom AB*, 1997.
- [10] B. S. Baker, E. G. Coffman, Jr., R. L. Rivest, "Orthogonal Packings in Two Dimensions," *SIAM Journal of Computing*, Vol. 9, Issue 4, pp. 846-855, 1980.
- [11] P. H. Bardell, W. H. McAnney, "Self-Testing of Multichip Logic Modules," *IEEE International Test Conference*, pp. 200-204, 1982.
- [12] P. H. Bardell, W. H. McAnney, J. Savir, "Built-In Test for VLSI Pseudorandom Techniques," *John Wiley and Sons*, 1987.
- [13] N. Z. Basturkmen, S. M. Reddy, I. Pomeranz, "A Low Power Pseudo-Random BIST Technique," *IEEE International On-Line Testing Workshop*, pp. 140-144, 2002.
- [14] B. Beizer, "Software Testing Techniques," (2nd ed.), *Van Nostrand Rheinold*, New York, 1990.
- [15] M. Bershteyn, "Calculation of Multiple Sets of Weights for Weighted Random Testing," *IEEE International Test Conference*, pp. 1031-1040, 1993.
- [16] D. Brahme, J. A. Abraham, "Functional Testing of Microprocessors," *IEEE Transactions on Computers*, Vol. 33, No. 6, pp 475-485, 1984.

- [17] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, Vol. 35, No. 8, pp 667-690, 1986.
- [18] F. Brglez, H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," *IEEE International Symposium on Circuits and Systems*, pp. 663-698, 1985.
- [19] F. Brglez, D. Bryan, K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *IEEE International Symposium on Circuits and Systems*, pp. 1929-1934, 1989.
- [20] K. Chakrabarty, "Test Scheduling for Core-Based Systems Using Mixed-Integer Linear Programming," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 19, No. 10, pp. 1163-1174, October 2000.
- [21] S. Chakravarty, V. Dabholkar, "Minimizing Power Dissipation in Scan Circuits During Test Application," *IEEE International Workshop on Low Power Design*, pp. 51-56, 1994.
- [22] A. Chandrakasan, T. Sheng, R. W. Brodersen, "Low Power CMOS Digital Design," *Journal of Solid State Circuits*, Vol. 27, No. 4, pp. 473-484, 1992.
- [23] M. Chatterjee, D. K. Pradhan, "A Novel Pattern Generator for Near-Perfect Fault-Coverage," *IEEE VLSI Test Symposium*, pp. 417-425, 1995.
- [24] C. L. Chen, "Linear Dependencies in Linear Feedback Shift Registers," *IEEE Transactions on Computers*, Vol. C-35, No. 12, pp. 1086-1088, 1986.
- [25] S. Chiusano, F. Corno, P. Prinetto, "Exploiting Behavioral Information in Gate level ATPG," *Journal of Electronic Testing; Theory and Applications (JETTA)*, No. 14, pp. 141-148, 1999.

- [26] Core Test Language Web site,
<http://grouper.ieee.org/groups/ctl>.
- [27] F. Corno, P. Prinetto, M. Rebaudengo, M. Sonza Reorda, "GATTO: A Genetic Algorithm for Automatic Test Pattern Generation for Large Synchronous Sequential Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 15, No. 8, pp. 991-1000, August 1996.
- [28] F. Corno, M. Sonza Reorda, G. Squillero, "High Level Observability for Effective High level ATPG," *IEEE VLSI Test Symposium*, pp. 411-416, 2000.
- [29] E. Cota, L. Carro, M. Lubaszewski, A. Orailoglu, "Test Planning and Design Space Exploration in a Core-based Environment," *Design, Automation and Test in Europe Conference*, pp. 478-485, 2002.
- [30] F. DaSilva, Y. Zorian, L. Whetsel, K. Arabi, R. Kapur, "Overview of the IEEE P1500 Standard," *IEEE International Test Conference*, pp. 988-997, 2003.
- [31] 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, Apr. 1978.
- [32] S. Devadas, M. Malik, "A Survey of Optimization Techniques Targeting Low Power VLSI Circuits," *IEEE/ACM Design Automation Conference*, pp. 242-247, 1995.
- [33] K. I. Diamantaras, N. K. Jha, "A New Transition Count Method for Testing of Logic Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 10, No. 3, pp. 407-410, March 1991.
- [34] D. L. Dill, "What's Between Simulation and Formal Verification?" *IEEE/ACM Design Automation Conference*, pp. 328-329, 1998.

