Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (37^{th}cycle)

Towards Ultra-Reliable Automotive Systems-on-Chip

By

Giusy Iaria

******

Supervisor(s):
Prof. Paolo Bernardi, Supervisor

Doctoral Examination Committee:
Prof. Giorgio Di Natale, *Referee*, TIMA – Université Grenoble-Alpes, France
Prof. Angeliki Kritikakou, *Referee*, Rennes University, France
Prof. Zebo Peng, Linköping University, Sweden
Prof. Paolo Rech, Università di Trento, Italy
Prof. Massimo Violante, Politecnico di Torino, Italy

Politecnico di Torino
2025

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Giusy Iaria
2025

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

To my parents, my sister, my brother and little Nebbi

Acknowledgements

All the research projects carried out during this thesis have been supported by the Italian Ministry of Research and University under the DM1061.

I want to thank the people who supported me throughout my PhD journey.

I thank my supervisor, Professor Paolo Bernardi, who has always been *Paolo* to me, for trusting me from day *zero*. Thank you for all the valuable advice, opportunities, and availability. You have been a true mentor in guiding my research and academic life, giving me the right amount of freedom and independence.

I thank the entire CAD & Reliability research group for welcoming me and keeping me company every step of the way, day in and day out. Thanks to Andrea, without whom this PhD journey would never have started, for introducing me to the research group. Thank you to Francesco and Giorgio, companions in cycles, lectures, misadventures, and laughter. Thanks to Gabriele for all the chit-chat breaks from work, shared registration desks, and for making every research project we tackled together fun. Thanks to Nicolò for the shared volunteer collaborations, in which we had the opportunity to give space to our *non-technical* hobbies. Thanks to Anna, for sharing *not-so-bad* motels, giving wise advice, and for the best seat in the lab.

I thank Claudia Bertani's team, Claudia, Giuseppe, and Vincenzo, who welcomed me at STMicroelectronics and gave further meaning to my academic journey. Thanks to Vincenzo Tancorre for being a caring and helpful tutor.

Finally, I thank my family with little Nebbi, Clara, and all my friends, for their constant presence and affection, thanks to whom I went through all the moments of the journey peacefully. Thanks to mom, dad, Cetty, Davide, and Nebbi. Thanks to Clara for always being there and making everything straightforward. Thank you for accompanying me on this journey.

Abstract

Nowadays, the automotive industry has witnessed an exponential rise in the number and intricacy of Automotive Systems-on-Chip (SoCs). Proper device testing is crucial to avoiding life-threatening incidents. For this reason, devices must guarantee high reliability, undergoing several testing phases before their sale. However, their increasing complexity poses significant challenges for companies, especially in managing costs and returns from the field.

This thesis has two main research branches: the first focuses on the manufacturing testing phases (pre-sale), while the second focuses on the in-field phase (post-sale) of the device.

For the first research branch, the thesis makes multiple contributions to different phases of manufacturing testing. It first proposes a scan-based test cost model that takes the manufacturing yield and the costs of wafer sort, packaging, and package test as input. The final goal is to address the increasing complexity of devices, which require ever-increasing test costs. Such a model can estimate how many patterns can be cut from the tail of the pattern set during Wafer Sort, thus shifting right some test escapes to the Package Test, while still achieving an overall economic gain. Also, assuming that large devices have non-uniform failure distribution, the proposed methodology analyzes the device layout and a sacrificial lot of devices to extract each fault's layout characteristics and criticality levels. As a further by-product, the thesis shows how generating patterns following the criticalities extracted from the data analysis of sacrificial lots leads to even more significant economic gain.

In addition to scan-based testing, the manufacturing process includes a phase called System-Level Test (SLT), particularly for devices intended for safety-critical fields. The SLT is introduced just before the Final Test. It aims to cover the faults left uncovered by the structural tests of the previous phases. Precisely, it reproduces the system environment of the device as if it were in its final operation, mainly testing

the interconnections between the various modules. The generation of programs for the SLT is done holistically, and evaluating the goodness of these programs is not an easy task, given the extended time of functional fault simulations for very large devices. This thesis proposes a categorization of faults left uncovered by structural testing based on a preliminary analysis of the circuit and the trade-offs made by commercial tools to generate scan-based patterns automatically. This categorization allows for the evaluation and validation of holistically generated SLT programs by verifying their effectiveness in covering those fault categories that are difficult to cover when generating scan-based patterns for the whole circuit. The thesis also explores the application of pseudo-random-generated stress stimuli during the SLT through a low-cost tester architecture.

For the second research branch, the thesis focuses on device life in the field. Considering the problems of returns from the field faced by manufacturing companies, this thesis proposes a logical diagnosis method based on an in-field data collection phase. During its operational life, the device performs key-on and key-off self-tests at different frequencies. The results of these tests are collected within the device's nonvolatile memory. With such collected information, if the device malfunctions and returns to the manufacturing company, it is possible to start from the collected data to more accurately diagnose the failure. The methodology proposed in this thesis is highly adaptable to industrial realities, as it does not involve specific architectures but takes advantage of the self-test modules already present in devices intended for safety-critical fields.

All proposed methodologies are validated using industrial automotive devices produced by STMicroelectronics, as well as production data and failed devices provided by the company.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent decades, the rapid evolution of automotive and industrial electronics has been driven by an exponential increase in the complexity and functionality of System-on-Chip (SoC) devices [1]. In automotive applications, where SoCs play a key role in ensuring safety and reliability, this complexity is particularly pronounced due to the stringent requirements of standards such as ISO 26262 [2]. Modern automotive SoCs integrate millions, or even billions, of transistors into a single chip, enabling advanced features such as autonomous driving and real-time decision-making. However, this unprecedented scale and sophistication pose significant challenges to ensuring robust and cost-effective testing throughout the lifecycle of these devices.

Testing methodologies for SoCs must address two key issues: detection of safety-critical failures to avoid life-threatening accidents and monitoring the reliability of such devices during their mission mode. These challenges are particularly pronounced in the automotive industry, where failures in electronic components can lead to catastrophic consequences. The mentioned two issues are the main research branches of the thesis, that focuses both on *manufacturing testing* and *in-field data collection and diagnosis*, with the shared goal of optimizing and addressing the state-of-the-art issues regarding the reliability of very large SoCs.

The manufacturing test process typically involves wafer selection and package testing. For automotive devices, additional steps are needed to meet safety requirements. Each step requires a significant allocation of resources, including test time, equipment, and computational [3]. Most of the overall cost of testing comes from the Wafer Sort phase, in which dies are tested before packaging. The subsequent

Package Test reapplies the same patterns to the packaged devices, ensuring they meet quality standards. However, the increasing complexity of SoCs has led to more test patterns, which require more time and resources, thus significantly increasing testing costs.

Early detection of defective devices during Wafer Sort is critical to address these challenges, as it reduces the number of defective devices advancing in the manufacturing pipeline. However, achieving high fault coverage at the primary stage is resource-intensive due to the increasing pattern set size of the Automatic Test Pattern Generator (ATPG) and the limitations of the Automatic Test Equipment (ATE) [4]. Many studies have attempted to solve the problem of the increasing number of tests to be applied. For example, in order to reduce the testing time, the solution proposed by several studies [5, 6] is reordering the test patterns to first apply the most effective ones. In this way, considering a Stop-on-Fail test flow, the testing time would be reduced in case of failure. Adapting these studies to large industrial chips costs lot of computational efforts, and reordering patterns according to their effectiveness could not always lead to a significant economic gain. In particular, if the used testing environment is multi-site [7], the gain is lower because multiple dies are tested in parallel, and good and failed devices could be tested simultaneously, deleting the eventual economic advantage of detecting *in less time* the failed ones. Other studies [8–10] have investigated a way to adaptively predict the number of useless patterns, thus maintaining only the useful ones or privileging the best ones to be executed earlier. Such approaches can be adopted also in multi-site environments.

To address these limitations, this thesis aims to model the scan test costs to finally optimize the whole manufacturing test costs. The proposed scan test cost model permits the computation of the economic trends that can be obtained, in different conditions of Yield, Fault Coverage, and test equipment costs per second, by considering cutting patterns from the tail of the complete pattern set as a *straightforward* alternative to pattern selection or reordering at Wafer Sort. It is shown that, in normal productive conditions, it is easy to gain in the overall scan cost by removing the latest patterns at Wafer Sort and accepting more expenses for the Package test based on the fact that ATPG tools, at the end of the generation process, always cover fewer and fewer faults than the patterns before [10].

The model uses the Weighted Fault Coverage (Ω) to account for devices presenting non-uniform failure distributions, with some areas of the circuit having higher

fault densities due to systematic or random faults. The study analyzes the correlation between logical faults identified during testing and their physical layout characteristics. This refinement ensures that the prediction is as much accurate as possible. Differently from other approaches [9, 10] that require continuous production monitoring to implement adaptiveness to a specific defectivity, one-shot computation of Weighted Fault Coverage Ω is performed over a sacrificial population of dies which is tested according to a Continue-on-Fail strategy, while Stop-on-Fail is used to the volume of produced dies. As a further by-product, a pattern generation methodology that exploits the criticality levels computed of the device is proposed to achieve higher values of economic gain.

Despite the various stages of manufacturing testing to which devices are subjected, some likely failures are left uncovered. Ensuring high fault coverage through structural testing has become a challenge for very large and complex devices. Scan-based tests, in fact, could generate test escapes [11, 12], i.e., faults not captured by the structural pattern set may affect devices shipped to market and arise along the devices' mission mode. The System-Level Test (SLT) phase has been introduced as a complementary step in the manufacturing test flow, just before the Final Test phase, to cover further some device functionality not adequately covered by structural testing. A common holistic strategy for verifying the correct behavior of a device is to boot an Operating system [13], that exercises the interaction among the various modules of the device. However, grading functional programs is not an easy task. To the sake of properly grading an SLT suite written by following a holistic philosophy, the thesis proposes a methodology to perform a thorough algorithmic analysis of the residual fault list after structural patterns coverage, i.e., faults not detected by structural tests. Such information turns out to be crucial to validate the effectiveness of the SLT programs.

Along the System-Level Test phase it is also possible to apply stress stimuli through a low cost tester architecture [14], to exacerbate possible latent defects as in the Burn-In phase. The thesis also explores and presents experimental results regarding the application of pseudo-random generated stress stimuli on very complex automotive devices.

In addition to the manufacturing testing phase, which is the first research branch of the thesis, ensuring the reliability of SoCs during their operational lifecycle, particularly in safety-critical automotive applications, is an additional challenge

on which the thesis focused as the second research branch. Field testing, which monitors the health of devices after installation, has become increasingly important to identify issues such as aging-induced failures and reduce the risks associated with unexpected failures [15, 16]. Logic Built-In Self-Test (LBIST) engines [17, 18], integrated into SoCs for self-diagnostic purposes, play a crucial role in this context. LBIST engines generate and apply test patterns directly on the device, analyzing the results to detect faults. However, the configurational flexibility of LBIST engines is often significantly limited during field operation to avoid overstressing components and inducing premature degradation. These constraints, such as the fixed Linear Feedback Shift Register (LFSR) seeds and limited reprogramming capabilities, complicate fault diagnosis in the field. Consequently, identifying the root cause of failures in returned devices, particularly in No Trouble Found (NTF) [19] scenarios where the misbehavior cannot be replicated in controlled environments, remains an urgent problem for manufacturers.

In the state-of-the-art, numerous methodologies [20–25] have emerged for diagnosing failures detected by LBIST. In [21], the authors suggest strategies that integrate in-field self-test execution with the diagnosis process, though their effectiveness is contingent upon custom LBIST architectures. Furthermore, a more recent work [26] introduces a specific LBIST framework that improves the conventional ATPG diagnosis process by leveraging the on-chip pattern generation technique outlined in [27]. Additionally, there are examples [24, 25] that propose logic diagnosis strategies relying solely on the analysis of failing LBIST signatures without any diagnostic features. In a similar vein, but with greater efficiency, the suggested logic diagnosis methodology offers a means to diagnose LBIST failures captured in-field, all without necessitating architectural modifications or specialized on-chip pattern generation.

The thesis proposes a comprehensive approach for logic diagnosis of field returns using constrained LBIST engines in order to overcome the above stated challenges. By identifying the first failing pattern during LBIST runs, the logic diagnosis strategy dramatically reduces the number of candidate faults, therefore enabling successful field return diagnosis. This approach is particularly useful for automotive SoCs, when architectural changes are difficult to implement and limited LBIST setups are the norm. Moreover, the proposed solution offers data security and privacy first importance by storing diagnostic data direct on the device and so reducing the risks linked with data sharing on the cloud.

To sum-up, the key contributions of the thesis are the following:

- **Manufacturing Test Flow cost optimization**: a comprehensive cost model that combines yield, Wafer Sort, packaging, and Package Test costs. The model evolves known concepts and provides overall cost savings by moving some test escapes to the Package Test phase while maintaining the same accuracy at the end of the production flow. Assuming that mass production defect is unevenly distributed across the die population, a methodology that cross-references physical layout information with sacrificial batch results is proposed. The information regarding the criticality of faults is exploited to perform better predictions. As a further by-product, a pattern generation methodology that exploits the categorizations of criticalities of the possible faults is presented, showing how such a newly generated pattern set can significantly maximize the achievable economic gain. Thus, the study comprises at least three innovative contributions:

  1. a **scan test cost model** to predict the ideal point to cut patterns from the tail of the pattern set for devices population presenting a non-uniform failure distribution. It elaborates on [28] and [29] formulas to also consider the mass defectivity distribution;
  2. a **data analysis method** to determine the criticality level of the possible faults affecting the device, which is obtained from the analysis of failures resulted from a sacrificial lot of devices tested with the Continue-on-Fail test flow;
  3. a **criticality-oriented ATPG algorithm** to generate patterns targeting the faults labeled as the most critical in the analysis phase, first.

- **System-Level Test programs assessment**: a methodology to classify faults uncovered by structural tests (structural residual faults) and to quantify the impact of DfT configuration and usage on the structural coverage; such a contribution permits further guidance in the SLT generation and minimizes grading efforts.

- **Application of pseudo-random stress stimuli to a monster advanced driver assistance systems (ADAS) SoC**: evaluating and generating pseudo-random stress stimuli to enhance the stress tests applied along System-Level Test phase

for a monster ADAS device produced by STMicroelectronics. This activity has been carried out during a one year period spent as intern in the company.

- **Logic diagnosis of field returns**: a comprehensive methodology that leverages constrained in-field LBIST engines for field data collection and subsequent logic diagnosis based on the collected data. The methodology significantly reduces the number of candidate faults by identifying the first failing pattern, enabling efficient diagnosis of field returns. This solution is practical for industrial devices as it requires no architectural changes. Thus, this study comprises two key contributions:

  1. A comprehensive methodology for **in-field data collection** of failure information **from constrained LBIST engines** activated along the mission mode of the device;
  2. A **logic diagnosis strategy** to facilitate failure analysis of field returns by **leveraging in-field failure data** collected on the chip.

The proposed approaches in the thesis aim to contribute to state-of-the-art testing methodologies for industrial automotive devices. In particular, the thesis addresses the problem of their increasing complexity, contributing to each phase of the devices' life cycle, in terms of cost and patterns optimization based on scan circuitry, System-Level testing, stress stimuli, and diagnostic efficiency. The effectiveness of the proposed approaches is validated through experimental results conducted on both academic benchmarks and industrial case studies, underscoring their potential for effectiveness and adaptability in the industrial world.

The manuscript continues as follows. Chapter 2 provides the reader with the background concepts needed for the total comprehension of the thesis and details the state-of-the-art methodologies for the mentioned problems. Chapter 3 details the proposed methodologies and main contributions. Chapter 4 reports the experimental results of the proposed methodologies applied on devices produced by STMicroelectronics. Chapter 5 draws some conclusions and possible future developments.

Chapter 2

Background

This chapter aims to provide readers with the concepts guiding the research project and the state-of-the-art related works with similar objectives.

2.1 Manufacturing Test Flow

The density of transistors inside a certain area [30] is clearly increasing as transistors continue to shrink. This major development has resulted in a substantial increase in the complexity of integrated circuits, therefore generating more complicated testing requirements [30]. Before being delivered to consumers, devices must pass many test processes to ensure their reliability and safety.

The Manufacturing Test Flow consists of several phases applied one after the other to discard manufactured faulty devices [4, 13]. Figure 2.1 shows a possible manufacturing test flow of automotive SoCs, targeting different fault sets in each phase.

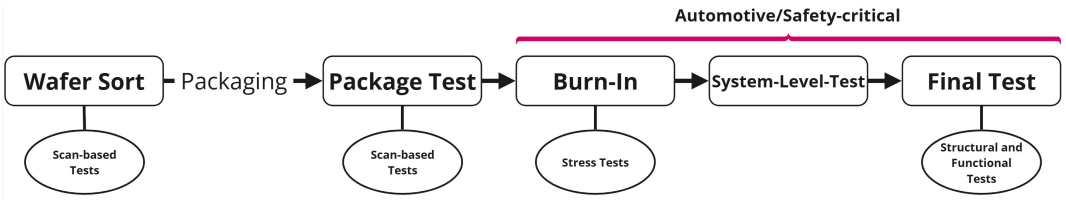


Fig. 2.1 Depiction of a possible Manufacturing Test Flow

The flow is company-dependent and could change depending on the device's field of application. The steps shown in Figure 2.1 are detailed below:

- Wafer Sort: it is performed at the Wafer-Level. It checks for the primary electrical functionalities of the chip by executing scan-based test patterns before the dies are cut.
- Package Test: it is performed at the Package Level. The dies targeted as *good* during the previous phase are cut, baked and packed. In order to ensure that such a process did not introduce defects, typically, the same pattern set composed of scan-based tests is applied again.
- Burn-In: it is performed primarily for automotive and safety-critical devices to exacerbate latent defects [31, 32]. This way, devices with latent defects can be successfully screened by later test steps after the Burn-In phase.
- System-Level Test: it has been added as an additional test for automotive and safety-critical devices [11] mainly to verify the peripherals and connections between the several units composing complex devices by the use of functional routines. It typically consists of running functional applications that emulate the in-field behavior, workload, and environment.
- Final Test: it is performed at the end of the process before the devices are shipped to the customers. It comprises a mix of scan-based and functional tests [33].

For each stage of testing, some devices will be discarded as defective. The weight of rejects on the total cost of the process depends on the stage at which they occur. The later failures are discovered, the more time and money is wasted on testing a defective device from the beginning. For this reason, the Wafer Sort phase is typically used as an initial screening to reduce the devices to be tested later. However, during the Package Test, companies typically apply the same pattern set immediately after the devices are packaged. For this reason, the following section will explore a case of reducing testing costs by considering a reduction of the pattern set during Wafer Sort.

2.2 Manufacturing Test Time reduction

A practical and impactful strategy for minimizing the costs associated with scan tests lies in reducing the test duration through the decrease in the number of patterns utilized during specific phases of the overall testing process. However, this reduction in patterns can also lead to an increase in test escapes in later stages. This paper emphasizes a scenario where the focus is on the Wafer Sort phase as the primary target for cuts in both time and cost; during the Package Test phase, the complete set of patterns is applied in its entirety, as is standard in production processes.

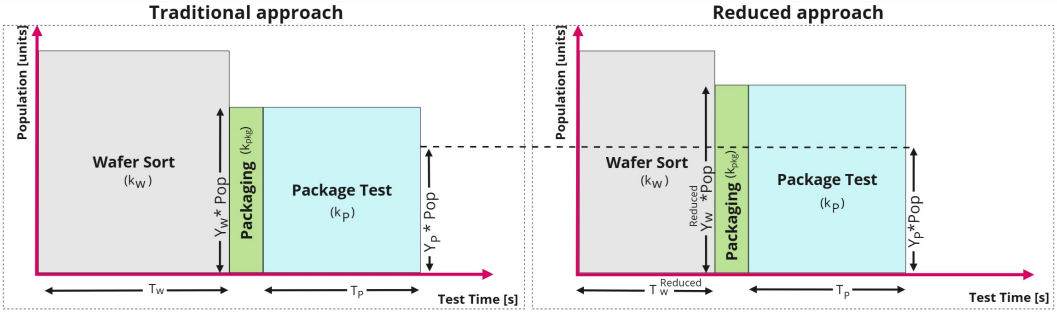


Fig. 2.2 Comparison at-a-glance between the Traditional Manufacturing Test Flow and the Reduced approach

Figure 2.2 visually illustrates the combined costs incurred during the Wafer Sort and Package Test phases, as well as packaging expenses. This is depicted through the area covered in the plot using two distinct methodologies. The cost associated with scan testing is influenced by three critical factors: the population size undergoing testing, the duration necessary for each phase of testing, and the fixed costs intrinsic to the process in question. The parameters k_W, k_{pkg}, and k_P denote the fixed costs associated with the Wafer Sort per second, packaging per die, and the Package Test per second, respectively. Conversely, T_W and T_P represent the required test duration for the Wafer Sort and Package Test phases.

On the right side of Figure 2.2, we find $T_W^{Reduced}$ and $Y_W^{Reduced}$, demonstrating the shortened duration of the Wafer Sort and the resulting change in yield. The calculations regarding these terms will be elaborated upon in the forthcoming formulas. To derive the cost associated with a single die, the following equations can be employed.

$$Cost_W^{Traditional} = k_W * T_W \tag{2.1}$$

$$Cost_P^{Traditional} = Y_W * (k_{pkg} + T_P * k_P) \tag{2.2}$$

The first formula serves to estimate the costs of the Wafer Sort scan test by multiplying the necessary testing time with a constant that corresponds to the testing equipment expenses. The second formula multiplies two key terms: Y_W, which indicates the yield from the first phase of testing, and the sum of the packaging cost (k_{pkg}) along with the product of the Package Test duration (T_P) and its associated operating cost (k_P). By multiplying the total of these two costs by the size of the initial population (Pop), we obtain the overall cost for the entire population.

$$Cost_{Tot}^{Traditional} = Pop * (Cost_W^{Traditional} + Cost_P^{Traditional}) \tag{2.3}$$

To model the reduction in Wafer Sort costs, we can utilize the following equation, which also represents the latter half of Figure 2.2:

$$Cost_W^{Reduced} = k_W * T_W^{Reduced} \tag{2.4}$$

In this equation, the time factor for calculating $Cost_W^{Traditional}$ is substituted with a generalized *reduced time*. In pursuit of this time reduction, specific applied patterns are eliminated, focusing on discarding the later patterns. Previous works have addressed the implications of these patterns on the test's efficacy in detecting faulty chips [10].

As a result of reducing the total number of patterns, the precision of the testing might diminish, subsequently increasing the yield because certain defective devices will no longer be subjected to screening. The new yield, denoted by $Y_W^{Reduced}$, combines the yield resulting from the classical method (Y_W) with an additional term that accounts for the test escapes (TE) that manifest as a consequence of shortening the Wafer Sort phase. This latter component can be approximated using the established formula [28], where FC denotes the function illustrating the relationship between fault coverage and the number of applied patterns (x).

$$TE = 1 - (Y_W)^{1-(FC(x)/100)} \tag{2.5}$$

$$\begin{aligned} Y_W^{Reduced} &= Y_W + TE \\ &= Y_W + 1 - (Y_W)^{1-(FC(x)/100)} \end{aligned} \tag{2.6}$$

Incorporating this new yield into the Package Test cost equation prompts a change in its formulation, as it follows:

$$Cost_P^{Reduced} = Y_W^{Reduced} * (k_{pkg} + T_P * k_P) \tag{2.7}$$

$$Cost_{Tot}^{Reduced} = Pop * (Cost_W^{Reduced} + Cost_P^{Reduced}) \tag{2.8}$$

The overall cost, represented in Equation 2.8, of the revised testing method directly correlates with the reduced Wafer Sort duration ($T_W^{reduced}$) and the adjusted yield ($Y_W^{Reduced}$). Hence, once the fixed constants of the testing environments (k_W, k_{pkg}, and k_P) are established, Equation 2.8 effectively models the cost behavior influenced by the time reductions and the subsequent escalation of test escapes, as delineated in Equation 2.6.

Notably, the cost associated with $Cost_W^{Reduced}$ escalates as the testing duration increases, while the cost for $Cost_P^{Reduced}$ diminishes with a reduced yield from the initial phase. As a result, a minimum point within the cost function can be identified.

