Collège Doctoral du Languedoc-Roussillon

THÈSE

Pour obtenir le grade de Docteur

Délivrée par l'**Université de Montpellier**

Préparée au sein de l'école doctorale **I2S**
Et de l'unité de recherche **LIRMM/UMR 5506**

Spécialité: **Systèmes Automatiques et Microélectroniques (SYAM)**

Présentée par **Aymen TOUATI**

Président du Jury: **M. Daniel CHILLET**

Amélioration des Solutions de Test Fonctionnel et Structurel des Circuits Intégrés

***

Improving Functional and Structural Test Solutions for Integrated Circuits

Soutenue le 15-10-2016 devant le jury composé de

M. Stefano DI CARLO	Professeur Associé	Politecnico di Torino	Rapporteur
M. Daniel CHILLET	Professeur	Université de Rennes	Président du jury | Rapporteur
M. Matteo SONZA REORDA	Professeur	Politecnico di Torino	Examinateur
M. Arnaud VIRAZEL	MCF	UM-LIRMM	Examinateur
M. Alberto BOSIO	MCF	UM-LIRMM	Directeur de thèse
M. Patrick GIRARD	DR CNRS	CNRS-LIRMM	Co-Directeur de thèse
M. Paolo BERNARDI	Professeur Associé	Politecnico di Torino	Invité

UNIVERSITÉ DE MONTPELLIER

Acknowledgements

I would like to express my deepest gratitude to my thesis director, Prof. Alberto Bosio. With his encouraging and supporting attitude, meeting him has always been a work-stress reliever for me. His profound guidance and invaluable advises helped keeping my research well directed and my progress on schedule while maintaining my autonomy.

My sincere thanks also goes to my co-director, Prof Patrick Girard, for his immense knowledge, motivation and support throughout the research. I also appreciate Prof. Arnaud Virazel's insightful comments and advises during my research.

My grateful thanks are also extended to Prof. Matteo Sonza Reorda and Prof. Paolo Bernardi for their generous support and effort to actively maintain the collaborative partnership, LAFISI, and their valuable and constructive suggestions that helped me enrich my ideas.

I wish to acknowledge the support received form my friends at LIRMM for the various discussions and brain storming sessions. Also for making the last three years memorable.

Finally, none of this would have been possible without the love and patience of my family (especially my sister Haifa) and my fiancee Syhem. Their constant support and strength has aided and encouraged me throughout this endeavor.

"A life spent making mistakes is not only more honorable, but more useful than a life spent doing nothing"
George Bernard Shaw

Abstract

In light of the aggressive scaling and increasing complexity of digital circuits, meeting the demands for designing, testing and fabricating high quality devices is extremely challenging. Higher performance of integrated circuits needs to be achieved while respecting the constraints of low power consumption, required reliability levels, acceptable defect rates and low cost. With these advances in the SC industry, the manufacturing process are becoming more and more difficult to control, making chips more prone to defects.

Test was and still is the unique solution to cover manufacturing defects; it is becoming a dominant factor in overall manufacturing cost. Even if existing test solutions were able to satisfy the cost-reliability trade-off in the last decade, there are still uncontrolled failure mechanisms. Some of them are intrinsically related to the manufacturing process and some others belong to the test practices especially when we consider the amount of detected defects and achieved reliability.

The main goal of this thesis is to implement robust and effective test strategies to complement the existing test techniques and cope with the issues of test practices and fault models. With the objective to further improve the test efficiency in terms of cost and fault coverage capability, we present significant contributions in the diverse areas of in-field test, power-aware at-speed test and finally scan-chain testing.

A big part of this thesis was devoted to develop new functional test techniques for processor-based systems. The applied methodologies cover both in-field and end-of manufacturing test issues. In the farmer, the implemented test technique is based on merging and compacting an initial functional program set in order to achieve higher fault coverage while reducing the test time and the memory occupation. However in the latter, since we already have the structure information of the design, we propose to develop a new test scheme by exploiting the existing scan chain. In this case we validate the complementary relationship between functional and structural testing while avoiding over as well under-testing issues.

The last contribution of this thesis deals with the test improvement of the most used DFT structure that is the scan chain. We present in this contribution an intra-cell aware testing approach showing higher intra-cell defect coverage and lower test length when compared to conventional cell-aware ATPG. As major results of this effective test solution, we show that an intra-cell defect coverage increase of up to 7.22% and test time decrease of up to 33.5% can be achieved in comparison with cell-aware ATPG.

Key words: Functional Test, Structural Test, ATPG, Power-Aware At-Speed Test, Test Compaction, Microprocessor Test, SBST, Fault simulation, Intra-Cell Defect, Scan-Chain Testing, Test Quality.

Résumé

OMPTE tenu de la complexité des circuits intégrés actuels qui ne cessent d'integrer de plus en plus de transistors, l'amélioration des tests fonctionnels et structurels de ces circuits est devenue une problématique de plus en plus critique.
En effet, la diminution de la taille de ces transistors induit implicitement une sensibilité toujours plus importante des circuits aux perturbations provenant de l'environnement, mais également aux défauts de fabrication.

Ce contexte technologique conduit donc les concepteurs à prévoir des techniques de test toujours plus élaborées, permettant de garantir des circuits sains après fabrication.

Les travaux présentés dans ce mémoire concernent directement cette problématique et proposent un ensemble de solutions qui satisfont à diverses contraintes, comme les contraintes de consommation de puissance au cours du test, les contraintes liées à la couverture de défauts temporels, celles liées au temps de test, à l'occupation mémoire générée par le stockage des vecteurs de test , etc.

Les solutions de test proposées dans cette thése, combinées avec d'autres techniques, permettent de détecter plus de fautes dans les cœurs de processeurs, mais aussi de considérer des modéles de faute plus réalistes dans le cas des circuits munis de chaînes de scan, technique de conception en vue de test DFT généralement utilisée dans l'industrie.

Les travaux sont illustrés sur la base de deux cœurs de processeur en particulier, mais peuvent s'étendre à d'autres cœurs sans réelles difficultés.

Mots clés: Test Fonctionnel, Test Structurel, DFT, Test des Microprocesseurs, ATPG.

Contents

Acknowledgements **i**

Abstract **iii**

Résumé **v**

List of Figures **xi**

List of Tables **xv**

1 Introduction **1**

2 A Comprehensive Evaluation of Functional Programs for Power-Aware Test **5**

2.1 Introduction 5

2.2 Evaluation Framework 8

2.3 Experimental Results 11

2.3.1 Experimental Setup 11

2.3.2 Functional Coverage Evaluation Metrics 12

2.3.3 Structural Coverage Evaluation Metrics 14

2.4 Summary 17

3 An Effective Approach for Functional Test Programs Compaction **19**

3.1 Introduction 19

3.2 Context and Background 21

3.3 Fault Coverage Analysis . . . 22
3.3.1 Fault Coverage Merging . . . 24
3.4 Compaction Technique . . . 27
3.5 Experimental Results . . . 28
3.6 Summary . . . 32

4 Exploring the Impact of Functional Test Programs Re-Used for Power-Aware At-Speed Testing 35

4.1 Introduction . . . 36
4.2 Background . . . 37
4.3 Proposed Methodologies . . . 38
4.4 Experimental Results . . . 41
4.4.1 Fault Coverage Enhancement Results . . . 41
4.4.2 A Global Test Solution Results . . . 44
4.5 Summary . . . 48

5 Scan-Chain Intra-Cell Aware Testing 49

5.1 Introduction . . . 49
5.2 Scan Chain Test . . . 50
5.3 Overall Flow of The Proposed Approach . . . 52
5.4 Defect Characterization . . . 53
5.5 Intra-Cell Defect Grading . . . 56
5.5.1 Scan Chain Test . . . 56
5.5.2 Logic Test . . . 58
5.6 Results and Analysis . . . 59
5.6.1 Defect Grading Results . . . 59
5.6.2 Learning . . . 63
5.7 Summary . . . 67

6 Conclusion 69

Scientific Contributions 73

Bibliography **a**

List of Figures

2.1 Over and under test phenomena 6
2.2 Test program generation 7
2.3 Functional coverage flow 10
2.4 Structural coverage flow 11
2.5 Functional program generation 12
2.6 Statement code coverage evaluation 12
2.7 Branch code coverage evaluation 13
2.8 Condition code coverage evaluation 13
2.9 Expression code coverage evaluation 14
2.10 Toggle code coverage evaluation 14
2.11 Stuck-at fault coverage 15
2.12 Stuck-at fault observability 15
2.13 Transition fault coverage 16
2.14 Transition fault observability 16

3.1 Stuck-at fault coverage achieved by functional test programs generated for the minimips 22
3.2 Test redundancy 23
3.3 Flow for determining the set of detected faults 24
3.4 Merged stuck-at fault coverage for the minimips 25
3.5 Merged transition fault coverage for the minimips 26
3.6 Compaction pseudo-code 27

3.7 Redundant pseudo-code 27
3.8 % of test reduction (stuck-at fault mc8051) 28
3.9 Test time distribution 30
3.10 Memory occupation distribution 31
3.11 Reduction and test Time (Transition faults minimips) 32

4.1 Functional test programs peak-power consumption (mc8051 core) 37
4.2 Observability issues (%) mc8051 37
4.3 Functional test program waveforms 38
4.4 LOC & Launch-off-Shift (LOS) test schemes 39
4.5 Mapping functional test program to Launch-off-Capture (LOC) test scheme . . 40
4.6 Functional programs to test patterns 41
4.7 LOC and LOS detected faults repartition 43
4.8 a) LOS vs Func2LOC, b) LOC vs Func2LOC 44
4.9 A global test solution 45
4.10 Transition fault improvement by applying a global test solutions 45
4.11 A global test solution (mc8051 case study) 46
4.12 Cumulative transition fault coverage : mc8051 48

5.1 MUX-scan flip-flop 51
5.2 Full scan architecture 51
5.3 Overall flow 52
5.4 Spice simulation 55
5.5 Schematic view of a MUX21 55
5.6 Scan-chain test intra-cell defect grading 57
5.7 Logic test intra-cell defect grading 58
5.8 Logic test functional window, corresponding to the period when scan enable = 0 59
5.9 Fault & defect coverage evaluation of the ATPG patterns 61
5.10 Defect coverage evaluation for scan-chain test 62
5.11 Defect coverage capabilities of the different test sets 63
5.12 Delay fault User-Defined Fault Modeling (UDFM) example: MUX21 65

5.13 Delay fault UDFM example: MUX-Scan Flip-Flop 65
5.14 Proposed test solution 66

List of Tables

3.1 Mc8051 & minimips gate-level characteristics 22
3.2 Functional test programs analysis 23
3.3 Test reduction (%) 29
3.4 Test time example 29
3.5 Test time reduction (%) 32

4.1 Mc8051: LOC and LOS test set characteristics 42
4.2 Functional test programs characteristics 42
4.3 Faults left undetected by LOC and LOS 44

5.1 Defect database 54
5.2 Multiplexer defect database example 55
5.3 Defect coverage evaluation for logic test 61
5.4 Defect coverage evaluation for scan-chain test 62
5.5 Logic test overall defect coverage 63
5.6 Intra-cell aware ATPG construction 66
5.7 Intra-cell aware ATPG comparison 67

List of Acronyms

IC Integrated Circuit 4

CPU Central Processing Unit 7

EA Evolutionary Algorithm 8

HDL Hardware Description Language 14

CDV Coverage-Driven Verification 9

CUT Circuit Under Test 5

FC Fault Coverage 59

SAF Stuck-at Fault 59

TF Transition Fault 59

SBST Software-Based Self-Test 3

DAGs Directed Acyclic Graphs 7

ALU Arithmetic Logic Unit 21

CISC Complex Instruction Set Computing 8

RISC Reduced Instruction Set Computing 21

CC Clock Cycles 29

DfT Design for Testability 35

LOC Launch-off-Capture xii

LOS Launch-off-Shift xii

Func2LOC Functional-programs mapped into LOC scheme 42

RTL Register Transfer Level 9

ATPG Automatic Test Pattern Generator 35

VCD Value Change Dump 11

FSM Finite State Machine 9

SE Scan Enable 50

SI Scan-In 50

SPICE Simulation Program with Integrated Circuit Emphasis 54

DC Defect Coverage 56

LT Logic Test 63

ST Scan-chain Test 63

UDFM User-Defined Fault Modeling xii

Chapter 1

Introduction

The ever-increasing advances in semiconductor technology continue to open new issues for research in the domains of digital circuit design and process development. Designing and manufacturing smaller, faster, cheaper and low-power devices are some of the main challenges that for the semiconductor industry. The incessant increase in density and the corresponding decrease in feature sizes of integrated circuits as evinced by Mooreś Law [1] have been a driving force in the progress of the industry for the past few decades. However, scaling of CMOS structures into the nanometer domain has posed new challenges to the physical design and reliability of circuits. The reasons for this are manifold [2] [3] [4]. Firstly, manufacturing structures much smaller than the wavelength of light used in modern lithography are difficult to fabricate and can be practically done only with certain coarse limits. Similarly it is difficult to control the doping concentration for transistors in the nanometer range. Further, the structures are located closer to each other with every technology node, resulting in even the smallest of impurities or metal silvers being able to create shorts or other defects. Lastly, as the number of transistors, wires, contacts and vias on a single chip increase, the probability of one or more being faulty increases. These limitations have resulted in increasing the probability of defects in advanced technology nodes.

A circuit defect may lead to a fault causing an error that can result in a system failure. Manufacturing defects are physical defects introduced during manufacturing that cause the circuit to fail to function properly. The diversity of defects makes difficult to generate meaningful tests maximizing the defect coverage. Fault models which are gate-level representation of physical defects are necessary for generating and evaluating a set of test patterns. Generally, a good fault model should accurately reflect the behavior of the physical defects and should be computationally efficient in terms of the time required for fault simulation and test generation [5]. Many fault models have been proposed so far, but unfortunately, no single fault model accurately reflects the behavior of all possible defects that can occur. As a result, a combination of different fault models is often used in the generation and evaluation of test patterns. Some well-known and commonly used fault models are the stuck-at, the bridging and the delay fault models. The stuck-at fault is a logical fault model that has been used successfully for decades. The gate-level stuck-at fault transforms the correct value on the

faulty signal line to appear to be stuck-at a constant logic value, either logic 0 or 1, referred to as stuck-at-0 (SA0) or stuck-at-1 (SA1), respectively. At the switch level, a transistor can be stuck-off or stuck-on, and these faults are also referred to as stuck-open or stuck-short faults respectively. A short between two wires is commonly referred to as a bridging fault. Delay faults have become more prevalent with decreasing feature sizes, and are used to model the excessive delays caused by resistive opens and shorts in wires as well as parameter variations in transistors. In the gate-delay fault and transition fault models, a delay fault occurs when the time interval for a transition through a single gate exceeds its specified range. The path-delay fault model, on the contrary, considers the cumulative propagation delay along any signal path through the circuit. The small delay defect model takes into consideration the timing delays associated with the fault sites and propagation paths from the layout [6].

This in turn necessitates the need for having effective test architectures and methodologies for efficiently capturing all modeled faults while maintaining ease of application. Different test methodologies like parametric, functional and structural testing are combined together to maximize the fault detection. Parametric testing is typically used to check the electrical properties of the device and can be used to characterize any potential systematic issues with the process node. These tests may not check any functionality of the device but can find gross shorts, opens, leakage issues, or current drive problems. One of the uses of functional tests is for design verification wherein it is checked if the output responses are inline with expected values according to design specifications.

However it is considered very time consuming and very expensive due to the exhaustive nature of applying test patterns to cover each known reachable functional state. Structural tests deal with observing the state of internal signals at the primary outputs of a circuit. Unlike functional testing, it does not require enumeration of all functional states to test the design, so test volumes are not as large.

For any modern chip design with a considerably large portion of logic, embedding design for test (DFT) structures has become a mandatory part of the design process that helps to reduce the complexity of testing sequential circuits. In this context, scan design has been widely adopted across the industry due to the ability to achieve high fault coverage with relatively low overhead. The basic concept of a scan test is to connect memory elements like flip-flops or latches forming chains, so that shifting through scan chains allows controlling and observing the states of the DUT.

Delay tests verify that a design operates correctly at the specified clock speed. Application of at-speed tests for detection of delay faults in synchronous sequential circuits can be done using scan-based structural tests generated by an automatic test pattern generator (ATPG) or they can also be applied using the functional patterns [7]. Depending on how the transition is launched and captured during test, there are two delay test pattern generation methods in scan test environment. In the first method, referred to as Launch-Off-Shift (LOS), the last shift of the scan chain load also serves as the transition launch event whereas in the second method, referred to as Launch-Off-Capture (LOC), the entire scan data shifting is done at slow speeds and then two at-speed clocks are pulsed for launch and capture [8] [9] [10].

As device geometries continue to shrink, deep sub-micron delay defects become prominent, thereby increasing the need for at-speed tests. As tests generated for one fault model can

potentially detect faults of other models, identifying a good order of fault models to target during test generation can help reduce the number of test vectors and, in turn, test time.

The ideal goal while developing test patterns is to achieve 100% fault coverage with no test escapes. In conventional test flows, usually multiple tests are performed with different patterns generated for targeting different fault models. In this scenario, there is a great need for a unified test solution which may deal with both design verification and manufacturing testing if a potential correlation exists between these metrics. Although structural ATPG tests generally provide high fault coverage, they may lead to significant yield loss due to over-testing. This is due to the fact that some of these patterns may activate paths which are not required to meet system speed (for e.g. non functional paths), and these patterns may fail during production test. Herein the challenge lies in complementing functional test patterns with the high fault coverage obtained by structural ATPG patterns. In this context, there is a need for efficiently merging functional and structural test sets to develop an optimized test solution. Furthermore, the test pattern generation process must efficiently take into account the multiple diverging metrics of high fault coverage, low memory requirement and test-time while respecting the test power budget.

This is particularly relevant for modern processor based systems where the most efficient strategy for applying functional tests is the Software-Based-Self-Test (SBST). SBST is a test methodology based on the use of test programs to cover the faults inside specific processor blocks. A test program is first loaded in the processorś program memory, then executed and its results collected. These results can be used as a signature to determine the correct functioning of the target block. SBST is employed for complying with new standards mainly in the critical aviation and automotive applications that specify hard constraints regarding execution time, memory occupation, test frequency and test coverage.

Another issue related to the modern deep submicron technologies, is the fact that physical defects do not appear only in the cell interconnections but also inside the cell itself. They are called intra-cell defects [11]. It has been observed that these defects can escape classical test solutions using classical fault models (i.e., the inter-cell fault models). This phenomenon has been already investigated in the literature but mainly for the combinational gates [12] [13]. However, intra-cell defects affecting sequential elements (i.e., flip-flops) have not been targeted so far. Usually, scan chain test is applied only when flip-flops are in test mode. This classical way of testing the scan flip-flops results in low intra-cell defect coverage. Hence we need to review the applied shift-test to overcome these systematic intra-cell defects.

The main contribution of this thesis consists in providing effective test solutions targeting the issues encountered at different abstraction levels starting from the transistor level up_to the system level. More in details, we will investigate the Software-Based Self-Test (SBST) techniques and propose a methodology able to increase the achieved fault coverage without affecting the test length. Moreover, we will show how the SBST technique can be used coupled with structural test to maximize the fault coverage. In the second part of the thesis, we target the test of intra-cell defects affecting sequential elements. We will show that combining together different tests can lead to a meaningful defect coverage compared to existing tools and solutions.

The manuscript is organized in five chapters: Chapter 2 details the context and motivation

of utilizing functional tests. It starts by discussing one of the most critical aspects of Integrated Circuit (IC) testing i.e. power-aware testing which has been the subject of many prior research works. The adopted technique consists of generating a set of power-hungry functional test programs which are used to tune the test power limits. Our first contribution provides a comprehensive evaluation of such programs taking into account other interesting test and verification metrics. Later in Chapter 2 we present a correlation analysis between all these metrics (i.e., the switching activity, the fault coverage and the verification of the design), collected at different levels.

Chapter 3 introduces the functional self-testing also called SBST strategy. The adoption of cumulative fault coverage of functional test programs and related constraints are discussed. Later in this chapter, we propose a static test compaction technique targeting a high test quality and minimum test length. Moreover in this chapter, the critical memory occupation is considered while implementing the test compaction technique.

Chapter 4 is devoted to describe functional test programs fault coverage analysis and test solutions targeting the observability issues and the property of each type of test (structural vs functional). This chapter first provides an in-depth analysis on the impact of reusing functional test programs for at- speed test by exploiting the existing scan chain circuitry. More specifically, we propose to map the functional test programs into a LOC-like test scheme. This analysis shows an increase of the fault coverage especially when the transition fault model is considered. In addition, a comparative analysis between the structural and the mapped functional test programs shows clearly the benefits of merging both sets and leads to a global test solution. In final part of chapter 4, an efficient test flow is proposed, based on a selection strategy thereby reducing the test length and the shifting power consumption since a scan-test scheme is used during the proposed mapping technique. We propose a refined test length compaction based on pattern fault simulation and a constrained static test compression.

Chapter 5 covers a more granular topic concerning basically the structural scan testing. This study presents an in-depth study of the impacts of intra-cell systematic defects that first of all escape the classical test solutions using classical fault models and also may affect the scan chain test. This topic represents a sort of bridge between transistor and gate abstraction levels. As a first step, the basics of scan chain design and test are reviewed. Next, we briefly introduce the defect characterization process and show the intra-cell defects escape rate when standard tests are applied. This was our motivation to propose a high quality test solution. This latter is based on combining different test sets, generated by a commercial ATPG and targeting different fault models. An experimental case study of the application of the developed high quality test solution and other available commercial intra-cell-Aware ATPG is presented. Finally, a comparative analysis concludes the chapter.

Chapter 6 summarizes the contributions of this work and presents some future perspectives.

Chapter 2

A Comprehensive Evaluation of Functional Programs for Power-Aware Test

THis chapter provides the first handling of the functional test programs and offers an overview of the context and the motivation of dealing with functional test in this thesis. In Section 2.1, a brief introduction of one of the most critical issues of IC manufacturing is presented (i.e, test overhead), presenting so one of the recent works which copes with test-power issues. Next, Section 2.2 provides a detailed and comprehensive evaluation of the generated functional programs taking into account other interesting test and verification metrics. Moreover, with this latter, we could detect any correlation between the corresponding switching activity and those testing and verification metrics and generate based on that a meaningful test solution. Finally, Section 2.3 presents the obtained preliminary results which will validate or not this correlation. Concluding remarks are given in Section 2.4. We note that since the Circuit Under Test (CUT) is a processor device, the considered functional patterns will refer to programs executed by the processor.

2.1 Introduction

IC manufacturers exploit the technology scaling to produce smaller and faster electronic devices. However, manufacturing defects affecting latest technologies lead to dynamic faults (i.e., delay faults) [14]. Such faults require at-speed test to be detected. Usually, at-speed test is achieved by exploiting structural test patterns.

At-speed structural test leads to excessive power consumption that can either damage the Circuit Under Test (CUT) or lead to yield loss [15]. In fact, the yield loss is a case in which a good chip fails the test. This happens when the test exercises intensively the device making a high number of the capacitance charging and discharging simultaneously and results in phenomena such as IR drop and crosstalk [16] which in turn cause the critical path to be decelerated and results in a false fault detection.

Reducing the power during test is a well-known technique, but reducing too much the power consumption leads to test escapes phenomena as depicted in Figure 2.1. Test escape, in the other side, is a case in which a faulty device passes the test. This happens when the test is not exercising the device to an extent where the switching activity is sufficient to cause the critical path to decelerate enough to make small delay defects observable and detectable. Thus remain undetected and the device escapes the test.

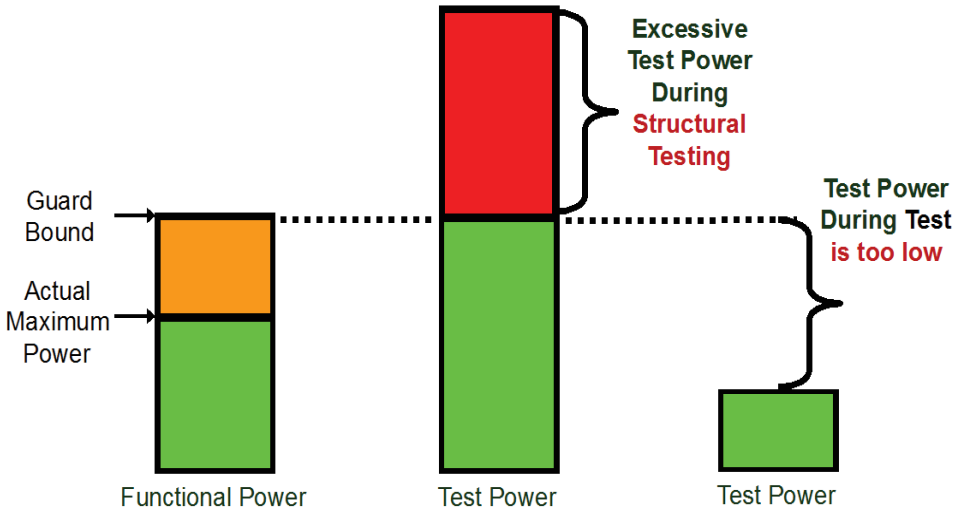


Figure 2.1: Over and under test phenomena

Therefore, to cope with the above issues, we have to control this risk by limiting the test power depending on the functional power of the device itself [15]. The knowledge of the actual functional power is thus mandatory. In [17], authors propose a generator of functional test programs for microprocessor core. The generator aims at maximizing the power consumption of the target microprocessor, thus the generated programs are good candidates to accurately estimate the functional power limits (i.e., to avoid both over- and under-test).