- [35] C. Dufaza, C. Chevalier, L. F. C. Lew Yan Voon, "LFSROM: A Hardware Test Pattern Generator for Deterministic ISCAS85 Test Sets," *IEEE Asian Test Symposium*, pp. 160-165, 1993.
- [36] G. Edirisooriya, J. P. Robinson, "Design of Low Cost ROM Based Test Generators," *IEEE VLSI Test Symposium*, pp. 61-66, 1992.
- [37] E. B. Eichelberg, E. Liddbloom, "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.
- [38] R. D. Eldred, "Test Routines Based on Symbolic Logic Systems," *Journal of the ACM*, Vol. 6, No. 1, pp. 33-36, January 1959.
- [39] P. Eles, K. Kuchcinski, Z. Peng, "System Synthesis with VHDL", *Kluwer Academic Publishers*, 1997.
- [40] F. Fallah, P. Ashar, S. Devadas, "Simulation Vector Generation from HDL Descriptions for Observability-Enhanced Statement Coverage," *IEEE/ACM Design Automation Conference*, pp. 666-671, 1999.
- [41] F. Ferrandi, F. Fummi, D. Sciuto, "Implicit Test Generation for Behavioral VHDL Models," *IEEE International Test Conference*, pp. 587-596, 1998.
- [42] F. Ferrandi, G. Ferrara, D. Scuito, A. Fin, F. Fummi, "Functional Test Generation for Behaviorally Sequential Models," *Design, Automation and Test in Europe*, pp. 403-410, 2001.
- [43] D. Flynn, "AMBA: Enabling Reusable On-Chip Designs," *IEEE Micro*, Vol. 17, No. 4, 1997, pp. 20-27.

- [44] L. Formaggio, F. Fummi, G. Pravadelli, "A Timing-Accurate HW/SW Co-simulation of an ISS with SystemC," *CODES+ISSS*, pp. 152-157, 2004.
- [45] H. Fujiwara, T. Shimono, "On the Acceleration of Test Generation Algorithms," *IEEE Transactions on Computers*, Vol. C-32, No. 12, pp. 1137-1144, December 1983.
- [46] R. A. Frohwerk, "Signature Analysis, A New Digital Field Service Method," *Hewlett Packard Journal*, Vol. 28, No. 9, pp. 2-8, May 1977.
- [47] D. D. Gajski, R. H. Kuhn, "Guest Editor's Introduction: New VLSI Tools," *IEEE Computer*, December 1983.
- [48] D. D. Gajski, F. Vahid, S. Narayan, J. Gong, "Specification and Design of Embedded Systems," *Prentice-Hall*, 1994.
- [49] S. Gerstendörfer, H.J. Wunderlich, "Minimized Power Consumption for Scan-based BIST," *IEEE International Test Conference*, pp. 77-84, 1999.
- [50] S. Ghosh, T. J. Chakraborty, "On Behavior Fault Modeling for Digital Designs," *Journal of Electronic Testing: Theory and Applications*, Vol. 2, No. 2, pp 135-151, 1991.
- [51] N. Giambiasi, "Test Pattern Generation for Behavioral Descriptions in VHDL," *VHDL conference*, pp 228-234, 1991.
- [52] P. Girard, L. Guiller, C. Landrault, S. Pravossoudovitch, "A Test Vector Inhibiting Technique for Low Energy BIST Design," *IEEE VLSI Test Symposium*, pp. 407 – 412, 1999.
- [53] P. Girard, L. Guiller, C. Landrault, S. Pravossoudovitch, J. Figueras, S. Manich, P. Teixeira, M. Santos, "Low-Energy BIST Design: Impact of the LFSR TPG Parameters on the Weighted Switching Activity," *IEEE International Symposium on Circuits and Systems*, pp 110-113, 1999.