Example 1 ■

This example delves into the consideration of an academic benchmark, which features a cluster of 24 OpenRISC1200 processors [34]. This particular configuration presents approximately 2 million Stuck-at faults and another 2 million Transition delay faults.

An exhaustive Automatic Test Pattern Generation (ATPG) process was conducted to cover all these faults. The flow generated a set of over $30,000$ unique patterns with 99% fault coverage. Figure 2.3 shows the relationship between fault coverage and the number of patterns, also offering the trend's view.

Furthermore shown are the expected expenses connected with these testing techniques by Figure 2.4 and Figure 2.5. Based on the above specified algorithms, these estimations include different values for Wafer Sort, Package Test expenses, and yield. We investigate two particular yield scenarios: a low yield of 90% and a high yield of 99%.

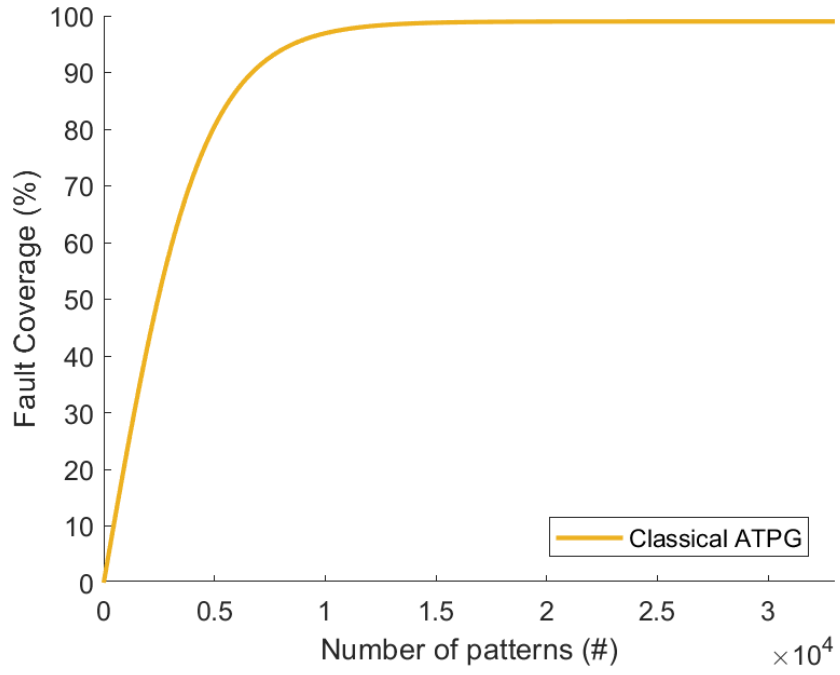


Fig. 2.3 Fault Coverage vs Percentage of applied patterns

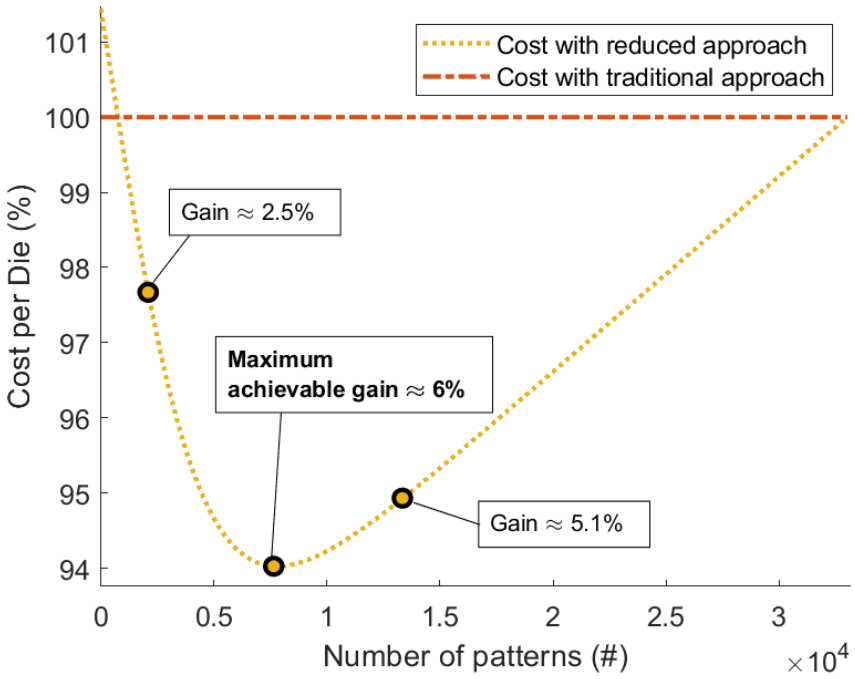


Fig. 2.4 Low yield (90%), $k_W/k_P \approx 0.15$

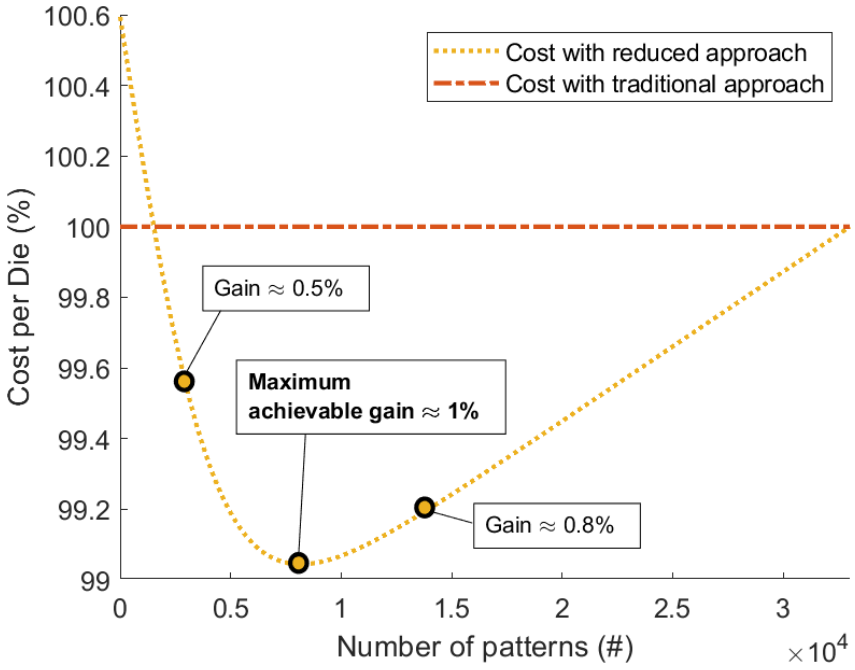


Fig. 2.5 High yield (99%), $k_W/k_P \approx 0.018$

However, if we were to explore even lower yield values, such as 80%, the potential economic benefits derived from the testing process could be even more pronounced. In fact, under these circumstances, the maximum achievable economic gain is approximately 6.3% with an 80% yield. It is worth noting, though, that observing such low yield percentages is relatively uncommon in well-established production environments, except during temporary crises that may arise. Therefore, managing low yield scenarios should primarily be approached from a technological standpoint rather than predominantly a testing perspective, which aligns with the central theme of the proposed model.

Indeed, the cost model finds a minimum where the increase in test escapes is negligible compared to the savings from reducing the number of patterns along Wafer Sort. With lower yields, the number of devices reaching the Package Test is lower, and, as a consequence, the pattern cut affects the overall cost by a more significant percentage of the economic gain. It should be noted that this behavior is explained by the fact that the cost model is only related to the scan-based test cost and does not refer to production costs before the Wafer Sort.

Furthermore, varying fixed cost values for Wafer Sort and Package Test are considered, incorporating two distinct ratios (k_W/k_P): 0.15 and 0.018. The packag-

ing cost (k_{pkg}) remains constant across both scenarios. The precise location of the minimum point is influenced by the values assigned to these costs as well as the yield levels. A higher value of k_W means that the Wafer Sort cost exerts more influence, and it causes the slope of the curve to steepen significantly beyond the minimum, owing to an increased duration for this process.

On the other hand, when the combined cost of $k_{pkg} + k_P$ is significant, the decrease in cost in the initial segment of the curve is more noticeable since less defective devices pass the Wafer Sort process. Figures Figure 2.4 and Figure 2.5 show how changes in the above specified values affect the cost dynamics.

One can determine the highest possible gain for both situations by using the described approach. The maximum gain—which corresponds to the smallest point of the curve—hovers about 6% for Figure 2.4. By contrast, for Figure 2.5, the gain comes to about 1%. Although these numbers might first seem to be similar, at the moment where the curve crosses its knee they actually indicate somewhat different possible cost savings.

Specifically, Figure 2.4 indicates a higher Wafer Sort cost, and if reductions were made to this cost, significant financial savings could be realized. On the other hand, Figure 2.5 features a lower Wafer Sort cost, which does not exhibit the same level of sensitivity or yielding improvements.

In addition, it is of crucial significance to emphasize that high yields make it possible to significantly cut down on the number of patterns before the expenses that are associated with defective chips escaping from the Wafer Sort have a negative impact on the overall costs of packing and package testing. The possible economic gain of the curve increases with its steeper slope. Apart from this minimum, two additional important spots are shown close by, roughly $5,000$ patterns apart. For example, the gain reduces to roughly 2.5% if the pattern cut in Figure 2.4 is too severe. On the other hand, should the cut be too mild, the gain just marginally drops to 5.1%. In the other situation (Figure 2.5, the response is identical albeit the values are quite lower. Maximizing the possible economic advantage thus depends on precisely determining the ideal point of cutting patterns.

End of the example 1. ■

2.3 Stress stimuli evaluation

In addition to the classical scan-based patterns, automotive devices undergo further testing steps, including stress tests to exacerbate latent defects.

Determining how effectively a device has been stressed during testing can be challenging. The availability of effective metrics to measure stress in a DUT is crucial to selecting and improving the stimuli applied to the device under test during the Burn-In phase.

To deal with evaluating the stress capabilities of pattern sequences, researchers have proposed various methods [35–37]. The study [35] introduced a new metric for evaluating stress during Burn-In testing, including layout analysis. Authors in [32] developed a tool-chain to analyze the actual activity of applied stress patterns, rather than relying on estimates from commercial tools, and, in addition, [14] compared stress stimuli generated by different techniques and proposed a low-cost tester based on a Microcontroller Unit (MCU) for applying stress during various test stages, including Burn-In and System-Level Test.

A stress pattern should be able to activate all circuit nodes, which means that the pattern set needs to force logic values of *0* and *1* to be applied to all gates and nets of the device. A stress pattern unable to toggle all nodes cannot exacerbate all probable manufacturing latent defects, such weak metallization or oxide problems. Hence, toggle coverage is the main indicator used to evaluate stress exposure during test pattern running. Toggle coverage measures the fraction of logic gates in a circuit that at least once go from a logic state of *0* to *1* and back to *0*.

Although toggle coverage offers a broad sense of stress application, the *toggle activity* is a more detailed estimation. Toggle activity probes farther, precisely counting the logic transitions across each gate in the circuit.

Another stress measure is the so-called *toggle activity*, which counts the number of toggle events per gate in the circuit. Uniformly distributing the toggle activity is also important because it ensures all nodes are similarly stressed.

2.4 Trade-offs of scan-based testing in large SoC

Ensuring that large System-on-Chips (SoCs) achieve high fault coverage depends critically on scan-based testing. But this testing approach presents various difficulties, especially in view of the growing complexity related to modern design methods.

Often needing multi-model incremental generations [38] to satisfy the strict fault coverage criteria mandated for devices running in safety-critical sectors, the strategies used for Automatic Test Pattern Generation (ATPG) in the context of scan-based testing can be notably resource-intensive.

Although ATPG's main purpose is to achieve complete fault coverage, there is a drawback to this aim: many created test patterns can be overly broad. Such patterns could unintentionally set the circuit in ways that do not reflect real operational conditions of the device during its intended use [39]. This phenomenon, referred to as over-testing, can lead to increased test times and a greater consumption of computational resources [40]. Furthermore, in large-scale SoCs, it is not uncommon for certain faults to remain uncovered by the ATPG process [11], indicating that the scan-based pattern set may not always provide exhaustive coverage.

Another potential limitation of scan-based test patterns arises when logic gates are governed by scan flops located in one clock domain yet are observed by scan flops situated in a different clock domain. This issue is illustrated in Figure 2.6, where the highlighted gate in yellow is managed by scan flops from two distinct clock domains, in contrast to the clock domain of the observing scan flops. Faults associated with such configurations can be particularly challenging for ATPG to address in a singular test run. Therefore, multiple generations utilizing varying configurations or dedicated testing strategies may be required [41, 42].

Usually based on heuristic approaches, the algorithms used in ATPG computation differ greatly within the body of current literature [43]. The FAN Algorithm [44] is among the most often used ones since it focuses on the fan-out of gates to efficiently excite and spread faults. This approach aims to maximize the possibilities of finding faults in observable flip-flops (FFs) by using cascading effects over larger input/output logic cones.

For instance, as depicted in Figure 2.7, consider a fault present at the output of a gate (marked in yellow). The ATPG must first generate the necessary input values to effectively excite this fault, thereby producing specific values to load into FFs

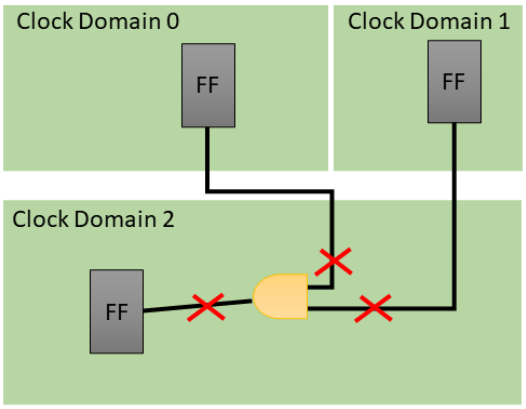


Fig. 2.6 Cross domain faults example.

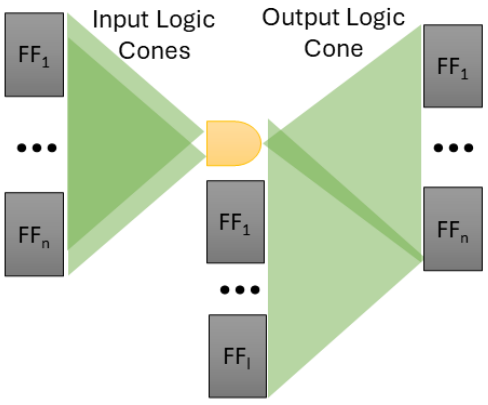


Fig. 2.7 ATPG fault excitement and propagation.

from 1 to n. The following step involves making sure the problem shows up in the FF located inside the output logic cone. To effectively spread the excited fault to those that can be seen, ATPG must hence set limitations on the values produced for different FFs. Usually, the output logic cone will converge on many FFs, therefore enabling a problem to be found as a cascade consequence of different test patterns. On the other hand, should the output logic cone result in a single observable FF, the ATPG is assigned to concentrate on designing a specific test pattern for the excitation and propagation of the fault. This frequently results in longer collection of test patterns devoted to a single, particular fault and more computing effort.

In recent decades, to mitigate the inherent weaknesses associated with scan-based testing, complementary strategies such as functional testing and System-Level Test have gained traction. These approaches work in tandem with scan-based testing, aiming to overcome its limitations while circumventing the need for netlist-based structural analyses. Instead, they rely on empirical observations of failures within faulty devices executing real-world applications during the production phase [13, 11, 12, 45].

2.5 Circuit topology

As automotive SoCs become more complex than precedent, topological information about cell placement plays an increasingly crucial role in developing and analyzing these devices.

The layout design phase creates the topology of the device once it is synthesized. Using the physical description of the transistors contained in a technology library in terms of the length, width, and height of the different transistors, the placement of the cells on the layout and the routing of the connections between the transistors take place to recreate the logic gate and, thus, the final functional behavior of the device. During the placement phase, high-density areas are typically generated, consisting of areas with high concentrations of cells, especially in modern devices with high complexity. Thus, an automotive system-on-chip does not generally have a uniform port distribution on its surface, especially in the case of very complex devices. Its central operating core includes denser areas and sparser parts where only a few gates are placed. In addition, large parts of the chip are dedicated to components such

as memories and the interconnection between these components and the central functional core.

Figure 2.8 shows an example of the layout for an Automotive SoC. The colors highlight the different density levels all over the circuit: parts with higher gate density are indicated by a lighter shade of green, while areas with fewer gates have been placed are represented by a darker shade of green.

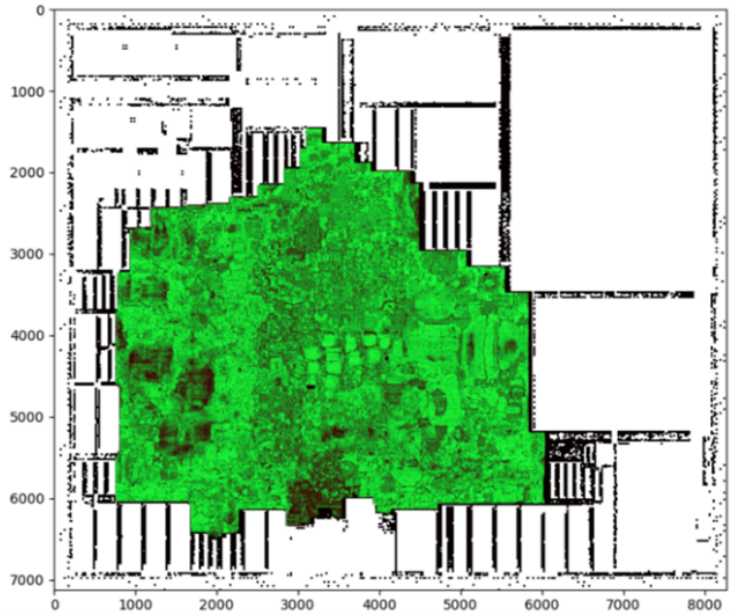


Fig. 2.8 Coloured gate-density distribution over the layout of a System-on-Chip.

The layout phase is critical in the production of modern Automotive Systems-on-Chip since it determines the circuit's topology, and it is essential to place cells correctly to avoid electromagnetic interference and defects. As the complexity of these systems increases, new approaches to analyzing layouts will become necessary to ensure that they are designed to the highest standards. Indeed, in recent years, researchers have proposed several approaches to analyze and exploit layout information.

The authors of [46–48] propose a new methodology that is based on extracting possible bridge faults from layout information. Precisely, the critical areas of all possible faults are calculated. The resulting list is then sorted according to the size of the critical area. This way, the list can be truncated after a certain threshold to prioritize bridge faults characterized by a larger critical area. The experimental results are reported on a 160*nm* automotive case study. The results show that patterns

generated by targeting bridging defects extracted in this way actually add coverage compared to patterns generated by traditional methods, even in the case of patterns generated by cell-aware. The study also presents analyses on the possible truncation of the list according to the ranking of defects while maintaining an increase over traditional coverage. Patterns generated by this methodology are applied in addition to traditional patterns, so the ultimate goal is to exploit the layout to improve total circuit coverage.

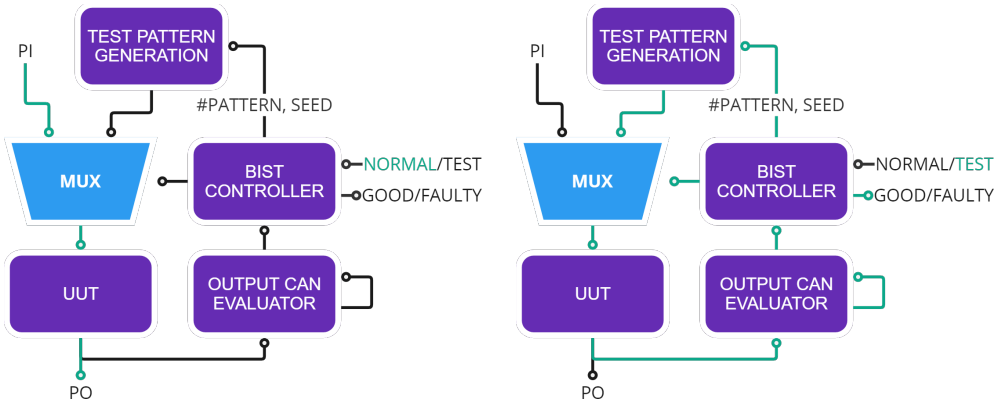
2.6 Built-In Self-Test

Very-Large-Scale-Integration (VLSI) technology has changed testing fundamentally. Boundary scan methods have been applied as a Design for Testability (DfT) approach targeted at identifying as many faults as feasible to address issues during the production phase. This approach stimulates the device under test (DUT) using specifically DfT hardware and scan chains. These structures' inputs are subjected to test patterns, whose ensuing outputs are tracked and matched against expected non-defective performance. However, the complexity of activating these structures necessitates the employment of automated testing equipment (ATE).

The continuous developments in VLSI technology have made the testing environment increasingly demanding. These novel developments make it challenging to keep up with the always increasing complexity of modern devices, which in turn complicates the testing process application. In safety-critical sectors like automotive, where strict reliability evaluation criteria are applied, this is especially true.

As operational frequencies, pin counts, and costs rise, traditional ATEs often prove impractical. To help to tackle these problems, manufacturers are including auxiliary modules into devices capable of doing internal self-tests, hence lowering dependency on outside equipment. Called Built-In Self-Test (BIST), this technology improves reliability and offers a more affordable substitute for expensive and sophisticated ATEs. Additionally, because BIST implementations can reuse scan chains that would otherwise be unneeded after production, testing can be done without disclosing important design knowledge, and they often require little additional hardware. Manufacturers must, however, offer instructions on how to do the tests and interpret the findings.

The unit under test (UUT) in the BIST framework can be configured to operate either functionally or test mode. Primary inputs (PI) and primary outputs (PO) are disconnected in test mode. Test patterns from a test pattern generator (TPG) are brought into the inputs of a BIST controller. The outputs link to an output-data-evaluator (ODE), which checks whether the produced signature matches the intended one. Typically, the ODE refreshes the signature following every BIST cycle using a Multiple Input Signature Register (MISR). This signature depends on the past test results as well as the current test outputs. Therefore, it is sufficient to identify the correct signature and verify it corresponds with what the ODE generates rather than comparing every output with its expected outcome on their own. This explanation relates to the basic STUMP architecture seen in Figure 2.9.



(a) Normal functional paths are enabled, PIs reach the UUT.

(b) Self-Test paths are enabled, the BIST Controller feeds test vectors to the UUT and capture its response in the ODE.

Fig. 2.9 Standard STUMPS BIST architecture composed of its major functional blocks.

LBIST refers to the application of BIST to test logic modules. Its implementation usually involves the following modules:

- LBIST Controller: it selects the test or functional mode and handles LBIST configuration;
- Pseudo Random Number Generator (PRNG): it generates patterns for the UUT;
- MISR: it collects the output signature obtained after executing the LBIST. The signature is updated at each LBIST cycle, hence it depends not only on the final test but also on the previous tests.

The LBIST Controller can be programmed with specific testing parameters reported below:

- Pattern-Count-Stop (PCS) indicates the number of patterns that the TPG generates;
- Initial seed of the PRNG for the pattern generation;
- Frequency of application, which can be controlled and configured from high to low, depending on the requirements.

After being programmed the LBIST starts the self-test by iterating the following:

1. LFSR generates a random number starting from the given seed;
2. Shift values in/out of the scan chain/s and update MISR;
3. if $\#shift < total\_shifts$ go to 1;
4. Apply pattern by pulsing capture clocks cycles;
5. if $\#pattern < PCS$ goto 1;
6. Self-Test ends and MISR can be observed and compared to golden value.

2.7 Logic Diagnosis

Minimizing yield loss during the production of integrated circuits is a primary focus within the modern semiconductor industry [49, 50]. Consequently, the relationship between defects and failures in digital circuits has emerged as a key research area.

Logic diagnosis plays a vital role in the testing and fault analysis of electronic circuits. This process focuses on identifying and isolating the faults that lead to malfunctions or failures in digital logic circuits. The objective is to accurately determine the root causes of these faults, enabling efficient repairs and enhancements for future designs. Typically, the process starts with the detection of unusual circuit behavior or failures during functional or manufacturing tests. Logic diagnosis involves systematically analyzing the circuit's responses to varying input stimuli,

often using test patterns that target potential faults. By meticulously comparing observed circuit outputs with the expected results, logic diagnosis techniques can help pinpoint faulty components responsible for the inconsistencies. The insights gained from logic diagnosis contribute to enhanced reliability and performance of digital circuits, promote effective debugging, and facilitate targeted repairs or improvements.