The functional programs generation has been carried out by Politecnico di Torino group [18][19] and hereunder to understand how this functional programs generator works. We refer to this generic and simplified version which will target the optimization of a given metric. It could be the induced power consumption, the fault coverage or also how much stressful are the generated test programs.

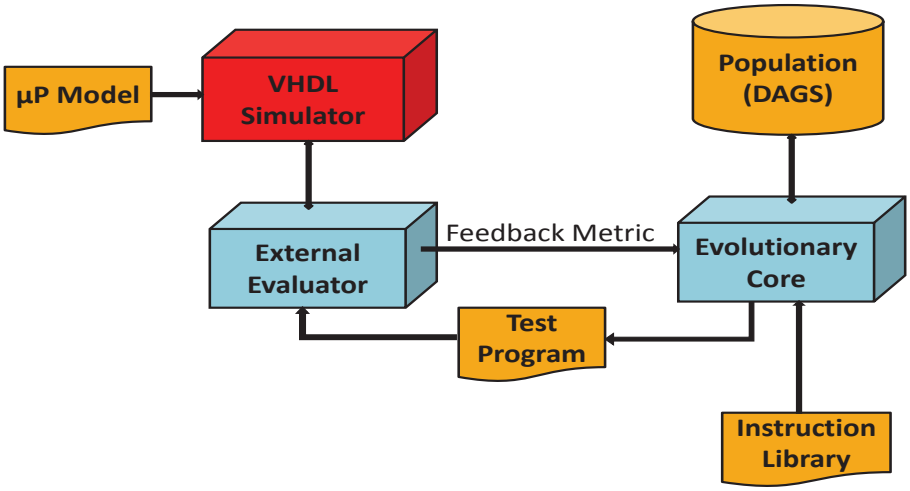


Figure 2.2: Test program generation

Since our target is Central Processing Unit (CPU) cores, the input functional patterns are assembly programs. Figure 2.2 sketches the overall routine of this proposed efficient test programs generator composed of an evolutionary core, an instruction library and an external evaluator, totally separate blocks.

The evolutionary core cultivates a population of assembly programs which are internally represented as Directed Acyclic Graphs (DAGs), composed of different types of nodes. Basically, it uses auto-adaptation mechanisms, dynamic operator probabilities, dynamic operator strength and variable population size. The instruction library is used to map individuals to valid assembly language programs. So according to the instruction library which contains a highly concise description of the assembly syntax, each node is mapped to a different instruction. Finally, the external evaluator simulates the assembly program providing the necessary feedback to the evolutionary core.

By looking to this figure, we can rapidly figure out the importance of the feedback mechanism which permanently guides the optimizer (implemented in the evolutionary core). This optimizer is based on a technique called MicroGP (μGP), an evolutionary system able to automatically devise and optimize a program written in a general assembly-like language [20]. Some features of μGP stem from genetic programming (GP) which is a technique whereby computers programs are encoded as a set of genes that are then modified using a generic population-based meta-heuristic optimization algorithms also known as Evolutionary Algorithms (EA). These latter, implemented in the evolutionary core in Figure 2.2, may be defined as local search algorithms since they sample a region of the search space dependent upon their actual state. The μGP is firstly fed by an initial set of test programs, randomly generated. These latter will be evaluated depending on the targeted metric using an external evaluator. Then, based on the feedback information (the estimated value), the generated test programs will be optimized.

Now back to our case study, the external evaluator tool corresponds to a power estimator tool which will determine the maximum peak power and thus driving the μGP for correctly generating functional patterns with maximum peak power consumption (survival of the

fittest). From an initial random population and resorting to Evolutionary Algorithm (EA), the μGP tool is able to maximize the power metric discarding the weak ones and improving the stronger patterns.

Since generated programs maximize the power consumption, they are definitively characterized by a high switching activity [17]. In other words, they could be also good candidates for delay fault testing. In this work, we would like to investigate more in detail the capability of such generated functional-programs to satisfy other metrics than the power consumption. In this chapter, we will focus on two metrics: structural and functional. The first one is related to the fault coverage, where the fault models are the stuck-at and the transition faults. The second one corresponds to the code coverage, such as statement, branch and toggle coverage.

The main goal of this comprehensive evaluation is to verify whether or not the above metrics are in some way correlated in order to provide a meaningful test that can be reused many times during the overall product flow (i.e., during verification and for power aware test). Our case study (i.e., targeted microprocessor) is the Intel mc8051 non-pipelined Complex Instruction Set Computing (CISC) processor, with 8-bit ALU and 8-bit registers, synthesized with a 65nm industrial technology as already presented in [17]. Next section will describe the evaluation framework for both structural and functional metrics.

2.2 Evaluation Framework

Previous work [17] proved that the switching activity correlates with the overall power consumption, therefore, we have to investigate if there exist some relations between the switching activity and the considered metrics (i.e., functional and structural). Basically, the question we would like to answer is: can the functional programs, generated to maximize the power consumption, be re-used for verification and test purposes?

In this chapter, we first evaluated the functional test programs generated in [17] with respect to the code and functional coverage. We measured how much these test programs can meet the design quality needs by exercising each piece of code in the behavioral description of the microprocessor.

Before starting the framework evaluation, a brief introduction of these new metrics (i.e., code & functional coverage metrics) could be helpful to understand the representation of each metric in the whole design verification flow. First of all, the code coverage concept is one of the most significant measurements to check the testbench completeness (how thoroughly the entire verification suite exercises the source code). It can help increase our confidence that the verification job is complete, but it should not be our only indicator. In fact, it gives us a great head-start on full verification coverage. Results from code coverage tools should be interpreted with a great precaution. They should be used to help identify corner cases that were not exercised by the verification suite.

Statement coverage : it measures whether or not a given statement was executed which is different from the line coverage for the simple reason that we can find multiple statements on a line.

Branch coverage : it is considered as a refinement of the statement coverage (i.e., more granular). It checks if each possible branch in an "if/else" or "case" statement has been exercised or not.

Usually during the Coverage-Driven Verification (CDV) process, it is easy with these two metrics (i.e., statement and branch coverage metrics) to achieve 100% coverage with the appropriate exclusions to meet the verification intent. In some cases, it is normal for some statements not to be executed (i.e., dead code). As effective practice, as mentioned in [21], this additional code simply monitors for conditions that should never occur and reports that an unexpected condition happened. Every statement and every branch was executed correctly, however the functionality are not completely tested. So, we need to consider more granular metrics.

Condition & Expression coverages : these metrics check if each possible sub-expression has been evaluated or not.Thus would detect the missing evaluations.

Toggle coverage : it measures the amount of activity in the design, such as unused signals, signals that remain constant, or signals that have too few value changes. This metric is very useful for testing tri-state wires.

Finite State Machine (FSM) coverage : this metric enters the realm of functional verification, since it monitors the coverage of the FSM representation of control logic blocks in the design. It is more sophisticated metric and based on behavioral not just language elements.

Additionally, high functional coverage does not necessarily correlate with high code coverage. Whereas code coverage is concerned with recording the mechanics of code execution, functional coverage is concerned with the intent of the implemented function. So two can serve to complement each others. We started by the most basic ones, which are the statement, toggle and branch coverage. Figure 2.3 depicts the overall evaluation frame. Basically, each test program is simulated using Modelsim [22] that reports the verification coverage. For this evaluation the Register Transfer Level (RTL) description of the microprocessor is used.

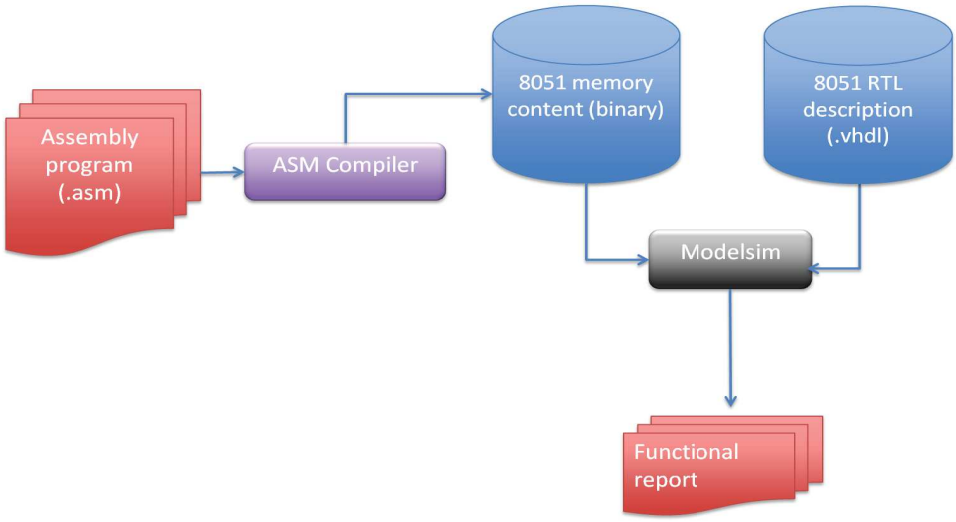


Figure 2.3: Functional coverage flow

The second evaluation has been done with respect to the structural coverage. The test programs can act as test patterns to excite faults inside the gate-level description of the microprocessor. Targeted faults are the stuck-at and transition fault models. Similarly, we take here few lines to recall the definition of stuck-at and transition fault models that will be used for the rest of this thesis as the most relevant structural coverage metrics.

One of the most popular fault models in manufacturing test is the stuck-at fault model. As defined in [23], it is a logical fault model where any wire in the logic circuit can be stuck-at-1 or stuck-at-0. A test vector that produces the opposite value (zero for a stuck-at-1, and one for a stuck-at-0) will excite the fault. The effect of the fault has to be propagated to an observable circuit output in order for the fault to be detected by the vector.

For the second fault model (i.e., the transition fault model), there are two transition faults associated with each signal: a slow-to-rise fault and a slow-to-fall fault. A slow-to-rise (slow-to-fall) transition fault is a logical model for a defect that delays a rising (falling) transition. The extra delay caused by the fault is assumed to be large enough to prevent the transition from reaching the primary outputs at the time of observation. The transition fault is considered to be detected if a transition occurs at the fault site and if a sensitized path extends from the fault site to any primary output.

For any fault model, given a test vector set, the fault coverage of the test vector set can be computed using fault simulation. For every possible fault in the fault model, it is checked for each vector in the vector set whether the fault is excited and propagated to a primary output. Fault coverage for a vector set is defined as the number of detected faults divided by the total number of faults. Fault coverage measures the "goodness" of a vector set in detecting all faults. A test set with higher fault coverage is more likely to detect bad integrated circuits and so fault coverage is used to drive the test generation process.

Figure 2.4 shows the corresponding evaluation flow. In this case each test program is still simulated using Modelsim [22] but the output is the mc8051 activity. The activity is stored

into a Value Change Dump (VCD) file. This file is then used as input for the fault simulator which is Tetramax [24].

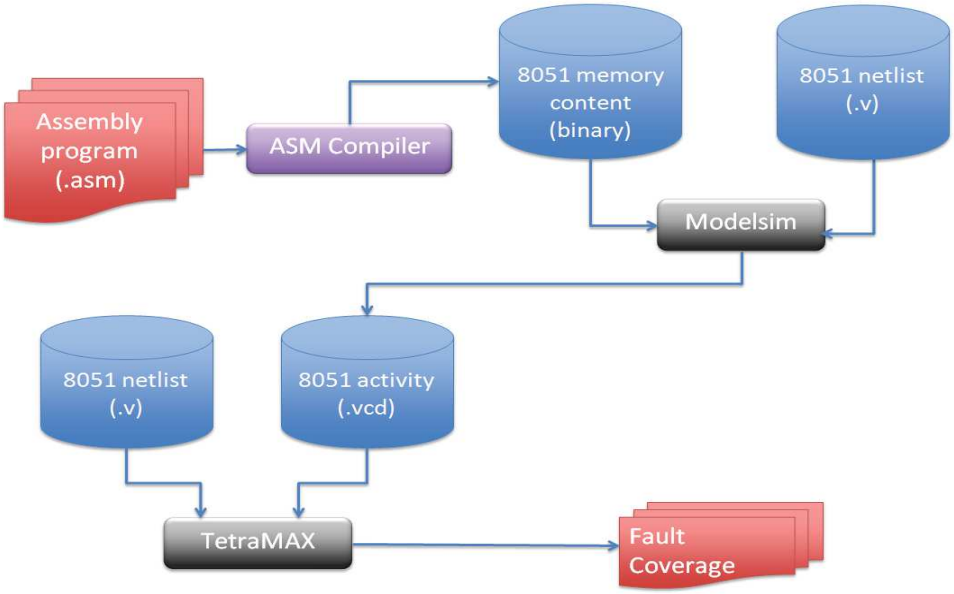


Figure 2.4: Structural coverage flow

2.3 Experimental Results

In this section, we present a set of experiments, which provides an evaluation of functional test programs according to the metrics discussed previously: the code, functional and structural coverages.

2.3.1 Experimental Setup

The population of functional test programs has been generated by using the methodology presented in [17]. The target circuit is the mc8051 synthetized using a 65nm technology. A total number of 117 functional test programs has been generated. Figure 2.5 depicts the generated programs on the X-axis and the peak power on the Y-axis. By construction, the peak power increases at each new test program, leading to have a very effective test set for maximizing the power consumption of the circuit.

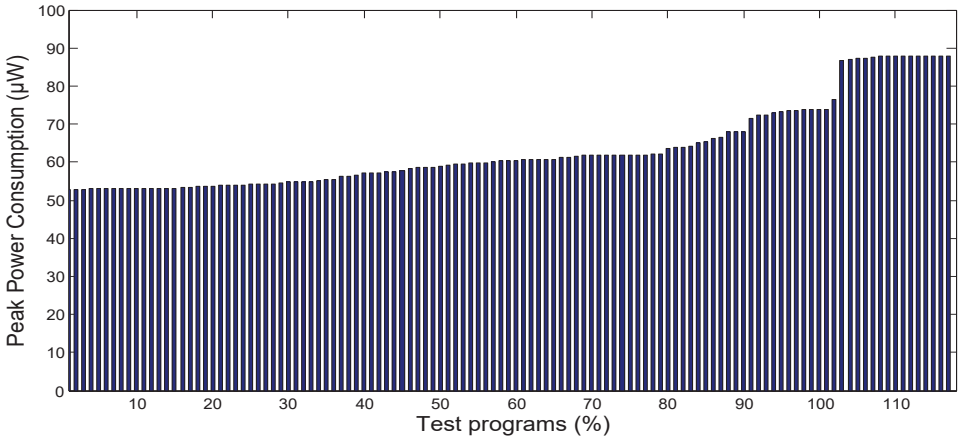


Figure 2.5: Functional program generation

Since all experiments will focus on evaluating these test programs, we keep the same order as the one shown in Figure 2.5. As we mentioned before, the goal of this contribution consists in evaluating functional test programs considering other metrics, nevertheless satisfying the power consumption one. In this way, we can detect prospective correlation between these metrics.

2.3.2 Functional Coverage Evaluation Metrics

The functional coverage metrics are exploited by the CDV. It is usually used in order to improve prototype quality by identifying untested areas of the design and locating potential overlaps (i.e., redundancies). It measures how well these test programs exercise both structural and functional codes of the design. As we mentioned in the section 2.2, functional coverage is measured by using the behavioral description of the mc8051 microprocessor.

In this section, we present the most common functional code coverage to evaluate our test programs, which are Statement, Branch, Condition, Expression and Toggle coverage, shown in Figures 2.6, 2.7, 2.8, 2.9 and 2.10, respectively.

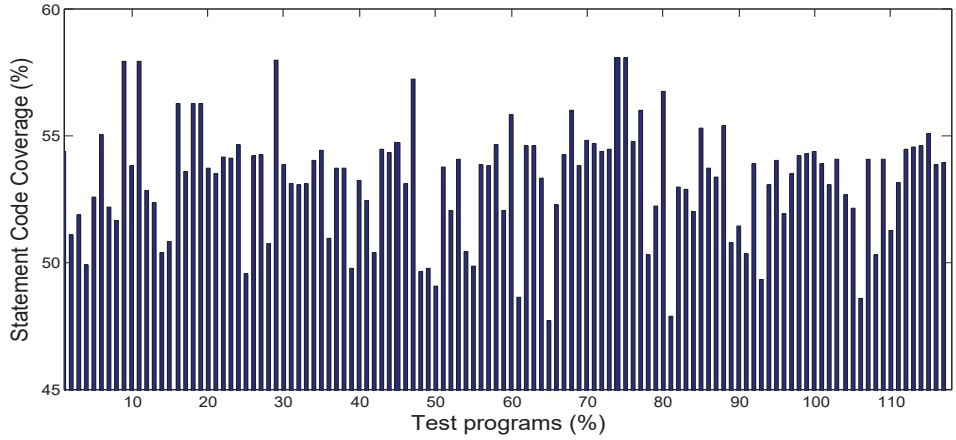


Figure 2.6: Statement code coverage evaluation

We started the code coverage evaluation metric by the most basic criterion, which is the Statement coverage, measuring whether or not a given statement in the RTL code of the design was executed. The vertical axis in Figure 2.6 represents the average statement code coverage of the whole behavioral description with its different modules. The statement coverage, exercised by this test programs set, varies between 47% and 59%, but no correlation with the switching activity has been observed with this distribution.

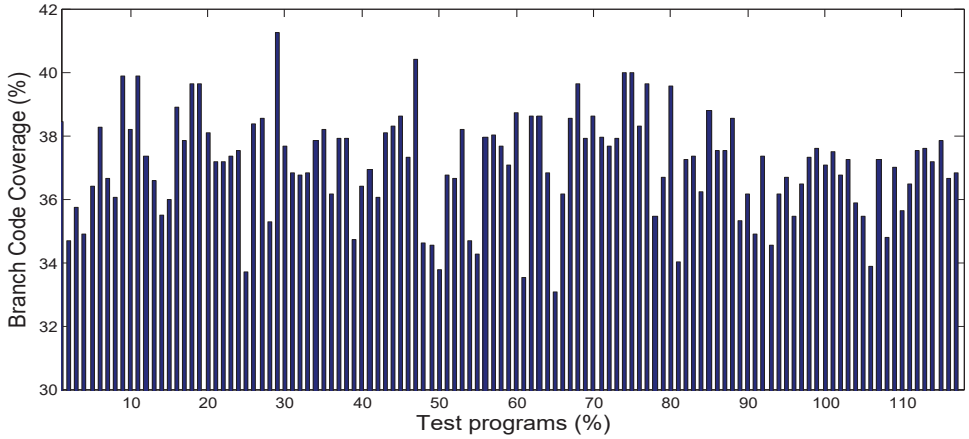


Figure 2.7: Branch code coverage evaluation

Let us move to the branch coverage, which is more granular. Figure 2.7 shows an average branch coverage variation estimated between 33% and 41%. As for the statement code coverage, it does not represent correlation with the activity induced by test programs. These results may depend on the targeted module during the switching activity extraction process.

We also consider the verification of the design functionality by evaluating our test programs in terms of Condition, Expression and Toggle coverage in Figures 2.8, 2.9 and 2.10. These criteria would detect the missing evaluations in the design. We recall that the X-axis in these figures corresponds to the evaluated functional test programs in an ascending order of their switching activities.

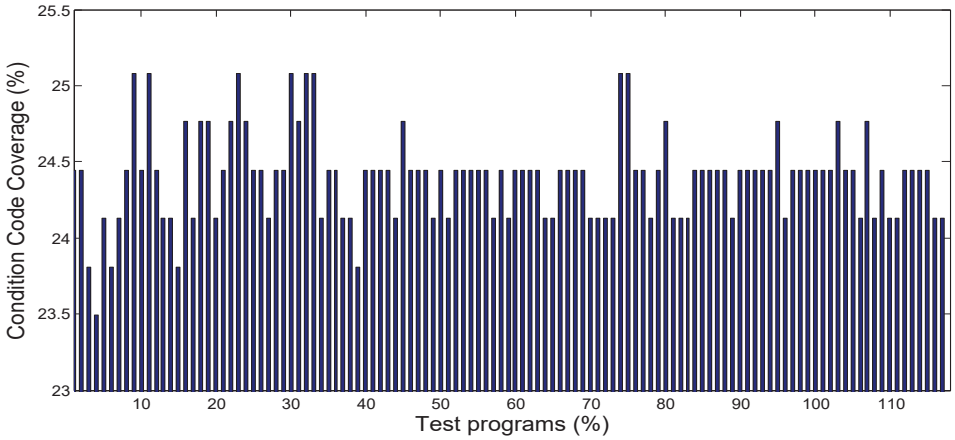


Figure 2.8: Condition code coverage evaluation

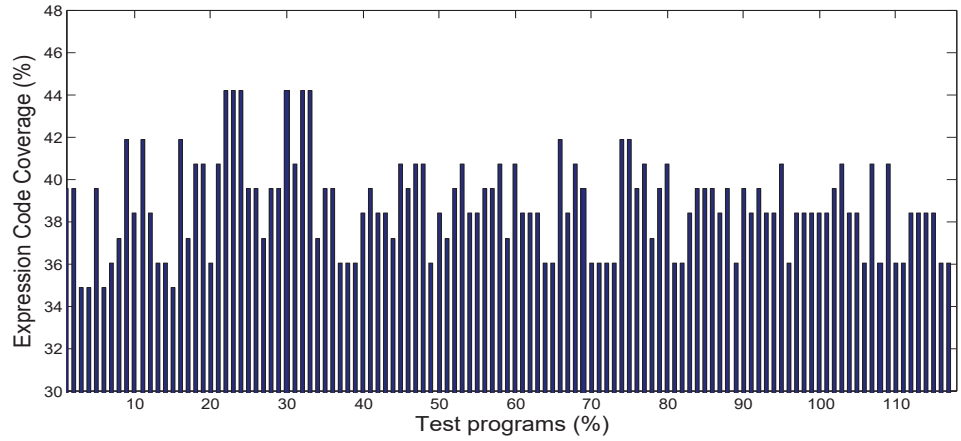


Figure 2.9: Expression code coverage evaluation

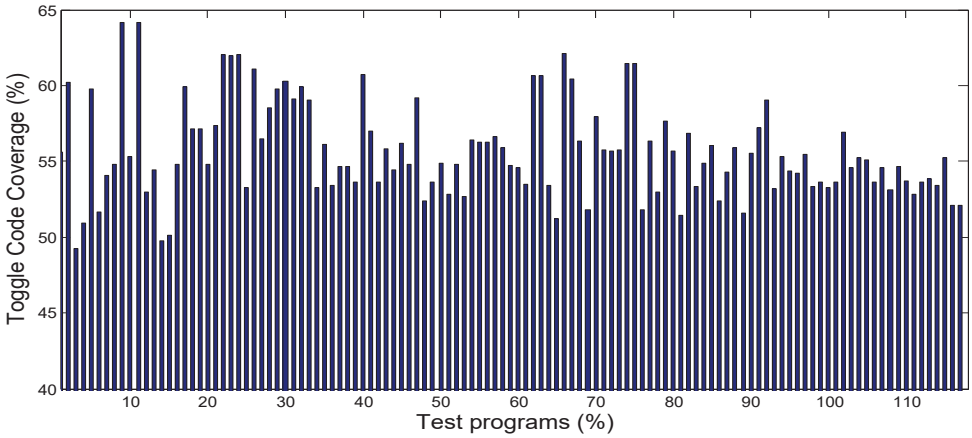


Figure 2.10: Toggle code coverage evaluation

Similarly to the previous cases, we do not observe any correlations between the induced switching activity of the generated programs and code coverage. These results are unexpected. However, a possible explanation could be the following: The programs generation process [17] exploits the gate-level netlist of the microprocessor. It means that the switching activity is maximized at gate-level and this does not necessary lead to a high activity at RT-level. Moreover, as cited in [23], statements in the Hardware Description Language (HDL) description may correspond to hundreds of gates and wires in the final design, which also could explain the obtained results.

2.3.3 Structural Coverage Evaluation Metrics

The last part of this comprehensive evaluation focus on the structural coverage of the considered test programs. As mentioned in the introduction, this study targets stuck-at and transition fault models. Please note that in order to identify any potential correlation between the switching activity and the structural fault coverage metrics, the functional test programs in the subsequent figures of this section are sorted in ascending order of their switching activity

like in Figure 2.5.

We started this evaluation, by estimating the stuck-at fault coverage of these test programs, using TetraMAX as a fault simulator. Definitively, we do not expect a higher coverage for the simple reason that these programs do not target structural coverage. In this experiment, we aim at measuring the impact of the switching activity in detecting stuck-at faults.