- [54] P. Girard, "Low Power Testing of VLSI Circuits: Problems and Solutions," *IEEE International Symposium on Quality Electronic Design*, pp. 173-179, 2000.
- [55] P. Girard, L. Guiller, C. Landrault, S. Pravossoudovitch, H.J. Wunderlich, "A Modified Clock Scheme for a Low Power BIST Test Pattern Generator," *VLSI Test Symposium*, pp. 306-311, 2001.
- [56] D. Gizopoulos, N. Kranitis, A. Paschalis, M. Psarakis, Y. Zorian, "Low Power/Energy BIST Scheme for Datapaths," *VLSI Test Symposium*, pp. 23-28, 2000.
- [57] F. Glover, "Future Paths for Integer Programming and Links to Artificial Intelligence," *Computers & Operations Research*, No. 5, pp. 533-549, 1986.
- [58] F. Glover, E. Taillard, D. de Werra, "A User's Guide to Tabu Search," *Annals of Operations Research*, No. 41, pp. 3-28, 1993.
- [59] F. Glover, M. Laguna. "Modern Heuristic Techniques for Combinatorial Problems", *Blackwell Scientific Publishing*, pp. 70-141, 1993.
- [60] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Transactions on Computers*, Vol. C-30, No. 3, pp. 215-222, March 1981.
- [61] S. K. Goel, K. Chiu, E. J. Marinissen, T. Nguyen, S. Oostdijk, "Test Infrastructure Design for the Nexperia™ Home Platform PN8550 System Chip," *Design, Automation, and Test in Europe*, pp. 108-113 (Designer's Forum Proceedings), 2004.
- [62] S. W. Golomb, "Shift Register Sequences," *Aegan Park Press*, 1982.

- [63] A. Gupta, "Formal Hardware Verification Methods: A Survey," *Formal Methods in System Design*, Vol. 1, pp. 151-238, 1992.
- [64] A. K. Gupta, J. R. Armstrong, "Functional Fault Modeling and Simulation for VLSI Devices," *IEEE/ACM Design Automation Conference*, pp 720-726, 1985.
- [65] R. K. Gupta, Y. Zorian, "Introduction to Core-Based System Design," *IEEE Design and Test of Computers*, Vol. 14, No. 4, pp. 15-25, 1997.
- [66] P. Hansen, "The Steepest Ascent Mildest Descent Heuristic for Combinational Programming," *Congress on Numerical Methods in Combinatorial Optimization*, 1986.
- [67] P. Harrod, "Testing Reusable IP - A Case Study," *IEEE International Test Conference*, pp. 493-498, 1999.
- [68] Z. He, G. Jervan, Z. Peng, P. Eles, "Hybrid BIST Test Scheduling Based on Defect Probabilities," *IEEE Asian Test Symposium*, pp. 230-235, 2004.
- [69] S. Hellebrand, S. Tarnick, J. Rajske, B. Courtois, "Generation Of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," *IEEE International Test Conference*, pp. 120-129, 1992.
- [70] S. Hellebrand, J. Rajske, S. Tarnick, S. Venkataraman, B. Courtois, "Built-In Test for Circuits with Scan Based on Reseeding of Linear Feedback Shift Registers," *IEEE Transactions on Computers*, Vol. 44, No. 2, pp. 223-233, February 1995.
- [71] S. Hellebrand, B. Reeb, S. Tarnick, H.-J. Wunderlich, "Pattern Generation for Deterministic BIST Scheme," *IEEE/ACM International Conference on CAD-95*, pp. 88-94, 1995.