Still, logic diagnosis can be computationally difficult and memory-intensive. One of the main areas of research lately has been the construction of systems able to store all required data to isolate errors. Referred to as *fault dictionaries*, such structures help to identify a group of possible fault locations causing the exhibited behavior.

Studies on fault dictionaries have drawn a lot of attention [51, 52]. Many methods and tools have been studied to efficiently build these types of structures. The design of fault dictionaries is largely influenced by the number of faults that have to be considered. As the number of probable faults increases, designing a fault dictionary that is both computationally reasonable and memory-efficient gets harder.

Creating fault dictionaries presents still another difficult task in balancing the size of the dictionary with the precision of problem identification. The size of the fault dictionary usually rises as the accuracy of fault identification rises, which can lower computing complexity and memory needs in the fault diagnostic process. Thus, a careful balance has to be done between the dictionary size and the defect identification accuracy.

Boppana *et al.* [53] introduced a tree-like structure for representing faults. This approach is attractive because it preserves the fault classification derived from the pattern set. The set of faults responsible for the circuit's faulty behavior can be identified by starting at the root of the tree and moving down to a leaf node. Each leaf represents a set of faults (*equivalent class*) that exhibit the same faulty responses for all applied patterns. Figure 2.10 illustrates an example of this structure, assuming three patterns ($T1$, $T2$, and $T3$) and a fault universe represented by the letters from A to I.

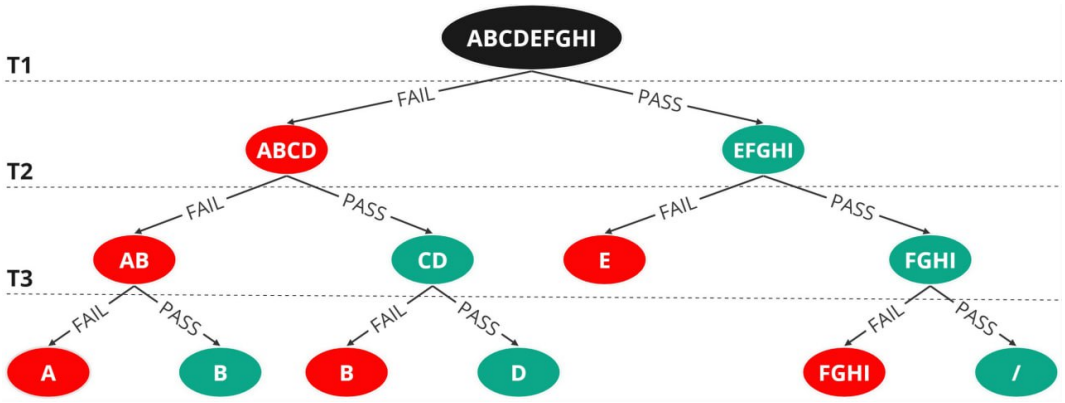


Fig. 2.10 Tree-based fault dictionary with an example of three patterns.

2.8 Silicon Lifecycle Management

For safety-critical devices, the testing flow does not end after the manufacturing process detailed in Section 2.1. Indeed, such devices should be continuously monitored throughout their whole lifecycle to ensure their reliability does not decrease. As a result, Silicon Lifecycle Management (SLM) has become a topic of great interest to industries in recent decades. SLM encompasses various phases, including the entire device lifecycle, from conception to end-customer use. Its main objective is to collect and analyze data during the device's operational life, thereby acquiring information that may prove important when used to improve its performance and reliability [15].

In SLM, field testing is critical to continuously test the reliability of devices during their operational phase and ensure that the device has not degraded due to use. Meeting the reliability requirements of different application domains is becoming increasingly critical, especially for devices used in safety-critical areas. Several solutions can be developed and implemented to address this issue. Regulations and standards such as ISO26262 [2] may impose specific failure coverage targets based on different levels of safety. Depending on the application context, field tests can be conducted during device power-up and power-down phases or application execution [54].

In particular, on-chip LBISTs (described in detail in Section 2.6) are exploited to detect device failures and enable SLM, as discussed in [55] and [56].

2.9 State-of-the-art about Automatic Test Pattern Generator and test cost reduction methodologies

Seeking to cover Device Under Test (DUT) faults as much as feasible, Automatic Test Pattern Generator (ATPG) tools are assigned to create patterns to test digital devices. But in terms of logic gates in contemporary automotive SoCs, increased complexity has also brought complexity to the test vector generating process. Even with regard to the terms of computation time [31].

Depending on the case study, the fault coverage acquired by creating a pattern by ATPG is typically incremental in the first few patterns, even rapid, in absence of specific generation direction. But concurrently, it often stabilizes or rises gradually following many test patterns. This is why an incremental technique is usually employed [57, 58] to split the circuit into small sections to target and to cover several fault models.

In light of the ever-increasing rise in testing costs, there are numerous examples of studies that have been published in literature that make an effort to reduce these costs using a variety of approaches. For instance, [8] aims at reducing the patterns to be applied for every core of the device during Wafer Sort. Combining statistical yield modeling with integer linear programming forms the basis of the chosen approach. The study seeks to ascertain the ideal number of patterns to apply for every device core in case wafer sort presents test time restrictions. Screening during wafer sort shows from experimental data that the suggested sorting method maintains excellent performance. Five of the ITC'02 benchmarks [59] are used as the case studies.

Reordering the patterns going forward the ones with greater failure rate as suggested in [5] is another way of reducing the test cost. The method discovers the ideal pattern sequence by means of an SVMRank method. The experimental results are obtained on simulations since the reported case studies are some benchmarks from [60].

The authors of [6] suggest an adaptive testing method to apply to the current die, based on the findings of the previous tested dies. Considering a multi-site wafer testing environment, this approach could be difficult to implement. ATEs with this type of capability is needed and the economic gain achievable is limitated to the

failed devices. The paper presents results of simulations for a benchmark derived from [60].

Furthermore, [10] introduced a methodology to estimate the count of patterns that could be actually beneficial during the testing phase. The study suggests a statistical methodology to identify the number of relevant patterns to apply and hence lower the ultimate test cost by removing the useless patterns. Though it depends on the number of tested devices, such a model is cost-independent. It is specifically based on the function $F_D(k)$, which is defined as the chance of spotting a faulty chip under application of patterns 1 through k. Production data helps to extend the function; the study indicates that based on the case study, it may be readily modified. The latter will be used in a comparative analysis with the proposed methodology to identify the ideal point of cutting the patterns (in Chapter 4).

Table 2.1 briefly shows the comparison between the described methods and the one proposed in the thesis.

The proposed methodology, detailed in Section 3.1 differs from the other ones, since it proposes a cost model for locating the optimal point to cut patterns from the tail, even in non-uniform failure distribution and considering the total cost of Wafer Sort and Package Test. Beyond that, the proposal introduces a novel pattern generating technique, as a by-product, such that the most critical faults are initially covered. Furthermore, the proposed approach reduces the amount of patterns for every evaluated device, including the good ones. Studies emphasizing reordering patterns, meanwhile, actually diminish patterns solely for the failing devices.

2.10 State-of-the-art about Logic diagnosis for LBIST caught failures

LBIST-based logic diagnosis is a hot topic given the increasing use of in-field testing methodologies, with many papers presented in the literature.

Elm *et al.*[21] introduced a built-in schema-based architecture called Built-In Self-Diagnosis (BISD). This architecture leverages BIST to collect data and exploit it for field diagnosis. Specifically, after collecting fault signatures in a dedicated memory, using the proposed methodology for diagnosis results in a final

Table 2.1 Test Cost Reduction methodologies

Method	**Brief strategy and goal**	**Reported case of study**
A test-application-count based learning technique for test time reduction [6]	Adaptive technique to reorder the patterns based on previously tested dies in real production	s38585 from ISCAS'89 [60]
Pattern Reorder for Test Cost Reduction Through Improved SVMRANK Algorithm [5]	Machine Learning based reordering by means of the weighted SVMRANK algorithm	s5378, s9234, s13207, s15850, s38417 and s38584 from ISCAS'89 benchmarks [60]
Wafer-Level Test-Length Selection [8]	Combining statistical yield modeling and integer linear programming to select the number of patterns needed	ITC'02 Test benchmarks [59]
How Many Test Patterns Are Useless? [10]	Statistical model based on production yield, the fault coverage characteristics, and the number of tested chips that forecasts the number of useless patterns	IBM devices
Proposed methodology	A comprehensive cost model to find the best point to cut patterns during Wafer Sort and accepting extra costs in packaging and Package Test	Academic benchmark with about 1 million gates and 40*nm* real-world Automotive SoC with about 20 million gates

rank on the entire universe of device faults. The rank of a fault depends on the bitwise similarity between the collected signatures and the *fault reference signature* previously computed for that fault.

Jayalakshmi *et al.* [22] proposed a methodology that aims to reduce the time and memory requirements of the tester for logic diagnosis on high-volume machines. The study proposes to use a two-step flow procedure with two different types of patterns. A series of GO/NOGO patterns are used in the first pass, highlighting the need for the eventual second pass of diagnostic patterns.

Ubar *et al.* [23] has proposed a methodology that considers diagnostic weight for logical diagnosis in BIST environments. The approach makes use of deterministic patterns saved in a dedicated memory. The number of patterns results from a trade-off between required memory and final diagnostic resolutions.

Cheng *et al.* [24] proposed a methodology that uses the error propagation function, which maps the scan cell faults in MISR, to extract a final list of candidates. The candidates are located at the intersection of the logic cones connected to the flip-flops that capture the logic response of the circuit. The method requires no architectural modification and relies solely on the resulting MISR signature. In [25], another improved methodology based on the logic cones of the scan cells is proposed. This study improves the methodology described in [24] by further optimizing the diagnostic accuracy by using the information also from the scan cell logic cones that map to *corrected* MISR bits.

Leveraging on on-chip test generation, Gopalsamy *et al.* [26] presented a logic diagnosis approach based on the technique described in [27]. The authors of prior work [27] suggested a technique for permuting current test vectors to generate deterministic test sets. These test sets are used in the logic diagnostic technique in [26] to reduce the overall amount of candidate sets, hence enhancing accuracy when used on academic benchmarks.

As previously detailed, the diagnosis phase described in [21] is based on the BISD custom architecture. Specifically, after failure signatures are stored in a dedicated memory, the method performs diagnosis by ranking faults across the entire fault universe. The ranking of a fault is determined by the bitwise similarity between the collected signatures and the corresponding *reference faulty signature* precomputed for that fault.

Table 2.2 summarizes the previously discussed and most relevant approaches as follows:

- The name of the methodology;
- A brief description of the approach;
- The reported case of study;
- The field application of the study.

Previously described studies mainly focused on architectural changes in LBIST and considered only failure models. In addition, many of them could not provide helpful guidance for NTF situations. One of the methodologies proposed in this thesis, deeply detailed in Section 3.4, solves this problem by proposing a diagnosis methodology that does not require architectural modification but fits any type. It considers both successful patterns and the first failed pattern to improve the logical diagnosis process. It also deals mainly with transition (TRN) failures, the primary source of aging problems.

Table 2.2 Logic diagnosis through LBIST methodologies.

Method	Brief Description	Reported Case of Study	Application
BISD: Scan-based Built-In self-diagnosis [21]	Authors propose a novel architecture, named BISD, capable of grading the fault candidates from saved faulty signatures	NXP industrial circuits $\approx 100-350$K gates	In-field Diagnosis
Signature Based Diagnosis for LBIST [24, 25]	Authors propose exploiting error propagation function that maps scan capture failures to MISR	Industrial Circuits ≈ 1.3, 1.9M gates	Logic Diagnosis
A Storage Based LBIST Scheme for Logic Diagnosis [26, 27]	Authors propose a way to enhance the diagnosis by adding test sets obtained through the computation of subsets of scan vectors and permutations	ISCAS'89 [60], ITC'99 [61], IWLS'05 benchmarks and logic blocks of the OpenSPARC T1	Logic Diagnosis
Proposed methodology	By exploiting the in-field captured first failing LBIST pattern's index, a list of candidates is obtained through pre-computed fault dictionaries	ITC'99 benchmakrs [61] and **Automotive SOC** $40nm \approx$ **20M gates**	In-field Data Collection & Logic Diagnosis

Chapter 3

The proposed methodologies

This chapter details the proposed methodologies and significant contributions made over the years of the research project. As mentioned earlier, the reliability and testing of automotive devices at various stages of manufacturing and in-field testing have been studied. For this reason, the proposed approaches are varied and belong to different steps of the device testing process. The subsections will follow the order in which the steps are applied.

Thus, Section 3.1 details the proposed approaches that optimize scan-based testing of Wafer Sort and Package Test steps. Section 3.2 discusses the critical issues of scan-based testing and how the categorization of such criticalities can assess the System-Level Test programs. Section 3.3 explores of pseudo-random generated stress stimuli along SLT. Finally, Section 3.4 covers the post-manufacturing phase, where it is critical to continue monitoring the device's reliability during its operational phase.

In each section, the main contributions with the common goal of improving the testing process of automotive devices are highlighted.

3.1 Scan-based Test Cost optimization

This section details the main contributions of the thesis primarily focusing on manufacturing scan-based tests and their optimization. Part of the content of this section has been published in [62].

Considering two successive test flow processes, such Wafer Sort and Package Test, the proposed approach offers a comprehensive and efficient scan test cost model capable of enabling optimization of the scan test cost.

It is based on the presumption that scan-based tests used at Wafer Sort are then repeated along the Package Test phase following device packaging in productive environments. In this setting, cutting patterns to apply from the tail of the pattern set allows one to reduce the cost of scan tests, as explained in Section 2.2. The suggested calculations finds the optimal point to cut patterns during the Wafer Sort and enable still an economic benefit despite more test escapes forwarded to the Package Test following the Wafer Sort, where they are finally screened out. These calculations apply when the failure distribution is uniform and valid for any pattern set. The main focus of study is on modifying the formulations to fit actual industrial conditions. Thus, the cost model is extended, leveraging the study published in [29], to be effective also for real-world devices with complicated designs, which may have areas with more faults likely to occur than others.

Hence, a comprehensive and extended scan test cost model is proposed. The latter is capable of predicting the cost reduction in the case of non-uniform failure distribution. Such model extends the concept of Fault Coverage to *Weighted Fault Coverage* (Ω) [29], which takes in consideration also the probability for each fault to occur.

This work aims to optimize the economic gain achievable by reducing the number of patterns. An effective pattern generation strategy is proposed to maximize the gain. The outcome is a final pattern set that fits better with the circuit criticality, bringing a higher economic gain than a classical application flow would.

Figure 3.1 shows briefly the steps of the proposed methodology, in which three branches can be distinguished:

A. *Cost Modeling for non-uniform failure distribution*: by modeling the scan test cost, an ideal point for cutting off patterns is estimated, where, while

increasing Wafer Sort escapes, it is still possible to achieve overall cost savings due to the reduction in Wafer Sort test time. The model uses the Weighted Fault Coverage (Ω), which involves the probability for each fault to occur, making the methodology suitable for devices presenting non-uniform failure distribution;

B. *Data analysis for the extraction of the criticality level of faults*: by analyzing failures testing with the Continue-on-Fail flow a sacrificial batch of devices, the faults more prone to be candidate are found, and each fault is assigned with a criticality index (weight). The computed weights enable the calculation of the Weighted Fault Coverage (Ω), used in the cost model. The latter exploits the weights to guarantee an accurate prediction in case of non-uniform failure distribution;

C. *Criticality-oriented ATPG for pattern set generation*: by exploiting the analysis of the step B, in which faults are ranked depending on a criticality index, a new pattern set is generated following the priority of the ranking.

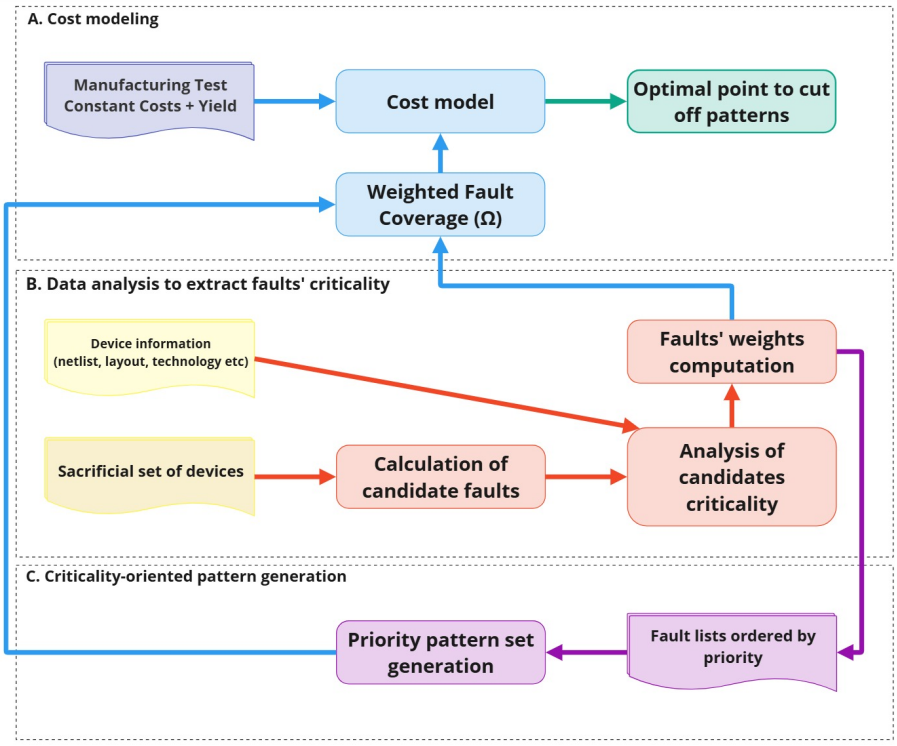


Fig. 3.1 Brief workflow of the proposed methodology to reduce scan-based test cost

3.1.1 Test cost modeling

The proposed cost model, for non-uniform failure distribution, extends the formulas described in Section 2.2.

In particular, Equation 2.6, describing the $Yield_W^{Reduced}$, is based on the well-known formula proposed by [28], in which uniform distribution is assumed.

Equation 2.6 ($Yield_W^{Reduced}$) takes into account the test escapes (Equation 2.5) introduced by the reduction of test time along Wafer Sort. Such term is significant because it describes the increase of the Package Test cost due to the increase number of test escapes coming from Wafer Sort. Equation 2.6 is based on the well-known formula proposed by [28], in which uniform distribution is assumed. The latter formula depends on the Fault Coverage. Considering non-uniform distribution, the formula for the yield changes taking into account also the probability for a fault to occur, following the derivation process described in [29]. Thus, the new formula that describes the yield resulting from a reduced application of Wafer Sort, for non-uniform failure distribution is:

$$Y_W^{Reduced} = Y_W + 1 - (Y_W)^{1-(\Omega/100)} \tag{3.1}$$

Where Ω represents a Weighted Fault Coverage that also considers the probability for each fault to occur, contrary to the classical Fault Coverage metric that assumes each fault to share the same weight. The formula to compute such value is shown in Equation 3.2, that is a weighted average.

$$\Omega = \frac{\sum_{n=1}^{N} W_n \cdot FD_n}{\sum_{n=1}^{N} W_n}, \text{ where } FD_n \in \{0,1\} \tag{3.2}$$

N represents the total number of faults. The numerator of the division is composed of the weighted summation of all the detected faults. Each fault is weighted depending on the W_n value ($W_n \in [0,1]$), and each weight is multiplied by a binary variable (FD_n), which represents whether that fault was detected (1) or not detected (0) by the considered pattern. Repeating the computation for each pattern (or chunks of patterns) gives the trend vs the number of applied patterns.

The Weighted Fault Coverage (Ω) enhances the accuracy of predicting how many test escapes would occur for devices presenting a non-uniform failure distribution.

The prediction of the number of test escapes directly affects the Package Test cost. Since the aim of the proposal is finding the optimal point where reducing Wafer Sort test time leads to an economic gain despite the increase of test escapes and Package Test cost. Thus, the more accurate the prediction of test escapes, the more accurate the cost gain estimation.

An effective way how to compute the Ω in real production cases is detailed in the following subsection.

3.1.2 Production data and device information analysis

In order to extract the criticality of the faults (and to compute thus the Ω), a strategy is proposed which encompasses the following elements:

- A corner lot of sacrificing devices tested with a Continue-on-Fail flow, applying the pattern set used in production. The selection of the corner lot to analyze that is used to extract information on the distribution of failures is left to the company and the judgment of experts in the quality field for the chosen device. In particular, it could be selected at random or in one of the particularly unfortunate lots, which presented more failures than the average. The procedure is recommended in a device ramp-up phase, during which companies typically conduct several phases of testing on a multitude of different lots.
- A ranked list of the total candidate faults computed by considering the number of occurrences of each candidate per each failing die.
- A list of criticality classes extracted through the device information (netlist, layout, technology) to extract the correlation between the extracted candidate faults and their criticality index, in order to compute the weight of each fault and finally the Ω.

The criticality indexes could diverge depending on the used technology. Thus, in order to make the methodology independent from the latter, the use of a sacrificial batch of devices can be exploited to tune the critical areas for all the devices using the same technology, i.e. belonging to the same chip family.

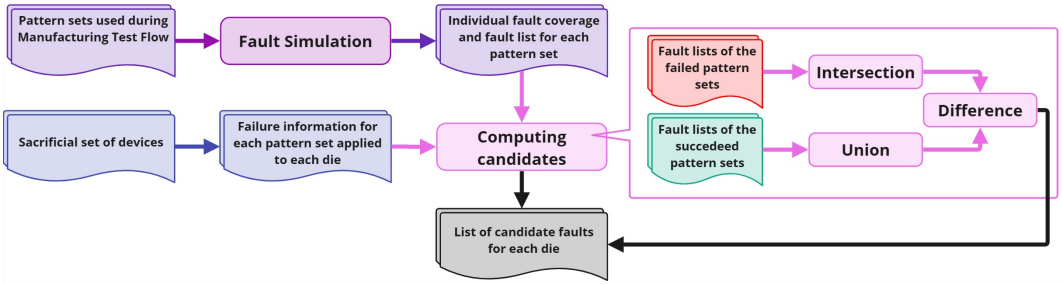


Fig. 3.2 General overview of the calculation of the candidate faults

The key elements of the Ω calculation are: 1) computation of candidate faults per each failing die, 2) criticality analysis of such candidates.

Figure 3.2 shows the proposed flow to calculate the list of candidate faults for each failing die. The calculation can be performed by dividing the process into two distinct parts. The final goal is to have a complete list of candidate faulty gates for extracting their criticality in a real production case of study. The input data needed are the pattern sets applied during the testing process and the information about the failure or success of each pattern set for each failing die.

Therefore, the first step is to fault simulate each pattern set individually because, typically, in productive environments, the pattern sets are generated incrementally without storing the individual list of covered faults. Doing so calculates the individual coverage of each applied pattern set.

A developed algorithm that intersects all the faults covered by the respective pattern sets is used at this point. For each die, the intersection of the faults' sets covered by the failed pattern sets was calculated, and the union of the faults' sets covered by the successful pattern sets was removed from the result of the previous intersection.

The single-fault hypothesis enables considering as candidates only those faults covered by all failed pattern sets, thus by the intersection of their coverage sets, allowing for removing all faults covered by the pattern sets that did not fail. The algorithm is detailed in Algorithm 1.

Considering F as the total number of failed pattern sets, and the set of covered faults for each failing pattern set ($Fail^{per\_pat}$), the intersection of all the failing pattern sets is the starting point of the candidates calculation:

Algorithm 1 Computing Candidates of a die

Input: Covered faults for each pattern sets (*coverages*), Indexes of failed pattern sets ($failures$), Indexes of successful pattern sets (*success*)
Output: Candidate faults ($candidate\_faults$)

```
candidate_faults ← [∅]
inter_fail ← coverages[failures[0]]
for each fail in failures: do
    inter_fail ← inter_fail.intersect(coverages[fail])
end for
union_succ ← [∅]
for each succ in success: do
    union_succ ← union_succ.union(coverages[succ])
end for
candidate_faults ← inter_fail.difference(union_succ) return candidate_faults
```

$$Fail_{die} = \bigcap_{f=1}^{F} Fail_f^{per\_pat} \tag{3.3}$$

$Fail_{die}$ represents the intersection between all the failing patterns. At this point, the set can be reduced further, considering also the succeeded pattern sets. Indeed, the faults covered by succeeded patterns cannot be candidates.