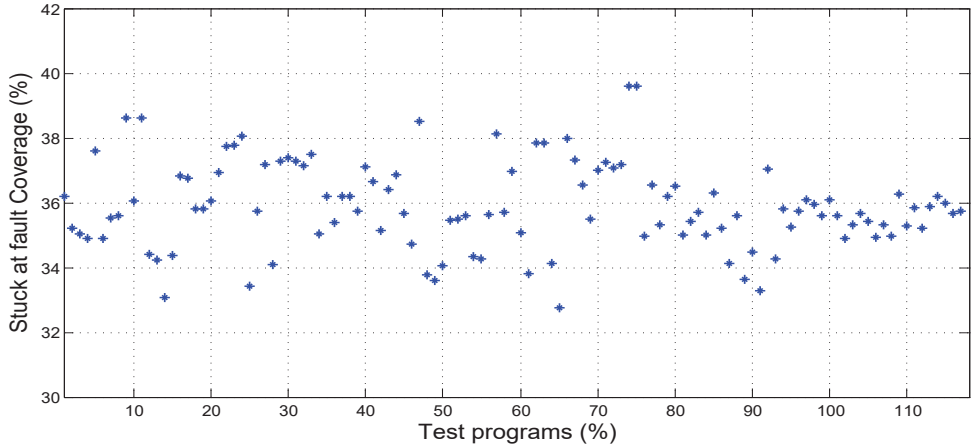


Figure 2.11: Stuck-at fault coverage

A first impression given by Figure 2.11, does not really invalidate our hypothesis since stuck-at faults are not mandatorily sensitized when increasing the switching activity of the circuit. Stuck-at faults coverage achieved by functional test programs varies between 33% and 40% without any special monotony which confirms our expectation.

Refined analysis could be more expressive if we focus on the observability of stuck-at faults, which means the capacity of test programs to sensitize stuck-at faults. In this way, we separate sensitization issues from propagation ones and try an optimistic evaluation.

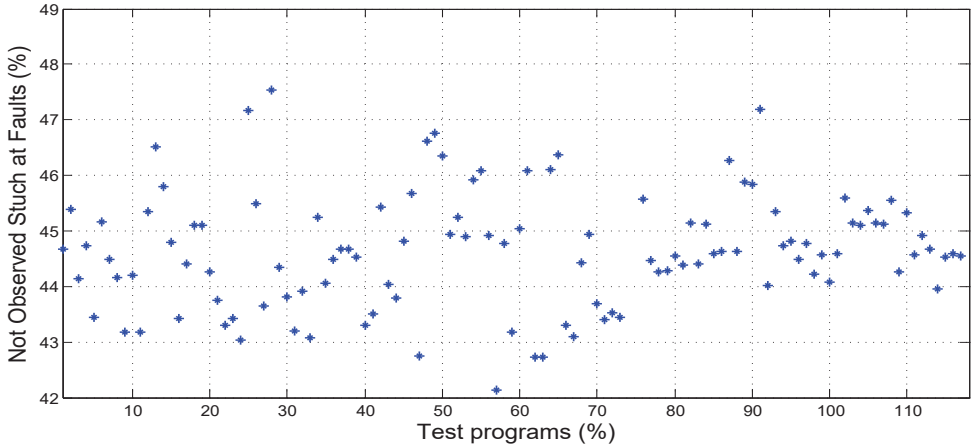


Figure 2.12: Stuck-at fault observability

As we can observe in Figure 2.12, we get a slight variation between 42% and 47.5% of, sensitized but not observed, stuck-at faults. The slope of the trend line is close to zero, which

invalidates any possible correlation with activity induced by the evaluated test programs. However the interesting thing is that a big amount of stuck-at faults is sensitized but not observed.

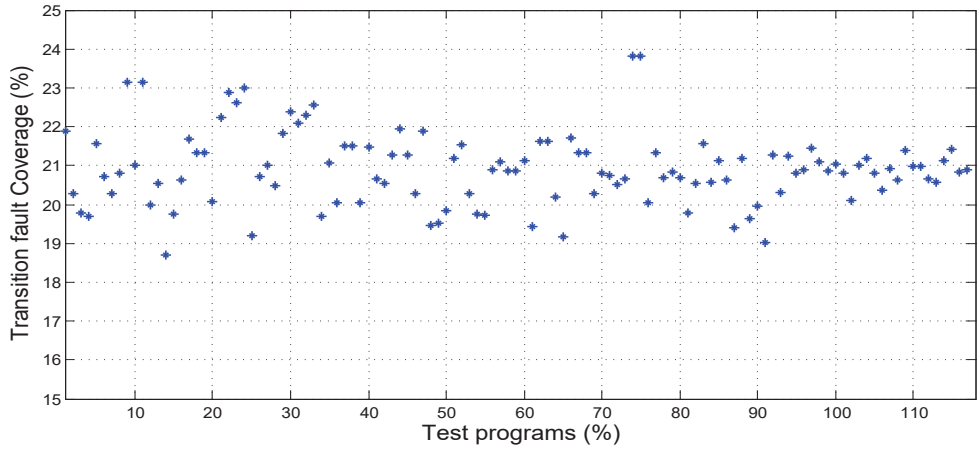


Figure 2.13: Transition fault coverage

Therefore, if we proceed similarly to evaluate the transition fault coverage, achieved by the same functional test programs set, we obtain as depicted in Figure 2.13 a variation between 18.7% and 23.9% without any special monotony. However we measure a considerable percentage of sensitized but not observed faults with the same tests set. This percentage can reach 65.1%, as depicted in Figure 2.14. Even if these results do not allow us to confirm a good correlation between structural coverage and circuit activity, we can anyway make one consideration. A large number of faults (i.e., stuck-at and transition) are sensitized but not observed. Thus, further investigation has to focus on how to improve the observability to increase fault coverage.

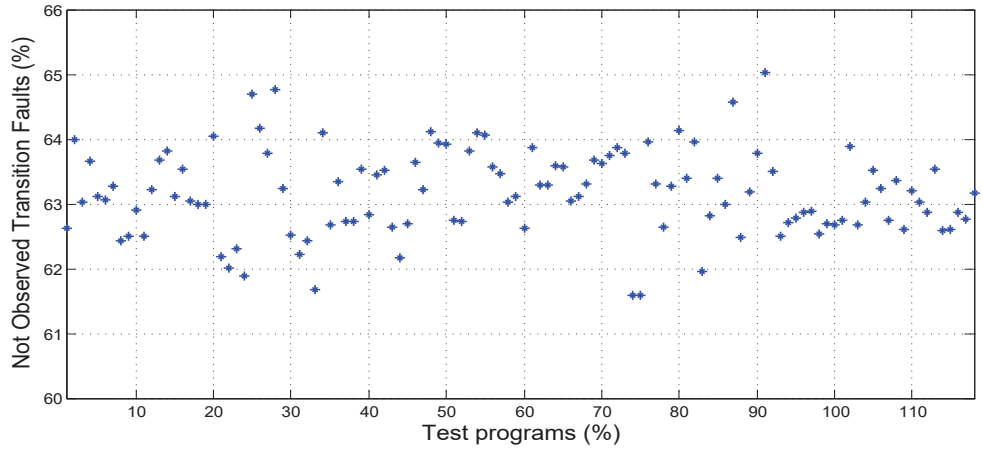


Figure 2.14: Transition fault observability

2.4 Summary

In this chapter, we evaluated a set of test programs able to maximize the peak power consumption, thus characterized by higher switching activity, with respect to the code and structural coverage metrics. During this study, we analyzed different sub-metrics to examine a potential correlation between the induced switching activity and the code or/and the structural coverages. The main goal is to propose a comprehensive test solution which deals with both verification and testing at the same time. First results, which are not expected, reject any correlation with the code coverage metric. However, we observed that many faults are sensitized but not observed by our functional programs. Only comparing the obtained fault coverage with the induced switching activity does not show any direct correlation in the observed trends. This necessitates further investigation into the occurrence of non-observable faults to have a more meaningful evaluation.

Before moving to any observability enhancement, which will be discussed in chapter 4, let us consider a more common situation where no information about the internal structure of the design are available. In such cases, it gets difficult to verify the effectiveness of the functional programs for in-field test. Next chapter deals with the factors required for in field test. Moreover, we also discuss and propose techniques for test program compaction for a better test coverage evaluation with lower test length and test time.

Chapter 3

An Effective Approach for Functional Test Programs Compaction

Contents

2.1 Introduction 5
2.2 Evaluation Framework 8
2.3 Experimental Results 11
2.3.1 Experimental Setup 11
2.3.2 Functional Coverage Evaluation Metrics 12
2.3.3 Structural Coverage Evaluation Metrics 14
2.4 Summary 17

IN this chapter, we first investigate on increasing the fault coverage by merging together the functional test programs and then we compacted them while conserving the obtained fault coverage. Section 3.1 introduces the context of dealing with SBST strategy and their related constraints which lead us to focus on test program compaction techniques. Next, Section 3.2 introduces the targeted microprocessors and we detail herein the test program generation process for each one by using the μGP tool [18]. Then, Section 3.3 presents the fault coverage analysis by giving the main idea of the proposed compaction technique that is further detailed in Section 3.4. Section 3.5 analyzes the experimental results. Finally, Section 3.6 concludes this chapter.

3.1 Introduction

With the incredible fast advances in very large scale integrated circuit technology, processors design become more and more complex. This complexity combined to the limited accessibility of the microprocessors internal logic, makes the testing task harder and then

classical approach of deriving tests based on the gate level description of microprocessors inefficient. Many reasons could justify that. The size overhead of the design coupled with a lack of DFT techniques in some microprocessors, make the use of gate level in sequential ATPG impractical. Also some physical failures in integrated circuits result in logic behavior which can not be modeled by stuck-at faults. Moreover, the test generation using a sequential ATPG is time-consuming process. On the other hand, at-speed system testing and embedded cores in SOC, require faster and autonomous test methods which are resumed in the functional test properties. Functional test allows the execution of test programs at the same frequency which means testing the system at-speed. In addition, this kind of test is easier to implement and test programs, through low frequency interfaces, are rapidly accessible by the processors once uploaded in the memory. Last but not least, with functional test we gain a high flexibility (i.e., it could be adjusted for different targets) and a comprehensive test by means the functional test could cover the whole system including each single module and their interconnections. For these reasons, functional test is widely adopted and could be a complementary or alternate solution for the classical approaches, depending on whether it is adopted for end-of-manufacturing or in-field test.

In functional test, only the functional input signals of a circuit (or module) are stimulated and only functional output signals are observed. Therefore, functional test guarantees that the circuit is tested under normal conditions thus avoiding any over- as well as under-test [15].

For processor-based systems, the most efficient strategy for applying a functional test is called Software-Based Self-Test (SBST) [25]. This strategy is based on the execution of a set of functional test programs on the target processor. Although SBST solutions are usable by the manufacturer for post-production test and by the customer for incoming inspection and on-line test, they are particularly suited for the latter case when processor-based systems are considered (i.e., since it may be the only available option). For example, a dedicated functional test program (or a set of test programs) can be periodically executed during the mission time.

In this context, it is clear that the test length (i.e., how many programs) as well as the test time is a very important issue for this type of test (i.e., the size of the test set is large in the case of functional test generation compared to structural). Here, we distinguish the test application time from the test length since conversely to the test pattern application, there is no linear relationship between them and the reason could be simply explained if we look at the structure (i.e., content) of test programs. They include flow-control instructions (i.e, including loops, conditional execution, subroutines, and so on). In addition to that, SBST test solutions should be minimally intrusive in the system's operation [26]. Indeed, functional test programs have to be as small as possible while achieving the highest fault coverage.

Some approaches have been proposed so far in the literature to deal with both the memory occupation and the test program length. Although all these approaches share the same title, test compaction process, two main classes of techniques can be identified: **dynamic compaction** and **static compaction**. The first one acts during test generation. It can usually achieve good results but at the cost of a high generation time. On the other hand, static compaction is applied after the test generation process. Even if it does not impact the generation time, it can lead to less compaction compared to dynamic compaction since it does

not allow any pattern regeneration.

In [27] a dynamic compaction method is proposed. It extracts from a functional test program an independent fragment (called a spore) whose iteration allows achieving a given fault coverage. The authors describe a method to identify the best sequence of spore activations that can achieve the same initial fault coverage, by resorting to an evolutionary approach. The method is effective, but could only be applied under strict constraints as, for example, the test of an arithmetic unit. In [28] authors propose an algorithm able to automatically compact an existing functional test program. Based on instruction removal and restoration solutions, they showed great compaction capabilities keeping in mind the computational costs. In [29] a new static compaction procedure was proposed to better fit in-field constraints for cores in embedded systems. By using an evolutionary-based strategy, authors were able to discriminate which are the instructions in a given test program are not contributing on the testing goals.This procedure was applied to a set of test programs (i.e., generated using different approaches) targeting a couple of modules inside the minimips pipelined processor core.

The main goal of this chapter is to investigate the static test compaction of a given set of functional test programs. Which means that test compaction does not interfere with the test generation process, and compacts the test program after it is generated. The input is a functional test set composed of many functional test programs. The investigation aims at understanding and determining how to select the best functional test program candidates to obtain the smallest set having the best fault coverage. Conversely to [28] and [29], this approach is applied at test program level and not instruction level. By means, the entire functional test program could be removed or restored based on its contribution in the cumulative test coverage in a specific order. In other word, we consider the context in which we do not have any specific information about the test program itself, which is usually the case in practice, and we aim at compacting it while preserving the initial fault coverage with respect to a given fault model.

Moreover, we assume in this work that test programs are totally independent from each other in term of fault coverage. Hence, for each test program, we could compute the set of detected faults with a single fault simulation. Results carried out on two different microprocessors show that a 49% reduction in test length and a 28.7% reduction in test application time can be achieved.

3.2 Context and Background

In this work, to evaluate the effectiveness of the functional test programs compaction, we used two different processors having different architectures. The first processor, already described in Chapter 2, is the **Intel mc8051** which represents a classical Harvard architecture, non-pipelined CISC architecture, with 8-bit Arithmetic Logic Unit (ALU) and 8-bit registers [30]. The second processor is the **minimips**, 5-stages-pipelined Reduced Instruction Set Computing (RISC) processor, with 32-bit ALU and 32-bit registers [31]. Both processors have been synthesized with the same industrial 65nm technology. Table 3.1 gives the main characteristics of the synthesized processors in terms of number of Gates, Flip-Flops, Primary Inputs and Primary Outputs.

Table 3.1: Mc8051 & minimips gate-level characteristics

Cores	#Gates	#FFs	#Scan-Chains	#PIs	#POs	#TFs
Mc8051	4307	576	1	65	94	30706
Minimips	8612	1785	1	38	67	79690

We started from two different functional test sets targeting two different microprocessors. Both tests have been generated using the μGP tool [32]. This tool, based on a genetic approach, is able to generate a population of functional test programs optimizing one or more constraints.

The first set of functional test programs, as presented in Chapter 2 has been generated in order to maximize the power consumption of the mc8051. Till now, we did not prove yet that functional programs maximizing power consumption can achieve important fault coverage. We just marked a high amount of not observed faults when fault simulating these functional test programs (see chapter 2). We will explore later how to deal with observability issue and then we may validate this assumption. On the other hand, the second test set has been generated to maximize the stuck-at fault coverage on minimips. Using the same μGP tool but constrained to maximize the stuck-at fault coverage, as already explained in chapter 2, an efficient test set of 110 functional test programs has been generated showing high stuck-at fault coverage figures.

3.3 Fault Coverage Analysis

As already described, the two sets of functional test programs have been generated by considering different targets (i.e., maximizing the power consumption and maximizing the stuck-at fault coverage). Therefore, the first part of our analysis is about the fault coverage achieved by those test sets. The considered fault models are the stuck-at and the transition fault models. This kind of analysis has been already presented in detail in chapter 2.

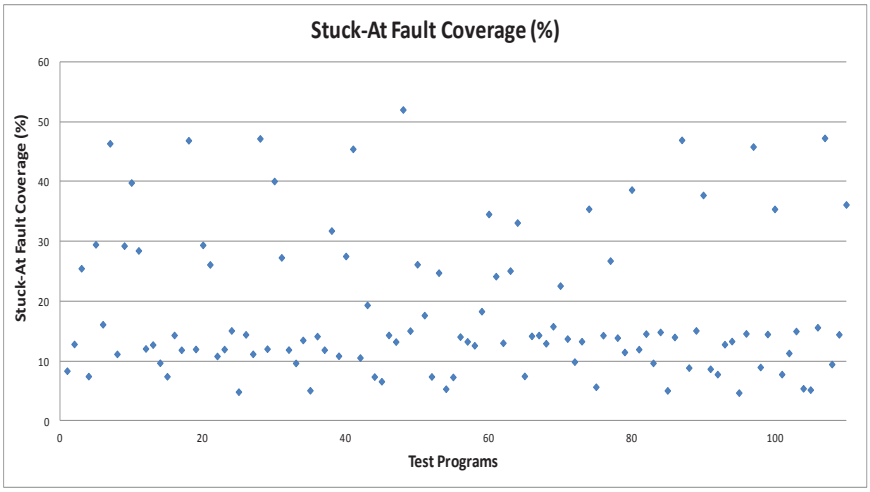


Figure 3.1: Stuck-at fault coverage achieved by functional test programs generated for the minimips

Figure 3.1 plots the stuck-at fault coverage distribution achieved by the functional test programs generated for the minimips. All fault simulations have been carried out by using

a commercial tool [24]. Those functional test programs correspond to the last generation carried out by the μGP tool. As can be seen, the fault coverage varies significantly depending on the programs. Actually, the minimum fault coverage is 4.70% while the maximum is 52%. Table 3.2 reports the same analysis for both processors. For each one, we give the maximum and minimum stuck-at as well as transition fault coverage achieved by the functional test programs. Also, we do not forget to give more details about the minimum and maximum memory requirements and test time in columns 5 and 6. The last column reports the number of functional test programs generated for each processor.

Table 3.2: Functional test programs analysis

	Metrics	SAF (%)	TF (%)	Size (Bytes)	Duration (Clock Cycles)	Prog
Mc8051	Max	43.45	25.07	618	$1.158 * 10^3$	117
	Min	35.67	19.61	346	$1.070 * 10^3$	
Minimips	Max	52.00	43.49	1540	$19.61 * 10^3$	110
	Min	4.70	1.02	32	21.5	

As we can see for the minimips, and conversely to the mc8051, from one test program to another, we can have a huge difference in terms of memory requirement (i.e., from 32 bytes to 1540 bytes) and test application time (i.e., from 21.5 to 19615 clock cycles). This could be explained by the fact that these test programs were generated to enhance stuck-at fault coverage which does not require obligatory a high number of clock cycles. In fact there is no linear relationship between the stuck-at fault coverage and the required clock cycles among the generated test programs. We will investigate later the impact of this in the proposed test compaction methodology. The next analysis we made was about the redundant tested faults (i.e., faults detected by more than one functional test program).

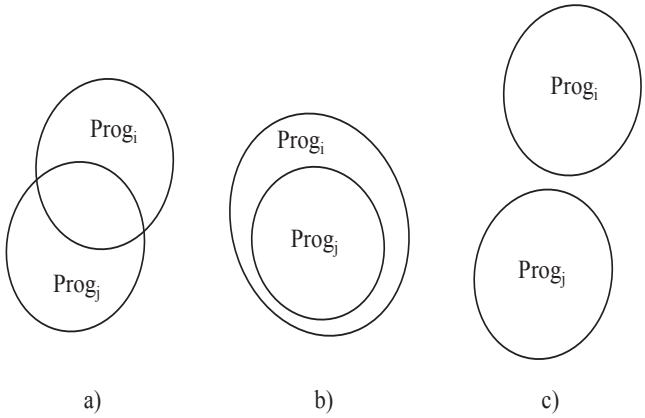


Figure 3.2: Test redundancy

Figure 3.2 shows the three possible cases of equivalent faults when considering two functional test programs i and j. For each program, we plot the set of detected faults defined as $Prog_i$ and $Prog_j$. In the first case shown in 3.2.a), some faults are covered by both $Prog_i$ and $Prog_j$ programs. In this case, the same faults are determined by the intersection between detected fault sets. In the second case shown in 3.2.b), all faults covered by $Prog_j$ are also covered by $Prog_i$. In this case, $Prog_i$ is the superset of $Prog_j$. Finally, in the last case shown in 3.2.c), there is no relation between the two sets. The two programs detect different faults.

The described cases can be formalized as follows:

a) $Prog_i \cap Prog_j \neq \oslash$

b) $Prog_i \cap Prog_j = Prog_j$

c) $Prog_i \cap Prog_j = \oslash$

In the next subsection, we describe how the three cases have been considered in order to deeply analyze the achieved fault coverage.

3.3.1 Fault Coverage Merging

The main idea behind Fault Coverage Merging is to understand whether or not it is possible to combine together functional test programs to achieve the highest fault coverage. The final goal is clearly to remove as many test programs as possible without impacting the fault coverage. Clearly, each program has to be fault simulated in order to compute the set of detected faults.

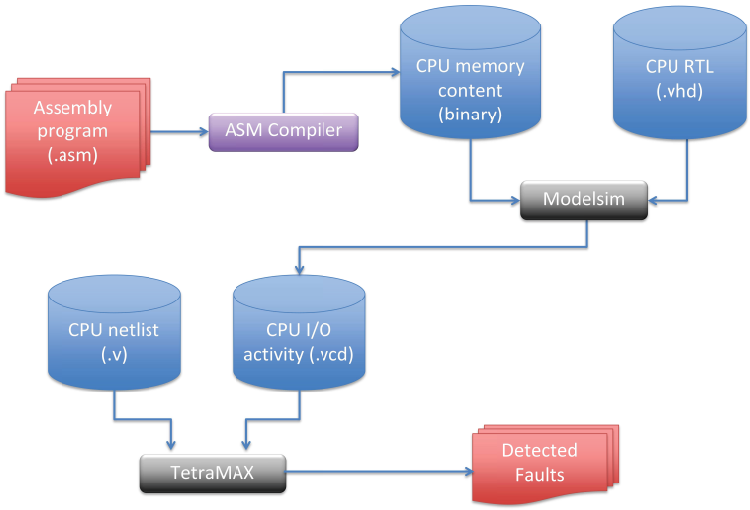


Figure 3.3: Flow for determining the set of detected faults

Figure 3.3 sketches the flow for determining the set of detected faults. Basically, each functional test program is simulated using Modelsim [22] in order to store the Input/Output activity into a VCD file. For this evaluation the RTL description of the microprocessor is used. The VCD file is then used as input for the fault simulator, which is TetraMAX [24]. The fault simulator requires the processor described at gate-level (i.e., the netlist). The result of the fault simulation of a given test program i, is a set of detected faults which is simply defined as follows:

$$DT_i = \{f_1, f_2, \cdots f_n\} \tag{3.1}$$

In the charts shown in Figure 3.4 and Figure 3.5, we plot the merged stuck-at and transition fault coverage respectively, obtained for the minimips. As can be seen in both curves, the fault coverage increases every time a functional test program is fault simulated. From a formal point of view, we see that the relations between the set of detected faults mostly have the same behavior as the cases shown in Figures 3.2.a and 3.2.c. In other words, many functional test programs detect different faults. The limit reached by the merged fault coverage is about 90% for the stuck-at faults and 80% for the transition faults. This is actually very good fault coverage for a functional test.

Another interesting property of the two curves is the fact that they are composed of steps. For example, for the stuck-at faults, we can notice that once the fault coverage reaches 70%, it remains constant for a while and then it directly ramps up to 79%. It means that after reaching 70% of faults, the next ten fault simulated test programs do not contribute to further improve it. Thus, their set of detected faults is completely included in the previous one. This is the case shown in Figure 3.2.b. In other words, those programs are redundant.

Results for the mc8051 have the same characteristics. The limit reached by the merged fault coverage is about 60% for stuck-at faults and 39% for transition faults. The difference in terms of fault coverage between the two processors is easily explained by the fact that the test set generated for the mc8051 maximizes the power consumption and not the fault coverage.