- [72] S. Hellebrand, H.-J. Wunderlich, A. Hertwig, "Mixed-Mode BIST Using Embedded Processors," *Journal of Electronic Testing: Theory and Applications*, No. 12, pp. 127-138, 1998.
- [73] F. C. Hennie, "Fault Detecting Experiments for Sequential Circuits," *Annual Symposium on Switching Theory and Logical Design*, pp. 95-110, 1974.
- [74] G. Hetherington, T. Fryars, N. Tamarapalli, M. Kassab, A. Hassan, J. Rajski, "Logic BIST for Large Industrial Designs: Real Issues and Case Studies," *IEEE International Test Conference*, pp. 358-367, 1999.
- [75] P. D. Hortensius, R. D. McLeod, W. Pries, D. M. Miller, H. C. Card, "Cellular Automata-Based Pseudorandom Number Generators for Built-In Self-Test," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 8 , No. 8, pp. 842-859, August 1989.
- [76] S.-Y. Huang, K.-T. Cheng, "Formal Equivalence Checking and Design Debugging," *Kluwer Academic Publishers*, 1998.
- [77] Y. Huang, W.-T. Cheng, C.-C. Tsai, N. Mukherjee, O. Samman, Y. Zaidan, S. M. Reddy, "Resource Allocation and Test Scheduling for Concurrent Test of Core-based SOC Design," *IEEE Asian Test Symposium*, pp. 265-270, 2001.
- [78] S. D. Huss, R. S. Gyurcsik, "Optimal Ordering of Analog Integrated Circuit Tests to Minimize Test Time," *IEEE/ACM Design Automation Conference*, pp. 494-499, 1991.
- [79] H. Higuchi, N. Ishiura, S. Yajima, "Compaction of Test Sets Based on Symbolic Fault Simulation," *Synthesis and Simulation Meeting and International Interchange*, pp. 253-262, 1992.

- [80] O. H. Ibarra, S. Sahni, "Polynomially Complete Fault Detection Problems," *IEEE Transactions on Computers*, Vol. C-24, No. 3, pp. 242-249, March 1975.
- [81] "IEEE Standard Test Access Port and Boundary-Scan Architecture," IEEE Standard 1149.1, *IEEE Press*, 1990.
- [82] V. Iyengar, K. Chakrabarty, B. T. Murray, "Deterministic Built-in Pattern Generation for Sequential Circuits," *Journal of Electronic Testing: Theory and Applications*, pp. 97-114, No. 15, 1999.
- [83] V. Iyengar, K. Chakrabarty, E. J. Marinissen, "Test Wrapper and Test Access Mechanism Co-optimization for System-on-Chip," *IEEE International Test Conference*, pp. 1023-1032, 2001.
- [84] M. Jacomino, J.-L. Rainard, R. David, "Fault Detection in CMOS Circuits by Consumption Measurement," *IEEE Transactions on Instrumentation and Measurement*, Vol. 38, No. 3, pp. 773-778, June 1989.
- [85] W. J. Jiang, B. Vinnakota, "Defect-Oriented Test Scheduling," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 9, No. 3, pp. 427-438, 2001.
- [86] G. Jervan, A. Markus, P. Paomets, J. Raik, R. Ubar, "A CAD system for Teaching Digital Test," *European Workshop on Microelectronics Education*, pp. 287-290, 1998.
- [87] G. Jervan, P. Eles, Z. Peng, "A Hierarchical Test Generation Technique for Embedded Systems," *Electronic Circuits and Systems Conference*, pp 21-24, 1999.
- [88] G. Jervan, Z. Peng, R. Ubar, "Test Cost Minimization for Hybrid BIST," *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 283-291, 2000.