Considering S as the total number of succeeded pattern sets, and the set of covered faults for each successful pattern set ($Success^{per\_pat}$), the union of all the sets represents the faults that cannot be candidate and that must be deleted by the $Fail_{die}$ set. Thus, the final set representing the candidate faults for a die is Equation 3.5.

$$Success_{die} = \bigcup_{s=1}^{S} Success_s^{per\_pat} \tag{3.4}$$

$$Candidates_{die} = Fail_{die} - Success_{die} \tag{3.5}$$

Once the candidate faults are extracted, they are ranked depending on how many dies have that fault in their candidate list. Such ranking is then exploited by Algorithm 3 to cross the information about the criticality levels.

After having extracted the candidate faults for each die, it is possible to compute how many times a fault has been a candidate, that is the same of saying how many dies have that fault in their candidate faults. Having this information enables also the computation of a ranking of the candidates and to know which one were found the most. The algorithm is detailed in Algorithm 2.

Algorithm 2 Computing occurrences of each candidate fault over all failing dies

Input: Candidate faults of all failing dies (*total_candidates*)
Output: Ranked candidate faults with the number of occurrences of each (*ranked_candidates*)

```
1:  ranked_candidates ← dictionary()
2:  for each candidates_of_die in total_candidates: do
3:      for each c in candidates_of_die: do
4:          if c in ranked_candidates then
5:              occurrences ← ranked_candidates[c]
6:              ranked_candidates[c] ← occurrences + 1
7:          else
8:              ranked_candidates[c] = 0
9:          end if
10:     end for
11: end for
12: sort_dictionary_by_value(ranked_candidates)
13: return ranked_candidates
```

To better explain how the calculation works, an example with three failing dies is considered. The complete pattern set is composed of 4 pattern sets ($T1, T2, T3, T4$). The sets of covered faults by each applied pattern set are shown in Table 3.1. The respective failed and succeeded ones are shown in Table 3.2. Thus, with the above mentioned information, the resulting list of candidates, with also the number of dies whose the candidate belongs to, is shown in Table 3.3. The list is also ranked depending on the how many times the fault has been found as a candidate. In this way, in the next criticality analysis phase it is possible to cross such information to extract which criticality have the faults more prone to occur.

To extract the criticality of a device, its information, such as netlist, layout and technology, could be exploited. Different studies have proposed for example to exploit the layout information. In [46] a study is presented about increasing the total coverage in production exploiting critical areas of the layout. Moreover, [63] and [64] discussed how in modern complex devices the layout of the circuit is

Table 3.1 Pattern sets' coverages

Pattern set	Covered Faults
T1	F1, F2, F3
T2	F2, F3, F4
T3	F4, F5, F6
T4	F3, F5, F6

Table 3.2 Pattern sets' failures during the Test Flow

Die ID	Pattern sets' failures	Pattern sets' fail coverage intersection	Pattern sets' success coverage union	Resulting candidate faults
1	T1, T2	F2, F3	F5, F6, F4, F3	F2
2	T3, T4	F5, F6	F1, F2, F4, F3	F5, F6
3	T3, T4	F5, F6	F1, F2, F4, F3	F5, F6

Table 3.3 Resulting list of candidates

Fault alias	# Dies
F5	2
F6	2
F2	1
F1	0
F3	0
F4	0

characterized by non-uniform distribution of gates. Such studies proposed hence a way to correlate the density and the device criticality.

Algorithm 3 Faults' weight computation

Input: Population of faults ($fault\_list$), Ranked candidate faults with the number of occurrences of each ($ranked\_candidates$), Criticality classes with the faults belonging to them ($criticality\_classes$)
Output: Weighted fault list ($weighted\_fault\_list$)

1: $weighted\_fault\_list \leftarrow [\emptyset]$
2: $N\_classes \leftarrow len(criticality\_classes)$
3: $classes\_freq \leftarrow N\_classes * [0]$ ▷ it keeps count of the candidates' occurrences for each criticality class
4: **for each** $fault$, $occurrences$ in $ranked\_candidates$ **do**
5: $class \leftarrow criticality\_classes$.find($fault$)
6: $classes\_freq[class] \leftarrow classes\_freq[class] + 1$
7: **end for**
8: $ranked\_classes \leftarrow$ get_ordered_classes($classes\_freq$) ▷ ranked classes by the candidates' occurrences
9: **for each** $fault$ in $fault\_list$ **do**
10: $class \leftarrow criticality\_classes$.find($fault$)
11: $rank \leftarrow ranked\_classes$.find($class$)
12: $fault.weight \leftarrow \frac{N\_classes - rank + 1}{N\_classes}$
13: Append $fault$ to $weighted\_fault\_list$
14: **end for**

However, the method of extracting the criticality classes is of no importance. Once the latter is obtained, the calculation of the weights for the faults and the Ω can be executed independently of the extraction process. The sole prerequisite is to possess criticality classes alongside the faults associated with each class. The algorithm for calculating the weights is outlined in Algorithm 3. The weights are subsequently utilized to obtain the Weighted Fault Coverage (Ω) as demonstrated in Equation 3.2.

Example 2 ■

Building upon the example presented in Section 2.9, the identical academic circuit benchmark is employed to demonstrate the computation of the Weighted Fault Coverage (Ω).

Due to the impracticality of obtaining an authentic sacrificial set of dies for the benchmark, many populations of around $1M$ dies with elevated yield (refer to

Figure 2.5) have been generated by randomly picking faults from the fault universe using a Monte Carlo approach [65]. The selection of faults has not been conducted randomly, as a non-uniform failure distribution is taken into account.

The circuit has been generated, and the placement and routing phase has been finalized using the same tool and technology (40*nm*) employed in the industrial case study, which will be detailed in Chapter 4. The selected criticality index is the density metric [35].

Specifically, in the selection of faults to emulate a realistic testing scenario, it was acknowledged that the fault distribution is non-uniform. As a consequence of this, the selection probability of a fault was determined based on whether or not it was classified as belonging to a more or less critical class. Due to the fact that the same technology is utilized for the academic benchmark, it is possible to make the same assumption regarding the association between the criticality and the density areas. Figure 3.3 illustrates the average distribution of candidates across the Monte Carlo runs. It offers a comprehensive insight into the distribution of candidates about gate density. On the x-axis, the density classes of the device are represented, and the bars indicate the proportion of the total number of candidates that were discovered in that particular class.

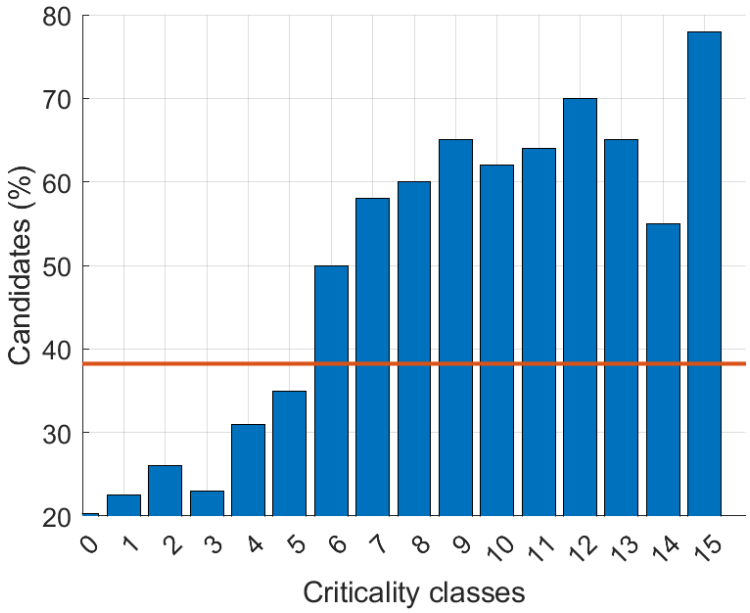


Fig. 3.3 Percentage of candidates for each criticality class (number of neighbors)

After calculating the weights of the faults as per Algorithm 3, the cost estimation for the non-uniform failure distribution can be conducted.

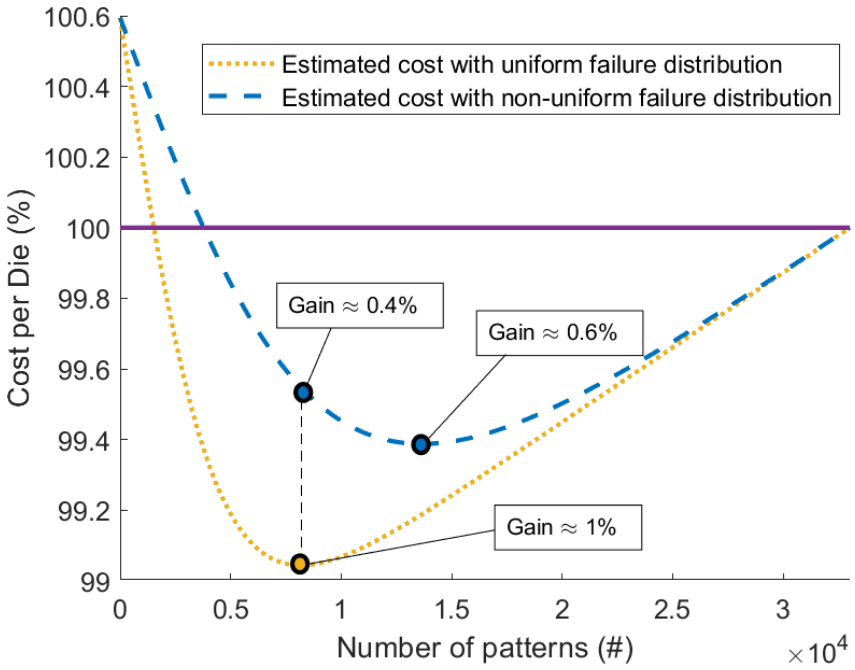


Fig. 3.4 Cost per Die (%) vs number of applied patterns considering both uniform and non-uniform failure distribution

Figure 3.4 illustrates the trend of the projected cost with a uniform failure distribution (as seen in Figure 2.5) alongside the curve representing the cost estimates derived from the proposed model, which incorporates the Ω calculation. The uniform model forecasts a maximum attainable gain of approximately 1% for cutting patterns at approximately $8,000$. Simultaneously, the non-uniform model forecasts a maximum benefit of approximately 0.6% at around $13,500$ patterns.

The cost model that accounts for uniform failure distribution is marginally more optimistic in forecasting the greatest attainable gain. A comparable trend will be exhibited in Chapter 4 for the real-world industrial case study, whereby both estimations will be juxtaposed with actual production data gathered over a six-month period. Nonetheless, the optimal point for cutting patterns, as forecasted by the *uniform* model, would yield an approximate gain of 0.4% when accounting for the *non-uniform* distribution, which is more precise (as demonstrated in Chapter 4).

Example 3 will demonstrate how the classification of criticality will be utilized to create a criticality-oriented pattern set that maximizes potential economic gain.

End of the example 2. ■

For any given pattern set, the Ω facilitates a more accurate estimation of economic cost benefits in comparison to the uniform model. Furthermore, the economic benefit can be optimized based on this information, as demonstrated in the next subsection.

3.1.3 Criticality-oriented pattern generation

The outcome of the analysis step will be a list of weighted faults, with each weight reflecting the fault's criticality classification. Consequently, this information can be utilized to establish a pattern generation depending on the criticality of the faults. Generation can be accomplished using a conventional ATPG tool; however, the ATPG will sequentially address the faults considered most vital to the device, thereafter progressing to encompass all progressively less critical errors until the fault list is depleted.

The objective of the proposal is to give more importance to the most critical faults, while the less critical ones are shifted to a lower priority. Generating the pattern set following such a flow ensures that eliminating patterns from the tail of the pattern set helps one to guarantee that the missed faults are the least likely to present. The strategy used to carry out this procedure, known as *criticality-oriented ATPG*, is defined in Algorithm 4. This pattern generation strategy is a by-product of the proposed scan test cost model. It aims to improve the economic gain achievable by reducing patterns during Wafer Sort for devices showing non-uniform failure distribution.

Algorithm 4 Criticality-oriented ATPG pattern generation

Input: List of criticality classes ordered by their priority (*criticality_classes*), Fault Lists for each criticality classes ($fault\_lists$)
Output: criticality-oriented pattern set (*criticality_pattern_set*)

1: Run ATPG tool
2: $criticality\_pattern\_set \leftarrow [\emptyset]$
3: **for each** *class* in *criticality_classes*: **do**
4: Read faults from $fault\_lists[class^{th}]$
5: Create patterns
6: Store patterns to *criticality_pattern_set*
7: **end for**

Example 3 ■

Referring to the academic benchmark considered so far, to better clarify the methodology through actual examples, a new pattern set has been generated following the Algorithm 4. The new pattern set has been generated with the same number of patterns of the traditional one ($\approx$ 30 thousands), but guiding the generation as described in Algorithm 4. The comparison of the two pattern sets is shown in Figure 3.5.

The trend of the criticality-oriented generation is a bit slower. This behaviour is reasonable because the required efforts for the guided ATPG are higher than the classical procedure. The reached coverage is a bit less than the original one, with the same number of patterns.

Figure 3.6 shows the curves trend for two different pattern sets, computed by the proposed cost model considering non-uniform failure distribution, thus exploiting the previously computed Ω (see Example 2).

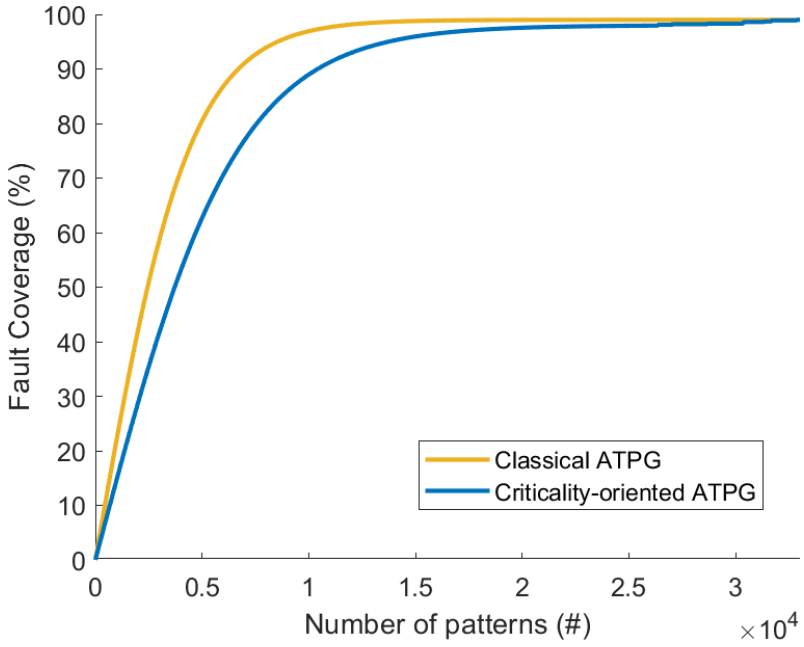


Fig. 3.5 Fault coverage (%) comparison between classical ATPG and criticality-oriented pattern generation

The model can predict which is the best point to cut patterns to achieve the maximum economic gain, also considering the non-uniform failure distribution, for any given pattern set. Thus, the methodology can be applied just to know where to cut the pattern set already in use. Moreover, Figure 3.6 also shows the points

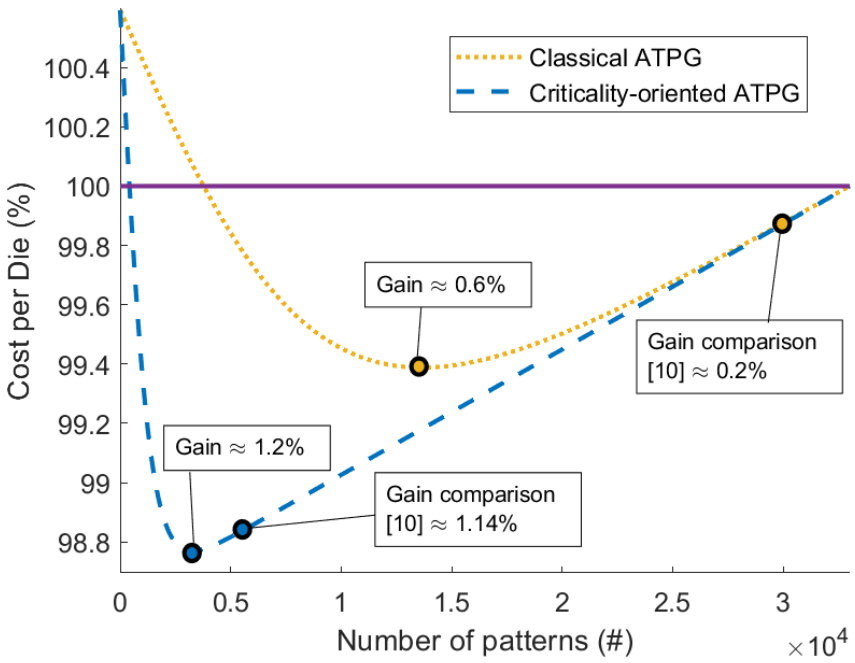


Fig. 3.6 Non-uniform failure distribution: Cost per Die (%) vs number of applied patterns generated by classical ATPG and criticality-oriented ATPG algorithm

to cut patterns predicted by the model presented in [10]. They are reported in the non-uniform curves because they more accurately predict the production trend (as it will be shown in Chapter 4 with real production data).

As a by-product of the analysis, the extrapolated information about the failure distribution can also be exploited to generate a more accurate pattern set, which follows the algorithm shown in Algorithm 4. The predicted maximum economic gain for the pattern set generated by classical strategy is about 0.6%. Meanwhile, exploiting the criticality information, the newly generated pattern set achieves double the gain, reaching 1.2%.

End of the example 3. ■

3.2 Assessment of System-Level Test programs

This section outlines the primary contributions of the thesis focusing on the identification of structural weaknesses in very large system-on-chips (SoCs) as detailed in Section 2.4, with the aim of assessing System-Level Test (SLT) programs. Part of this content is currently under major revision in *IEEE Transactions on Computers* as part of the article titled: "*Automatic Generation of System-Level Test for un-core logic of large Automotive SoC*".

Typically, SLT procedures are generated automatically or manually based on holistic assumptions. Their main purpose is to identify any uncovered faults that remain after extensive structural scan-based tests, emphasizing the architectural interconnections. To analytically determine whether this holistic generation effectively addresses the inherent challenges of structural testing, an analysis of residual (not-detected) faults not covered by scan-based tests is presented, building on the concepts discussed in Section 2.4.

Specifically, uncovered faults may be categorized as follows:

A. *Clock-domain crossing*: faults present in the intersection of flop logic cones triggered by different clock domains.

B. *Chain configurations crossing*: faults occurring in the intersection of flop logic cones that are part of different chain configurations.

C. *ATPG critical*: faults that are challenging to observe due to their association with a limited number of scanned flops.

D. *Functionally untestable*: faults that cannot be excited or observed during typical functional program execution, such as those associated with debug circuitry and test enables for flip-flops; these are identified using the method proposed in [66] and are incorporated as constant constraints in the fault simulator.

In order to assign categories to the uncovered faults, it is necessary to obtain some important information from the circuit. In particular, it is necessary to extract from the netlist the information of all the affected gates, with the inputs/outputs and related endpoints. Then, it is possible to process the above information and by means of special algorithms, derive the categories exposed above.

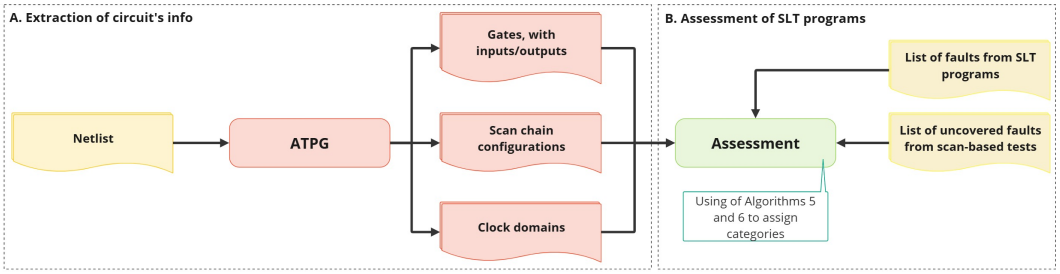


Fig. 3.7 General overview on the assessment of SLT procedures

Thus, the assessment procedure of SLT programs, as shown in the figure, is done through two key steps:

A. **extraction** of circuit's information by the means of a commercial ATPG tool, i.e., gates with inputs/outputs, scan chain configurations, clock domains;

B. **assessing** uncovered faults by the application of algorithms (Algorithm 5 and Algorithm 6) on the previously extracted information.

The algorithms adopted in the classification of these faults are described below. Algorithm 5 is capable of extracting faults in both the clock-domain and chain configuration crossing categories. A catalog of circuit gates, including the flop endpoints and the corresponding clock domain and chain configuration information, is essential before utilizing these algorithms. Algorithm 6 describes the extraction method depending on the set of faults not covered by the current structural patterns in the ATPG critical category. This enables the method to identify faults with more importance than the average ones.

Once all remaining faults that are not covered by the structural tests have been categorized, it becomes possible to identify the areas where the SLT was most effective in mitigating the structural weaknesses of the Design for Test (DfT) in complex case studies. This analysis thus confirms that the holistic and automated generation of SLT programs within the proposed methodology successfully achieves its goal of addressing a larger number of critical structural testing faults.

Algorithm 5 Finding faults between different scan chain configurations or clock domains

Require: *gates*, list of gates including inputs, outputs and corresponding endpoints with the belonging to the scan chain configuration

```
1:  faults ← ∅
2:  for each g in gates do
3:      scan_conf_in ← ∅
4:      for each inp in g.inputs do
5:          for each endp in inp.endpoints do
6:              chain_conf_in.add(endp.scan_conf)
7:          end for
8:      end for
9:      scan_conf_out ← ∅
10:     for each out in g.outputs do
11:         not_found ← FALSE
12:         for each endp in out.endpoints do
13:             scan_conf_out ← endp.scan_conf
14:             if endp.scan_conf not in scan_conf_in then
15:                 not_found ← TRUE
16:             end if
17:         end for
18:         if not_found is TRUE then
19:             faults.add(out)
20:         end if
21:     end for
22:     for each inp in g.inputs do
23:         not_found ← FALSE
24:         for each scan_conf in scan_conf_in do
25:             if scan_conf not in scan_conf_out then
26:                 not_found ← TRUE
27:             end if
28:         end for
29:         if not_found is TRUE then
30:             faults.add(inp)
31:         end if
32:     end for
33: end for
34: return faults
```

Algorithm 6 Extracting ATPG critical faults depending on the fan-in/fan-out ratio

Require: *gates*, list of gates including inputs, outputs and corresponding endpoints
Require: *residual_faults*, set of faults not detected by structural-based tests
1: $atpg\_critical\_faults \leftarrow \emptyset$
2: **for each** g in $gates$ **do**
3: $fan\_in \leftarrow 0$
4: **for each** inp in $g.inputs$ **do**
5: $fan\_in \leftarrow fan\_in +$ len($inp.endpoints$)
6: **end for**
7: $fan\_out \leftarrow 0$
8: **for each** out in $g.outputs$ **do**
9: $fan\_out \leftarrow fan\_out +$ len($out.endpoints$)
10: **end for**
11: $g.ratio \leftarrow \frac{fan\_in}{fan\_out}$
12: $faults$.add_with_ratio($g.inputs$, $g.ratio$)
13: $faults$.add_with_ratio($g.outputs$, $g.ratio$)
14: **end for**
15: $unique\_ratios \leftarrow$ get_unique_ratios($faults$)
16: $tot\_occurrences \leftarrow \emptyset$
17: **for each** not_det in $residual\_faults$ **do**
18: $not\_det\_ratio \leftarrow faults[not\_det].ratio$
19: $tot\_occurrences[not\_det\_ratio]$.increment()
20: **end for**
21: $avg\_occurrences \leftarrow \frac{tot\_occurrences}{\text{len}(tot\_occurrences.\text{keys}())}$
22: **for each** $fault$ in $residual\_faults$ **do**
23: **if** $fault.ratio \geq avg\_occurrences$ **then**
24: $atpg\_critical\_faults$.add($fault$)
25: **end if**
26: **end for**
27: **return** $atpg\_critical\_faults$

3.3 Stress stimuli application along System-Level Test

This section details the primary investigation of the thesis regarding the application of stress stimuli, typically intended for the Burn-In phase, also along the System-Level Test phase. In particular, the research activity and subsequent experimental evaluations presented in Section 4.4 have been conducted during a period as an intern in STMicroelectronics. The main goal of the investigation is to improve the stress stimuli for a very large ADAS device.