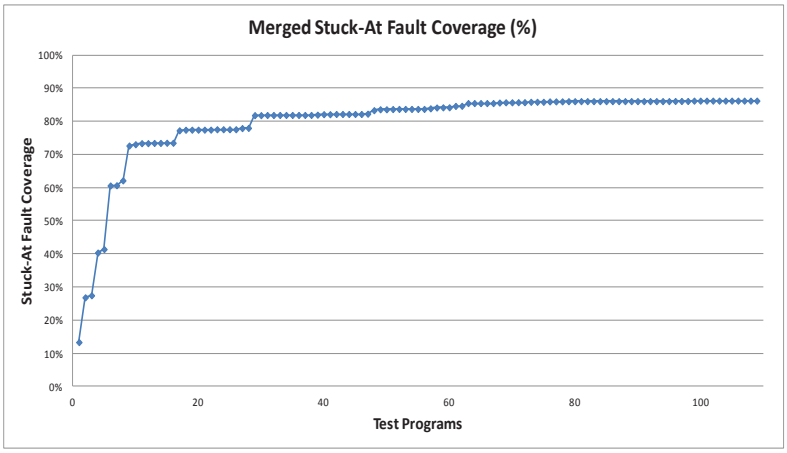


Figure 3.4: Merged stuck-at fault coverage for the minimips

The most important result obtained by this analysis is the demonstration that test programs may easily be redundant. Those programs can be removed since they do not contribute to the overall fault coverage. An interesting property of merging the fault coverage is that the order of test programs execution impacts the number of redundant programs. Let us consider the following example. We have three test programs P1, P2 and P3. Those programs are characterized by the following DTs:

- $DT_1 = \{f_1, f_2, f_3, f_5\}$
- $DT_2 = \{f_2, f_4\}$
- $DT_3 = \{f_4, f_5\}$

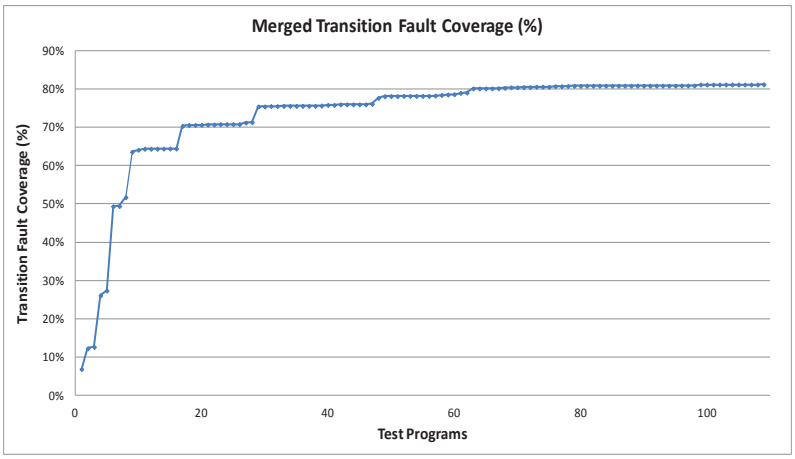


Figure 3.5: Merged transition fault coverage for the minimips

Let us now consider two different program execution orders: P1-P2-P3 and P3-P2-P1. For the first execution order, we start from the execution of P1 that detects the faults in DT1. Then, we execute P2. The overall detected faults is computed by the union between DT2 and DT1:

- $DT_{2-1} = DT_2 \bigcup DT_1 = \{f_1, f_2, f_3, f_4, f_5\}$

P2 is not redundant with respect to P1 because DT1 is not a superset of DT2. We finally execute P3 and we still compute the final set of detected faults.

- $DT_{3-2-1} = DT_3 \bigcup DT_{2-1} = \{f_1, f_2, f_3, f_4, f_5\}$

In this case the number of detected faults does not increase because the set DT2-1 is a superset of DT3. Thus, for this program execution order, P3 becomes redundant and thus, it can be removed. We come back now to the second execution order P3-P2-P1. We apply the same steps as we did for the first one:

- $DT_{3-2} = DT_3 \bigcup DT_2 = \{f_2, f_4, f_5\}$
- $DT_{3-2-1} = DT_1 \bigcup DT_{3-2} = \{f_1, f_2, f_3, f_4, f_5\}$

For the second execution order, we do not identify any redundant program because at each step we increase the set of detected faults. Obviously, the final fault coverage remains the same, irrespective of the order. To conclude, for the first execution order, we can apply only two test programs while for the second order, we must apply all the three programs.

In the next section, we will describe the proposed compaction technique based on the program execution order. The goal is to identify the execution order which is able to reduce as much as possible the functional test length.

3.4 Compaction Technique

In this section, we aim at compacting the given set of functional test programs. Our target is so to keep the achieved fault coverage while removing as much redundant functional test programs as possible. As already stated in the introduction, the proposed approach is a static compaction technique that involves the program execution order to maximize the number of redundant functional test programs. The proposed approach is similar to the restoration-based technique described in [28], but with the difference that our approach is processed at higher abstraction level (i.e., test program level instead of removing subsets of instructions in one test program). Moreover, since it is applied after the test generation, it is completely independent of the used test generation tool.

The preliminary step required by the compaction technique is the fault simulation of each test program in order to determine the *DTs* as defined in equation 5.1.

```
1. Compaction (T)
2.      while (iteration < K)
3.        exe_order = random;
4.        T_tmp = T
5.        cost = remove_redundant (T_tmp, exe_order)
6.        actual_cost = |T|
7.        while (actual_cost > cost)
8.                          exe_order = random;
9.                T_tmp = T
10.               actual_cost = remove_redundant (T_tmp, exe_order)
11.       iteration = iteration +1
```

Figure 3.6: Compaction pseudo-code

Figure 3.6 depicts the pseudo-code of the proposed compaction technique. The input is a given test set *T*. Actually the main core of the compaction algorithm is composed of two loops. We loop for a given number of iterations *K*. For each iteration, we randomly specify an execution order for the test programs and we compute the cost by removing the redundant programs (i.e., procedure remove_redundant). Then we enter in the second loop. Here we continue to generate random execution orders until the cost (i.e., actual_cost) is lower or equal to the old one. The first loop is executed in order to reduce the risk of ending the compaction at a local minimum.

```
1. remove_redundant (T, exe_order)
2.      D = ∅
3.      foreach test_program i using exe_order
4.        if  (DT[i] ∩ D = DT[i])
5.             remove T[i]
6.        else
7.             D = D ∪ DT[i]
```

Figure 3.7: Redundant pseudo-code

Figure 3.7 depicts the pseudo-code of the remove_reduntant procedure. Here inputs are

the test set T and a given execution order. We apply the given execution order in the for-loop and during each iteration we check if the current program i is redundant with respect to the test programs already applied. We note once again that the order is specified by the input exe_order. The program i is redundant if the intersection between the detected fault set of i ($DT[i]$) and the global detected fault set D is equal to the detected fault set of i as described in Section 3.3 (i.e., D is a superset of $DT[i]$). If i is redundant we remove it from T, otherwise we compute the new global set of detected fault D as the union between D and $DT[i]$.

The computational cost of the compaction procedure is, in the worst case, equal to $K * |T|$, where $|T|$ is the number of test programs included in the test set T and K is the maximum number of iterations. Please note that the detected fault sets DTs have been computed before the compaction execution.

3.5 Experimental Results

This section presents experimental results carried out on the two case studies: mc8051 and minimips. For all experiments, we fix the maximum number of iterations K equal to 300. For all reported experiments, the compaction CPU time does not exceed one minute.

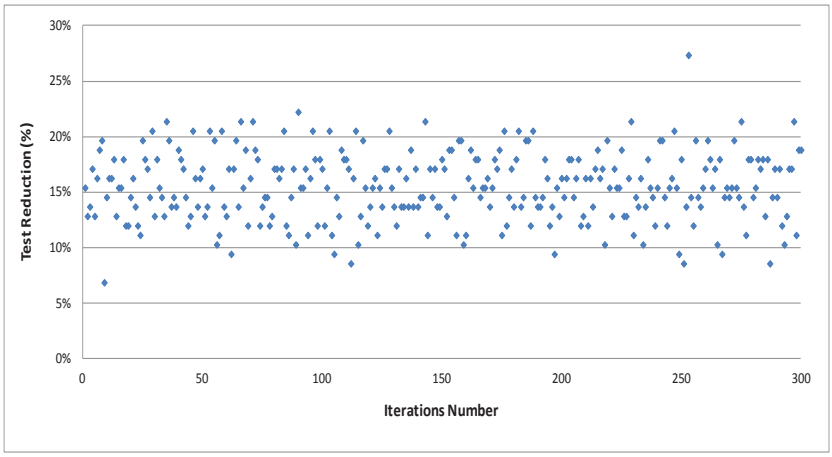


Figure 3.8: % of test reduction (stuck-at fault mc8051)

Figure3.8 plots the chart giving the percentage of test reduction at each iteration of the compaction procedure. On the horizontal axis we plot the iterations (i.e., 300 random execution orders); while on the vertical axis we plot the percentage of test reduction. The test reduction is computed as the number of redundant functional test programs over the total number of programs (i.e., for the mc8051, the number is 117 test programs as reported in Table 3.2). The data are related to the mc8051 and the stuck-at fault coverage. It is interesting to note that the maximum percentage of reduction is 27% while the minimum is only 7%. Please note that irrespective of the execution order, the stuck-at fault coverage is always the same: 60% (see Section 3.3.1).

We performed the same type of experiments for both processors and fault models. Table 3.3 summarizes the obtained results. For each processor and fault model, we report the maximum and minimum test reduction. For all the cases, the maximum reduction is important since we

can remove from 27% up to 49% of test programs without affecting the fault coverage. For all experiments, we fix the number of iterations K equal to 300.

Table 3.3: Test reduction (%)

Processor	Test Reduction (%)	SAF	TF	K
Mc8051	Max	27	29	300
	Min	7	19	
minimips	Max	**49**	38	300
	Min	27	22	

The above results show that by changing the program execution order, it is possible to remove many redundant test programs. Despite this important result, it could be difficult to quantify the real gain. Indeed, we cannot state anything about the test length except regarding the number of test programs, but what about the test time? To answer this question, we have to consider the test time required by each test program.

Let us resort to another example to explain this point. We have four test programs P1, P2, P3 and P4. These programs are characterized by the following DTs:

- $DT_1 = \{f_1, f_2, f_3, f_5\}$
- $DT_2 = \{f_2, f_4\}$
- $DT_3 = \{f_4, f_5\}$
- $DT_3 = \{f_1, f_4\}$

For our example, we also consider the test execution time expressed in terms of clock cycles (i.e., how many clock cycles are required by each program):

- $TestTime(P1) = 50$
- $TestTime(P2) = 5$
- $TestTime(P3) = 15$
- $TestTime(P4) = 10$

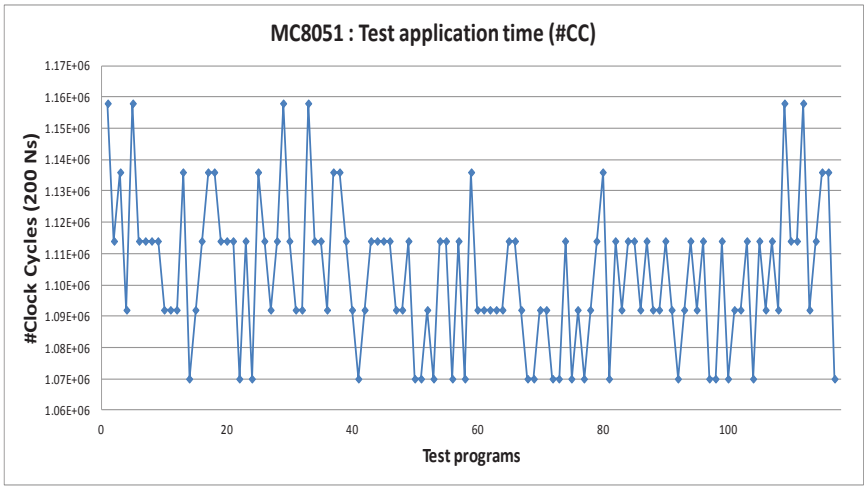
Table 3.4: Test time example

Order	Redundant Programs	Clock Cycles
P1-P2-P3-P4	P3, P4	55
P1-P3-P4-P2	P4, P2	65
P1-P4-P3-P2	P3, P2	60

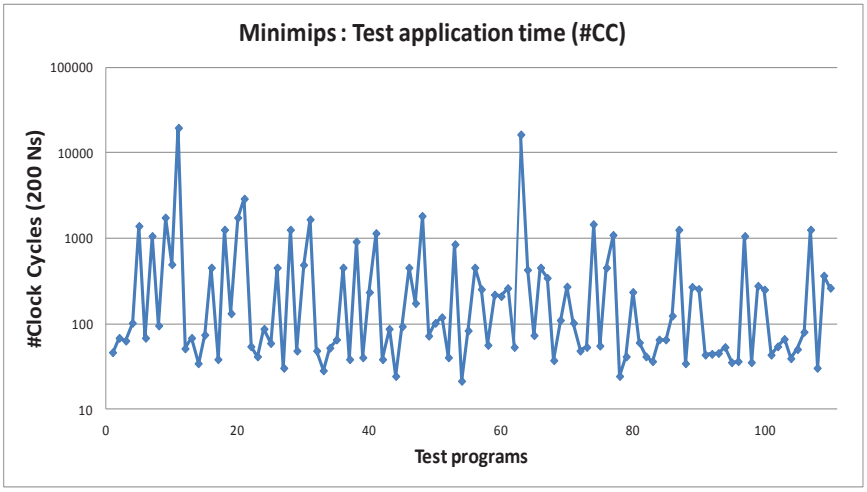
Table 3.4 summarizes our example. We consider three execution orders reported in the first column. For each execution order, the second column reports the redundant programs. Those programs are determined by following the compaction procedure described in Section 3.4. The last column reports the clock cycles required by the merged test sequences. In other words, we compute the Clock Cycles (CC) value as the sum of the clock cycles of each program composing the final test sequence. Thus, for the first execution order, we have 55 clock cycles including the 50 required to execute P1 plus the 5 clock cycles required to execute

P2. Once again we do not consider P3 and P4 since they are identified as redundant programs for this order. In the same way we computed the CC for the other execution orders. As it can be seen, the CC varies depending on the programs composing the final test sequence. However, all the execution orders require the execution of two programs over the initial four. The above example clearly illustrates that by only looking at maximizing the number of redundant programs, we could not reach the shortest test time.

We plot the test time distribution of the generated test programs for both cores to add more details to our analysis. A first observation concerns the test time variation among these test programs, it varies from $1.07 * 10^{(06)}$ up to $1.16 * 10^{(06)}$ clock cycles for the mc8051 case study. However higher variation with completely different distribution is observed for the minimips case study. In fact the test time metric varies from 21.5 up to 19,615 clock cycles which will certainly impact the test compaction results from one order to another.



(a) mc8051

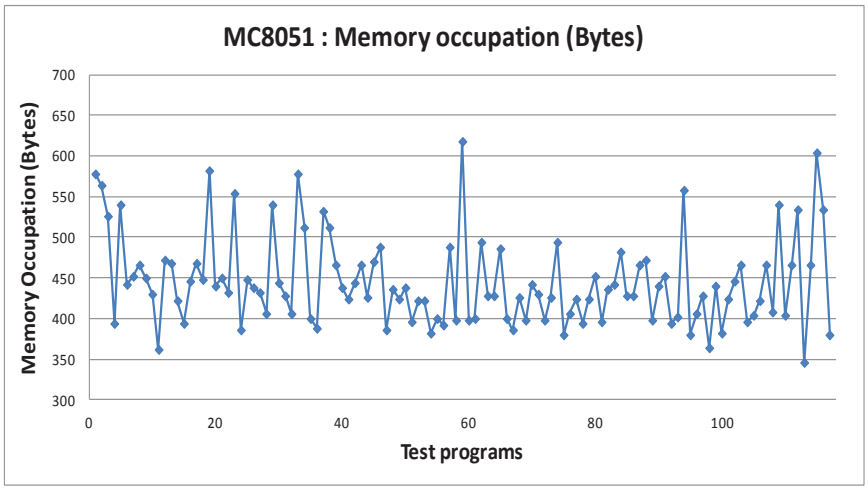


(b) minimips

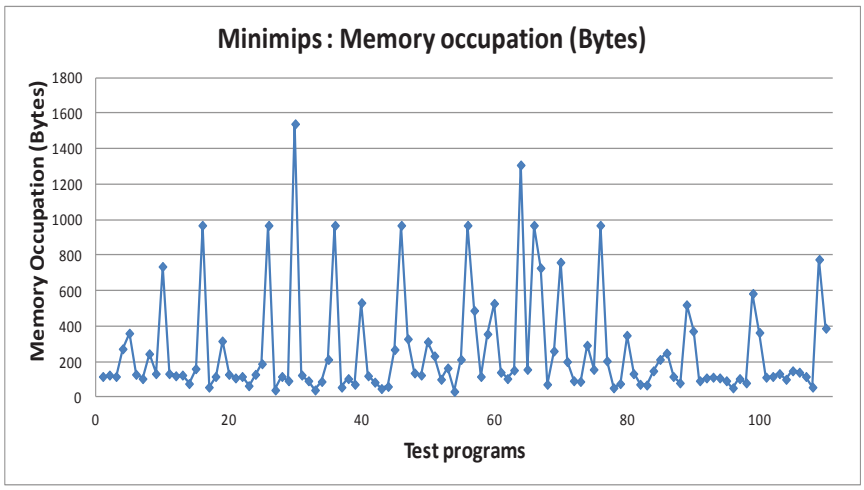
Figure 3.9: Test time distribution

Similarly, Figures 3.10 depicts the distribution of the memory occupation of the different

generated test programs. We report here a variation of 44.01% for the mc8051 (from 346 up to 618 bytes). Again the minimips shows different figure by reaching a memory occupation variation of 97.92% (from 32 up to 1540 bytes). Figure 3.11 reports two curves. The red



(a) mc8051



(b) minimips

Figure 3.10: Memory occupation distribution

one reports the percentage of test reduction achieved by the compaction procedure during each iteration (the values are reported on the left side of the chart). The blue one reports the percentage of test time reduction (the values are reported on the right side of the chart). The test time reduction is computed as the sum of clock cycles of the final test sequence over the total number of clock cycles. The latter is the sum of all the 110 programs clock cycles. The data are related to the minimips and the transition fault coverage. This plot is interesting because it shows that the maximum test reduction (equal to 34%) is obtained at the iteration 186. However, that iteration does not correspond to the lowest number of clock cycles. Indeed, at the iteration 186 we have 15% of test time reduction. The maximum of test time reduction (18%) is obtained at the iteration number 111.

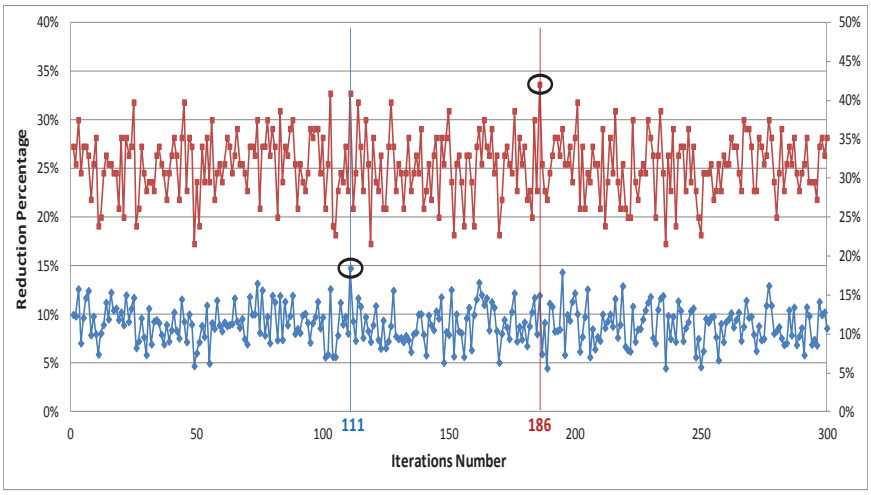


Figure 3.11: Reduction and test Time (Transition faults minimips)

Table 3.5: Test time reduction (%)

Processor	Test Reduction (%)	SAF	TF	Clock Cycles
Mc8051	Max	28	28.7	$129.084 * 10^3$
	Min	8	19	
minimips	Max	20	18	72.290
	Min	7	6	

We performed the same type of experiments for both the processors and the fault models. Table 3.5 summarizes the results obtained. For each processor and fault model, we report the maximum and minimum test time reduction. For all the cases, the maximum reduction is important since we can reduce up to 28.7% the test time without affecting the fault coverage. Please note that 300 iterations are considered for all the experiments.

To summarize, the proposed compaction technique is able to determine the shortest test set in terms of number of test programs and test time. This kind of technique can be very useful when the test is performed during the mission time where a short test time window is available. In this case the main goal is to reach the maximum of the fault coverage by running the shortest number of test programs as possible or by running the shortest test programs. The user can tune the proposed technique to determine the best test suite.

3.6 Summary

In this chapter, we investigated the static test compaction level that can be achieved starting from a given set of functional test programs. The input of the proposed compaction technique is a functional test set composed of many functional test programs. A cumulative fault coverage approach was carried out showing the impact of reordering the application of such programs in both test length and test time reduction efforts. These constraints have been analyzed and discussed at the end of this chapter. The compaction technique is able to determine the best functional test program candidates to obtain the smallest set while preserving the original fault coverage. Targeting the same final fault coverage, we figure out that the lowest memory occupation, once the compression technique is applied at K different application orders (i.e., iterations), did not correspond to the lowest test length. Results

carried out on two different microprocessors show that a 49% reduction in test length and a 28.7% reduction in test time can be achieved. Again we have to mention that the efficiency of this technique (test reduction) depends clearly on the initially generated test set.

Chapter 4

Exploring the Impact of Functional Test Programs Re-Used for Power-Aware At-Speed Testing

Contents

3.1 Introduction **19**

3.2 Context and Background **21**

3.3 Fault Coverage Analysis **22**

3.3.1 Fault Coverage Merging 24

3.4 Compaction Technique **27**

3.5 Experimental Results **28**

3.6 Summary **32**

In chapter 2, during the evaluation of the power-hungry test programs considering the structural coverage metric, one of our questions (i.e., does the switching activity induced by the test program execution correlates in some way with the structural test coverage?) were not answered. The reason is that, many faults are sensitized but not observed when applying the test programs in a sequential way (i.e., in functional mode). The main goal of this chapter so, is to investigate more on how to enhance the observability of these sensitized faults in order to refine our comprehensive evaluation and show better delay fault coverage figures.

In Section 4.2, we introduce the background of this work. Section 4.3 presents the proposed methodologies describing how to apply functional test programs by re-using existing Design for Testability (DfT) architecture, how to improve the global fault coverage based on a deep comparative analysis. Section 4.4 analyzes the collected results and compares them with the structural test patterns generated by using commercial Automatic Test Pattern Generator

(ATPG) and discusses a global test solution. Concluding remarks are given in Section 4.5.

4.1 Introduction

Nowadays, electronic product design faces various issues that are becoming more important with the CMOS technology scaling and the requisite request of both high operation speed and high frequency [14]. Testing for performance, which is required to catch timing or delay faults is therefore mandatory and often implemented through at-speed structural scan testing for digital circuits. Considering at-speed scan testing, two different schemes are used in practice: Launch-off-Shift (LOS) and Launch-off-Capture (LOC). They consist of using a rated (nominal) system clock period between launch and capture for each delay test pattern, while a longer clock period is normally used for scan shifting (load/unload cycles) [33].

For processor-based systems, the most efficient strategy for applying a functional test is called Software-Based Self-Test (SBST) [25]. This strategy is based on the execution of a set of functional test programs on the target processor. The use of functional stimuli for manufacturing testing has already been investigated in the literature. Authors in [34] propose a fast fault grading approach to rank a pool of functional tests depending on their fault coverage. This approach avoids the use of gate-level fault simulation. In [35], authors present a DfT architecture applied at RT-level to increase the fault coverage of functional tests. Basically, they add test-points to improve the testability of the device during the application of functional test. In [36], authors present a methodology able to concatenate a set of functional tests to maximize the fault coverage while minimizing the test length. The latter is also reduced thanks to a static compaction algorithm.

In this chapter, we reuse the functional test programs generated according to the method described in [17] for the mc8051 core, already described in chapter 2. These programs adheres to functional power constraints as the power consumption of a functional test sequence is expected to be close to operation conditions. Then, we investigated the impact of re-using such functional test programs for delay fault testing. In particular we show how these functional tests can be applied to improve the transition delay fault coverage. Conversely to [35], we propose to re-use the DfT circuitry already present in the circuit under test, thus avoiding any further modifications of the device. Basically, we intend to map functional test programs into a structural scan testing scheme (i.e., LOC or LOS). The result will be a test scheme applied to the circuit through existing scan chains. We perform a full analysis of the improved delay fault coverage using the classical at-speed LOC and LOS delay fault testing schemes.

Results carried out on the mc8051 microprocessor case study show that the proposed way to apply functional test programs can detect delay faults escaping the structural test sets. Finally, compared to [36] and the methodology presented in chapter 3, we combine functional test with structural one to prove that it is possible to maximize the delay fault coverage while respecting the initial functional power consumption constraints. These steps lead to an efficient global test solution.

4.2 Background

This section describes the main concepts of the methodology proposed in this chapter. As stated in the introduction, we consider a pool of functional test programs automatically generated as described in [17] targeting the Intel mc8051 non-pipelined CISC processor. All the details about the microprocessors characteristics are given in chapter 3.

A total number of 117 functional test programs have been generated. Figure 4.1 depicts the generated programs on the X-axis and the corresponding peak power (in μW) on the Y-axis. By construction, the peak power increases at each new test program, leading to a very effective functional test programs set for maximizing the power consumption of the microprocessor.

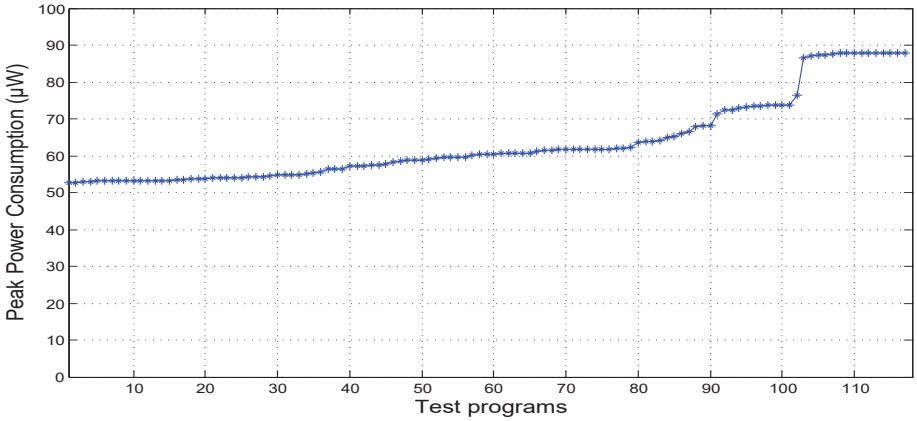


Figure 4.1: Functional test programs peak-power consumption (mc8051 core)

In this chapter we perform a comprehensive analysis of such programs. First of all, we consider the achieved transition fault coverage. Basically, we fault simulated each functional test program (i.e., in a sequential way) to obtain the transition fault coverage. The fault simulations were carried out exploiting the commercial tool TetraMax^{TM} [24].