- [89] G. Jervan, Z. Peng, R. Ubar, H. Kruus, "A Hybrid BIST Architecture and its Optimization for SOC Testing," *IEEE International Symposium on Quality Electronic Design*, pp. 273-279, 2002.
- [90] 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*, pp 169-174, 2002.
- [91] G. Jervan, P. Eles, Z. Peng, R. Ubar, M. Jenihhin, "Hybrid BIST Time Minimization for Core-Based Systems with STUMPS Architecture," *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 225-232, 2003.
- [92] G. Jervan, P. Eles, Z. Peng, R. Ubar, M. Jenihhin, "Test Time Minimization for Hybrid BIST of Core-Based Systems," *IEEE Asian Test Symposium*, pp. 318-323, 2003.
- [93] G. Jervan, Z. Peng, R. Ubar, O. Korelina, "An Improved Estimation Methodology for Hybrid BIST Cost Calculation," *IEEE Norchip Conference*, pp. 297-300, 2004.
- [94] G. Jervan, R. Ubar, Z. Peng, P. Eles, "An Approach to System-Level DFT" in "System-level Test and Validation of Hardware/Software Systems," M. Sonza Reorda, Z. Peng, M. Violante (editors), *Springer-Verlag*, 2005 (in print).
- [95] G. Jervan, R. Ubar, Z. Peng, P. Eles, "Test Generation: A Hierarchical Approach" in "System-level Test and Validation of Hardware/Software Systems," M. Sonza Reorda, Z. Peng, M. Violante (editors), *Springer-Verlag*, 2005 (in print).
- [96] N. K. Jha, S. Gupta, "Testing of Digital Systems," *Cambridge University Press*, 2003.

- [97] M. W. Johnson, "High Level Test Generation Using Software Testing Metrics," M.Sc. Thesis, *University of Illinois at Urbana-Champaign*, 1994.
- [98] R. Kapur, S. Patil, T. J. Snethen, T. W. Williams, "Design of an Efficient Weighted Random Pattern Generation System," *IEEE International Test Conference*, pp. 491-500, 1994.
- [99] F. Karimi, Y. B. Kim, F. Lombardi, N. Park, "Compression of Partially Specified Test Vectors in an ATE Environment," *Instrumentation and Measurement Technology Conference*, Vol.2, pp. 999-1004, 2003.
- [100] C. Kern, M. R. Greenstreet, "Formal Verification in Hardware Design: A Survey," *ACM Transactions on Design Automation of Electronic Systems*, Vol. 4, No. 2, pp. 123-193, April 1999.
- [101] J. Khare, W. Maly, N. Tiday, "Fault Characterization of Standard Cell Libraries Using Inductive Contamination Analysis (ICA)," *IEEE VLSI Test Symposium*, pp. 405-413, 1996.
- [102] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol. 220, No. 4598, pp. 671-680, 1983.
- [103] B. Koenemann, "LFSR-Coded Test Patterns for Scan Designs," *European Test Conference*, pp. 237-242, 1991.
- [104] S. Koranne, "On Test Scheduling for Core-based SOCs," *International Conference on VLSI Design*, pp. 505-510, 2002.
- [105] A. Krasniewski, S. Pilarski, "Circular Self-Test Path: A Low Cost BIST Technique of VLSI Circuits," *IEEE Transactions on Computer-Aided Design*, Vol. 8, No. 1, pp. 46-55, January 1989.

- [106] B. Könemann, J. Mucha, G. Zwiehoff, "Built-In Test for Complex Digital Integrated Circuits," *IEEE Journal of Solid-State Circuits*, Vol. SC-15, No. 3, pp. 315-319, June 1980.
- [107] M. Lajolo, L. Lavagno, M. Rebaudengo, M. Sonza Reorda, M. Violante, "Behavioral level Test Vector Generation for System-on-Chip Designs," *IEEE International Workshop on High Level Design Validation and Test*, pp. 21-26, 2000.
- [108] W. K. Lam, "Hardware Design Verification: Simulation and Formal Method-Based Approaches," *Prentice Hall*, 2005.
- [109] E. Larsson, Z. Peng, "An Integrated Framework for the Design and Optimization of SOC Test Solutions," *Journal of Electronic Testing; Theory and Applications*, Vol. 18, No. 4, pp. 385-400, August 2002.
- [110] E. Larsson, H. Fujiwara, "Test Resource Partitioning and Optimization for SOC Designs," *IEEE VLSI Test Symposium*, pp. 319-324, 2003.
- [111] E. Larsson, J. Pouget, Z. Peng, "Defect-Aware SOC Test Scheduling," *IEEE VLSI Test Symposium*, pp. 359-364, 2004.
- [112] J. J. LeBlanc, "LOCST: A Built-In Self-Test Technique," *IEEE Design and Test of Computers*, Vol. 1, No. 4, pp. 45-52, 1984.
- [113] J. Lee, J. H. Patel, "Architectural Level Test Generation for Microprocessors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13, No. 10, pp. 1288-1300, October 1994.
- [114] K.-J. Lee, J.-J. Chen, C.-H. Huang, "Broadcasting Test Patterns to Multiple Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol.18, No.12, pp.1793-1802, 1999.