As previously discussed in the manuscript, especially in Section 2.1, SLT typically involves functional tests aiming at emulating the device's operational environment, mainly covering the interconnections among the various modules. Nevertheless, using the tester usually intended for SLT, it is also possible to apply pseudo-random stress stimuli. Indeed, it is possible to enter the test mode typically used for the Burn-In phase and apply internal stress stimuli through scan ports, as proposed in [14]. Applying stress stimuli during the System-Level Test phase can complete and enhance the applied tests before the device is shipped to the market.

Moreover, with such a tester architecture, it is possible to apply pseudo-random stress stimuli generated on the fly; an endless stream of bits generated by software permits the application of many more patterns without memory limits.

As detailed in Section 2.3, evaluating stress stimuli can be challenging. Commercial tools typically compute only the *toggle coverage* metric. However, there are other ways to extract the *toggle activity*, which is a more precise way to know how many nodes in the circuit did toggle, as discussed in [32, 14], also giving some statistical insights.

In order to investigate the applicability of such methodologies in very large industrial devices, two primary generation strategies have been used to compare their effectiveness: ATPG-based and Pseudo-random LFSR-based.

Figure 3.8 depicts the flow for generating and evaluating the stress stimuli. Both toggle coverage and toggle activity are computed. Commercial ATPG tools compute the first by fault simulating the desired pattern set. The second one needs the generation of the Verilog testbench corresponding to the desired pattern set and the logic simulation of the latter to obtain a final Value Change Dump (VCD) file that can be analyzed to compute both the toggle coverage and activity [32], including

also phases that are not taken into account by commercial tools, such as the test mode entry sequence.

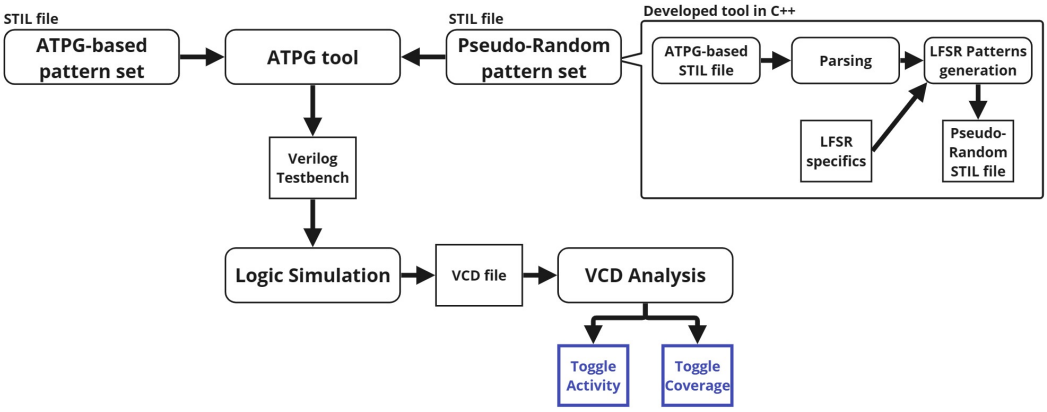


Fig. 3.8 Stress stimuli generation and evaluation phases.

Stress stimuli can be applied during the System-Level Test phase using an MCU-based tester that is programmed to send commands to the Device Under Test (DUT) through the DMA controller, using a tester architecture similar to [14]. In the case of pseudo-random pattern generation, the MCU is also programmed to generate the patterns on the fly by implementing a software LFSR module in firmware. Such a methodology has been chosen because of the significant advantage of not having memory limit requirements. Storing static patterns is memory-consuming for very large case studies because of the significant number of scan cells.

3.4 Logic diagnosis based on data collected in-field

This section details the main contributions of the thesis about the post-sale life of automotive devices. Part of the content of this section has been published in [56] and [67].

The proposed methodology comprises the collection of failure data in-field and then the exploitation of them to conduct logic diagnosis based solely on such collected information obtained from the constrained LBIST execution by CPUs.

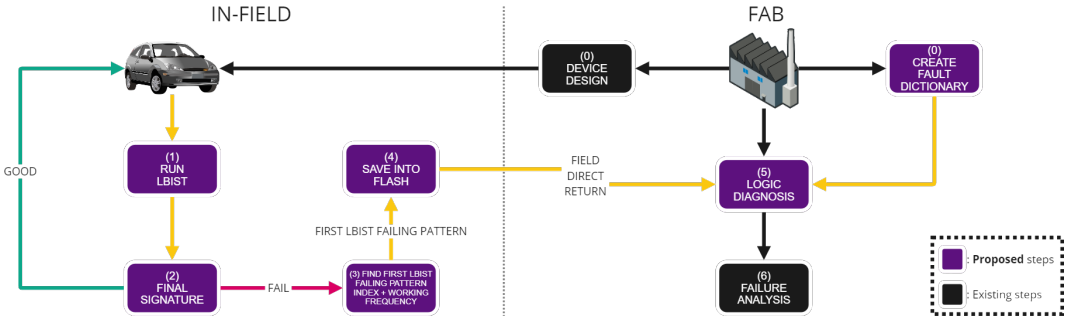


Fig. 3.9 Flow of the proposed approach to compute logic diagnosis of mission mode failures

Figure 3.9 shows the flow of the proposed approach. It starts with step (0), in which the FAB designs and markets a device. As this thesis describes, the FAB creates fault dictionaries before to device shipment to customers. Under mission restrictions, the LBIST is carried out in step (1), during the key-on/off events of the vehicle. Maintaining the LFSR seed and MISR settings unaltered helps to prevent possible hazards to the DUT while in-field. The first phase generates a 2-final signature as output. Should the signature match the expected one, the car is deemed safe, so the process is repeated. On the other hand, a mismatch suggests a logical error in the UUT, which makes the car dangerous and calls for research.

Because it does not show which pattern index caused the UUT to fail, the signature captured in step (2) offers minimal diagnostic information. This restriction narrows the field of reasoning diagnosis. Therefore, knowing the index of the first failing pattern will provide the final signature some important information.

A dichotomic search algorithm (3) is employed to find the index of the first failing pattern. Additionally, this step extracts the working frequency at which the tests pass. In order to do this, precise control over the number of patterns to apply and the ability to read the final signature is required.

Once the pattern index is identified, the resulting signature and frequency are stored (4) in the device's flash memory. This enables a more refined logic diagnosis when the device is returned to the manufacturer (5). Finally, in step (6), the FAB can conduct a failure analysis.

The faults covered by each pattern can be determined through fault simulation. Based on this, the candidates for a faulty device are identified by analyzing the differences between fault sets. Specifically, the first failing pattern should cover faults not detected by previous patterns. Consequently, the faults detected by earlier patterns can be excluded from the candidate set, leaving only the unique coverage of the first failing pattern.

Thus, the proposed approach consists of two phases:

A. *In-Field data collection*: During each key-on/off event, LBIST is executed at the maximum allowable frequency to target specific transition (TRN) faults. If the resulting signature does not match the golden signature, the frequency at which LBIST produces a correct signature is determined by gradually decreasing it during subsequent runs. The index of the first failing LBIST pattern is then identified using a dichotomic search algorithm (as described in Algorithm 7) and stored along with the corresponding signature.

B. *Logic Diagnosis of field return*: When a device is returned to the manufacturer from the field, the information stored in the FLASH memory is employed to help diagnose potential failures. Fault dictionaries are used to identify the failing class of the device, and the index of the first failing pattern ensures that previous failures do not corrupt the final result.

3.4.1 In-field data collection through LBIST

This section explains the in-field data collection process as outlined in [67]. It first discusses the application of a dichotomic search to identify the index of the first failing LBIST pattern and the method for computing the golden signature dynamically. Finally, it details the storage requirements necessary for implementing the in-field approach.

First Failing LBIST Signature Collection

When an LBIST signature does not match the expected one, identifying the index of the first failed pattern is not easy. The LBIST signature is calculated incrementally, and each new signature is based on data from previous patterns. Consequently, any corrupted value caused by a failure propagates to the final signature, available only at the end of the test sequence (as illustrated in Figure 3.10). Consequently, tests must be run sequentially for each model until the failed model is identified.

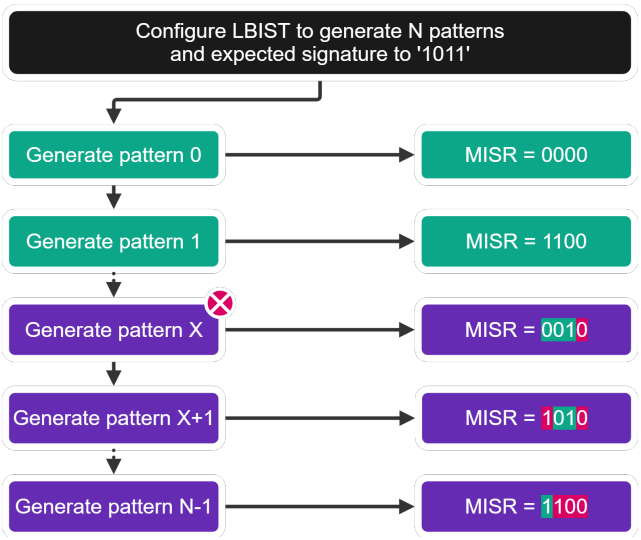


Fig. 3.10 LBIST generates N patterns, the X_{th} fails and propagates the error till self-test end in the MISR.

To address this challenge, the study uses a well-known algorithm in computer science: dichotomic search. This algorithm is very versatile and widely used in various applications. This work's novelty is applying a dichotomic search to identify the index of a faulty device's first failed LBIST pattern. This way, the number of tests required is significantly reduced, scaling logarithmically concerning the number of patterns. The pseudo-code of the dichotomous search algorithm is shown below:

The LBIST signature meets all the criteria required for applying the dichotomic search algorithm described in Algorithm 7, making it suitable for this context. These criteria include:

- The signature is computed incrementally, relying on previous signatures, ensuring any deviation propagates through the entire test sequence.

Algorithm 7 Pseudo-code of dichotomic search

Require: k, cardinality of the set
Require: $f : K \rightarrow \{False, True\}$
1: $low \leftarrow 0$
2: $high \leftarrow k - 1$
3: **while** $high > low$ **do**
4: $current \leftarrow (high + low) \gg 1$
5: **if** $f(current) = True$ **then**
6: $low \leftarrow current - 1$
7: **else**
8: $high \leftarrow current - 1$
9: **end if**
10: **end while**

- The cardinality corresponds to the number of generated patterns, which can be programmed via the LBIST Controller.
- A comparison function is available that evaluates the expected signature against the obtained signature and outputs a boolean result.

The process flow, as illustrated in Figure 3.11, is designed to leverage the stepwise programmability of the LBIST. To enhance the targeting of TRN faults, each LBIST execution is initially performed at the maximum operating frequency.

The process follows these steps:

1. It begins with running LBISTs from the firmware utilizing the maximum number of patterns #$total$ at the highest running frequency;
2. If the resulting signature matches the predicted one, all patterns produced between 0 and #$total$ have passed (indicated as green). In this situation, the device is verified to operate correctly as shown in Figure 3.12, so the testing is finished.
3. Should a signature mismatch arise, the proposed method firstly finds the frequency at which tests pass by progressively lowering the working frequency by a designated Δ. The Δ influences of the architectural constraints of the register range. It is found in the minimal working frequency steps that devices show problems in.

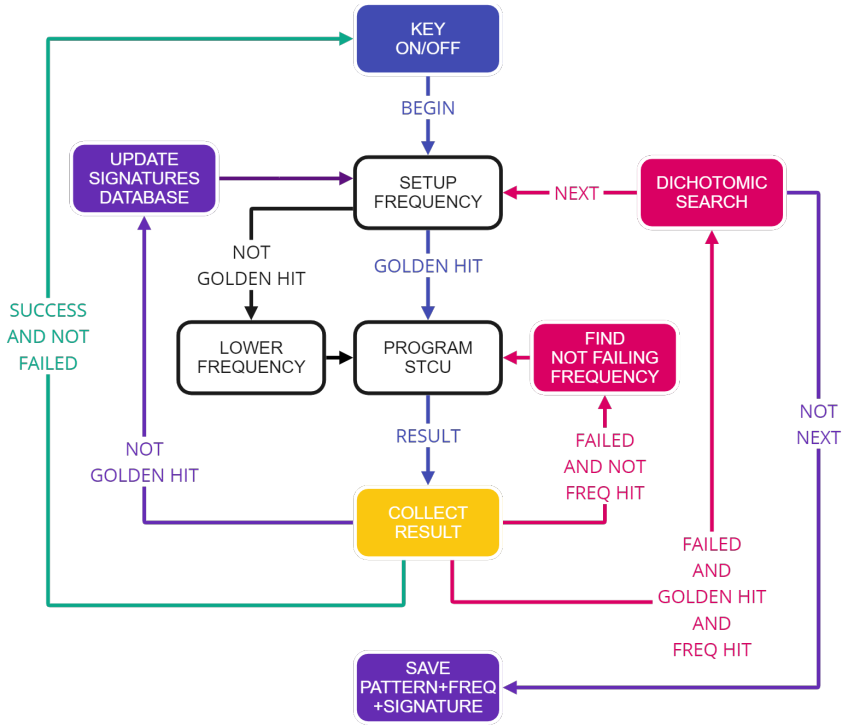


Fig. 3.11 Simplified Finite State Machine (FSM) version of the logic to be implemented for the proposed work.

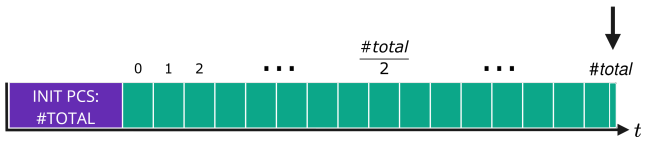


Fig. 3.12 If the signature is good, test ends immediately

Once the appropriate frequency is identified, the dichotomic algorithm is initiated by configuring the LBIST to run with half of the total patterns $\frac{\#total\ patterns}{2}$. If the resulting signature remains incorrect, all signatures generated using a number of patterns between $\frac{\#total\ patterns}{2}$ and $\#total\ pattern$ patterns can be discarded since a pattern prior, or equal to $\frac{\#total\ patterns}{2}$, produced a wrong signature. Consequently, these signatures can be marked in red without further executing the LBIST, as the fault is covered in an earlier pattern;

4. This process continues by iteratively applying the logic described in Algorithm 7. The procedure ends when no further divisions are possible, leaving only a single number of patterns to apply, corresponding to the first failing pattern. This final scenario is represented in Figure 3.13.

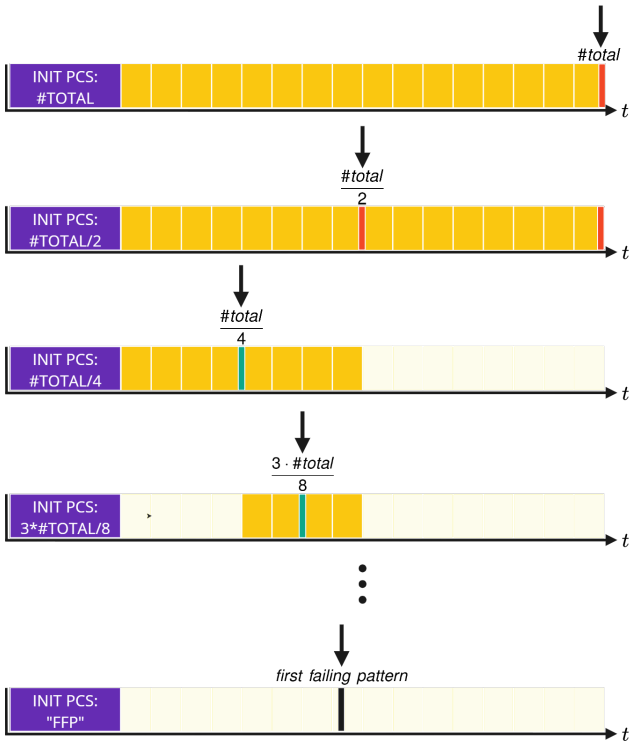


Fig. 3.13 If the signature is wrong, the dichotomous search algorithm is applied, which stops when two different dichotomies cannot be distinguished

The index of the first failed model, named p_{ff}, which is derived from applying the dichotomous search algorithm, contains key information. It allows for a refined logical diagnosis by considering the coverage of previous successful patterns.

Considering p_{ff} as the first failed pattern, the set composed of patterns between $[p_0, p_{ff})$ does not cover the observed failure, but p_{ff} must cover it.

Golden signatures are retrieved from a database stored in FLASH memory for test comparisons. If a signature is unavailable in the database, a procedure is initiated to extract the required signature. The process for obtaining a golden signature and the organization of FLASH memory is described in detail in the following sections.

Golden Signature Computation

Golden signatures play a key role in LBIST to detect anomalies during the comparison process, as discussed in Section 2.6. The tests, illustrated in Figure 3.11, are conducted at high frequency to detect TRN faults effectively. However, performing tests at such high frequencies increases the risk of compromising the final signature. Suppose a golden signature is not already stored in FLASH memory. In that case, it can still be generated and stored safely by following the steps described below.

- Suspend current test;
- Lower the frequency to a safe LBIST execution range;
- Collect golden signature and save it in FLASH at self-test end;
- Restart the initial suspended test.

The purpose of frequency switching is to reduce the probability of capturing potential faults in the signature, which might otherwise compromise test results and subsequent comparisons.

Although this study focuses primarily on TRN faults, the dichotomic search methodology can also be applied to LBIST signatures to address Stuck-At (SA) faults. However, in these cases, a precompiled signature database should replace the current approach of extracting gold signatures from a field device.

It is important to note that based on the bathtub curve [68], which models the occurrence of failures in devices during production and over their lifetime, and considering Burn-In (BI) [69] tests to detect production failures, it is reasonable to assume that failures initially occur as TRN failures before evolving into SA failures. Nevertheless, the main objective of this study is to detect faults as early as possible, making the coverage of TRN faults the primary objective.

Data Storage

The structure of the nonvolatile memory required for this approach, as shown in Figure 3.14, is divided into three distinct regions:

1. The *frequency region* stores the working frequency at which tests pass;
2. The *golden region* is dedicated to the storage of golden signatures;
3. The *failure region* contains information about the fist failed pattern identified through the dichotomic search method.

The second and last region share the same structure composed of:

- The index representing the number of patterns for the test;
- The resulting LBIST signature.

The required memory footprint varies depending on the fault model considered.

For TRN fault modeling during device shipment, the memory structure starts empty and is populated during device operation. Initially, only the good signature occupied the first entry in the *golden region* since no faults were detected at this stage. If LBIST detects misbehavior, the *frequency region* stores the safe working frequency at which the tests are passed. The *golden region* is then fully populated using the dichotomous search algorithm (Algorithm 7). At the same time, the *failure region* records the first failed pattern. The number of entries in each region is determined by the number of possible divisions by the number of patterns, calculated as $\#entries = log_2(\#total\ patterns) + 1$.

The *golden region* must be precomputed for SA failure modeling since no computation involving gold signatures takes place. In this case, the number of entries in the region is given by $\#entries = \#total\ patterns + 1$.

In addition, the *frequency region*, and the indexes of the golden signatures can be omitted since the frequency adjustments are irrelevant, and the indices can be derived as offsets from a base address in FLASH memory.

Regardless of the fault model, this memory scheme must be replicated for each LBIST partition of the device to ensure proper operation.

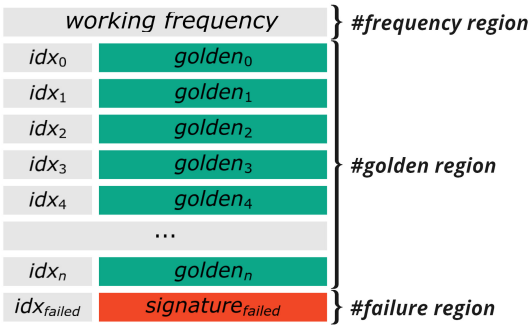


Fig. 3.14 Memory layout required

3.4.2 Logic diagnosis of field return devices

The increasing number of devices returned from the field presents a significant challenge for manufacturing companies. As modern system-on-chips (SoCs) expand in complexity, particularly in safety-critical applications, pinpointing the source of failures has become increasingly intricate.

To address this issue, the suggested in-field data collection system enhances the diagnostic process for manufacturers by capturing and storing the index and corresponding signature of the first failing pattern during key-on and key-off tests. This is crucial because the functionality of LBIST leads to the corruption of information obtained from patterns that fail after the initial one, as MISR signatures are derived from prior states. Consequently, all signatures following the first failure are rendered uninformative for diagnosis and could even hinder accurate interpretation (refer to Figure 3.15).

As illustrated in Figure 3.15, the fault dictionary generated for diagnosing failures through this proposed methodology is inherently incomplete. This is due to the fact that, for each applied pattern, the focus remains solely on the first two *pass/fail* outcomes, continuing the investigation only for *pass* nodes. Therefore, each *failed* node effectively becomes a terminal point, making it impossible to determine subsequent pass or fail outcomes accurately due to the aforementioned limitations of MISR.

The process for diagnosing failures is depicted in Figure 3.16. The diagnosis comprises two main phases:

- **Fault Dictionary Calculation**: A tree-based fault dictionary is established through fault simulations, which help extrapolate the unique coverages of

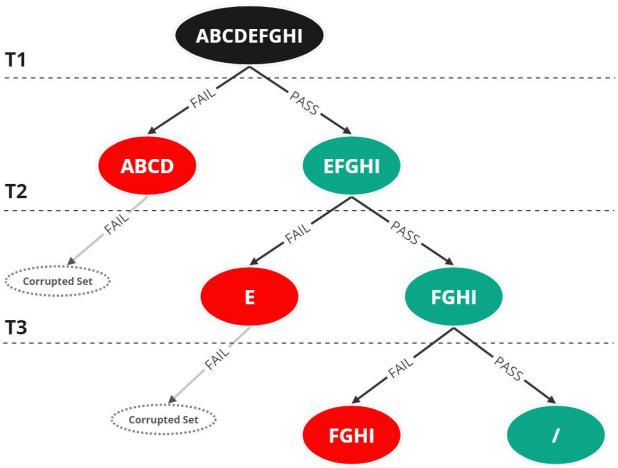


Fig. 3.15 Tree-based fault dictionary for LBIST with an example of three test patterns

each pattern generated by the LFSR, and this information is stored for future reference in case of returns.

- **Logic Diagnosis of In-Field Returns**: When a device is returned to manufacturers due to a misbehavior, the fault dictionary is utilized to identify potential faults, leveraging the data stored in the device's FLASH memory via the in-field data collection system. By identifying the index of the first failing pattern, the associated set of candidates can be pulled from the pre-computed tree-based fault dictionary.

To compute the fault dictionary for the LBIST, it is necessary to conduct a fault simulation campaign beforehand to determine the unique coverage of each pattern, generated by the LFSR, which can be utilized during the device's in-field data collection.