Figure 4.2 reports the achieved transition fault coverage for the mc8051.

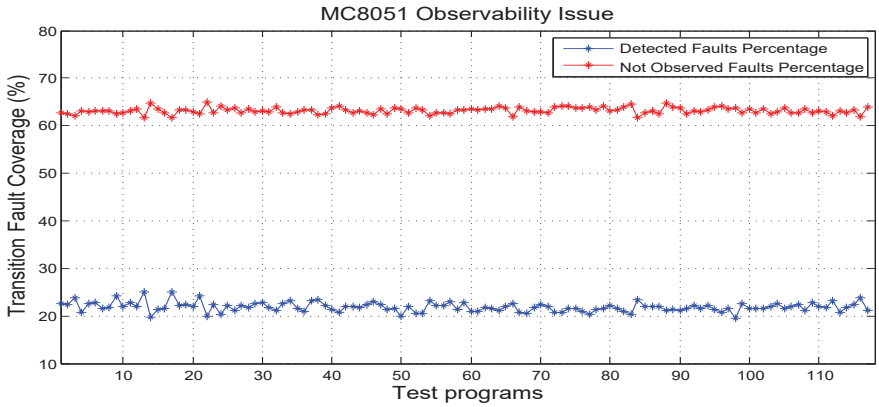


Figure 4.2: Observability issues (%) mc8051

The transition fault coverage varies from about 19% up to 25% as shown in blue color. We

further investigated on the transition fault coverage since our goal is to re-use the functional test programs for at-speed delay testing.

The next evaluation was about the percentage of transition faults that are sensitized but not observed. For the transition fault model, this percentage varies from 61.59% to 65.03%, as depicted in Figure 4.2 in red. It means that a huge number of transition faults are indeed potentially detectable if we could increase the CUT observability. For example, the first functional test program detects 22.5% of transition faults while the 62.7% of the undetected transition faults are not observed.

In the rest of the chapter, we investigate how to improve the observability to increase the transition fault coverage. Next section will describe the proposed methodology explaining how we can apply the functional test programs to achieve this goal.

4.3 Proposed Methodologies

In order to improve the observability, a simple solution is to add test-points in the CUT. This type of solution has been proposed in [35]. In this chapter, the main goal is to avoid any extra area overhead. Therefore, instead of modifying the architecture of the microprocessor we intend to re-use the existing DfT circuitry.

Actually, the microprocessor has been synthesized by inserting one scan chain (see Table. 3.1) that is used for the structural test application. Thus, we would like to exploit any existing scan chain to improve the observability during the application of our functional test programs. To do that, we have to first understand how a functional test program is executed in sequential way and then how it can be executed in test mode (i.e., using the scan chain).

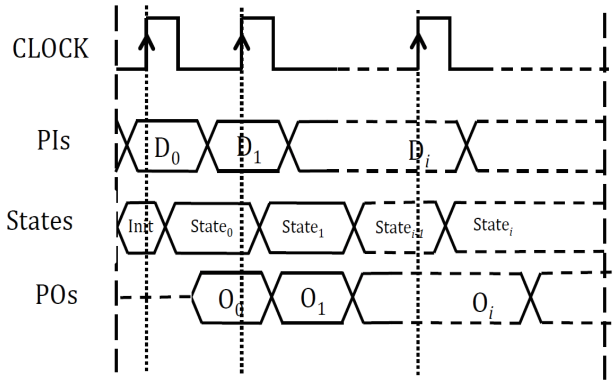


Figure 4.3: Functional test program waveforms

In Figure 4.3, we report the waveforms corresponding to functional test program execution. We have traced the activity of the CLOCK, Primary Inputs (PIs), States and Primary Outputs (POs). This functional test program can be designed using a simple FSM. Being at a known state $State_i$ (i.e., current state), the state machine (the microprocessor) can switch to another state when initiated by the clock edge (i.e., triggering event) depending on the applied Primary-Inputs (D_i) (i.e., condition). This is called a transition. A state is a description of

the status of the microprocessor (i.e., the content of the flip-flops) that is waiting to execute a transition. This transition is a set of actions to be executed when a condition is fulfilled or when an event is received (i.e., the clock edge in this case). The next state ($State_{i+1}$) and output (O_{i+1}) of this FSM is a function of the input (D_i) and the current state ($State_i$).

Now that the application of the functional test programs is clear, we have to understand how to apply them in test mode. Thus, we have to map them to a structural test scheme and first of all we have to determine the best structural test scheme to use. As well known, two test schemes are used for structural transition fault testing during at-speed scan testing: LOS and LOC [33].

The typical waveforms of the clock and Scan Enable (SE) signals for LOS and LOC testing schemes are given in Figure 4.4. Both schemes use a two-vector test $< V1, V2 >$ to detect the targeted transition delay fault. In both schemes, the test vector V1 is shifted into the scan chain(s) at slow speed, while the launch-to-capture cycle (also called "test" cycle and defined as the time interval between the launch and capture edges) is applied at rated speed. In LOS, the SE signal remains at '1' (shift mode) and test vector V2 is obtained by one bit shifting of vector V1. Transitions are launched and propagated in the CUT. Right before the capture cycle, the SE signal is switched from 1 to 0 (functional mode), and then the response to V2 is captured in the scan flip-flops. In LOC, after test vector V1 has been shifted into the scan chain(s), the SE signal has a large time window to be switched from '1' to '0'. This time, the vector V2 is obtained by the functional response of the CUT to vector V1. Transitions are launched and propagated in the CUT, and the response to V2 is captured in scan flip-flops during the capture cycle [33].

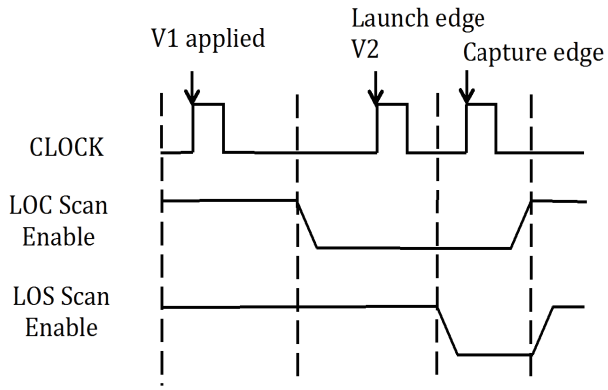


Figure 4.4: LOC & LOS test schemes

In this work, we use the LOC test scheme for mapping the functional test programs. The main reason is because vector V2 is the functional response to the vector V1. In this sense, the test scheme is more similar to a functional stimuli application.

The problem now is mapping the functional test program into a LOC test scheme. From Figure 4.4, it is easy to understand that once we have the knowledge of PIs, FFs and POs values for each clock cycle, we can map them to a LOC test scheme.

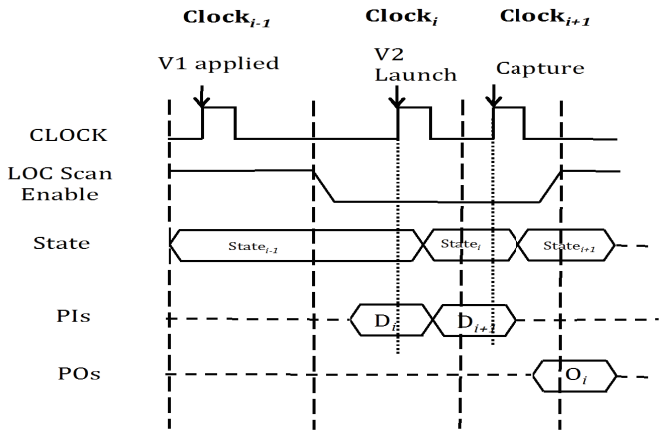


Figure 4.5: Mapping functional test program to LOC test scheme

Figure 4.5 depicts the mapping of a functional test program into a LOC test scheme. By simulation we can identify the state ($State_i$) reached by the circuit at the generic i^{th} clock cycle during the execution of the test program. Let us consider three generic clock cycles taken from the functional test program waveforms:

- $Clock_{i-1}$: it is mapped to the V1 application. Actually we shifted into the scan chain the logic values corresponding to the $State_{i-1}$;
- $Clock_i$: it is mapped to the V2 Launch cycle. Actually, before applying the i^{th} clock cycle, the SE signal switches from '1' to '0', thus the circuit is now in functional mode. We have to apply the expected primary inputs (i.e., D_i) to obtain V2 as the functional response of the circuit to V1. V2 corresponds to the $State_i$;
- $Clock_{i+1}$: it is mapped to the Capture cycle. The circuit is still in functional mode and the response to V2 is captured in the scan flip-flops.

From the above scheme, we can formally define that the functional test program is mapped into a sequence of test patterns. Each test pattern is composed of a couple of test vectors $< V1, V2 >$ where:

- V1 = $\{State_{i-1}, PI_i\}$;
- V2 = $\{State_i, PI_{i+1}\}$.

The output values are $State_{i+1}$, that are shifted out through the scan chain, and the primary outputs corresponding to the values obtained after the $i+1^{th}$ clock cycle (i.e., O_{i+1} in Figure 4.5). The above scheme is repeated for each clock cycle from 1 to $N-1$, where N is the last clock cycle of the applied functional test program.

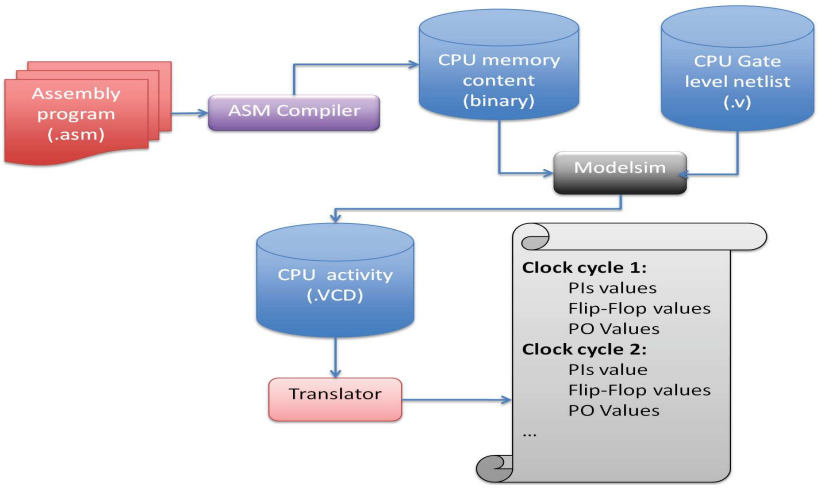


Figure 4.6: Functional programs to test patterns

Figure 4.6 illustrates the flow implemented to translate a functional test program into test patterns to be applied under a LOC test scheme. Each functional test program (i.e., the assembly program in Figure 4.6) is compiled in order to obtain the memory contents to be loaded into the targeted microprocessor (mc8051). This latter with the test program loaded in memory is simulated using ModelSim^{TM} [22] and the related switching information: primary inputs, primary outputs and flip-flops are stored (dumped) in a VCD file (Value Change Dump). Thus, in the resulting VCD file, we have saved the waveforms depicted in Figure 4.3 when the applied functional test program is executed. The last step of the flow actually extracts from the VCD file the information to obtain V1 and V2 for each clock cycle. The test vectors are stored into a STIL format file that will be used during the test application.

4.4 Experimental Results

4.4.1 Fault Coverage Enhancement Results

This subsection presents the results obtained by applying the proposed methodology for enhancing the fault coverage to the set of functional test programs on the mc8051 processor. Thus, we first show the benefits of mapping functional test programs into a LOC-like test scheme (i.e., by comparing the fault coverage before and after applying the mapping methodology for each functional test program). Moreover, we compare the results achieved with our methodology with those obtained with the classical LOC- and LOS-based structural tests.

Let us first describe the conditions and the tools used to generate the structural test sets and to fault simulate the functional test programs. Transition fault test sets (LOC and LOS) are generated using the commercial ATPG TetraMax^{TM} [24]. Note that we used the random filling option, while static and dynamic compactions were not used during test generation. The Peak Power consumption has been evaluated by using Prime_Power^{TM} [37]. Note that

for both LOC and LOS, we simulated the power consumed during the at-speed clock cycle, i.e., the test cycle (see Figure 4.4). In other words, we did not consider the power consumed during scan shifting. Also note that we used the *waveform_interval* option to specify the sampling interval of 0.01 that is used for power waveform. The default value is the timescale from the VCD file. In our case the timescale is 1ns. We performed that in order to be in the same conditions existing during the functional test programs execution, where all the clock cycles are applied at-speed. In this way the structural power consumption is comparable with the one achieved by the functional test programs reported in Figure 4.1. Finally, we exploited TetraMax™ [24] to fault simulate the functional test programs. These latter are applied in two different ways: The first one is referred to as "Sequential" which means that the programs are executed in sequential way, so that we applied the sequential transition fault simulation. The second one is referred to as "Func2LOC" which means that we applied the methodology described in the previous section to map the functional programs into a LOC-based structural test". Therefore, the fault simulation procedure is the same as the one used to simulate a LOC test set.

Table 4.1: Mc8051: LOC and LOS test set characteristics

Scheme	Length	TF (%)	Peak Power (μW)
LOC	1,063	78.18	168.3
LOS	1,232	88.89	175

Table 4.1 summarizes the characteristics of the generated LOC and LOS test sets. The structural test sets (LOC and LOS) will be our reference during the functional test programs evaluation. Both LOC and LOS test sets have been generated by using TetraMax™ [24]. We report the test length expressed in terms of pattern number (Conversely to the Stuck-at patterns, each transition pattern is a couple of vectors $< V1, V2 >$), the percentage of detected transition faults and the estimated peak power (expressed in μW). As expected, the LOS test scheme achieves a higher transition delay fault coverage than the LOC one, while the peak power consumption does not vary so much between one test scheme to the other. However the test length is about 16% higher for LOS than LOC. These results are not surprising, since the literature already reported similar data [38].

Table 4.2: Functional test programs characteristics

	Transition Fault Coverage (%)		Length	Peak Power (μW)
	Sequential	Func2LOC		
Max(TF)	25.07	**43.33**	1,118	61.8
Min(TF)	20.02	**34.47**	1,052	60.8
Max(L)	23.18	40.52	**1,140**	57.4
Min(L)	20.02	34.47	**1,052**	60.8
Max(PP)	21.99	38.48	1,096	**88.0**
Min(PP)	22.79	40.34	1,140	**52.8**

Table 4.2 reports the results obtained from the analysis of the functional test programs. We report the transition fault coverage, the length and the peak power. As described above, the transition fault coverage has been estimated in two ways, corresponding to the sub columns Sequential and Functional-programs mapped into LOC scheme (Func2LOC). Note that once again the length is expressed in terms of patterns (couples of vectors <V1, V2>). As shown in

Section 4.3, we associated each couple of functional clock cycles to a LOC pattern. Therefore, if we have *N* functional clock cycles then the number of patterns will be *2*N*. We resort to this way of expressing the length because it is comparable with the classical LOC and LOS test sequence lengths reported in Table 4.1. Finally, we give the maximum and minimum of each characteristic: Transition Fault coverage (TF), Length (L) and Peak Power (PP). The first two rows of the table give the max and min transition fault coverage considering the Func2LOC technique and its corresponding Peak Power and Length. The third and fourth rows give the max and min test length and their corresponding TF and Peak Power. Finally and similarly, the last two rows give the max and min peak power and their corresponding Transition Fault and Length.

A first comment is that the test length does not vary so much among the functional test programs. The variation is about 7.71% of patterns (from 1,052 up to 1,140). Conversely, the peak power varies of about 41%; this result was actually expected, since these programs have been generated for this reason (see Chapter 2 Section 2.1). The second and most important observation is related to the improvement of the fault coverage. In fact, by applying the proposed methodology (i.e., Func2LOC) we can increase by 16.46% on average the transition fault coverage compared to when the test programs are executed in the classical sequential way.

Deeper investigations into the fault coverage analysis show the benefits of merging different test sets. Another analysis has been carried out about the capability of one test scheme to detect a specific set of faults that is actually undetected by the other test scheme. As reported in [39], LOS-untestable faults may be tested by LOC and vise versa LOC-untestable faults may be tested by LOS as illustrated in Figure 4.7.

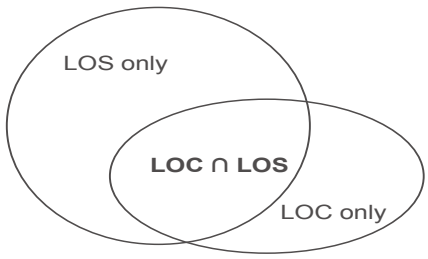


Figure 4.7: LOC and LOS detected faults repartition

Following this analysis, we want to verify if the same phenomena happens between functional test programs and structural test scheme. Thus we consider the two cases: LOC versus Func2LOC and LOS versus Func2LOC.

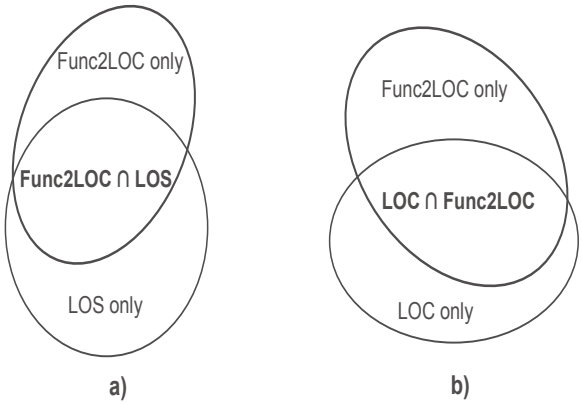


Figure 4.8: a) LOS vs Func2LOC, b) LOC vs Func2LOC

As reported in Figure 4.8, the interesting result is that we can really detect some faults only by applying the functional test programs using the proposed methodology. In other words, some faults can be detected neither by the LOC nor by the LOS but only by the Func2LOC scheme.

Table 4.3: Faults left undetected by LOC and LOS

	TF(%)	**TF tested by Func2LOC only (%)**		
		Max	**Avg**	**Min**
LOC	78.18	7.05	5.76	4.98
LOS	88.89	3.35	2.97	2.58

Table. 4.3 quantifies the percentage of transition faults that are undetected by the structural test schemes but detected by the Func2LOC. As reported in Table 4.3, we got the maximum of 7.05% transition faults that are undetected by the LOC, but detected by the Func2LOC, and 3.35% of transition faults undetected by the LOS scheme but detected by the Func2LOC scheme. Note that we performed the same analysis by applying the functional test programs in purely functional mode (i.e., the sequential fault coverage reported in Table 4.2). Even in this case we got some transition faults only tested in sequential way, even if the percentage for that case is less than 1%. Moreover, the detected transition faults in the sequential mode are completely included in the Func2LOC detected transition fault set. Finally, due to this new Func2LOC test scheme, we were able to further detect 19.31% of transition faults which were not observable when we ran a simple sequential fault simulation. Once again, please note that the Func2LOC test scheme does not introduce any over-testing issues.

4.4.2 A Global Test Solution Results

Based on the observed enhancement, we decided to merge together the LOC and the LOS test schemes and see if some transition faults are still undetected by the structural test while detected by the proposed Func2LOC scheme.

For this purpose, we simply applied the two sets of test patterns to the circuit, without using fault dropping during test generation. So, the test length of LOC+LOS is simply the

sum of the test length of LOC and the test length of LOS that is equal to 2,295 test patterns. The achieved transition fault coverage is 94.16%, that corresponds to the union between the set of faults detected by the LOS scheme and the faults detected by the LOC scheme (see Figure 4.7)

Since our goal is to evaluate the proposed Func2LOC scheme compared to the union of structural test sets, we have arbitrarily applied LOS patterns after LOC patterns (The transition fault coverage does not change if we change the application order and apply LOS patterns first). Even in this case, there are some transition faults that are detected only by the proposed Func2LOC scheme. Based on these results, we propose to create a global test scheme by merging the LOC+LOS and the Func2LOC (see Figure 4.9). As previously, we simply apply the three test sets in sequence, so that the total length is the sum of LOC+LOS+ Func2LOC.

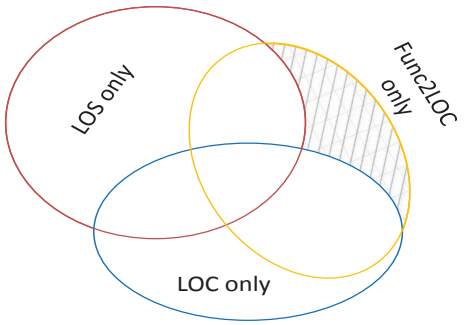


Figure 4.9: A global test solution

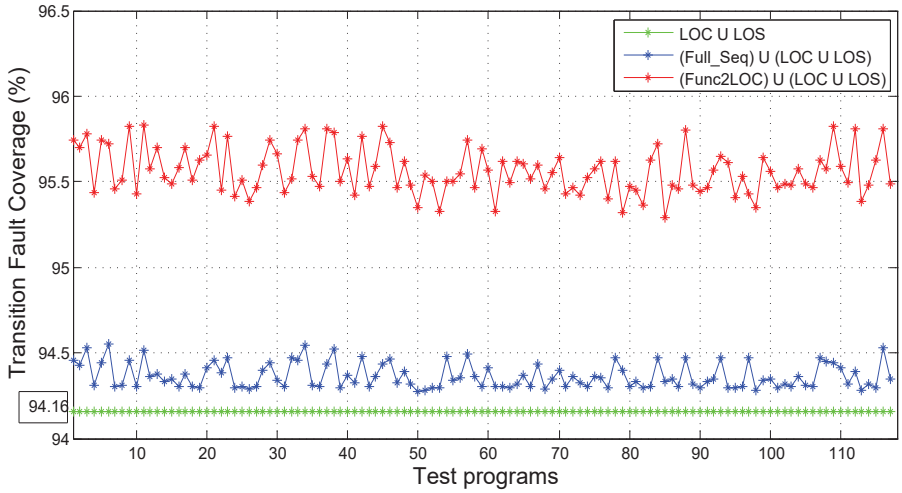


Figure 4.10: Transition fault improvement by applying a global test solutions

Figure 4.10 reports the achieved fault coverage (by considering one functional test program at a time). The green line corresponds to the coverage of the LOC+LOS. The blue graph corresponds to the LOC+LOS merged with the functional test program applied in sequential way. In this case, if we consider the most efficient test program, only 0.39% of faults will be escaped by the structural test. The red line corresponds to LOC+LOS+Func2LOC. In this case, a larger number of transition faults can be detected, corresponding to 95.83% of faults.

The final comment is related to the power consumption of the global test solution. Since we simply apply in sequential order the schemes (i.e., first LOC, then LOS, then Func2LOC) the peak power is dominated by the LOS. However, it is possible to tune it to be close to the functional one by applying the techniques described in [40]. Finally, we would like to remember that the above results depend on the quality of the given pool of functional test programs. The technique presented in [17] describes how to generate meaningful functional test programs that are good candidates for at-speed delay testing.

Having this great result, based on a simple comparative analysis, we tried to investigate more into the Func2LOC patterns and merge the overall Func2LOC test set. At that point, we can consider it as "A Global Test Solution". So the global fault coverage that we can reach is equal to:

$$G_{FC} = FC[LOC \cup \text{LOS} \cup (\cup \text{Func2LOC})] \tag{4.1}$$

Where G_{FC} is the global fault coverage achieved after merging the different test sets, LOC and LOS are the structural test sets generated by the commercial TetraMAX ATPG, and $(\cup Func2LOC)$ refers to the union of the overall Func2LOC test set.

We report in Figure 4.11 the final global test solution obtained for the mc8051 considering finally the overall test sets as shown in Equation 4.1. This histogram shows firstly the fault coverages obtained by using the classical ATPG test schemes (Structural LOC, LOS and the LOC U LOS test patterns) which are represented in blue.The red portion represents the maximum functional program fault coverage enhancement compared to the structural obtained fault coverage. As we can see, we can reach 7.05% as enhanced fault coverage compared to the LOC test scheme by choosing the appropriate functional test program (i.e., most efficient one). Similarly, if we consider only the LOS test scheme as reference, our methodology can detect in advantage 3.35%. Now if we consider both LOC and LOS detected faults, the transition fault coverage could be further improved by 1.67%.