- [115] P. C. Li, T. K. Young, "Electromigrations: The Time Bomb in Deep-Submicron ICs," *IEEE Spectrum*, Vol. 33, No. 9, pp. 75-78, 1996.
- [116] C. Lin, Y. Zorian, S. Bhawmik, "PSBIST: A Partial Scan Based Built-In Self-Test Scheme," *IEEE International Test Conference*, pp. 507-516, 1993.
- [117] E. J. Marinissen, R. Arendsen, G. Bos, H. Dingemanse, M. Lousberg, C. Wouters, "A Structured And Scalable Mechanism for Test Access to Embedded Reusable Cores," *IEEE International Test Conference*, pp. 284-293, 1998.
- [118] E. J. Marinissen, Y. Zorian, "Challenges in Testing Core-Based System ICs," *IEEE Communications Magazine*, pp. 104-109, June 1999.
- [119] E. J. McCluskey, S. Bozorgui-Nesbat, "Design for Autonomous Test," *IEEE Transactions on Computers*, Vol. C-30, No. 11, pp. 866-875, November 1981.
- [120] P. Michel, U. Lauther, P. Duzy, "The Synthesis Approach To Digital System Design," *Kluwer Academic Publishers*, 1992.
- [121] A. Miczo, "The Sequential ATPG: A Theoretical Limit," *IEEE International Test Conference*, pp. 143-147, 1983.
- [122] L. Milor, A. L. Sangiovanni-Vincentelli, "Minimizing Production Test Time to Detect Faults in Analog Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13, No. 6, pp. 796-813, 1994.
- [123] S. Minato, "BDDs and Applications for VLSI CAD," *Kluwer Academic Publishers*, 1996.

- [124] G. E. Moore, "Cramming More Components Onto Integrated Circuits," *Electronics*, Vol. 38, No. 8, pp. 114-117, 1965.
- [125] D. Moundanos, J. A. Abraham, Y. V. Hoskote, "A Unified Framework for Design Validation and Manufacturing Test," *IEEE International Test Conference*, pp. 875-884, 1996.
- [126] B. T. Murray, J. P. Hayes, "Hierarchical Test Generation Using Precomputed Tests for Modules," *IEEE International Test Conference*, pp. 221-229, 1988.
- [127] B. T. Murray, J. P. Hayes, "Testing ICs: Getting to the Core of the Problem," *IEEE Transactions on Computer*, Vol. 29, pp. 32-39, November 1980.
- [128] H. T. Nagle, S. C. Roy, C. F. Hawkins, M. G. McNamer, R. R. Fritzemeier, "Design for Testability and Built-In Self Test: A Review," *IEEE Transactions on Industrial Electronics*, Vol. 36, No. 2, pp. 129-140, May 1989.
- [129] F. Najm, "Transition Density: A New Measure of Activity in Digital Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 12, pp. 310-323, February 1993.
- [130] W. M. Needham, "Nanometer Technology Challenges for Test and Test Equipment," *IEEE Computer*, Vol. 32, No. 11, pp. 52-57, November 1999.
- [131] T. M. Niermann, J. H. Patel, "HITEC: A Test Generation Package for Sequential Circuits," *European Design Automation Conference*, pp. 214-218, 1991.
- [132] P1500 Web site. <http://grouper.ieee.org/groups/1500>.
- [133] M. Pedram, "Power Minimization in IC design: Principles and Applications," *ACM Transactions on Design Automation of Electronic Systems*, Vol. 1, No. 1, pp. 3-56, 1996.