The algorithmic steps used to derive the unique coverage can be outlined as follows:

1. Perform a fault simulation with the maximum number of applicable patterns, while also tracking the state of the LFSR for each pattern applied;
2. Utilize the tracked LFSR states to apply the various patterns produced on silicon;
3. For each pattern:

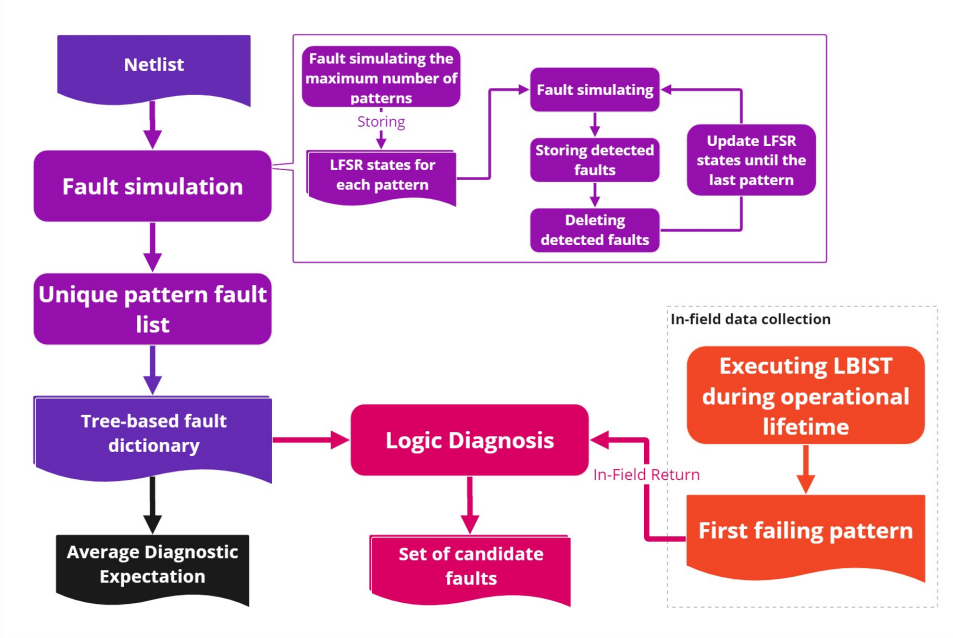


Fig. 3.16 General overview on the Logic Diagnosis of field return devices.

(a) Initialize the LFSR with the corresponding seed;

(b) Conduct fault simulation based on the fault list;

(c) Update the fault list by eliminating any detected faults;

(d) Collect the information necessary to construct the tree-based fault dictionary.

This process enables the extraction of all relevant information from the faults gathered during in-field data collection. By varying the seed of the LFSR, it becomes feasible to simulate faults for each pattern independently, facilitating the extraction of its unique coverage. After determining the unique coverages, the tree-based fault dictionary for the LBIST can be established following the algorithm described in Algorithm 8. The proposed methodology employs a tree structure [53, 70] to create a fault dictionary that includes all potential patterns the LBIST can generate.

Figure 2.10 illustrates a potential implementation in a scenario where the maximum number of patterns to be applied via the LBIST is three ($T1$, $T2$, and $T3$), and the fault universe (FU) is defined as $FU = \{A,B,C,D,E,F,G,H,I\}$, with each letter signifying a possible fault. By applying each pattern, only two new nodes are created, which is contrary to what occurs in non-LBIST situations, as described in

Section 2.7 and shown in Figure 2.10. In essence, for a tree-based fault dictionary for LBIST, the *pass* node forks into two *fail/pass* nodes for each applied pattern. On the other hand, the *fail* node does not fork; instead, it denotes all equivalence classes within the fault dictionary. The reasoning behind this is that the information obtained from the LBIST does not allow identification of which other test patterns may have also resulted in a failure after the initial one.

This restriction arises from the first failure the MISR encounters compromising its integrity. It thus fails to keep clear information regarding any later pattern that might also have failed. As such, information acquired from the very first failing pattern gathered in real-world circumstances is really important. Identification of candidate faults using the pre-computed fault dictionary relies on such information.

In particular, it stands as the only reference that can provide crucial insights necessary for effective diagnosis in these situations.

Algorithm 8 Tree-based fault dictionary for LBIST.

Require: $fault\_universe$, set containing all the faults.
Require: *unique_coverages*, list of sets containing the unique covered faults for each pattern.

$prev \leftarrow fault\_universe$
$fault\_dictionary \leftarrow [\emptyset]$
$i \leftarrow 0$
$N \leftarrow$ length(*unique_coverages*)
while $i < N$ **do** ▷ for each pattern
    $pass\_node \leftarrow prev - unique\_coverages[i]$
    $fail\_node \leftarrow prev \bigcap unique\_coverages[i]$
    $fault\_dictionary[i].pass \leftarrow pass\_node$
    $fault\_dictionary[i].fail \leftarrow fail\_node$
    $prev \leftarrow pass\_node$
    $i \leftarrow i + 1$
end while

Taking into account the tree-based fault dictionary illustrated in Figure 3.15, when a device is returned to the manufacturers with the indication that the initial failure pattern detected was the third one ($T3$), the set of potential faults corresponds to the *fail* node associated with that pattern, which is $\{F, G, H, I\}$. Therefore, once the fault dictionary has been established, the most direct method to conclude the logic diagnosis process is to access the *fail* node at the index of the first observed failure pattern, as detailed in Algorithm 9.

Algorithm 9 Logic diagnosis exploiting both tree-based fault dictionary and in-field collected information.

Require: $fault\_dictionary$, tree-based fault dictionary
Require: p_{ff}, index of the first failing pattern
1: $candidate\_faults \leftarrow fault\_dictionary[p_{ff}].fail$
2: **return** $candidate\_faults$

Chapter 4

Experimental results

This chapter reports the experimental results obtained for all the proposed methodologies and the main contributions illustrated in the previous chapters. It first describes the general experimental setup with the case study provided by STMicroelectronics, which has been used to demonstrate the proposals' effectiveness. The following sections detail the results obtained for each proposed approach.

4.1 Experimental setup

The considered industrial case study to validate all the proposals of the thesis is a large Automotive System-on-Chip produced by STMicroelectronics, with the following characteristics:

- 40*nm* technology;
- about 20 million gates;
- about 700 thousand flip-flops;
- multi-core architecture;
- ASIL-D compliant.

The SoC has a multicore architecture with three 32-bit cores using the PowerPC Variable-Length Encoding (VLE) instruction set. It has 6Mbyte of Flash memory

and 128Kbyte of general-purpose SRAM. There are several peripheral bridges to access a set of on/off-chip peripherals. Regarding the communication between components, they are interconnected to two, not equivalent, fast crossbar switches AHB-AMBA [71] at 64-bit and capable of working up to 200Mhz. Moreover, the two crossbars are linked with a cross-lake, which allows to tie-up crossbar one master with crossbars two slaves and vice-versa.

The complete fault list size of the case of study is composed of about 80 million Stuck-at and Transition delay faults. Table 4.1 reports the 7 LBIST partitions of the case study with the corresponding number of scan cells and scan chains, and faults summary, including SA and TRN faults.

Table 4.1 Faults summary for LBIST partitions of the industrial case of study.

LBIST partition	Number of scan cells	Number of scan chains	Number of SA + TRN faults
0	28K	800	4M
1	82K	1K	15M
2	56K	800	6M
3	61K	1K	5M
4	72K	1K	5M
5	69K	1K	6M
6	68K	800	9M

The case study supplied by STMicroelectronics includes also the industrial ATPG pattern set used during the production process.

To prove the methodologies an experimental setup has been implemented [56, 67] composed of:

- The physical board containing the SoC;
- An external tool responsible for managing LBIST parameters and facilitating serial communication with the board.

All the experiments regarding the pattern generation and the fault simulation have been executed on a high-performance multi-processor server equipped with a 64-bit 16-core processor AMD EPYC 7301, 256GB of RAM, a storage system of 10 TBytes, and a Centos Linux 7 operating system by using *TestKompress* from the Tessent-ATPG suite [72].

4.2 Scan-based Test Cost Optimization

This section details the experimental results obtained using the proposed comprehensive scan-based cost model presented in Section 3.1. The results present in this section have been published in [62] and [64].

Initially, the cost analysis of the pattern set that is at present in use in manufacturing is presented. Then, using information from a sacrificial lot of devices, the failure distribution of the tested devices is investigated and candidate faults are categorized according to their criticality level. Examined will be the test cost analysis, enabling for both uniform and non-uniform failure distributions, juxtaposing their cost projections with real-world production data collected over a six-month period.

At last, the pattern set generated by the *criticality-oriented ATPG* algorithm is shown to show how the proposed approach maximizes test costs for devices reflecting non-uniform failure distribution.

The comparison with the model presented by [10] for determining the quantity of useless patterns is also documented.

Figure 4.1 illustrates the increase in fault coverage of the pattern set relative to the percentage of patterns applied. The company supplies and rearranges the patterns, hence the trend does not adhere to the generation process of the ATPG tool. The reordering of patterns arises from a field evaluation procedure conducted during mass manufacturing in accordance with established industrial techniques that remain confidential. The current application of the pattern set serves as the foundation for the analysis, aiming to demonstrate that even with a consolidated pattern set, it is feasible to get a substantial benefit by eliminating patterns from the tail during Wafer Sort.

In Figure 4.2 the trend cost, assuming uniform failure distribution, is pictured using the described formulas in Section 2.2, with high production yield and high

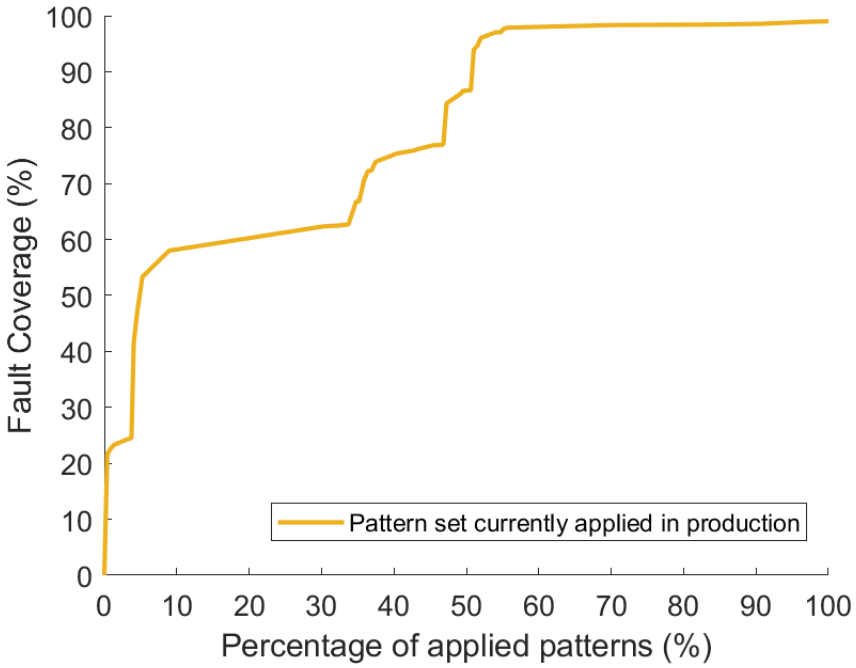


Fig. 4.1 Pattern set currently used in production: Fault coverage vs Percentage of applied patterns, ordered by the sequence of application

packaging and package test cost. Exact numbers cannot be disclosed and percentages are reported.

Using the cost model considering the uniform failure distribution, the estimated economic gain that would have been achieved reducing the pattern set currently used in production, only during Wafer Sort, is around 3.17% per device.

Moreover, in the next section, the data analysis of the failure distribution will be detailed. Such an analysis will then be exploited to estimate the cost trend using the proposed cost model, which also considers the non-uniform failure distribution. Both estimations will also be compared to real production data.

4.2.1 Faults' Weight Computation

The proposed cost model uses density as an indicator and supposes a non-uniform failure distribution. Using the Continue-on-Fail flow, a sacrificial set of devices of a meaningful population has been examined to identify which faults are more likely to occur and to assign them a weight dependent on a criticality index.

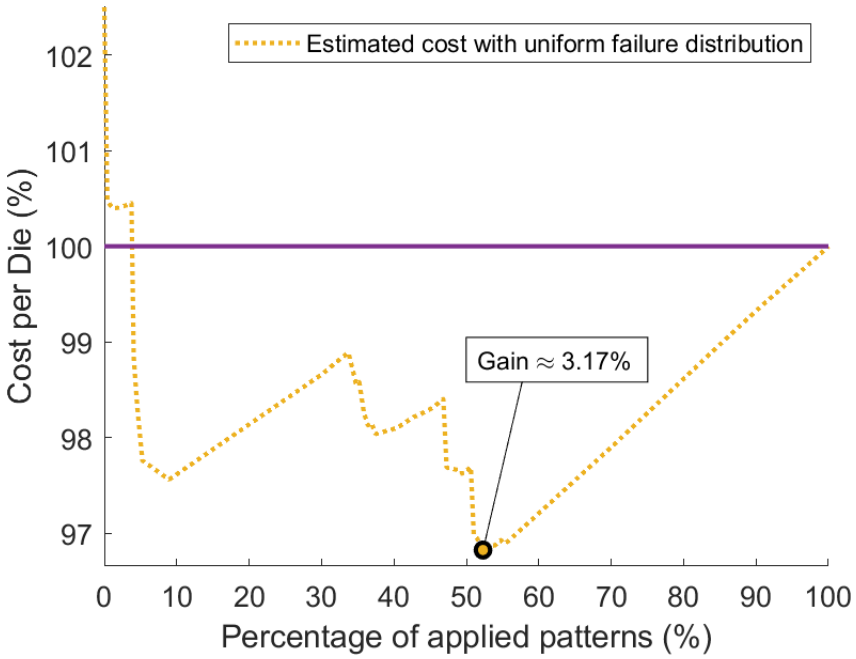


Fig. 4.2 Pattern set currently used in production: Estimated Cost per Die (%) considering uniform failure distribution vs Percentage of applied patterns (%)

Following fault simulation each applied pattern set to extract the individual list of covered faults and the matching fault candidates, a Rust language tool has been built to apply the algorithm detailed in Algorithm 3. This step produces a ranked list of candidate faults of the failing dies at its conclusion. The density metric (deeply discussed in [35]) has been applied during the phase of assignment of the weights. A *density class* is then regarded as a criticality class. Usually defined as the input pins distance of a *AND* gate [35], a density class especially refers to the number of neighbors each gate has upon such specified distance.

The case study layout is shown in Figure 2.8 as an example of non-uniform gate distribution in the front-end. The gate density on the physical layout is not uniformly distributed over the entire device surface.

The faults are categorized by their density class and the such information are then crossed with the candidate faults resulted from the tested sacrificial lot. As described in subsection 3.1.2, the weights of each fault are computed and used to finally calculate the Weighted Fault Coverage (Ω) that will be used in the next section to illustrate the proposed cost model.

The representation in Figure 4.3 provides a general understanding of candidates distribution in relation to gate density. The x axis represents the density classes of the device, and the bars identify the percentage of how many candidate were found in that specific class. It shows that regions with medium to low gate density tend to have fewer faults.

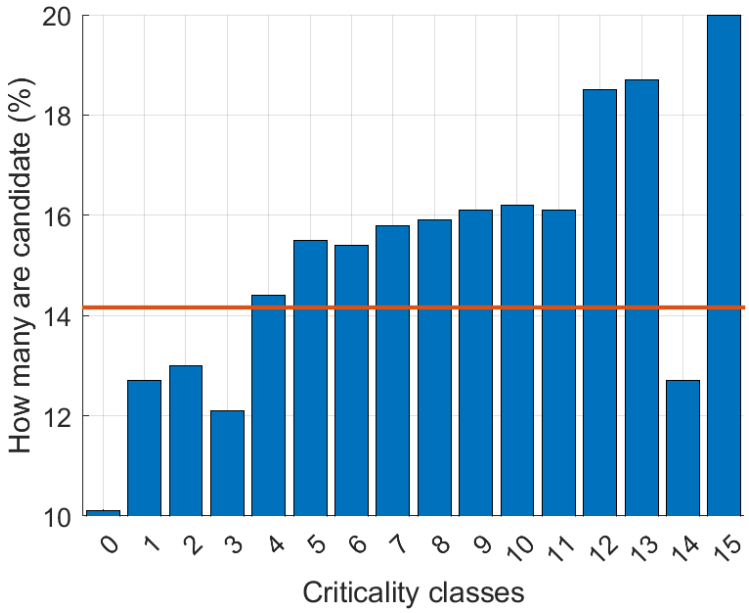


Fig. 4.3 Percentage of candidates for each criticality class (number of neighbors)

In particular, the latter wants to highlight the correlation between the number of elements in a given criticality class and how many of them were found as a candidate. A horizontal line labeled 14.16% shows the fraction of all the cells that have been a fault candidate at least once. As the experiments show, the density classes 0 to 3, which contain around 60% of all gates, remain below the 14.16% mark. From classes 4 to 15, instead, clearly above the demarcation line, demonstrating that a more significant amount of faults tend to occur in the densest areas of the device.

The criticality classes are ranked depending on the percentage of candidates that were found in that specific class. Thus, the computation of the weights for the faults is done by considering the maximum critical class (15 neighbors) the one with the maximum weight (1), and then the assignment proceeds following the ranking, as described in Algorithm 3.

4.2.2 Test Cost Analysis

Already presented in Section 2.2 is an equation to approximate the cost of cutting patterns at Wafer Sort accepting extra charges for packaging and during Package Test. That has been calculated, nevertheless, under an uniform failure distribution. The actual production statistics can vary from the made estimate, that is based just on the fault coverage: not all the faults are always equally likely.

Figure 4.4 shows:

1. the cost trend modeled by the cost model assuming uniform failure distribution;
2. the real costs coming from six months of production;
3. the trend modeled by the proposed cost model considering non-uniform failure distribution.

All the curves are dependent upon the amount of patterns utilized during Wafer Sort, as the objective of the work is to demonstrate how cutting time in Wafer Sort might yield economic benefits, despite the rise in test escapes to Package Test. In the context of actual production data, the quantity of test escapes is not an estimation but rather the real number.

Although the estimation of uniform failure distribution does not quite align with the actual production cost curve, they exhibit similar characteristics. Upon zooming the figure at the minimum point of the cost function (Figure 4.5), it is evident that the estimation of the non-uniform model, predicated on the Weighted Fault Coverage (Ω), exhibits marginally greater precision than the uniform model, which relies exclusively on the Fault Coverage. The points depicted in Figure 4.5 relate to the actual production benefits that could have been realized based on the cut point. The zoomed-in section of Figure 4.5, focusing on the area of minimal cost, illustrates that a minor error in cutting patterns during Wafer Sort could result in diminished economic returns, despite the error being quite insignificant. The largest economic benefit is approximately 3.1% per device, corresponding to about 52% of applied patterns. If the reduction is approximately 49% of patterns, the economic benefit diminishes to 2.3%; if the reduction is approximately 47%, the economic benefit diminishes to 1.3%. Consequently, if the cut occurs prematurely, it may result in diminished economic benefits or, worse, an increase in costs. A similar outcome

may occur if the incision is made excessively distant. Identifying the appropriate point for cutting the pattern set is essential for maximizing economic benefit.

Based on data collected over a six-month period, the greatest economic gain achievable by cutting the current production pattern merely during Wafer Sort is roughly 3.1% per device. While the proposed non-uniform model yields a more exact measurement of 3.13%, the uniform cost model calculated the largest economic gain at 3.17%. Because the pattern set was constructed without a precise guide, it is expected that the values at the minimum point are comparable. As therefore, the coverage of the faults has been evenly spread.

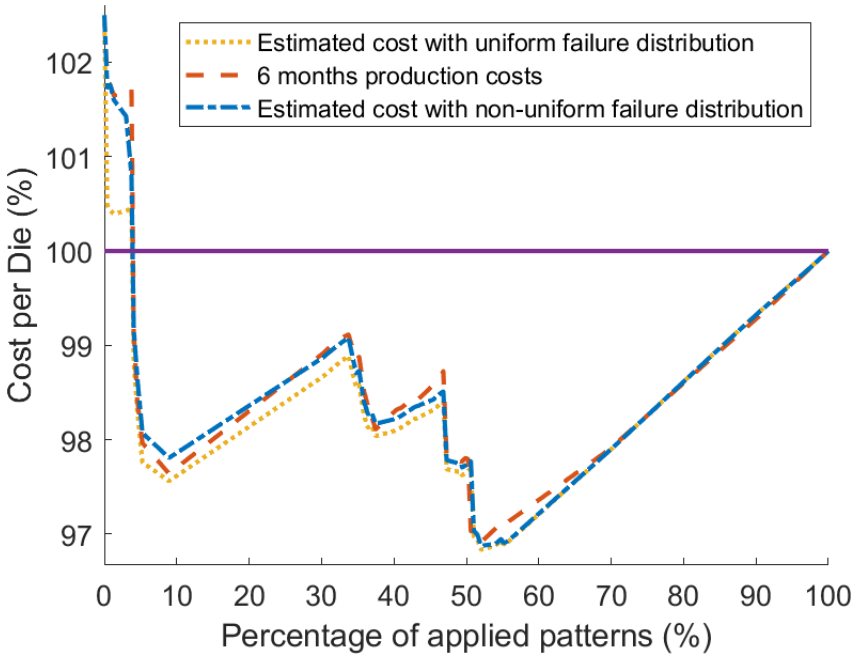


Fig. 4.4 Estimated costs and six months production cost vs Percentage of applied patterns

Table 4.2 shows the Pearson correlation index and the average percentage errors along six months of production for both uniform and non-uniform failure distribution model. The latter achieves 2% more correlation and it has a lower average error, looking at the cumulative values (0.07% vs 0.16%).

To further optimize the achievable economic gain, in the next section results for a new pattern set that also considers the non-uniform failure distribution are reported.

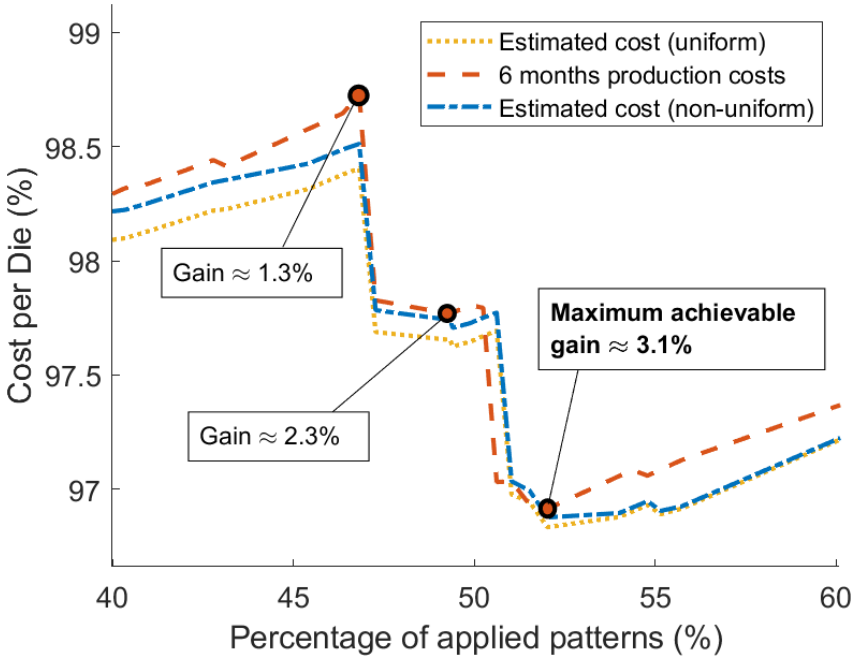


Fig. 4.5 Zoom over the region of minimum cost

Table 4.2 Accuracy of the models in respect to the results collected along six months of production

Month [#]	**Pearson correlation index**		**Average error [%]**	
	Uniform	**Non-uniform**	Uniform	**Non-uniform**
1	0.966	0.991	0.17	0.07
2	0.967	0.991	0.17	0.07
3	0.968	0.991	0.16	0.07
4	0.968	0.990	0.16	0.08
5	0.967	0.991	0.17	0.07
6	0.969	0.990	0.15	0.08
Cumulative	0.968	0.991	0.16	0.07

4.2.3 Criticality-oriented pattern set generation

To achieve greater economic benefit by exploiting the non-uniform failure distribution of the device, a novel pattern set that also accounts for the criticality of faults has been generated. Patterns are created to address both Stuck-at and Transition delay faults. These fault models for the examined technology more accurately describe the population's failure distribution. Further reasoning is required for not-modeled faults.

A comparison can now be presented between the existing production pattern and the proposed criticality-oriented pattern. The *criticality-oriented ATPG* algorithm, detailed in Algorithm 4, has been employed to assess the ranking of fault criticality for pattern generation. The number of patterns generated equals the number utilized in production.