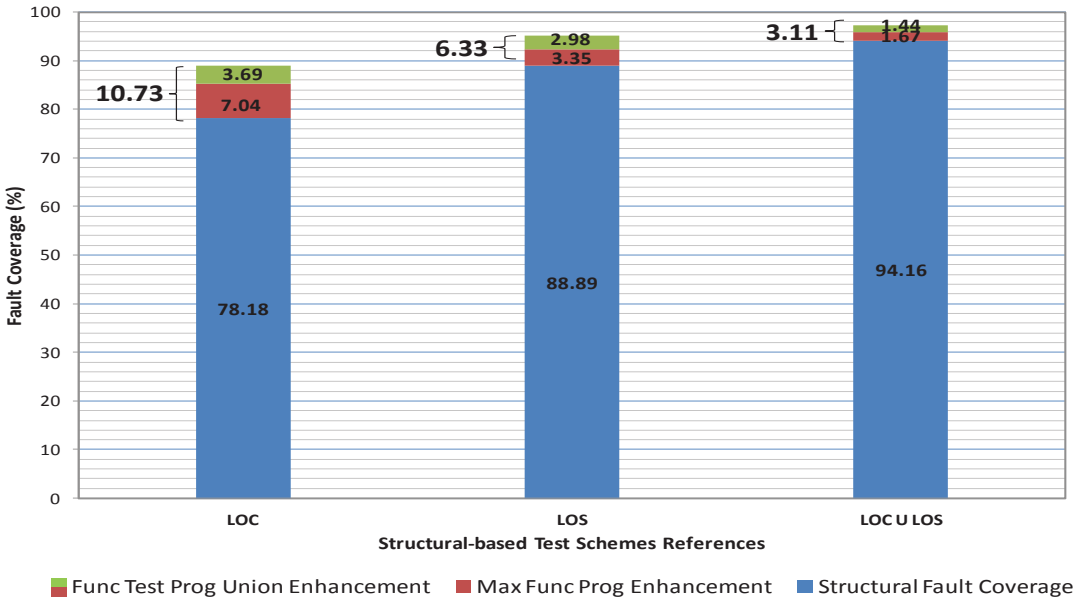


Figure 4.11: A global test solution (mc8051 case study)

Now the most relevant scenario is to consider the fault coverage improvement due to the merge of the overall functional test programs. In fact, if we just consider the maximum functional program fault coverage enhancement, we could escape the benefit of some relevant functional programs which are able to detect some faults labeled as undetected till here (i.e., the first approach). Thus we propose to take a look on the last green portion on these histogram, which shows clearly the benefit of merging the functional test programs. By construction, and if we consider the final global test solution (i.e., the last histogram bar), we can achieve 97.27% for transition faults.

To understand more the improved fault coverage, we tried to give more details on how the ATPG has classified these faults (i.e., both red and green portions in Figures 4.11). If we follow the fault class hierarchy of the ATPG, these faults (escaped by the ATPG) may be included in one of these classes: *Possibly detected* (PT), *Undetected* (UD), *ATPG untestable* (AU) or *Not detected* (ND) faults. By merging 117 functional test programs following the mapping technique (Func2LOC), we figure out that 970 transition faults are detected only by the Functional test programs. Theses faults corresponds in fact to the hatched part in Figure 4.9 and are classified entirely as ATPG Not detected by both LOC and LOS test schemes.

This benefit in term of enhanced fault coverage could be easily traced and understood if we do similar analysis of the functional test programs applied in sequential way. This latter shows that by only merging the functional test programs applied in a sequential way, we are able to detect some ATPG escaped faults. This analysis shows that 279 transition faults (detected only by the functional test set) are entirely classified as *ATPG Not detected* (ND). This allows us to highlight the benefit of merging the test programs apart from the one related to the mapping technique especially when we figure out that the faults represented by hatched portion in Figure 4.9 in case of Full-sequential application (i.e., 279 transition faults) are entirely included in the set of faults detected only by Func2LOC test programs (i.e., 970 transition faults).

Another interesting result we could achieve by considering different order of test programs while applying this global test solution. Starting from the first motivation of using the functional test programs which is to benefit from the actual functional power consumption and avoid any power issues as discussed in the introduction, we proposed to evaluate the functional test programs and firstly see what happens if we merge them together. As shown in Figure 4.12, we collected the cumulative fault coverage by fault simulating the functional test programs applied in two different ways (sequential way as shown in blue and Func2LOC as shown in red). One more time, by considering this approach, we can evaluate globally the benefit of our approach by just measuring the achieved fault coverage difference. For the mc8051, we reached around 61.64% of transition fault coverage once we applied all the 117 test programs using the Func2LOC test scheme, however by applying the same test set in a sequential way, we achieved only 39% as transition fault coverage.

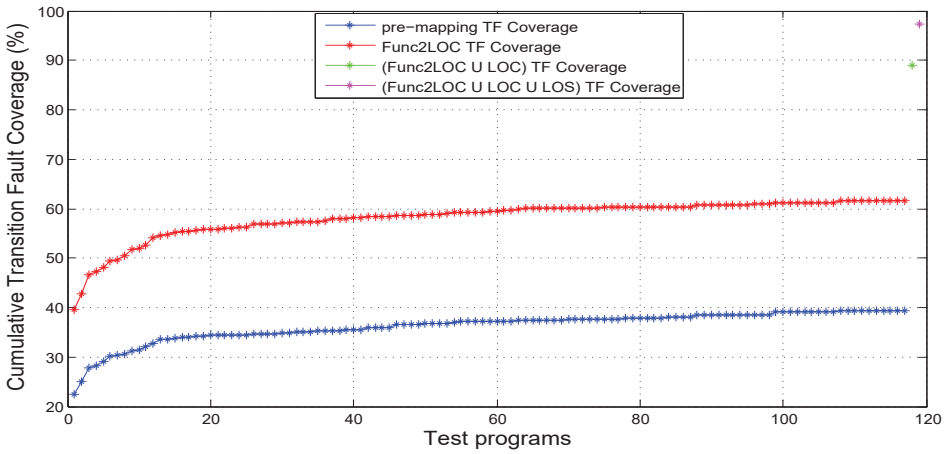


Figure 4.12: Cumulative transition fault coverage : mc8051

Second comment concerning the cumulative functional test programs approach is that the obtained fault coverage could be further enhanced but to the detriment of the test cost and the structural test power issues risk. Depending on the requirement fixed by the test engineer, we proposed a flexible way to satisfy the fault coverage/test length tradeoff. If, for specific cores designed for specific applications, a high fault coverage is required, we can benefit from the structural test patterns capability and target the remaining undetected faults and achieve the expected fault coverage.

Again considering the obtained results on the mc8051 case study, as already shown; we reached 61.64% for the transition fault coverage, using the Func2LOC cumulative fault coverage approach. With the help of the ATPG, using the LOC test scheme and targeting the remaining transition faults, we can reach 88.91% as shown in green color in Figure 4.12. Getting the remaining undetected faults, we can exercise the LOS test scheme as last resort and achieve the 97.27% transition fault coverage, shown in magenta color.

4.5 Summary

In this chapter, we presented a comprehensive analysis of the effects of re-using existing functional test programs for transition delay test in the context of power aware and at-speed microprocessor testing. In this study, we proposed to map the functional programs into a LOC-like test scheme. This mapping allows applying functional programs exploiting the existing scan chain. This increases especially the transition fault coverage without adding any extra test-point or dedicated DfT architecture. In the last part of the presented analysis, we prove that some transition faults are indeed undetected by the classical structural test scheme while the functional programs can detect them. At the end of this work, we proposed to use a global test scheme which benefits from both structural and functional test sets in order to maximize the achieved transition fault coverage by avoiding any over as well under-testing issues.

Chapter 5

Scan-Chain Intra-Cell Aware Testing

Contents

4.1 Introduction 36
4.2 Background 37
4.3 Proposed Methodologies 38
4.4 Experimental Results 41
4.4.1 Fault Coverage Enhancement Results 41
4.4.2 A Global Test Solution Results 44
4.5 Summary 48

5.1 Introduction

The endless advance of semiconductor technologies results in an increasing complexity of digital circuits. Designing and manufacturing smaller, faster, cheaper and less power consuming devices are the main challenges in semiconductor industry. The reduction of transistor size and the latest packaging technology (i.e., System-On-a- Chip, System-In-Package, Through Silicon Via 3D Integrated Circuits) allow the semiconductor industry to satisfy the latest challenges. Although producing such advanced circuits can benefit users, the manufacturing process is becoming finer and denser, making chips more prone to defects. In modern deep submicron technologies, systematic defects are becoming more frequent than random defects [41].

Today, systematic defects appear not only in the cell interconnections, but also inside the cell itself (intra-cell defects). In the literature, many works prove that these defects can escape classical test solutions. In [11] a statistic carried out over 1 million tested devices showed that a significant number of defects appear inside standard cells (i.e., intra-cell defects). In [13][12][42], it is shown that these defects cannot be detected by using approaches based

on classical fault models (i.e., stuck-at[43], transition[44], bridging[45]). Despite the fact that previous works already proved that classical test sets lead to a low coverage of intra-cell defects, none of them deeply investigated the issues related to scan chain testing in the presence of intra-cell defects.

Usually, scan chain testing is performed by applying a so-called **shift test** [7]. A toggle sequence "00110011..." is shifted into the scan chain and values appearing at the other extreme of the chain are checked. The applied sequence produces all possible transitions at the scan-input of each scan flip-flop. In this way, the correctness of the shift operations is verified and the presence of possible stuck-at and transition faults in the scan flip-flop interconnections can be detected. Moreover, the work in [46] shows that the above sequence can only cover the intra-cell defects affecting the scan-path of each scan flip-flops.

Despite the fact that the shift test is widely used in practice, authors in [47] prove that some intra-cell defects can escape because the scan chain test is applied only when flip-flops are in test mode. The intra-cell defect coverage is indeed too low. It is thus mandatory to analyze and quantify the intra-cell defect escapes to eventually develop meaningful test solutions.

In this chapter, we first show that the percentage of intra-cell defects escaping the standard tests is indeed very high (up-to 60%). Then, we propose a high quality test solution (based on combining different test sets generated by a commercial ATPG targeting different fault models) achieving high intra-cell defect coverage. Finally, we compare the defect coverage of the proposed test solution with the one produced by an available commercial tool when directly targeting intra-cell defects. Experimental results show that on average our solution outperforms the latter in terms of both defect coverage and test length.

The chapter is organized as follows: Section 5.2 presents the basics of scan chain design and test, while Section 5.3 sketches the overall flow. The defect characterization is reviewed in Section 5.4. In Section 5.5, the methodology for scan chain intra-cell aware test is presented. Experimental results and defect coverage analysis are presented in Section 5.6. Conclusions are given in Section 5.7.

5.2 Scan Chain Test

This section presents the basics of scan chain design and test. Figure 5.1 shows the well-known MUX-scan flip-flop library cell architecture. It is composed of a D Flip-Flop (DFF) plus a multiplexer. The input signal Scan Enable (SE) allows to select between the scan mode (SE = '1') and the functional mode (SE = '0'). In scan mode, the flip-flop stores the value coming from the Scan-In (SI), while in the functional mode it stores the value coming from D.

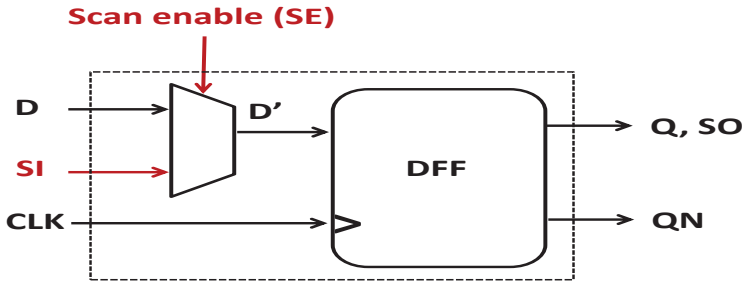


Figure 5.1: MUX-scan flip-flop

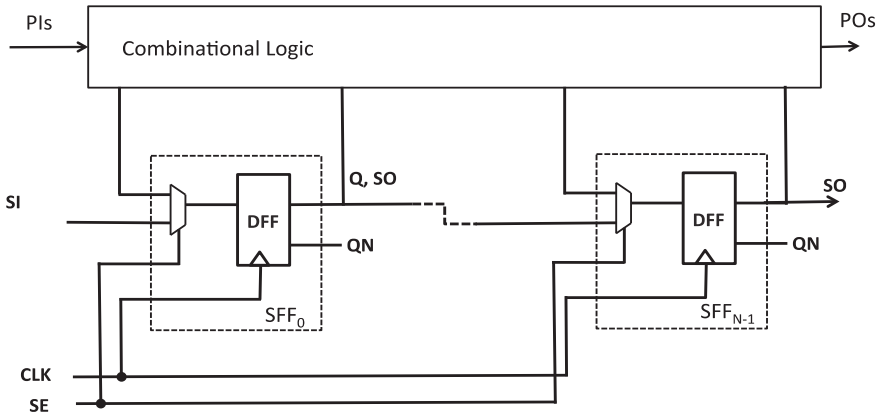


Figure 5.2: Full scan architecture

Figure 5.2 depicts the classical full scan architecture. Here, each scan flip-flop corresponds to one element of the scan chain (from SFF_0 to SFF_{N-1}). The scan chain is controlled by the SE signal and latched by the clock (CLK).

Scan designs are usually tested in two steps. First, a **Scan Chain Test** is applied. A single test pattern composed of a sequence of alternated "00" and "11" bits ("00110011...") is shifted in and out of the scan chain. This sequence produces all possible transitions on the input of each scan flip-flop. In this way, the ability of the scan chain to support the shift operations is verified and the presence of possible stuck-at and transition faults in the scan flip-flop interconnections is detected. If the scan chain test succeeds, the scan design test goes through the second step, corresponding to the **Logic Test**. It aims at testing the combinational logic in between the scan chains. It resorts to test patterns generated by an ATPG tool targeting several fault models (e.g., Stuck-at Faults, Transition Faults, ...). Logic test is applied and the output of the combinational blocks is latched by switching scan flip-flops from test to functional mode [7].

As reported in [46], scan chain test is efficient in testing intra-cell defects affecting the D flip-flops. However, since the scan flip-flops always work in test mode during the scan chain testing, some defects in the scan flip-flops cannot be detected since they can be sensitized only in functional mode. Experimental evidence of the correctness of this hypothesis has been provided in [47], whose authors show that some intra-cell defects actually escape. Physical

Failure Analysis proves that those defects affect the multiplexer controlling the scan flip-flop input.

The main contribution of this chapter is the analysis of the test quality with respect to intra-cell defects affecting the multiplexer of scan flip-flops. We first evaluate the percentage of intra-cell defects detected when scan chain test is performed. Secondly, we compute the intra-cell defect coverage when logic test is performed. The latter is done by applying three types of test sets, developed for the Stuck-at fault model and for the Transition Delay Fault model, adopting the LOC and the LOS schemes [7]. Finally, we propose to merge together the above test sets, thus achieving a high test quality for the scan chain (intra-cell defect aware testing).

5.3 Overall Flow of The Proposed Approach

Figure 5.3 sketches the overall flow of the proposed approach, which is composed of three steps. The first one is the **technology library characterization** (i.e., the grey box). In this step, an automatic tool extracts all possible defect locations for every library cell. Then, for each location, a defect injection campaign is executed. It exploits a transistor-level simulator to determine the faulty behavior of the injected defect. The result is the **Defect Database**. Please note that this step is applied only one time for a given technology library. The details about the considered defects and the location extraction will be given in the next section.

The second step is the **Intra-Cell Defect Grading**. Three inputs are required: (i) the previously computed defect database, (ii) the circuit under evaluation and (iii) the test sets generated by the ATPG. For each test set it computes the defect coverage.

More in detail, by using a logic simulator and for each ATPG test set, we extract the flip-flops activity. Then an ad-hoc Defect Coverage Estimator tool, based on both flip-flops activity and the previously computed defect database, will provide the defect coverage.

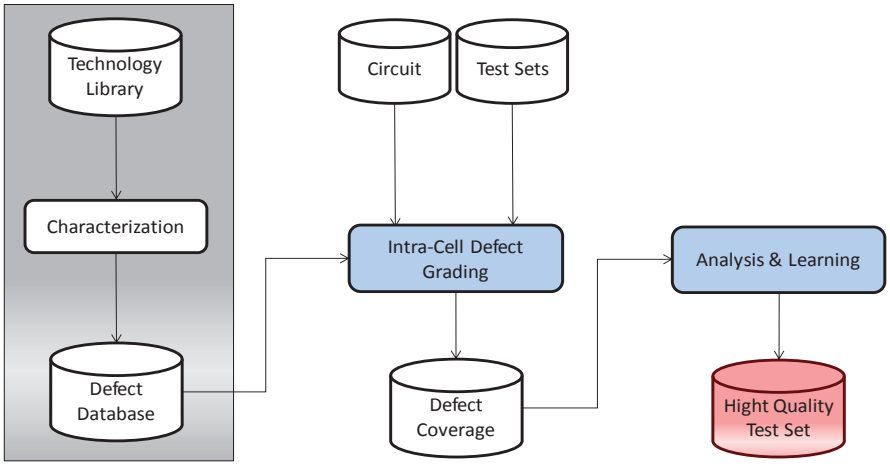


Figure 5.3: Overall flow

The last step is the Analysis and Learning. It aims at merging together the graded test sets by applying an in-depth analysis. The goal is to obtain a single high quality test set.

Two inputs are required: the test sets and the defect coverage determined in the second step.

Please note that the main focus of this chapter is on the intra-cell defect grading and on the defect coverage analysis steps. The latters can be applied on any technology library. We present in the next section the main guidelines about the considered defects and the location extraction performed during the technology library characterization.

5.4 Defect Characterization

The defect characterization for a given technology library is done by means of a defect injection campaign. Several papers described this process [13][12][42]. In this work we exploit the approach published in [42]. In this section, we recall the main concepts of [42].

For each library cell, we must determine all the possible defect locations (i.e., where a defect can appear) and the type of defect. In our work, the location can be any cell internal net. As already described in previous work [42] [48], cell layout analysis can be used to identify the realistic defect locations. Then, for each realistic defect location, defect injection is performed to evaluate if the behavior induced by the injected defect is covered or not by the applied set of stimuli. Finally, the defect database is created. Note that any transistor-level simulator can be used to perform this analysis.

Injected defects are modeled in the transistor-level domain as: (i) an unexpected connection between two nets associated to a specific resistance value (resistive-short), (ii) an unexpected resistance value on a given net (resistive-open) [49]. Injected defects can lead to either static faults (i.e., stuck-at faults) or dynamic faults (i.e., delay faults).

More in details, when considering the short resistive behavior (i) the choice of the resistance value will either let the circuit to behave like a short circuit (i.e., resistance value close to zero) leading then to stuck-at and bridging defects or introduce a time constant when well chosen resistance value is considered ($0 << R < R_open$). This latter case will lead to transition delay defects. However in case of resistive-open defects (ii), when the resistance value is close to infinite, the circuit will behave like an open circuit and lead to static faults. Similarly to the previous case (i.e., resistive-short), when lower resistance values are considered ($R_short < R << \infty$) a time constant is introduced and again a transition delay defect come out. Please note that R_open and R_short correspond the resistance values limits where the circuit start behaving like an open and short circuit respectively. It is clear so that an accurate defect model should consider meaningful resistance values to enclose all possible defects discussed above. In our case, the obtained defect database is built considering the following scenario:

- For the resistive-short approach, only static faults have been considered (i.e., resistance value less than 1Ω);

- For the resistive-open approach, static faults (value of resistance more than $1G\Omega$) and dynamic faults ($2.0\text{x}10^8$ Ω or $2.0\text{x}10^7$ Ω, depending on the net type) had been injected;

To sensitize the static faults, test patterns are made with only one test vector, while for the dynamic ones, we have to apply test patterns that contain a couple of test vectors in order to force a transition that actually sensitizes and propagates the fault.

Since we target library cells, we can resort to an exhaustive test pattern generation in order to insure that all possible defects are sensitized and observed. As discussed above, we consider couples of test vectors in order to be sure that even dynamic faults are sensitized. We resort to the following notation derived from [50]:

- C0: Static 0, this symbol corresponds to the couple "00";
- C1: Static 1, this symbol corresponds to the couple "11";
- R1: Rise to 1, this symbol corresponds to the couple "01";
- F0: Fall to 0, this symbol corresponds to the couple "10";

Four symbols are used, thus for a given library cell having n inputs, we have to simulate all possible 4^n input combinations. Once again, since we target library cells the number of input signals is low (i.e., four in average, up to 10 for the most complex cells) thus making the exhaustive generation and simulation feasible. The created database is a simple table having one row per input pattern. For each test pattern, the table stores the list of detected defects as shown in Table. 5.1.

Table 5.1: Defect database

IP	Detected Defects
Pattern1	Defects list
...	...
PatternN	Defects list

Let us resort to an example to show the process of library characterization. Figure 5.5 gives the schematic view of a two-ways multiplexer (MUX21) from an industrial 90nm technology library.

Thanks to the layout analysis we identify R10 as a realistic defect location (orange connection in 5.5, between net098 and net8). Then, as a defect type we consider a resistive short, thus the resistance value is set to 1Ω. Finally, for each possible input pattern we run a Simulation Program with Integrated Circuit Emphasis (SPICE) simulation. Simulations are done by fixing the temperature at 25°C and the Vdd at 1V. Results are collected in the defect database, as shown in Table. 5.2. Among all the possible 64 patterns (i.e., 4^3) only two of them detect the injected defect: "C1,C0,C1" and "R1,F0,R1". This happens because the injected defect leads to a static fault, thus we can detect it either when we apply stable values ("C1,C0,C1") or when we apply transitions ("R1,F0,R1"). In this example, we omit to put in the table the remaining patterns and remaining defects only for the sake of readability.

Table 5.2: Multiplexer defect database example

Input A, input B, input S	Detected Defects
C1,C0,C1	R10, ...
R1,F0,R1	R10, ...
...	...

Let us refer to Figure 5.4 [51] which plots the waveforms of the two-ways multiplexer (MUX21) when "C1,C0,C1" test pattern is considered among others. The output signal (shown in red) highlights the misbehavior in presence of the injected resistive-short defect (i.e., R10). As it can be seen, the test patterns are applied with a clock period of 50ns.

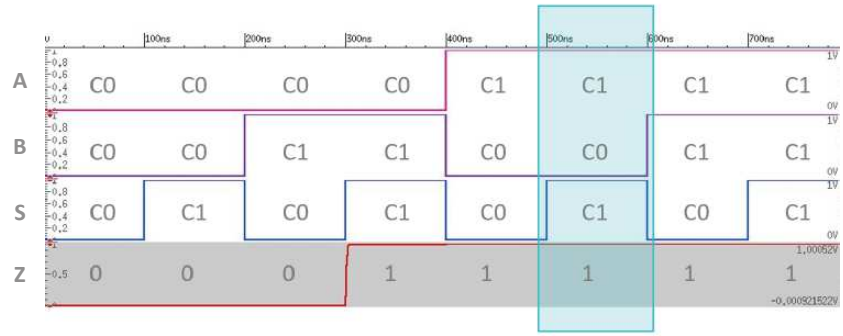


Figure 5.4: Spice simulation

The same process should be repeated for all other defect types and defect locations.

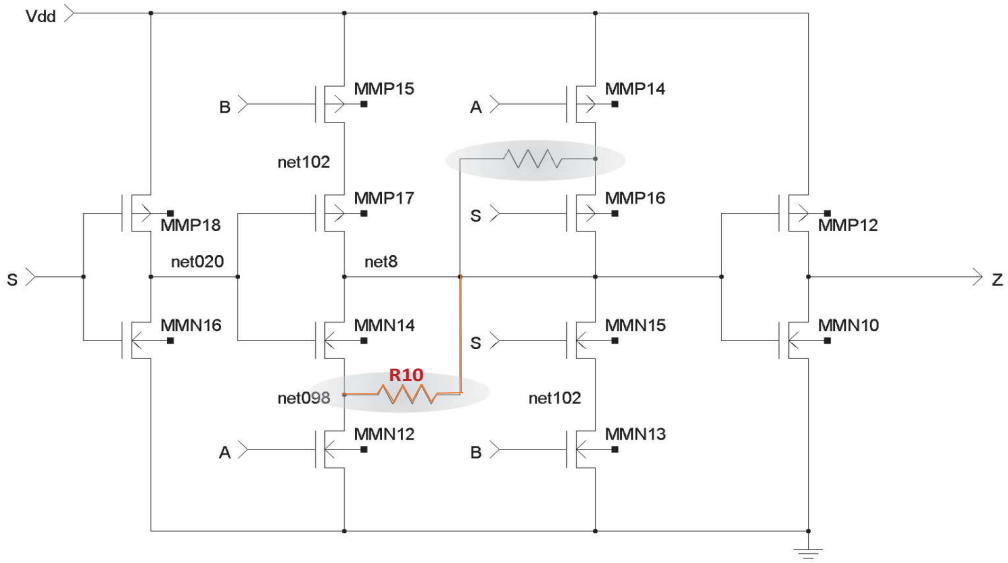


Figure 5.5: Schematic view of a MUX21

To quantify the complexity of one library cell characterization we resort to Equation 5.1:

$$O = \#DL * \#DT * \#IP \tag{5.1}$$

Where:

- O: represents the order of magnitude of the computation;

- #DL: is the defect location number, the possible locations are extracted during the cell layout analysis;

- #DT: is the defect type number, the range of possible resistance values varies from few Ωs (short) up to $1M\Omega$ (open);

- #IP: is the input pattern number;

5.5 Intra-Cell Defect Grading

This section describes the methodology followed during the intra-cell defect grading process. Once again, the target is the defect affecting the multiplexer of the scan flip-flop. The intra-cell defect grading is performed by exploiting a classical serial fault simulation technique [7]. After the library characterization, we identified 162 defects affecting the multiplexer of the scan flip-flop cell. Knowing the overall number of intra-cell defects affecting the multiplexers, we can define the Defect Coverage (DC) metric as follows:

$$DC = \frac{\#DD}{162 * \#N} \tag{5.2}$$

where

- DC: is the defect coverage;

- #DD: is the number of detected defects;

- #N: is the number of flip-flops in the target circuit.