- [134] I. Pomeranz, L. N. Reddy, S. M. Reddy, "Compactest: A Method to Generate Compact Test Sets for Combinatorial Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 12, No. 7, pp. 1040-1049, July 1993.
- [135] I. Pomeranz, S. M. Reddy, "3-Weight Pseudo-random Test Generation Based on a Deterministic Test Set for Combinatorial and Sequential Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 12, No. 7, pp. 1050-1052, July 1993.
- [136] J. Raik, R. Ubar, "Sequential Circuit Test Generation Using Decision Diagram Models," *Design, Automation and Test in Europe*, pp. 736-740, 1999.
- [137] J. Raik, R. Ubar. "Fast Test Pattern Generation for Sequential Circuits Using Decision Diagram Representations." *Journal of Electronic Testing: Theory and Applications*, Vol. 16, No. 3, pp. 213-226, June, 2000.
- [138] P. M. Rosinger, B. M. Al-Hashimi, N. Nicolici, "Scan Architecture for Shift and Capture Cycle Power Reductions," *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 129-137, 2002.
- [139] J. P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method," *IBM Journal of Research and Development*, Vol. 10, No. 4, pp. 278-291, July 1966.
- [140] J. A. Rowson, "Hardware/Software Co-Simulation," *IEEE/ACM Design Automation Conference*, pp. 439-440, 1994.
- [141] E. M. Rudnick, R. Vietti, A. Ellis, F. Corno, P. Prinetto, M. Sonza Reorda, "Fast Sequential Circuit Test Generation Using High Level and Gate Level Techniques," *European Design Automation and Test Conference*, pp. 570-576, 1998.

- [142] S. M. Sait, H. Youssef, "Iterative Computer Algorithms with Application in Engineering. Solving Combinatorial Optimization Problems," *IEEE Computer Society Press*, 1999.
- [143] R. Sankaralingam, B. Pouya, N. A. Touba, "Reducing Power Dissipation During Test Using Scan Chain Disable," *IEEE VLSI Test Symposium*, pp. 319-324, 2001.
- [144] M. B. Santos, F. M. Gonçalves, I. C. Teixeira, J. P. Teixeira, "RTL-Based Functional Test Generation for High Defects Coverage in Digital Systems," *Journal of Electronic Testing, Theory and Application*, Vol. 17, No. 3/4, pp. 311-319, 2001.
- [145] 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.
- [146] L. Shen, S. Y. H. Su, "A Functional Testing Method for Microprocessors," *IEEE Transactions on Computers*, Vol. 37, No. 10, pp 1288-1293, 1988.
- [147] "SICStus Prolog User's Manual," *Swedish Institute of Computer Science*, 2001.
- [148] 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>
- [149] S. Y. H. Su, T. Lin, "Functional Testing Techniques for Digital LSI/VLSI Systems," *IEEE/ACM Design Automation Conference*, pp 517-528, 1984.
- [150] Y. Sun, "Automatic Behavioral Test Generation By Using a Constraint Solver", Final Thesis, *LiTH-IDA-Ex-02/13, Linköping University*, 2001.

- [151] S. Sze, "VLSI Technology," *McGraw-Hill*, 1983.
- [152] S. M. Thatte, J. A. Abraham, "Test Generation for Microprocessors," *IEEE Transactions on Computers*, Vol. 29, No. 6, pp 429-441, 1980.
- [153] "The International Technology Roadmap for Semiconductors. 2003 Edition (ITRS 2003)," *Semiconductor Industry Association*, 2003. <http://public.itrs.net/>
- [154] N. A. Touba, E. J. McCluskey, "Transformed Pseudo-random Patterns for BIST," *IEEE VLSI Test Symposium*, pp. 410-416, 1995.
- [155] N. A. Touba, E. J. McCluskey, "Synthesis of Mapping Logic for Generating Transformed Pseudo-Random Patterns for BIST," *IEEE International Test Conference*, pp. 674-682, 1995.
- [156] "Turbo Tester Reference Manual," *Tallinn Technical University*, 1999. <http://www.pld.ttu.ee/tt>
- [157] R. Ubar, "Test Synthesis with Alternative Graphs," *IEEE Design and Test of Computers*, Vol. 2, pp 48-57, 1996.
- [158] R. Ubar, "Multi-Valued Simulation of Digital Circuits with Structurally Synthesized Binary Decision Diagrams," *Multiple Valued Logic*, Vol. 4, pp 141-157, 1998.
- [159] R. Ubar, "Combining Functional and Structural Approaches in Test Generation for Digital Systems," *Microelectronics Reliability*, Vol. 38, No. 3, pp 317-329, 1998.
- [160] R. Ubar, G. Jervan, Z. Peng, E. Orasson, R. Raidma, "Fast Test Cost Calculation for Hybrid BIST in Digital Systems," *Euromicro Symposium on Digital Systems Design*, pp. 318-325, 2001.