Figure 4.6 illustrates the coverage achieved with the suggested criticality-oriented methodology in comparison to the pattern set presently utilized in production. At approximately 50% of the applied patterns, the original pattern set surpasses the criticality-oriented set in terms of coverage. This behavior is expected as the Algorithm 4 method necessitates increased ATPG efforts. Consequently, it is plausible that the trend will increase at a diminished rate following the identification of the most critical faults. To achieve optimal fault coverage with the new generation, following the completion of the subsequent generation and the ranking of faults (approximately 75% of the original patterns), an additional pattern generation (with a *abort limit* set to 1000, *maximum compaction effort*, and *high coverage effort*) is performed on the remaining faults.

Nonetheless, the primary objective of the technique is to shift the most critical faults to the left as much as possible.

This is further illustrated by comparing the resultant test costs in Figure 4.7. A more pronounced rise in fault coverage during the initial segment of the curve allows for a more substantial reduction in costs if the number of applied patterns is decreased beyond the original curve.

The cost of the scan tests is even more optimized based on the newly created pattern set. In Figure 4.7, the highest economic benefit attainable using these patterns is around 6.1% per device, nearly double the benefit obtained by minimizing the initial pattern set, which was generated without any guidance on failure distribution.

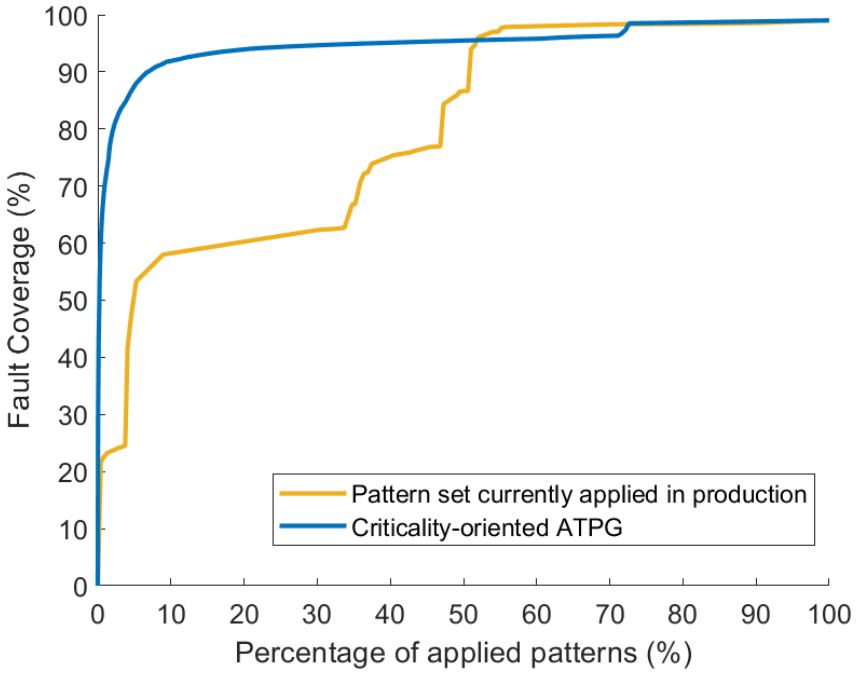


Fig. 4.6 Fault coverage vs Percentage of applied patterns

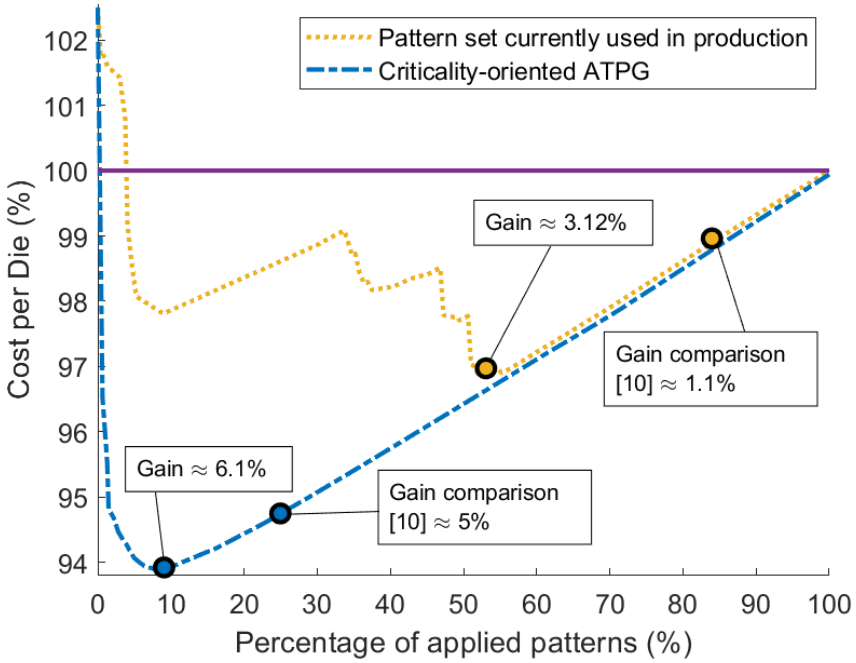


Fig. 4.7 Non-uniform failure distribution: Cost per Die (%) vs Percentage of applied patterns (%)

4.2.4 Comparison with another test cost model

As outlined in Chapter 2, the authors in [10] examine potential useless patterns employed in industrial production. The study presents a model that considers the quantity of tested devices, the manufacturing yield, and a function (F_D) that indicates the number of tested devices identified as failed for each pattern. The equation for determining the quantity of useless patterns is presented in Equation 4.1, where T denotes the total number of patterns, N represents the device population, and Y signifies the production yield.

$$E = \sum_{k=1}^{T} (1 - F_D(k) + F_D(k-1))^{N \cdot (1-Y)} \tag{4.1}$$

The points identified by the model proposed by [10] are also depicted in Figure 4.7. The formula proposed by the latter study is specifically utilized for the purpose of estimating the number of patterns that are not useful, for both the pattern set that is utilized in production and the pattern set that is generated by the criticality-oriented ATPG algorithm. To estimate the F_D function, production data spanning six months were gathered, and as illustrated in Figure 4.4 and Figure 4.5, the non-uniform model precisely forecasts the production trend (Table 4.2). Consequently, the proposed model is also suitable to approximate the F_D function for both sets of patterns.

Actually, about 86.5% of the patterns in the currently used pattern set in manufacturing would be beneficial. Eliminating all other patterns would thus lead to an estimated 1.1% economic benefit. According to the patterns generated by the proposed approach, about 26% are projected to be truly advantageous, so generating an economic advantage of almost 5%.

The next section offers a thorough comparison among multiple state-of-the-art test cost-cutting strategies.

4.2.5 Summary comparison with state-of-the-art test cost reduction methodologies

The proposed methodology depends on the implementation of a comprehensive test cost model that can properly forecast optimal point where to cut patterns during

Wafer Sort to minimize test duration and expenses while managing additional costs during Package Test.

State-of-the-art approaches for reducing test costs predominantly depend on pattern reordering [5, 6]. Prioritizing patterns for optimal effectiveness influences the final cost; nonetheless, the potential economic benefit is solely restricted by the number of faulty devices. The good devices would, in any event, undergo testing with the entire pattern set, hence confining the methodology for reduction to single-site test environment. In contrast, the proposed method and that outlined in [10] propose a strategy to achieve economic gain across all devices, including the good ones. Companies producing high-yield items, along with a limited number of defective devices, would gain more from employing a system that minimizes costs across all devices, rather than solely addressing the failed ones.

Table 4.3 Summary comparison among Test Cost Reduction methodologies

Method	**Economic Gain [%]**	**Type**	**Tester configuration**	**Test escapes handling**	**Data input**	**Outcome**	**Reported case study**
[5]	No gain	Static	Single-site	Not handled	Statistical data from production	Reordered patterns based on production	ISCAS'89 benchmarks [60]
[6]	0.8	Adaptive	Single-site	No test escapes	Information about the previous tested wafer	Reordered patterns at each tested wafer	ISCAS'89 benchmarks [60]
[10]	1.1	Static	Single-site and multi-site	Not handled	Statistical data from production	Prediction of useless patterns	IBM devices
Proposed methodology	6.1	Static	Single-site and multi-site	Covered by successive test phases	Circuit information and statistical data from production	Model to find the best point to cut patterns	Academic benchmark and 40*nm* real-world Automotive SoC

Table 4.3 presents a comparative summary of previously mentioned state-of-the-art test cost reduction methodologies. It is important to highlight that the proposed methodology yields the most economic benefit when utilizing the criticality-oriented pattern set. Furthermore, it is the sole solution capable of adapting to both single and multi-site testing environments, effectively managing potential further test escapes throughout subsequent testing phases. Actual production data for the industrial case study further corroborate the proposed research. Ultimately, it offers a model that is adaptable to any existing pattern set while simultaneously yielding a newly generated one that accounts for the device's criticality to optimize economic benefits.

4.2.6 Computational time and resources

Table 4.4 details the computational time and resources needed for the proposed methodology, categorized into points *A* and *B* shown in Figure 3.1. The *calculation of the candidate faults* phase is the longest. It includes fault simulation for all patterns unless the lists of covered faults by every pattern are already available. The *analysis of the candidates' criticality* phase refers to the density analysis of the layout. The *weights' computation* phase crosses the previously computed data to find the criticality level of the candidate faults. Finally, the proposed cost model processes all this information quickly.

Table 4.4 Computational time and resources needed.

Circuit	Gates [Million]	Calculation of candidate faults		Analysis of candidates criticality		Weights' computation		Cost modeling	
		Time [d]	Threads [#]	Time [s]	RAM [GB]	Time [s]	RAM [GB]	Time [s]	RAM [GB]
Academic	1	0.5	16	85	2	$\ll 1$	2	$\ll 1$	0.25
Industrial	20	6	32	180	8	$\ll 1$	8	$\ll 1$	0.25

Scalability discussion

The proposed methodology has been applied to an industrial device with 40 nm technology and with approximately 20 million gates. For larger and more complex devices, as shown in Table 4.4, the most time-consuming step in creating the model is the calculation of candidate faults. For instance, for a device with 90 million gates, model creation could take up to fifteen days, using 64 threads. In any case, because this step is performed only once to act as training, it can be reused for other devices having the same technology.

Another critical step in terms of scalability lies in the fault simulations phase, but in the industrial processes of any device, this phase is already part of the manufacturing and engineering process.

The final, additional phase, optional to maximize gain, is the generation of a new criticality-oriented pattern set. For this last step, the timing depends on the ATPG tool being used and the resources employed. Generating a new criticality-oriented pattern set is comparable in duration to the original pattern set creation process.

Beyond the time considerations, the proposed methodology exhibits strong scalability in the context of various testing environments; in fact, it can be employed in both single-site and multi-site setups, without losing its economic benefits, as it happens with methodologies based on pattern reordering [5, 6]. In addition, adaptive methodologies [6] also require real-time analyses performed by ATEs that may not be available in all test environments, further restricting the applicability of the approach.

In contrast, the methodology proposed by [10], relying solely on production data modeling, is shown to be scalable even for larger and more complex devices. However, as already seen in subsection 4.2.4, the proposed model is more accurate in predicting the optimal point to cut patterns. In addition, the proposed methodology also handles test escapes, which are critical for industries that produce large-scale products, unlike some of other methodologies discussed [5, 10].

4.3 System-Level-Test programs assessment

This section details the experimental results obtained grading a suite of System-Level-Test programs that has been generated to specific target un-core logic of very large SoCs. Firsty, a small summary, together with the fault coverages, on the programs is given. Then, the assessment of the programs is described through the use of venn diagrams to show how the categorized faults overlay with the SLT programs coverages. The experimental results of this section are under major revision in *IEEE Transactions on Computers* as part of the article titled: "*Automatic Generation of System-Level Test for un-core logic of large Automotive SoC*".

4.3.1 The functional SLT suite

Name	Tot. # Actions	Execution Time [cc]	Memory Footprint [Kbyte]	Fault Sim. Time [days][1] (est.)	# Fault Sim. Partitions	Avg. Fault Sim. Time per Partition [days]
App1	3,707	26,481,222	695	129.3	5	12.9
App2	6,536	51,071,225	870	249.3	7	9.9
App3	2,755	20,279,995	497	99.0	5	3.45
App4	24,051	161,110,659	1,894	786.5	21	11.4

Table 4.5 Generated SLT application suite characteristics targeting the crossbar modules.

[1] Stuck-at + Transition Delay fault models for a total of ca. 270k faults.

Table 4.5 presents the characteristics of the considered and assessed SLT functional suite, that has been generated to target the un-core logic of very large SoCs.

4.3.2 Fault coverage

Considering the described SLT suite, the proposed categorization method described in Section 3.2 is used. Particularly, the two crossbars of the case study are deeply analyzed.

The functional programs have been simulated for the Stuck-at fault model (a total of 270k faults among the two crossbars), and experimental results are presented in Table 4.6, in which the column *Single* represents the coverage of a single test approach, the column *Incr* represents the cumulative coverage of a given test approach with the previous approaches, and the column Δ represents the increment for a given

test approach with respect to the previous one. Moreover, ATPG untestable identified faults have been added in the fault coverage of structural test patterns; meanwhile functionally untestable faults following the approach identified in [66] are added to functional approaches.

Test Nature	**Test Approach**	**Stuck-at Fault coverage [%]**					
		XBAR1			**XBAR0**		
		Single	**Incr**	**Δ**	**Single**	**Incr**	**Δ**
Structural	Scan-based	94.07	94.07	NA	93.19	93.19	NA
	LBIST	51.99	95.18	1.11	50.44	94.28	1.09
Functional	App1+App2+ App3+App4	49.7	97.08	1.9	53.66	95.95	1.67
	RTOS boot	25.31	97.08	0.0	19.13	95.95	0.00
		Total	**97.08**	**1.9**	**Total**	**95.95**	**1.67**

Table 4.6 Fault coverage for Stuck-at fault model (ca. 270k faults).

The Stuck-at fault coverage achieved by the structural tests (scan-based and LBIST) reaches 95.18% for crossbar 1 (XBAR1) and 94.28% for crossbar 0 (XBAR0), removing identified redundant faults. Combining functional programs in the SLT suite with visiting algorithms provides additional fault coverage for high results. It adds up to 97.08% for crossbar 1 (XBAR1) and 95.95% for crossbar 0 (XBAR0). The introduction of an RTOS boot shows that the incremental fault coverage gain is entirely contained in the proposed SLT suite for SAF model. At the same time, it has a low fault coverage that can be translated into less detection capabilities.

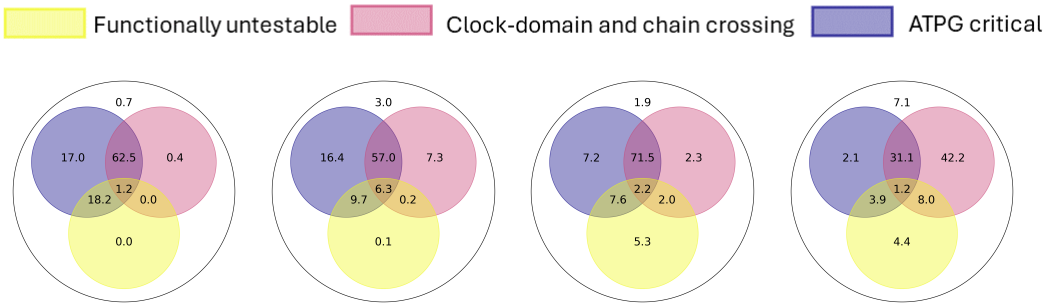
Moreover, the proposed SLT suite has been fault simulated also for the Transition Delay fault model (a total of 270k faults among the two crossbars), and experimental results are presented in Table 4.7.

The Transition Delay fault coverage achieved by the structural tests (scan-based and LBIST) reaches 88.71% for crossbar 1 and 87.49% for crossbar 0, removing identified redundant faults. Combining functional programs in the SLT suite with visiting algorithms provides additional fault coverage for high results. It adds up to 91.11 % for crossbar 1 (XBAR1) and 89.76% for crosbar 0 (XBAR0). The introduction of an RTOS boot shows that the incremental fault coverage gain is entirely contained in the SLT suite for TDF model. At the same time, it has a low fault coverage that can be translated into less detection capabilities.

Test Nature	Test Approach	Transition Delay Fault coverage [%]					
		XBAR1			XBAR0		
		Single	Incr	Δ	Single	Incr	Δ
Structural	Scan-based	87.28	87.28	NA	87.33	87.33	NA
	LBIST	4.08	88.71	1.43	3.15	87.49	0.16
Functional	App1+App2+ App3+App4	29.32	91.11	2.4	31.61	89.76	2.27
	RTOS boot	13.09	91.11	0.0	2.99	89.76	0.0
		Total	**91.11**	**2.4**	**Total**	**89.76**	**2.27**

Table 4.7 Fault coverage for Transition Delay fault model (ca. 270k faults).

Firstly, the undetected faults from structural tests are analyzed and labeled following the considerations presented in Section 2.4 in Figure 4.8. For readability reasons, the categories described in Section 2.4 as *clock-domain crossing* and *chain configurations crossing* are merged into a single category, *clock-domain and chain crossing*, in all the figures. Figure 4.8 shows that the majority of undetected stuck-at faults resides in the ATPG critical class. On the other hand, the number of undetected faults for the Transition Delay fault models is bigger compared to undetected stuck-at, and they spread among different classes.



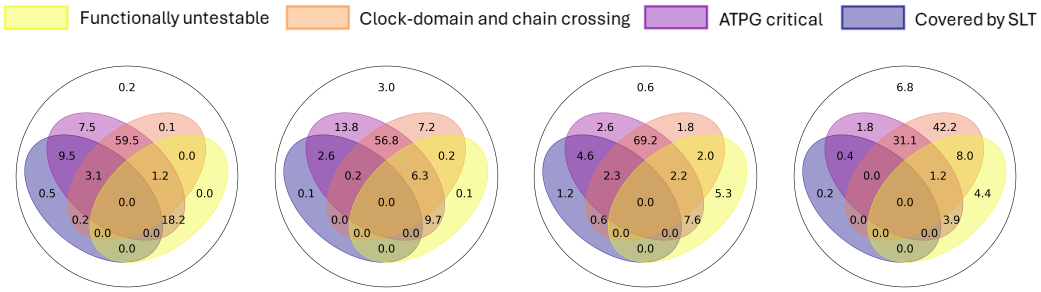
(b) Xbar1 SAF (ca. 5k). (c) Xbar1 TDF (ca. 12k). (d) Xbar0 SAF (ca. 8.5k). (e) Xbar0 TDF (ca. 19k).

Fig. 4.8 Classification of Undetected faults from structural tests for different fault models. All the values illustrated are shown in percentages.

Then, after having fault simulated the System-Level Test programs to extract the coverages for the modules, the final considerations follow. Figure 4.9 shows the classification of undetected faults from structural tests and the percentage for each labeled class of faults detected by the SLT suite. On one hand, some faults detected

by SLT are still unclassified; on the other hand, other faults detected by SLT resides in different classes (i.e., structural pattern weaknesses).

The total increase of fault coverage for the TDF model is more significant than the increase for the SAF model. The difference is because scan-based test patterns for modules between different cross-domain clocks are hard to detect [42], and they would require a big structural pattern set. Instead, experimental results show how SLT is capable of detecting such faults in a functional and in-field-like manner without adding a considerable set of structural patterns and that the presence of functionally untestable faults is considerable, as Figure 4.9 shows.



(b) Xbar1 SAF (ca 4k). (c) Xbar1 TDF (ca. 11k). (d) Xbar0 SAF (ca. 7.5k). (e) Xbar0 TDF (ca. 18k).

Fig. 4.9 Classification of Undetected faults from structural tests and SLT covered faults. All the values illustrated are shown in percentages.

Scalability discussion

The proposed assessment of SLT programs is realized through two key steps: the use of a commercial ATPG tool to extract circuit information (such as gates, logic cones, different scan chain configurations, and clock domains); a subsequent tool developed to analyze the information and extract the various categories of fault locations.

Table 4.8 shows the time required for the ATPG tool to extract the needed information for different modules, using 16 threads.

As reasonable, the extraction is longer for modules with a larger number of gates, so for very large devices this time could increase further. Nonetheless, to have proof of how the SLT is going to cover the un-core logic, it is possible to extract as an example the information for only a few modules, thus making the analysis more practical in terms of time.

Module	# of gates	Extraction time [d]
XBAR0	78k	2
XBAR1	56k	1.8
Peripheral bridge	38k	1
CAN Peripheral	217k	3

Table 4.8 Time needed by the ATPG tool to extract information about gates across various modules

4.4 Stress stimuli evaluation: a monster Automotive case study

All the results reported in this section have been carried out along a visiting research period at STMicroelectronics in Agrate Brianza site. Contrarily to the other experimental results shown in this chapter, this study has been conducted on a different, larger and more complex device: an Advanced Driver Assistance System (ADAS) SoC produced by STMicroelectronics. Table 4.9 summarizes its main characteristics.

Table 4.9 Case study specifications summary

Number of scan cells	Number of scan chains in Burn-In	Technology	Number of faults
4M	3	7 *nm*	187M

Table 4.10 shows the preliminary results obtained through different strategies of stress stimuli generation. All the values takes into account also the toggling activity happening during the test mode entry phase, that are typically not computed by the commercial ATPG tools.

The pattern set initially in use was the **5 ATPG** pattern set, composed of 4 patterns with chain integrity fixed sequences and 1 pattern generated by the ATPG. By using the strategy described in Section 3.3, the toggle coverage has been computed, showing a value of 84.7%.

Starting from that pattern set, two more ones have been generated through the use of the ATPG (**32 ATPG** and **64 ATPG**), both showing high values of toggle coverage, respectively 90.6% and 91.8%. The experiments have been conducted putting the maximum efforts to reach high coverage through the *IDDQ toggle* fault model. Such deterministic pattern sets would require a lot of memory to store them, considering the length of the scan chains detailed in Table 4.9. For this reason, for such a huge case study, pseudo-random generation techniques have been explored to further improve the stress applicable during the System-Level test phase.

For the pseudo-random generation a software LFSR module has been used, with a configuration with 32 bits for the seed. The initial seed to compute these experimental results has been chosen over a $1,000$ of possible random seeds, choosing the one

showing the maximum reached toggle coverage. Anyway, the toggle coverage reached by the pseudo-random patterns (**64 Pseudo-Random**) and the corresponding ATPG-based (**64 ATPG**) is pretty similar. The key difference, as discussed in [14], is that using the pseudo-random generator is more efficient in terms of memory because patterns do not need to be saved, as it happens in the case of applying ATPG-based deterministic patterns.

Moreover, also the case in which it is necessary to activate one chain at a time has been analyzed. Indeed, the last pattern set **12 * 64 Pseudo-Random** is composed of twelve pattern sets of 64 patterns each, for a total of 768 patterns. Each of the latter maintains the same state for two out of the three total scan chains and apply the pseudo-random generated sequence just to the remaining one. This specific type of pattern set could be useful especially if it is not possible to load multiple parallel scan chains during the System-Level Test because of buffer memory constraints of the application board.

Table 4.10 Comparison between different pattern set generation strategies

Pattern Set Name	Strategy	Toggle Coverage [%]
5 ATPG (production)	4 Chain Integrity + 1 ATPG	84.7
32 ATPG	ATPG-based	90.6
64 ATPG	ATPG-based	91.8
64 Pseudo-Random	LFSR	91.2
12 * 64 Pseudo-Random	LFSR (one chain at a time)	92.2

Scalability discussion

The results shown in Table 4.10 refer already to a very large case study, examples of which are hard to find in literature. Additionally, the same methodology was previously applied to a smaller device, as discussed in [35, 14]. Therefore, the presented results demonstrate how adaptability in larger devices is possible for such a methodology. Especially because the applied stress stimuli do not need to be saved

in memory; instead, they are generated on-the-fly by the tester, allowing the approach to adapt even more complex cases.

The most crucial step for the scalability of the approach is the preliminary evaluation of the stimuli through logic simulation and the subsequent generation of the VCD for analysis. Indeed, the time to perform such a process strictly depends on the size of the device and the used resources. However, this step is performed at the initial stage of LFSR seed selection and validation of the required toggle coverage and is no longer necessary thereafter.

4.5 In-field data collection and Logic diagnosis for a batch of faulty devices

This section outlines the experimental results derived from the methodology outlined in Section 3.4, which was applied to simulated circuits from the academic benchmarks ITC'99 [61] and the industrial case study detailed in Section 4.1. Part of these results have been published in [67].