For each circuit we simulate two types of test: scan chain and logic tests (see Section 5.2). For each type of test, different test sets are considered. The details are given in the following subsections.

5.5.1 Scan Chain Test

The scan chain test set is composed of load and unload operations. All scan flip-flops are set into scan mode (SE = '1') and a single pattern is shifted-in (load) and shifted-out (unload) through the scan chain.

In our work, we consider several test sets. Each set is composed of a single test pattern to be applied through load and unload operations. Applied patterns are the ones generated by the commercial ATPG tool used for our experiments: "0011", "0101", "1000" and "0111". Please note that patterns are repeated several times depending on the scan chain length. Each

pattern guarantees that each scan flip-flop is tested under the application of the same stimuli. For example, pattern "0011" produces the following stimuli: "00", "01", "11" in all flip-flops. Those values are applied through the SI input of each scan cell. For the same example, let us imagine that the scan chain is composed of 8 scan flip-flops, then the applied test pattern will be "00110011" and all possible 2-bits combinations (stimuli) will be checked. On the other hand, the logic value applied to the input D depends on the combinational logic of the circuit. Thus, this value can be different for every scan cell. Since we are looking for the intra-cell defects affecting the multiplexer, the knowledge of the value of the input D is mandatory.

We define the activity of the scan flip-flop i ($0 \leq i \leq N$-1, where N is the length of the scan chain) at a given clock cycle j as the following 4-tuple:

$$Activity_{i,j} = \{D, SI, SE, Q\} \tag{5.3}$$

where D, SI, SE are the inputs and Q is the output of the scan flip-flop i. Please note that each element of the 4-tuple is encoded by using the symbols of [50], as described in Section 5.4. Thus, the activity at the clock cycle j encodes the couple of logic values applied during the clock cycles $j-1$ and j.

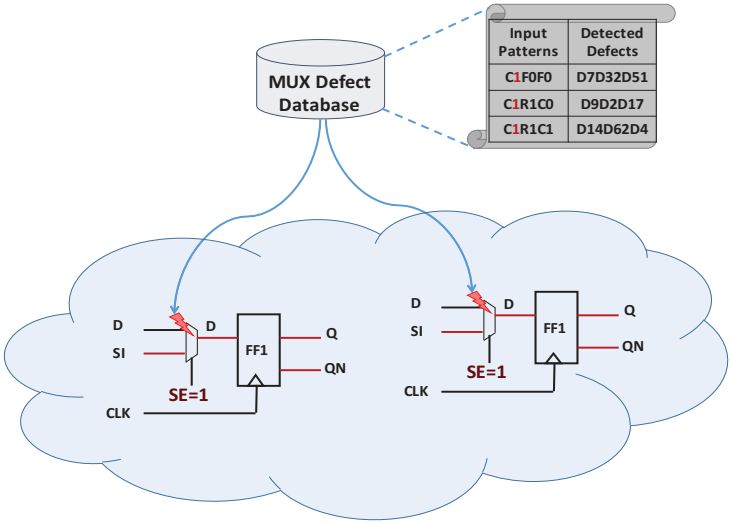


Figure 5.6: Scan-chain test intra-cell defect grading

In 5.6, we show the basic principle of defect grading. Since we apply the scan chain test, all scan flip-flops are set into scan mode(red connections in 5.6. Then, for each scan flip-flop i and for each target defect (among the 162 possible defects), we simulate a scan test pattern. During the simulation, at each clock cycle j we access the Defect Database by using the information stored in the related $activity_{i,j}$ tuple as defined by Equation 5.3 in order to determine if the target defect is **sensitized or not**. If yes, the value stored in the flip-flop i is inverted. At the end of the SI simulation, we compare the unloaded values with the golden ones to determine if the target defect has been **detected or not**. The simulation length is equal to $2 * N$ clock cycles. However, in practice, $N + 4$ clock cycles are sufficient to detect all possible detectable faults.

5.5.2 Logic Test

During the logic test, several test patterns are applied by using the scan chain, which is repeatedly switched from scan to functional mode. Logic test patterns are composed of *load*, *launch*, *capture* and *unload* operations (thus testing basically the combinational logic). The *load* and *unload* operations correspsond to serial shift-in and -out of a test vector (as described in the previous section), while the *launch* and *capture* correspond to the application of the vector to the combinational logic and to the capture of the test response respectively. Please note that the *launch* operation is only exploited during the test of delay faults (e.g., transition faults), while the *capture* is always used, independently on the target fault model.

Since intra-cell defects can lead to both static and dynamic faults (see Section 5.4), we resort to three widely used test sets: stuck-at fault test set, transition fault test set under the LOC scheme and transition fault test set under the LOS scheme [7]. In this case, we aim at quantifying the intra-cell defects detected when the scan flip-flops are in scan mode (SE = '1') as well as in functional mode (SE = '0').

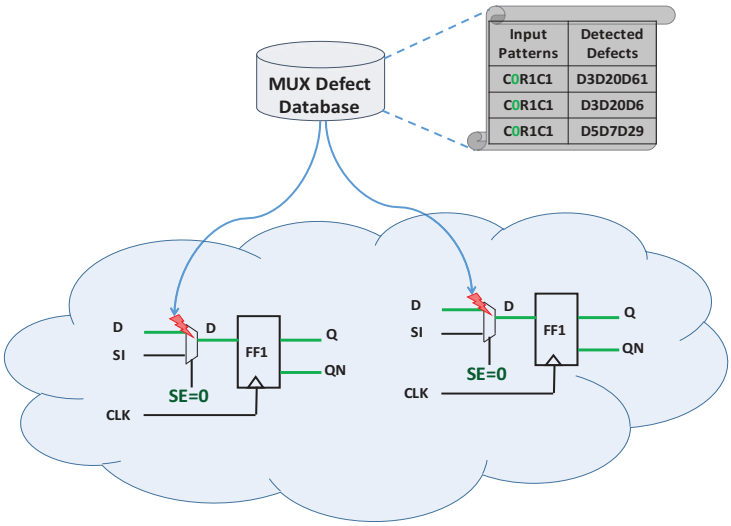


Figure 5.7: Logic test intra-cell defect grading

In 5.7, we show the principle of the logic test intra-cell defect grading. In this case, we have to consider the activity i,j not only during the *load* and *unload* operations, but also during the *launch* and *capture* (i.e., in functional mode) phases. The latter depends on the simulated test set, as depicted in 5.8. The figure reports waveforms related to the application of test sets: a) stuck-at test set, b) transition test set, further divided into the one adopting the LOC scheme, and the one adopting the LOS test scheme. For each one, we define the functional window as the time interval when SE $=' 0'$. This includes the capture edge for stuck-at and LOS test sets, while it includes both launch and capture edges for the LOC test set. Except for the functional window, the defect grading for the logic test is exactly the same as described in the previous section. The simulation length is equal to $[(N+1)*T+N]$ clock cycles for the stuck-at and LOS test sets, where N is the number of flip-flops composing the scan chain and T is the number of patterns composing the simulated test set. The term $+1$ is added to include the capture clock cycle during the functional window. For the LOC

test set, the simulation length is equal to $[(N+2)*T+N]$ clock cycles because in this case the functional window contains two clock cycles, corresponding to the *launch* and the *capture* operations.

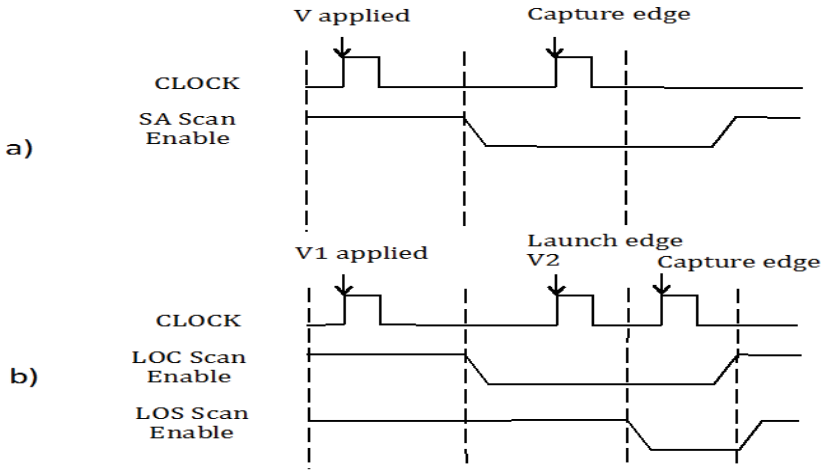


Figure 5.8: Logic test functional window, corresponding to the period when scan enable = 0

5.6 Results and Analysis

In this section, we present the results corresponding to the intra-cell defect grading process. Then, we present what it is possible to learn from the collected data in order to obtain a meaningful test solution. Finally, we compare the results achieved with the proposed test solution with the one from a commercial tool.

5.6.1 Defect Grading Results

We performed several experiments on the full-scan version of ITC99 benchmark circuits. All circuits were synthesized using an industrial 90nm technology library, and for each circuit one scan chain was inserted during synthesis. As mentioned in Section 5.5, during the library characterization we identified 162 defects affecting the multiplexer of the scan flip-flop cell. Thus, for each circuit the total number of defects is $162 * N$, where N is the number of flip-flops.

Table. 5.3 summarizes the defect grading results when logic test patterns are evaluated: results related to Stuck-at Fault (SAF), Transition Fault (TF)-LOC and TF-LOS patterns are reported in columns 3, 4 and 5, respectively. The first column reports the name of the ITC99 full-scan circuit. The corresponding number of scan flip-flops is reported in column 2. For each test set we report in the sub-columns the test length in terms of number of patterns (#patterns), the achieved fault coverage (Fault Coverage (FC)%) and the intra-cell DC (%).

As expected (and already mentioned and explained in a previous works [48] [12]), we obtained a large gap between fault and defect coverages. In fact, the gap can reach up to 43%

for the b09 circuit if we consider LOS patterns and 40% for b02 if we consider LOC patterns. More in detail, we obtained that in average SAF patterns achieved 60% of defect coverage, while TF-LOC patterns reached 71% and TF-LOS patterns reached 65%.

The difference in defect coverage between SAF and TF test patterns can be easily explained. TF test patterns (either LOC or LOS) lead to a greater number of signal transitions compared to SAF ones. Thus, the number of covered defects is higher because both static and dynamic faults are detected. Conversely, SAF test patterns mainly detect static faults, thus missing the dynamic ones. Please note that this depends not only on the applied test set, but also on the circuit structure and on how it propagates the fault effects. An example is the circuit b18, for which the defect coverage achieved by the SAF test patterns is comparable (and even higher) than the one obtained by TF-LOC and LOS patterns.

Table. 5.10 summarizes the defect grading results when scan chain test patterns are applied. The first column reports the name of the ITC'99 full-scan circuits. The corresponding number of flip-flops is reported in column 2. For each graded scan test pattern, we reported the achieved intra-cell DC (%) in columns 3, 4, 5 and 6, respectively.

The first comment refers to the difference in defect coverage average between scan chain test (i.e., by considering the average of all the scan chain tests defect coverage) and logic test (i.e., by considering the average of all the logic test defect coverage), which is about 56%. The observed gap can be explained by the fact that in logic test the scan chain works in both functional and scan modes, thus leading to a higher defect coverage.

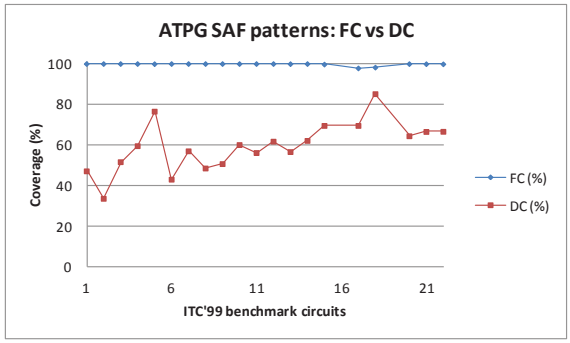
The second comment is about the fact that the defect coverage varies depending on the circuit. This result is somewhat expected since during the shift in/out operations the output of each scan flip-flop is propagated trough the combinational logic. Those propagations lead to have logic values applied to the D input of scan flip-flops. Thus, for some circuits the above values can be effective for defect coverage (for example for the circuit b15) while this does not happen for some others (circuit b18).

Finally, results in Table 5.10 do not vary among the applied test patterns. Only few exceptions can be noted for the b01 and b18. Again, here it is a matter of circuit structure and how test patterns are propagated trough the combinational logic.

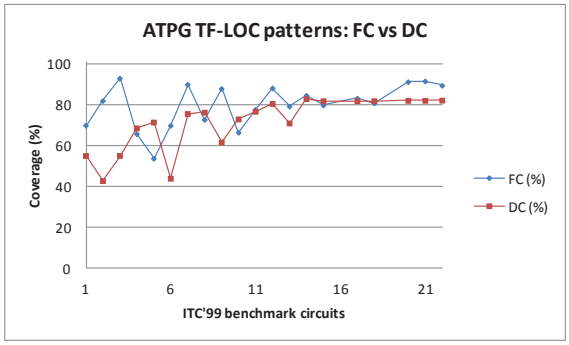
Figures 5.9 show clearly the large gap between the FC and the DC for the different classical fault models. Some exceptions, as already mentioned, are represented in Figures 5.9 b) & c) when transition patterns are considered. When LOC test patterns are applied, six out of twenty circuits show higher defect coverage compared to the classical fault coverage. However just one exception is observed when LOS test patterns are defect-graded (b05).

Table 5.3: Defect coverage evaluation for logic test

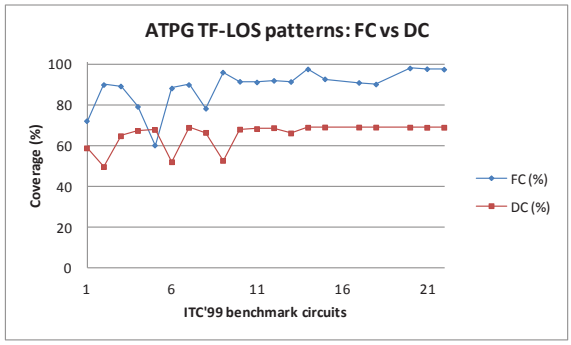
Circuit	#Flip-Flops	Logic Test								
		SAF			TF-LOC			TF-LOS		
		# patterns	FC (%)	DC (%)	# patterns	FC (%)	DC (%)	# patterns	FC (%)	DC (%)
b01	5	23	100	47.16	18	69.91	55.06	27	72.22	58.89
b02	4	17	100	33.64	13	82	42.75	17	90	49.69
b03	30	49	100	51.67	69	92.99	55.1	53	89.29	64.85
b04	66	71	100	59.57	44	65.74	68.58	103	79.23	67.43
b05	34	110	99.93	76.56	112	53.69	71.46	119	60.25	67.94
b06	8	17	100	43.06	13	69.82	43.9	22	88.29	52.08
b07	41	85	100	57.02	123	89.95	75.63	126	90.01	68.96
b08	21	79	100	48.56	54	72.73	76.3	65	78.33	66.49
b09	28	43	100	50.73	43	87.9	61.66	56	96.11	52.73
b10	17	70	100	60.1	48	66.4	72.98	97	91.57	68.08
b11	30	148	100	56.19	186	77.69	76.74	205	91.26	68.35
b12	119	191	100	61.78	450	88.15	80.54	292	92.04	68.81
b13	45	59	99.94	56.65	79	79.26	70.99	74	91.49	66.21
b14	215	413	99.99	62.16	440	84.57	82.93	608	97.67	69.02
b15	415	813	99.74	69.54	1.462	79.79	81.88	1.440	92.79	69.13
b17	1.311	1.780	97.83	69.53	4.130	83.15	81.9	3.152	91.02	69.11
b18	2.754	3.581	98.24	85.17	8.378	80.81	81.92	6.196	90.24	69.09
b20	429	577	99.97	64.44	1.219	91.23	82.1	947	98.13	69.06
b21	429	611	99.92	66.56	1.179	91.57	82.09	963	97.78	68.99
b22	611	748	99.86	66.64	1.281	89.43	82.18	1.239	97.56	69.02
Average	_		99.77	59.34	_	79.84	71.33	_	88.76	65.20



(a)



(b)



(c)

Figure 5.9: Fault & defect coverage evaluation of the ATPG patterns

Table 5.4: Defect coverage evaluation for scan-chain test

Circuit	#Flip-Flops	Scan chain test			
		"0011"	"0101"	"1000"	"0111"
b01	5	11.36	5.09	11.36	12.10
b02	4	9.88	11.11	9.88	11.11
b03	30	12.18	12.18	12.18	11.85
b04	66	3.22	3.25	3.17	3.25
b05	34	12.20	12.16	12.20	12.16
b06	8	10.34	9.88	10.80	9.72
b07	41	12.35	12.35	12.35	12.20
b08	21	12.35	12.35	12.35	12.22
b09	28	12.35	12.35	11.99	12.35
b10	17	12.06	12.35	11.18	11.76
b11	30	12.18	12.18	11.52	11.69
b12	119	3.16	3.18	3.13	3.18
b13	45	12.15	12.13	12.15	12.10
b14	215	3.13	3.14	3.11	3.14
b15	415	12.35	12.33	12.35	12.33
b17	1,311	12.34	12.34	12.34	12.34
b18	2,754	3.09	12.34	12.34	12.34
b20	429	3.11	3.11	3.10	3.11
b21	429	3.11	3.11	3.10	3.11
b22	611	3.10	3.10	3.10	3.10
Average		8.80	9.00	9.19	9.26

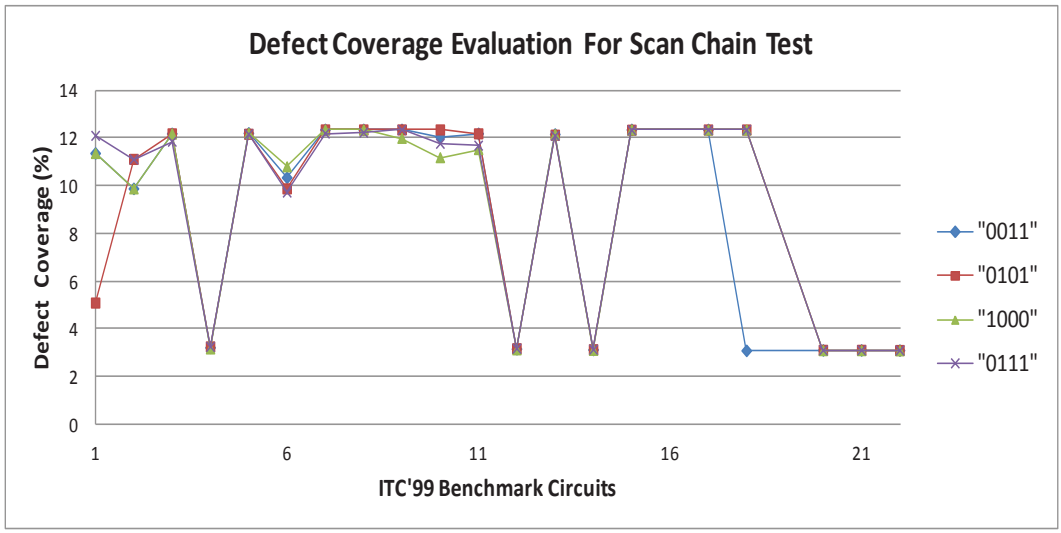


Figure 5.10: Defect coverage evaluation for scan-chain test

5.6.2 Learning

During the learning step, we aim at determining the capability of one test set to cover a specific set of defects that is actually uncovered by another test set. The goal of this analysis is to merge together different sets of test patterns to increase the overall defect coverage.

First of all, we compute the cumulative results for the three Logic Test (LT) sets: SAF, TF-LOC and TF-LOS as qualitatively reported in 5.11. Then, we further consider the contribution of the Scan-chain Test (ST) to the overall defect coverage.

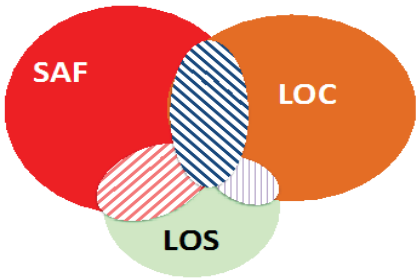


Figure 5.11: Defect coverage capabilities of the different test sets

Table 5.5: Logic test overall defect coverage

Circuit	LT DC (%)	LT & ST DC (%)
b01	72.84	73.09
b02	60.65	60.96
b03	76.44	76.44
b04	82.73	82.73
b05	80.27	80.27
b06	67.90	67.90
b07	81.62	81.62
b08	83.92	83.92
b09	75.44	75.53
b10	85.44	85.44
b11	84.44	84.44
b12	84.87	84.87
b13	82.11	82.13
b14	84.91	84.91
b15	86.4	86.4
b17	86.16	86.16
b18	87.71	87.72
b20	85.73	85.73
b21	85.84	85.84
b22	85.91	85.91
Average	81.07	81.10

5.5 quantifies the overall defect coverage by considering the union among the three logic test sets (second column). The obtained defect coverage increases for all the circuits; on average we detect about 23% more defects than considering a single test set. The third column reports the defect coverage achieved when considering the contribution of the scan chain test. From the reported data (shown also in Figure 5.14), it is easy to see that the scan chain test contribution is negligible (0.03% in average). This result is explained by the fact that in logic test, the scan chain works in both scan and functional modes. Therefore, most of defects covered by the scan chain test are indeed already covered by the logic test during

the load, unload and capture operations.

After the learning phase, we are able to propose a test solution composed of different test sets. However, the maximum value of the defect coverage (achieved for the b18 circuit) is only 87.72%, which could not be enough. As it can be seen, a number of defects are not covered by the standard fault models (already experienced) and are only detected by accident; these defects require specific conditions that cannot be defined for the existing fault models. Thus, we further investigate the possibility to eventually increase the defect coverage by generating a dedicated test set.

To do that, we exploit a commercial intra-cell aware ATPG tool to generate a test pattern set able to detect escaped defects. More in details, we opted to use UDFM [52], which basically extends the natively-supported faults models (primarily stuck-at and transition) by adding combinational or sequential constraints on other pins/nets. It enables us to generate specific test patterns for process-related defects that require additional constraints.

One of the restrictions we could encounter when applying the UDFM, is that we have to target static and delay faults separately since they cannot be handled together in one pattern generation step.

Each defect among the escaped ones is defined with the following information:

- It is associated to a specific scan flip-flop i in the CUT;
- It requires the knowledge of the logic values to be applied to the scan flip-flop inputs to sensitize the faulty behavior;
- It requires the knowledge of the fault effect.

Since the above information are already available from the defect database created during the technology library characterization and also from the learning phase, the UDFM perfectly matches our objective and will exploit the above information to generate a meaningful test pattern able to sensitize the targeted defect (i.e., it applies the required logic values to the scan flip-flop i inputs) and to propagate its fault effect to reach an observable point (i.e., either a primary output or a scan cell).

Let us refer to a small example that test a cell internal bridge fault of a 2-way multiplexer considered in 5.1. Example 5.12 specifies a delay test with the required activation conditions on the input pins ("D,SE,SI") for observing on the output pin ("D'") the effect of the bridge fault. It specifies a test scenario where the "D" input transitions from '1' to '0' while observing that the "D'" output is faulty if it remains at '1'.

```
UDFM {
    version : 1;
    UdfmType ("Special-delay") {
        Cell ("MUX21") {
            Fault ("myFlt-D0") {
                Test {
                        DelayFault {"D'" : 1;}
                        Conditions {"D" : 10; "SI" : 00; "SE" : 00;}
                    }
                }
            }
        }
    }
```

Figure 5.12: Delay fault UDFM example: MUX21

Once we know in which scan flip-flop the targeted multiplexer is associated, we could target the same defect (referring to the same example) by writing an equivalent UDFM file on the corresponding scan-flip-flop instance as shown in Figure 5.13.

```
UDFM {
    version : 1;
        UdfmType ("Special-delay") {
            UdfmType ("udfm_fault_model")    {
                Instance ("\stato_reg[1]") {

                Fault ("myFlt-0") { Test { DelayFault {"Q" : 1;}
                Conditions {"D" : 10; "TI" : 00; "TE" : 00;}}}

                Fault ("myFlt-1") { Test { DelayFault {"SO" : 1;}
                Conditions {"D" : 10; "TI" : 00; "TE" : 00;}}}
                }
            }
        }
    }
```

Figure 5.13: Delay fault UDFM example: MUX-Scan Flip-Flop

When the UDFM file is loaded, the defined delay faults are applied to the specified instance.

Table 5.6 reports the contribution to the defect coverage when adding extra patterns generated by the intra-cell aware ATPG tool. Column 2 reports the initial defect coverage by considering both logic tests & Scan chain Test. Column 3 reports the obtained defect coverage. Again, we could actually increase the coverage for all the circuits. In average, the coverage improvement is about 8.76%. The fourth column reports the overhead in terms of the percentage of extra patterns that have to be applied to improve the coverage, which is about 35%. Therefore, a high quality test solution has to be composed of logic tests plus a fragment of extra test patterns generated to specifically target the escaped defects.