- [161] R. Ubar, H. Kruus, G. Jervan, Z. Peng, "Using Tabu Search Method for Optimizing the Cost of Hybrid BIST," *Conference on Design of Circuits and Integrated Systems*, pp. 445-450, 2001.
- [162] R. Ubar, M. Jenihhin, G. Jervan, Z. Peng, "Hybrid BIST Optimization for Core-based Systems with Test Pattern Broadcasting," *IEEE International Workshop on Electronic Design, Test and Applications*, pp. 3-8, 2004.
- [163] R. Ubar, M. Jenihhin, G. Jervan, Z. Peng, "An Iterative Approach to Test Time Minimization for Parallel Hybrid BIST Architecture," *IEEE Latin-American Test Workshop*, pp. 98-103, 2004.
- [164] R. Ubar, T. Shchenova, G. Jervan, Z. Peng, "Energy Minimization for Hybrid BIST in a System-on-Chip Test Environment," *IEEE European Test Symposium*, 2005 (to be published).
- [165] P. Varma, S. Bhatia, "A Structured Test Re-Use Methodology for Core-Based System Chips," *IEEE International Test Conference*, pp. 294-302, 1998.
- [166] S. Wang, S. Gupta, "ATPG for Heat Dissipation Minimization During Test Application," *IEEE Transactions on Computers*, Vol. 46, No. 2, pp. 256-262, 1998.
- [167] P. C. Ward, J. R. Armstrong, "Behavioral Fault Simulation in VHDL," *IEEE/ACM Design Automation Conference*, pp. 587-593, 1990.
- [168] L. Whetsel, "Adapting Scan Architectures for Low Power Operation," *IEEE International Test Conference*, pp. 863-872, 2000.
- [169] M. J. Y. Williams, J. B. Angell, "Enhancing Testability of Large Scale Integrated Circuit via Test Points and Addi-

tional Logic," *IEEE Transactions on Computers*, Vol. C-22, No. 1, pp. 46-60, January 1973.

- [170] H.-J. Wunderlich, "Multiple Distributions for Biased Random Test Patterns," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 9, No. 6, pp. 594-602, June 1990.
- [171] H.-J. Wunderlich, G. Kiefer, "Bit-Flipping BIST," *ACM/IEEE International Conference on CAD-96*, pp. 337-343, 1996.
- [172] V. N. Yarmolik, I. V. Kachan, "Self-Checking VLSI Design," *Elsevier Science Ltd*, 1993.
- [173] N. Zacharia, J. Rajski, J. Tyzer, "Decompression of Test Data Using Variable-Length Seed LFSRs," *IEEE VLSI Test Symposium*, pp. 426-433, 1995.
- [174] Y. Zorian, "A Distributed BIST Control Scheme for Complex VLSI Devices," *IEEE VLSI Test Symposium*, pp. 4-9, 1993.
- [175] Y. Zorian, "Test Requirements for Embedded Core-based Systems and IEEE P1500," *IEEE International Test Conference*, pp. 191-199, 1997.
- [176] Y. Zorian, E. J. Marinissen, S. Dey, "Testing Embedded Core-Based System Chips," *IEEE International Test Conference*, pp. 130-143, 1998.