Table 5.6: Intra-cell aware ATPG construction

Circuit	LT & ST DC (%)	DC (%)	Pattern Overhead (%)
b01	73.09	85.58	26
b02	60.96	76.56	34
b03	76.44	89.62	62
b04	82.73	91.16	77
b05	80.27	90.23	30
b06	67.90	83.80	56
b07	81.62	89.92	27
b08	83.92	90.13	22
b09	75.53	86.89	49
b10	85.44	90.73	15
b11	84.44	91.39	12
b12	84.87	90.39	22
b13	82.13	89.85	50
b14	84.91	92.71	34
b15	86.40	26.24	24
b17	86.16	92.81	27
b18	87.72	93.72	30
b20	85.73	93.10	39
b21	85.84	93.13	35
b22	85.91	92.76	38
Average	81.10	89.86	35

Figure 5.14, plots for each circuit the defect coverage enhancement starting from the Logic test set and finishing with the last Intra-cell Aware ATPG pattern generated especially for the escaped defects. The green plot represents the final achieved defect coverage (i.e., the proposed test solution) while the red one represents the contribution of both Logic and Scan-chain Test sets. As already explained, it is clear when we look to the difference between the red and blue plots, that the ST contribution can be neglected.

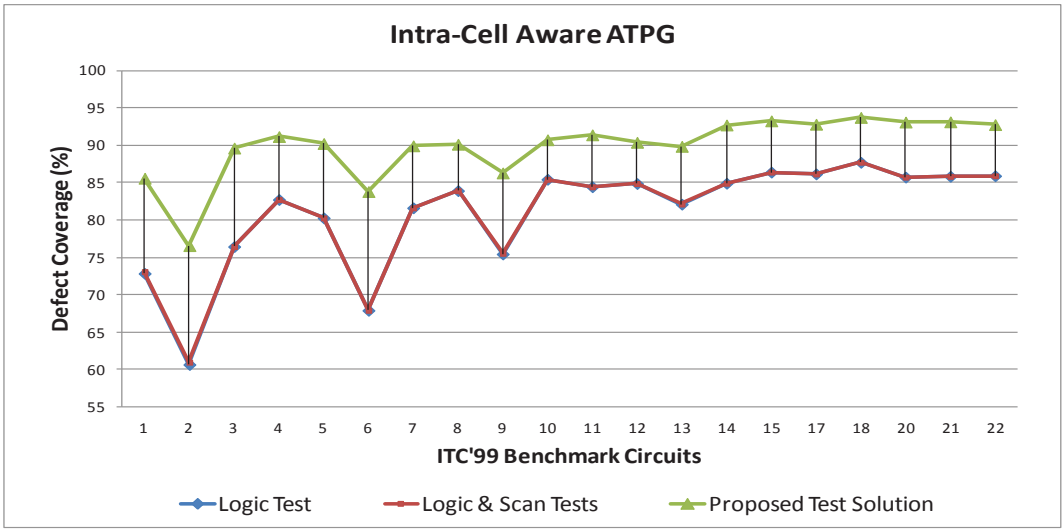


Figure 5.14: Proposed test solution

We now compare the above results with those produced by a single test set targeting intra-cell defects. Basically, we exploit a commercial intra-cell aware ATPG tool to target the whole set of defects for each scan flip-flop. The comparison is done in terms of achieved defect coverage and test time (i.e., number of test patterns).

Table 5.7 reports the comparison results. Columns 6 and 7 summarize the efficiency of our proposed approach in terms of defect coverage increase and test length decrease as compared to the intra-cell aware ATPG. It can be noted that in most of the cases (20 out of 22 cases, i.e., column 6) the coverage achieved by the proposed solution is higher than the one achieved by the intra-cell aware ATPG tool. Even if for the smaller circuits the difference is not so high, for the larger circuits the difference is significantly high. For example, for b18 the proposed test solution achieves 11.3% higher defect coverage than the intra-cell ATPG. If we consider the test pattern length, we lead to similar conclusions. In most of the cases the number of patterns of the proposed solution is smaller than the one of the intra-cell aware ATPG tool.

Table 5.7: Intra-cell aware ATPG comparison

Circuit	Proposed Test Solution		Intra-cell Aware ATPG		Comparison	
	# DC (%)	#p	# DC (%)	#patterns	% Increase in #DC	% Decrease in #patterns
b01	85.58	86	88.52	72	-2.94	-19.44
b02	76.56	63	73.77	45	2.79	-40.00
b03	89.62	277	84.96	550	4.66	49.64
b04	91.16	386	84.89	1.199	6.27	67.81
b05	90.23	443	83.42	576	6.81	23.09
b06	83.80	80.81	84.03	127	-0.23	36.22
b07	89.92	425	80.86	657	9.06	35.31
b08	90.13	241	80.25	343	9.88	29.74
b09	86.29	212	85.38	446	0.91	52.47
b10	90.73	247	78.36	261	12.37	5.36
b11	91.39	606	80.41	484	10.98	-25.21
b12	90.39	1.134	77.47	1.922	12.92	41.00
b13	89.85	317	75.51	639	14.34	50.39
b14	92.71	1.962	86.77	4.142	5.94	52.63
b15	93.26	4.610	82.24	7.471	11.02	38.29
b17	92.81	11.470	82.62	23.683	10.19	51.57
b18	93.72	23.681	82.45	49.369	11.27	52.03
b20	93.10	3.810	86.62	8.109	6.48	53.02
b21	93.13	3.272	86.83	8.192	6.30	54.50
b22	92.76	4.497	87.30	11.554	5.46	61.08
Average	89.86	2.913	82.63	5.992	7.22	33.47

Once again, for the largest circuits the difference is actually higher. If we consider b18, the proposed approach generates about 50% less patterns than the intra-cell aware ATPG (23,681 vs. 49,369 test patterns). It is important to mention that all the test sets have been generated by using the same commercial ATPG tool without any compaction option. The comparison results show that the proposed test solution definitely achieves better defect coverage with a lower test length.

5.7 Summary

Scan test is the most popular Design-for-Test technique. Assuring the correct behavior of the scan chain is therefore mandatory to ensure a high-test quality. In this chapter, we presented an intra-cell aware testing approach for the scan chain. We first evaluated the effectiveness of different test sets with respect to the intra-cell defects affecting the scan flip-flops. This evaluation considered both scan chain test and logic test, and was based on test patterns generated by a commercial ATPG. Then, we analyzed the obtained results and we proposed to combine together the test sets to achieve better defect coverage figures. Further analysis on the learning phase, allow us to target the missed intra-cell defects by using the

UDFM perfectly guided by the already built defect database. We exploited a commercial intra-cell aware ATPG tool to generate an efficient UDFM test pattern set able to detect escaped defects. We figured out that this defect grading procedure considering together Scan-chain, Logic and UDFM test patterns achieve higher defect coverage. This proposed test solution was validated for different ITĆ99 benchmark circuits. Finally, we compared the results produced by the combined test sets (i.e., the latter proposed test solution) with the ones obtained by using a commercial ATPG tool directly targeting intra-cell defects. The comparison shows that considering different test sets is more effective than straightforwardly activating the ATPG on the intra-cell defects, in terms of both test quality and test length.

Chapter 6

Conclusion

Contents

5.1 Introduction 49
5.2 Scan Chain Test 50
5.3 Overall Flow of The Proposed Approach 52
5.4 Defect Characterization 53
5.5 Intra-Cell Defect Grading 56
5.5.1 Scan Chain Test 56
5.5.2 Logic Test 58
5.6 Results and Analysis 59
5.6.1 Defect Grading Results 59
5.6.2 Learning 63
5.7 Summary 67

With aggressive scaling having become the norm in the semiconductor industry, meeting the demands for designing, testing and fabricating high quality devices is extremely challenging. Higher performance of integrated circuits needs to be achieved while respecting the constraints of low power consumption, required reliability levels, acceptable defect rates etc. In light of increasing complexity of digital circuits, test methodologies and adopted fault models should be reviewed to address the issues of over or under testing and accurate defect detection.

Moreover, diverse metrics of process variation, interferences, on-chip test switching activity could impact the quality and the cost of the final delivered device; some of them are related intrinsically to the manufacturing process and some others belong to the test practices especially when we consider the quantum of detected defects and achieved reliability. It has been observed that performance testing of low-power devices may reduce the production yield due to high test power. To cope with the issues of excessive test power consumption, we have demonstrated that functional tests effectively complement structural ones while optimizing

test power, achieving the required fault coverage and reducing both test time and memory occupation.

The focus of this work is aimed at the resolution of three main challenges.

1. Reducing both test time and memory occupation for in field-test SBST Despite the various advantages that SBST offers (flexibility, defect coverage, cost and online diagnosis), its adoption can be limited by the effort required to write the test program and can be also constrained by both memory and test time requirements to store and execute the test program respectively. Furthermore, it is essential that SBST programs must be able to achieve the required test standards and constraints that are specified by the end-user application.

2. Reuse of functional programs for power-aware at-speed testing At-speed tests performed using classical structural ATPG could be susceptible to causing yield loss or test escape because such patterns may induce excessive or negligible switching activity which does not accurately reflect the functional operational parameters of the CUT. Since these problems are inherently absent in functional patterns, it makes them an ideal candidate for power-aware at-speed testing.

3. Improving intra-cell defect coverage in scan-cells Since Scan-testing is one of the most widely adopted design for-test techniques, it is imperative to guarantee the absence of defects within the scan-cells. In this scenario, there exist a great need and scope to improve existing fault models and test patterns generation for achieving a high defect coverage when testing the scan-chain.

The work of this thesis is aimed at the resolution of the above listed challenges and is encompassed in the preceding four chapters and is briefly summarized hereunder.

- Chapter 2 evaluates functional programs considering both verification and test metrics in order to propose a comprehensive test solution. Evaluation of a wide set of functional programs, characterized by a high toggle rate, has led to the conclusion that no correlation exists between the corresponding switching activity with neither verification metrics nor the fault coverage.

- Chapter 3 focused on one the most critical test techniques based on SBST approaches. Considering an initial functional programs set, we applied an efficient test merging and compaction technique during the test application. The compressed test set does not compromise the achieved fault coverage while reducing the test time and the memory occupation. The presented experimental results for two different microprocessors show a high reduction in both test time and memory requirement.

- Chapter 4 presented a comprehensive analysis of the effects of re-using existing functional test programs for transition delay test in the context of power aware microprocessor testing. During this study, we proposed to map the functional programs into a LOC-like test scheme. This mapping allows applying functional programs exploiting the existing scan chain. This increases the transition fault coverage without adding any extra test-point or dedicated DfT architecture. In the last part of the presented analysis, we prove that some transition faults are indeed undetected by the classical structural test scheme while the functional programs can detect them. Finally, we propose to use a global test scheme to maximize the achieved

transition fault coverage by avoiding any over-testing issues.

- Chapter 5 deals with intra-cell defects that can appear in scan chains and presents an intra-cell aware testing approach. Initially the efficacy of different test patterns with respect to the intra-cell defects affecting the scan flip-flops is assessed. This evaluation takes into account both scan chain tests and functional tests, and was based on test patterns generated by a commercial ATPG. Upon analysis of the defect coverage achieved by logic & scan patterns, a combined test set which integrates logic, scan & UDFM patterns has been put forth. This combinational test pattern set manages to attain higher intra-cell defect coverage when compared to cell-aware ATPG as it succeeds in targeting the intra-cell defects missed by standalone ATPG patterns.

This work opens interesting perspectives concerning test methodologies/practices including the following:

1- Comparing the proposed method of using functional programs for power aware test with existing power ATPG techniques.

2- Evaluating the functional test programs based on their diagnosis capability. The diagnosis could be evaluated using the classical pass and fail procedure or also using both detection and primary outputs information, merged together, to define a more refined diagnosis metric. We plan first of all to build the diagnosis dictionary (based on individual functional patterns fault simulations) and then apply the two different diagnosis approaches for different microprocessor case studies.

3- Considering different scan-flip flops architectures which could include additional And-or logic as well further MUXs to multiplex different data inputs in addition to the presented Scan flip-flop in our intra-cell defect grading approach.

4- Since the functional test programs can be applied at speed, another direction is to study their effectiveness to detect small delay defects.

Scientific Contributions

[**IJ.1**] A. Touati, A. Bosio, P. Girard, A. Virazel, P. Bernardi, M. Sonza Reorda, A. Auvray. "Scan-Chain Intra-Cell Aware Testing", IEEE Transactions on Emerging Topics in Computing (Accepted journal).

[**IJ.2**] A. Touati, A. Bosio, P. Girard, A. Virazel, P. Bernardi, M. Sonza Reorda. "Microprocessor Testing: Functional meets Structural Test", Journal of Circuits, Systems, and Computers (Accepted journal).

[**IC.1**] A. Touati A. Bosio, P. Girard, A. Virazel, P. Bernardi, M. Sonza Reorda. "Improving the Functional Test Delay Fault Coverage: A Microprocessor Case Study", IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Pittsburgh, Pennsylvania, U.S.A, 2016.

[**IC.2**] A. Touati A. Bosio, P. Girard, A. Virazel, P. Bernardi, M. Sonza Reorda. "An Effective Approach for Functional Test Programs Compaction", Design & Diagnostics of Electronic Circuits & Systems (DDECS), 2016 (Best Award Paper).

[**IC.3**] A. Touati, A. Bosio,L. Dilillo, P. Girard, A. Virazel, P. Bernardi, M. Sonza Reorda. "Scan-chain intra-cell defects grading", Design & Technology of Integrated Systems in Nanoscale Era (DTIS), 2015 10th International Conference on, Naples, ITALY.

[**IC.4**] A. Touati A. Bosio,L. Dilillo, P. Girard, A. Virazel, P. Bernardi, M. Sonza Reorda. "Exploring the Impact of Functional Test Programs Re-used for Power-Aware Testing", Design Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, FRANCE, 2015.

[**IW.1**] A. Touati, A. Bosio, L. Dilillo, P. Girard, A. Todri-Sanial, A. Virazel, P. Bernardi, "A Comprehensive Evaluation of Functional Programs for Power-Aware Test", North Atlantic Test Workshop (NATW), IEEE 23rd Johnson City, NY, U.S.A, 2014.

[**S1**] A. Touati A. Bosio, P. Girard, A. Virazel. "Functional Testing Vs Structural Testing", IN: South European Test Seminar (SETS), 2014 & 2015.

[**S2**] A. Touati A. Bosio, P. Girard, A. Virazel. "Exploring the Impact of Functional Test Programs Re-used for Power-Aware Testing", IN: CNRS Research group colloquy on SOC-SIP (GDR SOC-SIP), 2014 & 2016.

[**S3**] A. Touati A. Bosio, P. Girard, A. Virazel. "Exploring the Impact of Functional Test Programs Re-used for At-Speed Testing", IN: Journées Nationales du Réseau Doctoral en Micro-nanoélectronique (JNRDM), 19ème. 2016.

Bibliography

[1] G. E. Moore. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff. *IEEE Solid-State Circuits Society Newsletter*, 11(5):33–35, Sept 2006.

[2] D. J. Frank, R. H. Dennard, E. Nowak, P. M. Solomon, Y. Taur, and Hon-Sum Philip Wong. Device scaling limits of si mosfets and their application dependencies. *Proceedings of the IEEE*, 89(3):259–288, Mar 2001.

[3] C. Hawkins, A. Keshavarzi, and J. Segura. View from the bottom: nanometer technology ac parametric failures - why, where, and how to detect. In *Defect and Fault Tolerance in VLSI Systems, 2003. Proceedings. 18th IEEE International Symposium on*, pages 267–276, Nov 2003.

[4] K. Banerjee and M. Ionescu. Emerging nanoelectronics: Life with and after cmos. In *Microelectronics Education*, pages 159–160. Springer, 2004.

[5] C. Stroud. *A Designer's Guide to Built-in Self-test*, volume 19. Springer Science & Business Media, 2002.

[6] Y. Sato, S. Hamada, T. Maeda, A. Takatori, Y. Nozuyama, and S. Kajihara. Invisible delay quality - sdqm model lights up what could not be seen. In *IEEE International Conference on Test, 2005.*, pages 9 pp.–1210, Nov 2005.

[7] M. Bushnell and V. Agrawal. *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*, volume 17. Springer Science & Business Media, New York, 2000.

[8] Miron Abramovici, Melvin A. Breuer, and Arthur D. Friedman. *Index*, pages 647–652. Wiley-IEEE Press, 1990.

[9] Xijiiang Lin, R. Press, J. Rajski, P. Reuter, T. Rinderknecht, B. Swanson, and N. Tamarapalli. High-frequency, at-speed scan testing. *IEEE Design Test of Computers*, 20(5):17–25, Sept 2003.

[10] N. Ahmed, C. P. Ravikumar, M. Tehranipoor, and J. Plusquellic. At-speed transition fault testing with low speed scan enable. In *23rd IEEE VLSI Test Symposium (VTS'05)*, pages 42–47, May 2005.

[11] S. Eichenberger, J. Geuzebroek, C. Hora, B. Kruseman, and A. Majhi. Towards a world without test escapes: The use of volume diagnosis to improve test quality. In *2008 IEEE International Test Conference*, pages 1–10, Oct 2008.

[12] F. Hapke, M. Reese, J. Rivers, A. Over, V. Ravikumar, W. Redemund, A. Glowatz, J. Schloeffel, and J. Rajski. Cell-aware production test results from a 32-nm notebook processor. In *2012 IEEE International Test Conference*, pages 1–9, Nov 2012.

[13] F. Hapke, W. Redemund, A. Glowatz, J. Rajski, M. Reese, M. Hustava, M. Keim, J. Schloeffel, and A. Fast. Cell-aware test. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(9):1396–1409, Sept 2014.

[14] ITRS. 2013.

[15] M. Valka, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, E. Sanchez, M. De Carvalho, and M. S. Reorda. A functional power evaluation flow for defining test power limits during at-speed delay testing. In *2011 Sixteenth IEEE European Test Symposium*, pages 153–158, May 2011.

[16] J. Saxena, K. M. Butler, V. B. Jayaram, S. Kundu, N. V. Arvind, P. Sreeprakash, and M. Hachinger. A case study of ir-drop in structured at-speed testing. In *Test Conference, 2003. Proceedings. ITC 2003. International*, volume 1, pages 1098–1104, Sept 2003.

[17] P. Bernardi, M. De Carvalho, E. Sanchez, MS. Reorda, A. Bosio, L. Dilillo, M. Valka, and P. Girard. Fast power evaluation for effective generation of test programs maximizing peak power consumption. *Journal of Low Power Electronics*, 9(2):253–263, 2013.

[18] Microgp.

[19] E. Sanchez, M. Schillaci, and G. Squillero. Evolutionary optimization: the μgp toolkit: The ugp toolkit. New York, 2011. Springer-Verlag.

[20] F. Corno, E. Sanchez, M. S. Reorda, and G. Squillero. Automatic test generation for verifying microprocessors. *IEEE Potentials*, 24(1):34–37, Feb 2005.

[21] J. Bergeron. *Writing testbenches: functional verification of HDL models*. Springer Science & Business Media, New York, 2012.

[22] MentorGraphics. ModelSim.

[23] F. Fallah, S. Devadas, and K. Keutzer. Occom-efficient computation of observability-based code coverage metrics for functional verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(8):1003–1015, Aug 2001.

[24] Synopsys. Inc. TetraMAX, User Guide, 2013.

[25] M. Psarakis, D. Gizopoulos, E. Sanchez, and M. Sonza Reorda. Microprocessor software-based self-testing. *IEEE Design Test of Computers*, 27(3):4–19, May 2010.

[26] A. Paschalis and D. Gizopoulos. Effective software-based self-test strategies for on-line periodic testing of embedded processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(1):88–99, Jan 2005.

[27] E. Sanchez, M. Schillaci, and G. Squillero. Enhanced test program compaction using genetic programming. In *2006 IEEE International Conference on Evolutionary Computation*, pages 865–870, 2006.

[28] M. Gaudesi, M. S. Reorda, and I. Pomeranz. On test program compaction. In *2015 20th IEEE European Test Symposium (ETS)*, pages 1–6, May 2015.

[29] R. Cantoro, M. Gaudesi, E. Sanchez, P. Schiavone, and G. Squillero. An evolutionary approach for test program compaction. In *2015 16th Latin-American Test Symposium (LATS)*, pages 1–6, March 2015.

[30] D. Appello, V. Tancorre, P. Bernardi, M. Grosso, M. Rebaudengo, and M. S. Reorda. On the automation of the test flow of complex socs. In *24th IEEE VLSI Test Symposium*, pages 6 pp.–171, April 2006.

[31] opencores.org miniMIPS, Overview. 2009.

[32] E. Sanchez, M. Schillaci, and G. Squillero. *Evolutionary Optimization: the μGP toolkit: The UGP Toolkit*. Springer-Verlag, New York, 2011.

[33] P. Girard, N. Nicolici, and X. Wen. *Power-aware testing and test strategies for low power devices*. Springer, 2009.

[34] H. Fang, K. Chakrabarty, A. Jas, S. Patil, and C. Tirumurti. Rt-level deviation-based grading of functional test sequences. In *2009 27th IEEE VLSI Test Symposium*, pages 264–269, May 2009.

[35] A. Sanyal, K. Chakrabarty, M. Yilmaz, and H. Fujiwara. Rt-level design-for-testability and expansion of functional test sequences for enhanced defect coverage. In *2010 IEEE International Test Conference*, pages 1–10, Nov 2010.

[36] I. Pomeranz. Concatenation of functional test subsequences for improved fault coverage and reduced test length. *IEEE Transactions on Computers*, 61(6):899–904, June 2012.

[37] PrimePOWER Synopsys Inc. User guide. 2013.

[38] F. Wu, L. Dilillo, A. Bosio, P. Girard, S. Pravossoudovitch, A. Virazel, J. Ma, W. Zhao, M. Tehranipoor, and X. Wen. Analysis of power consumption and transition fault coverage for los and loc testing schemes. In *Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2010 IEEE 13th International Symposium on*, pages 376–381, April 2010.

[39] I. Park and E. J. McCluskey. Launch-on-shift-capture transition tests. In *2008 IEEE International Test Conference*, pages 1–9, Oct 2008.

[40] A. Bosio, L. Dilillo, P. Girard, A. Todri, A. Virazel, K. Miyase, and X. Wen. Power-aware test pattern generation for at-speed los testing. In *2011 Asian Test Symposium*, pages 506–510, Nov 2011.

[41] B. Kruseman, A. Majhi, C. Hora, S. Eichenberger, and J. Meirlevede. Systematic defects in deep sub-micron technologies. In *Test Conference, 2004. Proceedings. ITC 2004. International*, pages 290–299, Oct 2004.

[42] A. Bosio, L. Dilillo, P. Girard, A. Todri-Sanial, A. Virazel, S. Bernabovi, and P. Bernardi. An intra-cell defect grading tool. In *Design and Diagnostics of Electronic Circuits Systems, 17th International Symposium on*, pages 298–301, April 2014.

[43] S. Gai, M. Mezzalama, and P. Prinetto. A review of fault models for lsi/vlsi devices. *Software Microsystems*, 2(2):44–53, April 1983.

[44] M. Geilert, J. Alt, and M. Zimmermann. On the efficiency of the transition fault model for delay faults. In *Computer-Aided Design, 1990. ICCAD-90. Digest of Technical Papers., 1990 IEEE International Conference on*, pages 272–275, Nov 1990.

[45] K. C. Y. Mei. Bridging and stuck-at faults. *IEEE Transactions on Computers*, C-23(7):720–727, July 1974.

[46] S. R. Makar and E. J. McCluskey. Functional tests for scan chain latches. In *Test Conference, 1995. Proceedings., International*, pages 606–615, Oct 1995.

[47] H. Tang, B. Benware, M. Reese, J. Caroselli, T. Herrmann, F. Hapke, R. Tao, W. T. Cheng, and M. Sharma. Diagnosing cell internal defects using analog simulation-based fault models. In *2014 IEEE 23rd Asian Test Symposium*, pages 318–323, Nov 2014.

[48] F. Hapke, R. Krenz-Baath, A. Glowatz, J. Schloeffel, H. Hashempour, S. Eichenberger, C. Hora, and D. Adolfsson. Defect-oriented cell-aware atpg and fault simulation for industrial cell libraries and designs. In *2009 International Test Conference*, pages 1–10, Nov 2009.

[49] James Chien-Mo Li and E. J. McCluskey. Diagnosis of resistive-open and stuck-open defects in digital cmos ics. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(11):1748–1759, Nov 2005.

[50] A. Bosio, P. Girard, S. Pravossoudovich, P. Bernardi, and M. S. Reorda. An efficient fault simulation technique for transition faults in non-scan sequential circuits. In *Design and Diagnostics of Electronic Circuits Systems, 2009. DDECS '09. 12th International Symposium on*, pages 50–55, April 2009.

[51] S. Bernabovi, A. Bosio, P. Bernardi, and M S. Reorda. Defect-based test: Simulation and diagnosis. pages 19–37, 2014.

[52] MentorGraphics. Tessent Scan and ATPG User's Manual.