The Diagnostic Expectation (DE) values were obtained by applying the complete flow to faulty devices during their operational lifetime. Furthermore, the outcomes from real faulty devices using the proposed methodology were compared with those from the methodologies outlined in [24, 25], as discussed in Section 2.10. These methodologies were selected for comparison because, similar to the proposed approach, they rely solely on LBIST signatures without necessitating architectural modifications to identify the cause of failure. These methods were also applied to real failed industrial devices to evaluate the resulting diagnostic results (DR).

Additionally, the proposed methodology was compared to the simulation in [26], which utilizes a storage-based LBIST scheme, focusing on benchmark circuits and the simulation of DE for the industrial case study.

Finally, simulation results are presented, illustrating how the methodology detailed in [21] might function with an architectural change in the referenced industrial case study. At the end of this section, a high-level comparison of the various state-of-the-art methodologies is provided, emphasizing key differences and scenarios where the proposed approach is particularly advantageous.

The main component of the external software tool used in the experimental setup is tasked with coordinating LBIST experiments, implementing a dichotomic search for the proposed method, and handling communication with both the circuit board and power supply.

It's important to note that environmental factors such as temperature and operating conditions can greatly influence the diagnostic information collected and, consequently, the final diagnosis. To address this, initial experiments were conducted on a non-faulty chip over several hours of continuous BIST execution under varying parameters. This characterization process enabled the identification of parameter sets where tests should ideally pass and those that should be avoided, as excessive

stress or temperature (like heating the Device Under Test due to BIST execution) could lead a non-faulty device to produce incorrect signatures.

4.5.1 FLASH Footprint

Each of the 7 LBIST partitions on the board is equipped with 16 bits of programmable PCS, which initially sets the LBISTs to a starting PCS value of $0xFFFF$. This value is subsequently halved or modified through a dichotomic search method in each iteration, as outlined in Algorithm 7.

In the framework presented in Section 3.4.1, an entry in the *frequency region* is represented as a 4-byte floating point number. For both the *golden and failure* regions, each entry has a width of 80 bits: 16 bits for the index and 64 bits for the signature. The FLASH memory footprint is dependent on the chosen fault model.

For TRN modeling, the total structure consumes $32+7\times80\times((\log_2(0xFFFF)+1))=9,552$ bits, or roughly 9 Kb. Conversely, in the SA model, since the *frequency region* and the indexes for the *golden region* can be omitted, the structure size balloons to $7\times64\times(0xFFFF+1)+16=29,360,144$ bits, which is approximately 29 Mb. To mitigate the memory footprint associated with SA fault modeling, a *skip* step technique can be applied to rows in the golden signature table. However, a signature mismatch in this case may result in a trade-off, potentially leading to a lower DR.

Nonetheless, if golden signatures are strategically stored—following the fault coverage curve, for example—the impact on resolution can be minimized.

It is important to highlight that the methodology described in this study is particularly effective for gathering data on TRN fault detection, which remains the main focus.

4.5.2 Signature collection time

In the case of study, the self-test process requires around 100 ms to complete when LBIST are set to simultaneously test the maximum quantity of patterns $0xFFFF$. When completed, the device performs a functional reset that resets the cores while preserving the activity of the STCU for signature collecting. Thus, assuming the

device is operating as expected, just one LBIS execution is needed throughout the power-on/off of the device, so determining the procedure length of 100 ms.

First estimated upon a preliminary signature inconsistency is the operational frequency. Starting at the highest frequency and lowering it by a designated Δ this method uses LBIS, making maximum use of patterns, until the tests are successful. Starting frequency 200 MHz and $\Delta = 10$ MHz, the ideal scenario mandates that the operating frequency be $max_f - \Delta$ thereby requiring only a single iteration and an execution time 100 ms. With an execution time of $100 \times \frac{200}{\Delta} = 2,000$ the worst-case scenario is represented by reaching Δ.

At this point a dichotomic search—described in Algorithm 7—which involves running LBIST a total of 16 times with either decreasing or increasing numbers of patterns—is started. Under the worst-case scenario, in which the dichotomic search finishes $0x655 - 1$ patterns, the total execution time needed is around $100 + \sum_{i=1}^{15} \frac{100 \cdot (2^i - 1)}{2^i} \sim 1,500$ ms. This estimate takes into for the reality that the upper dichotomy is always chosen. On the other hand, in the best-case scenario—where the index to be discovered is 1 indicating that the lower dichotomy is always selected—the necessary time is almost $\sum_{i=0}^{15} \frac{100}{2^i}$ ms. It should be mentioned that since a golden signature must be obtained for every execution, absent golden signatures from the database would double the execution time.

The given data just cover the LBIST execution time and neglect the time used for the FLASH erase and program operations. Though these activities are carried out in every iteration, their respective durations of 0.0192 ms and 0.00475 ms are regarded as minor in relation to the LBIST execution time.

4.5.3 Collected data from faulty devices

To identify a better range to target TRN faults, LBISTs signatures were collected in the experimental setup at the core reference voltage with increasing frequencies. The frequency used depends on a parameter Δ which denotes the quantity by which the frequency is lowered every time, equal to 10 MHz. Regarding registers, the least acceptable and lawful in the suggested scenario of research is 2 MHz. Finding at which frequency the DUT does not fail comes first in the proposed method. When using experimental observations in the industrial case study, it was more efficient to first decrease the frequency by an initial 10 MHz until the device under test (DUT)

did not fail, and then to begin a dichotomic search in order to fine-tune the computed frequency based on the increase or drop of 2 MHz. The results for twenty-three defective devices are displayed in Figure 4.10.

Fig. 4.10 Failures information for the industrial case study for field return devices depending on the BIST execution frequency.

4.5.4 Diagnostic Expectation

The average DE is computed following the building of the tree-based fault dictionary described in subsection 3.4.2. Average DE for every LBIST partition is shown in Table 4.12. The stated values reflect the average number of candidate faults once a misbehavior is recorded. The average DE is computed by means of a tree-based fault dictionary as the total number of candidate faults inside leaf nodes divided by the total number of patterns executed.

Building the fault dictionary for the LBIST entails omitting the failure branch of a previously *failed* node, hence limiting diagnostic information to merely the first failing pattern, as previously discussed in Section 2.7.

Under the worst-case situation, in which the first failing pattern either corresponds with one of the early patterns or develops shortly later, the average DE usually quite high.

For example, LBIST partition 5 in the industrial case study shows on average about 109 possible faults. When taken into account with the total fault list size for

the partition, which is roughly 6 million, this value stays fair even if it is somewhat high.

But since the failed pattern is found outside the 64^{th} location, concentrating on LBIST partition 5 causes the average DE to show a clear drop. Particularly, it is about 37.

Figure 4.11 demonstrates the link between the position of the first failing pattern and the resultant DE, which shows that DE is rather influenced by the position of the first failing pattern.

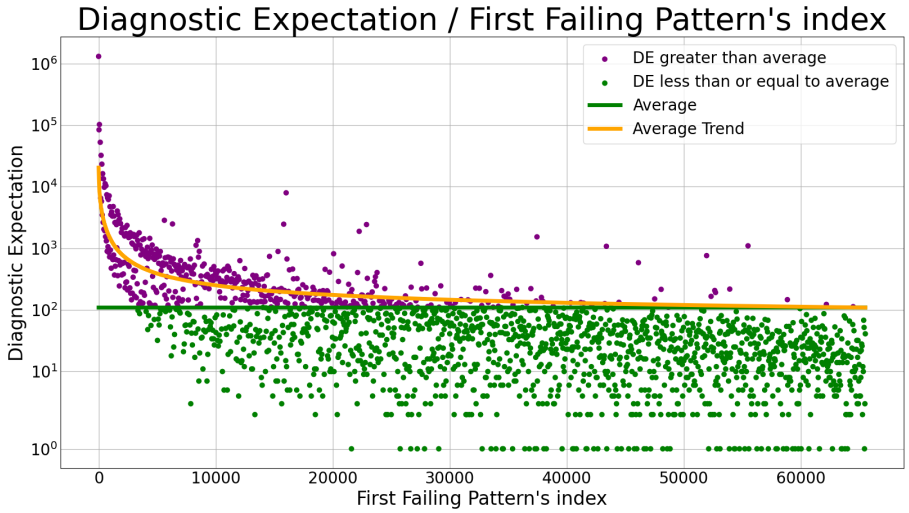


Fig. 4.11 Depiction of how the DE changes, in the industrial case study, based on the position of the first failing pattern for LBIST partition 5.

Indeed, Figure 4.11 shows the trend of the DR depending on the first failing pattern. The y-axis, on a logarithmic scale, represents the DE; the x-axis represents the index of the first failing pattern. The green horizontal line represents the average constant value of DE, which is approximately 109, whereas the orange line depicts the average trend of the DE. The logarithmic scale on the y-axis provides a more nuanced view of this relationship over a wide range of values.

All points with a DE greater than the average value are colored purple. Instead, those with a DE less than the average are colored in green.

In summary, this graph illustrates that as the index of the first failing pattern increases (moving right along the x-axis), there is a decrease in the DE (moving down along the y-axis). This decrease in DE corresponds to an increased diagnostic accuracy due to fewer possible candidates.

Table 4.11 shows results of average DE of the proposed logic diagnosis methodology, in comparison with other state-of-the-art studies, for benchmarks circuits from ITC'99 [61].

The method proposed in [25] gives the best outcome in the majority of the benchmarks circuits considered. Nevertheless the proposed methodology achieves anyway good and similar results to [25].

Table 4.11 Average DE for ITC'99 benchmarks.

Circuit	Avg. DE	Avg. DE [26]	Avg. DE [24]	Avg. DE [25]
b15	1.77	7.64	6.12	1.57
b17	1.87	7.55	3.47	1.11
b18	5.43	10.54	24.96	2.13
b20	1.37	14.43	7.74	1.21
b22	2.16	16.84	18.51	2.34

For every LBIST partition, Table 4.12 offers a summary of the average DE for the industrial case study against that obtained using the [26], [24], and [25] approaches. The average DE provides important new perspectives on the diagnosis accuracy of certain partitions. For instance, the proposed method reaches an average DE of 207, indicating a somewhat high DR, in LBIST partition 1, which consists of a complete fault list of 15 million faults. LBIST partition 0, with a fault list size of 4 million, has an average DE of 92, implying a more effective diagnostic method by contrast.

The storage-based LBIST scheme proposed in [26] has been used for comparison also for the industrial case study. The experimental procedure described in the study has been reproduced to compute the results, that is, the software procedure for diagnostic patterns generation and then the logic diagnosis methodology on $2,000$ random distinct faults injection campaign.

The approach described in [24] exploits the error propagation function of MISR to determine candidates for faulty behavior and the logic cones of the scan cells capturing the wrong values.

To calculate the values for the DE outcome using the [24] approach, the same method described in the cited paper has been used. That is, $2,000$ faults have been randomly injected into the combinational logic for each LBIST partition of the case

Table 4.12 Average DE for each LBIST partition of the industrial case study.

LBIST Partition	Avg. DE	Avg. DE [26]	Avg. DE [24]	Avg. DE [25]
0	92	44	12,362	7,305
1	207	81	38,253	29,846
2	103	52	18,872	10,751
3	100	65	23,114	17,539
4	101	58	22,020	12,878
5	109	56	25,595	17,231
6	123	77	26,412	19,945

study. The resulting faulty signatures have been diagnosed using the previously calculated error propagation function of the MISR.

The comparison with [25] followed a same technique. The latter is a development of what an earlier work [24] offered. The candidates were computed beginning from the cones of the logic of the detected scan cells; $2,000$ faults were thus inserted randomly and using the propagation errors of the MISR. The first approach differs in that, in this enhanced form, some trimming is done depending on the correct bits of the MISR. A few candidates are eliminated from the first list, so their mistake would have been spread even to MISR bits that, at last prove to be accurate.

For every LBIST partition, Table 4.12 presents the average number of candidates obtained using the suggested methodology and those specified in [24] and [25]. For the industrial case study, the later two methods fall short in low DE values. The justification for [24] is that it is based just on the MISR of the failing pattern following a single LBIST run, thus without reducing the possible candidates of successfully applied patterns, as suggested in this study. Although the improved technique [25] generates a high average number of candidates, it makes pruning for the proper MISR bits more efficient. The case study, and usually every large complex SoC, contains several scan chains, many LBIST partitions, and a compacted MISR, therefore causing severe error information loss during compaction. Therefore, for such highly big and sophisticated SoCs, only examining the spread of mistakes is insufficient to precisely identify failures. Furthermore consider the scenario whereby the case study features several scan chains varying in length. In such instance, for

some scan chains the total number of shifts will be more than the whole length; this behavior would produce a more complicated error propagation function.

This difference is significant for [24] and [25] for the motivations explained above. If considering LBIST partition 5, an impressive difference in the outcome can be observed. This is consequently because only 64 bits of MISR, 1 thousand scan chains, and partition 5 has around 6 million faults overall (see Table 4.1. Considering the DE value for LBIST partition 0, the difference between the approaches is still high; nevertheless, both [25] and [24] perform better for this partition than the other partitions because it has less faults and less scan chains, so producing a smaller test compaction.

The highest results for the industrial case study seem to come from [26]. Given that the employed patterns are predictable and diagnostic rather than pseudo-random and that the information used for the diagnosis is more precise, its efficacy is reasonable. As already discussed in the previous sections, the LBIST in-field has usually limits, so the proposed approach just evaluates the first failing pattern, aiming at capitalizing what is possible to recover. Concurrently, the [26] seeks to identify failing devices considering all the failure patterns, so implying the option to re-test the device to extract the necessary information for the complete diagnostic report.

In subsection 4.5.7, a final comparative summary explaining the different advantages of the considered methodologies is reported.

4.5.5 Logic Diagnosis of faulty devices

A set of failed devices taken from the field has been used to validate the experimental setting (as depicted in Figure 4.10. The proposed methodology gives as output the total number of candidates per failed device, acquired by leveraging the index of the first failing pattern to execute the diagnosis. In fact, the latter is used to find the particular node inside the tree-based fault dictionary, therefore enabling the evaluation of the set of possible faults.

As demonstrated in Figure 4.10, twenty-three devices in all have failed. Of these, eight are definitely diagnosed as combinational logic failures (as seen in Table 4.13), each linked with a particular collection of potential faults. Furthermore noted by the suggested approach are fifteen devices displaying chain failures. Indeed, the LBIST can be configured to run a specific number of patterns as scan chain integrity tests

(without capture) prior to running the effective logic tests. The method allows one to identify which devices displayed scan chain errors unrelated to combinational logic by finding the index of the first failing pattern.

The logic diagnosis results are presented in Table 4.13, where each row represents a faulty chip, indicating the DR achieved, that here is the number of candidate faults. The DR provides crucial insights into the effectiveness of the diagnostic process for each faulty chip. For instance, FAIL#2 displayed a DR of 2, suggesting a high level of diagnostic precision, whereas FAIL#5 exhibited a high number of candidate faults, considering only the information that can be retrieved from the vehicle.

Table 4.13 Logic DR for the industrial case study, obtained through the data collected in-field for a batch of faulty devices field return.

Faulty chip #	DR	DR [24]	DR [25]
FAIL#2	2	40,116	16,950
FAIL#5	25,299	4,105	3,280
FAIL#14	198	15,990	11,925
FAIL#15	41	11,764	9,530
FAIL#18	1,910	91,140	71,378
FAIL#19	1,910	7,948	5,347
FAIL#20	4,158	5,844	4,670
FAIL#23	586	13,553	8,132

Moreover, in Table 4.13, the last two columns indicate the DR obtained using the methodologies described in [24] and in [25]. The proposed method has a significant advantage for most analyzed devices. The second method enhances on average the DR obtained by the first one, but it still does not reach the proposed method's values. The only case in which the method lacks accuracy compared with the other two is FAIL#5. Indeed, this device reported a failure in the 16^{th} pattern. As depicted in Figure 4.11, the position of the first failing pattern (in this case, the 16^{th}) determines the value of the DR. The lower the position, the higher the value of the DR. In addition, [25] quite reaches the DR of the proposed approach for the FAIL#20, improving the one achieved by [24]. Nevertheless, considering the average (see Table 4.12), the method still achieves higher accuracy.

4.5.6 Using another architecture scheme

Many works in the state-of-the-art, as seen in Table 2.2, propose changes or extensions to the standard STUMPS BIST architecture to provide a better and more efficient diagnosis process. A comparison with this work would not be appropriate, as the latter's purpose is to provide a way for manufacturers to adapt the proposed methodology to any architecture scheme, without requiring a specific one.

However, a simulation based on an architectural change can be adapted to assess its effectiveness in the industrial case study. In particular, results are computed for the BISD architecture [21] because it has a similar goal to the proposal, as described in Section 2.10.

Firstly, the requirements of the selected study regarding needed memory have been assessed:

- Failing memory depth of 50;
- Test set size of $65,535$, hence pattern indexes are on 16 bits;
- Single-Input-Signature-Register (SISR) length of $log_2(\#shifts)$ bits.

Examining all LBIST partitions in Table 4.1, the necessary memory is $3,146,528$ bits or almost 3 Mb. The target fault model for this study is the Transition delay, which requires merely a fraction of the computed one; the reported memory is less than the one given in subsection 4.5.1 for the Stuck-at fault model. Furthermore, such memory must be accounted for the area overhead since the suggested method generates a new memory local to the BISD instead of using already existing ones.

With regard to the logic diagnostic phase, the method [21] starts with the computation via simulation of faulty signatures for every fault in the DUT, therefore generating scalability issues depending on the size of the fault universe. Once such dictionary is built, the BIDS in-field allows one to acquire incorrect signatures, if any. Described in the cited article, the result of the diagnosis method is a final ranking of all the faults sorted such that, should a fault rank first, that is exactly the candidate. Based on total 800 fault injections, the writers in [21] report experimental results. Considering the amount of times the first fault in the rank was the injected one, they determine the diagnostic resolution.

To apply the study in the industrial case study, an injection campaign of roughly $1,000$ faults has been carried out for every LBIST partition in the DUT; in the 23% of cases, the first fault in the rank was indeed the injected one. In the other circumstances, where the top-ranked fault is not the injected one, the number of candidates produced by the [21] is the maximum, being the set of candidate faults the full LBIST partition's faults.

Furthermore, unlike the [21], which offers a ranking of all the faults in the circuit, the proposed methodology strictly defines the set of candidates in any instance. Thus, in the case of actual defective devices, the computed ranking used in [21] can be followed during failure analysis; yet, the number of possible choices is not decreased at all.

4.5.7 High-Level Summary Comparisons

A comprehensive comparison of several state-of- the-art logic diagnosis techniques, including the proposed approach, is given in Table 4.14. This table highlights the several needs for data from the defective device, the required pre-compilation actions that have to be done, the kind of information generated in the framework of NTF devices, and the last results for each approach.

Applied to very complex real-world devices, the proposed method shows better efficiency according to the experimental results than the approaches described in [21, 24, 25]. Specifically, since the first mentioned approach [21] depends on getting a whole set of defective signatures, it requires more pre-computation than the alternatives. This is a resource-intensive procedure since it requires many fault injections all throughout the fault universe. Moreover, this approach [21] finally generates a ranked list of the fault universe instead of a succinct list of plausible candidates, therefore failing to really limit the alternative causes of failure as efficiently as it could.

On the other hand, the methods described in [24] and [25] perform well with benchmark circuits; however, they tend to struggle when faced with the complexities presented by more intricate case studies, resulting in less accurate diagnoses.

Furthermore, although [26] has exhibited superior efficiency in industrial case studies, it regrettably fails to offer insights into NTF scenarios, as seen in Table 4.14. This methodology advocates a systematic strategy for producing deterministic pat-

Table 4.14 High-Level Summary among different LBIST diagnostic methodologies.

Method	Required data of the failed device	Pre-computation	Insights in case of NTF	Limitations	Outcome
BISD: Scan-based Built-In self-diagnosis [21]	Signatures	Mapping of each fault to its signature through simulation	Yes	It does not provide a final list of candidates but only a ranking	Ranking of the fault universe
A Storage Based LBIST Scheme for Logic Diagnosis [26]	Diagnostic report with all the failed patterns	Software procedure to determine the patterns	No	Diagnostic features are required for the used LBIST scheme and fully deterministic patterns pre-computed are necessary	List of candidates
Signature based diagnosis for logic BIST [24]	Signatures	Logic cones of the scan cells linked to failing MISR bits	Yes	The accuracy strictly depends on the aliasing of the MISR	List of candidates
Improving the performance of signature based diagnosis for Logic BIST [25]	Signatures	Logic cones of all the scan cells	Yes	The accuracy strictly depends on the aliasing of the MISR	List of candidates
Proposed	Index of the first failing LBIST pattern	Tree-based fault dictionary	Yes	If the first failing pattern is at the beginning of the test set, the candidate set size could be large	List of candidates

terns for diagnosis, considering all failure patterns throughout the diagnostic process. This method enhances accuracy; nevertheless, obtaining all failure patterns to provide a thorough diagnostic report is not always practical, especially with NTF devices.

Choosing the suitable diagnosing technique is really crucial. The choice must satisfy several key requirements, including data availability, the capacity to re-test the malfunctioning device in debug mode, computing resource availability, and required diagnostic outcome granularity of information. In this regard, the proposed approach is efficient in handling limited knowledge from the device and it is mainly compatible with any LBIST scheme since it mostly employs the index of the first failed pattern.

Importantly, the approach can still generate a list of possible candidate faults even in cases when the equipment does not subsequently fail upon being returned to the manufacturers. This feature is especially helpful in cases when changing the LBIST design of the device is not possible and a thorough diagnostic report is not realistic. Regarding these circumstances, the strategy is clearly the most suitable and effective one among several approaches.

Scalability discussion

The proposed methodology for the diagnosis of devices returned from the field is based on two main stages: data collection and logic diagnosis.

For the data collection phase, dealing with a much larger device could generate problems in saving information. However, the memory available for a larger device is expected to be proportionately larger and thus able to store the necessary data. Possibly, if the latter is not the case and having stringent memory limits, it could be avoided saving the LBIST signatures and save only the index of the first failing pattern, or limiting the number of failing to store.

For the diagnosis phase, the biggest concern in scalability lies in the time needed to create the fault dictionary. However, industries typically generate this type of information during the design and engineering phase of the device. Thus, the methodology requires no additional effort.

Lastly, the proposed methodology is highly versatile because it does not involve specific LBIST schemes but can be used by any industrial device with few requirements, unlike other methodologies proposed in the literature [21, 26].

Chapter 5

Conclusion

This thesis addresses the challenges related to the increasing complexity of automotive devices and their reliability throughout their entire life cycle, starting from the production test process and continuing with the mission mode. The various proposed methodologies, focusing on the various stages of testing, are detailed in Chapter 3.

First, in Section 3.1, the trade-offs between reducing test patterns during Wafer Sort, increasing packaging costs, and screening more chips during Package Testing are analyzed. A comprehensive cost model is introduced. This introduced model considers the non-uniform failure distribution by exploiting a sacrificial lot to extract useful information. The main findings of the study are (1) strategically reducing the number of patterns to be applied along the Wafer Sort can lead to economic gains under specific conditions of yield, fault coverage, and manufacturing cost; (2) sacrificing a set of devices can help understand the distribution of faults over the die population; and (3) pattern generation taking into account the fault distribution ensures that even if patterns are cut from the queue, the remaining ones still cover the most fault-prone gates, increasing the economic gain. The methodology was successfully validated using production data collected for six months at STMicroelectronics. The study's results are described in Section 4.2, showing the proposal's effectiveness and applicability. The method was validated by focusing on the first two production test phases. However, further investigation could be done by applying the same methodology to subsequent test phases.

Turning to the later testing phases, specifically the System-Level Test, in Section 3.2, the thesis proposes a method for analyzing the netlist and its DfT to classify

structural test escapes and evaluate the SLT suites' abilities to cover the faults discovered by previous scan-based tests. Results presented in Section 4.3 were calculated on an automatically generated SLT suite to cover portions of the un-core logic of the devices, and demonstrates how such programs cover some categories of critical structural faults. Furthermore, in Section 3.3, a method for evaluating pseudo-random stress stimuli and applying them during the SLT phase is detailed. Section 4.4 presents novel experimental results for a large automotive device.

In addition to contributions related to production testing, the thesis also delves into a second branch of research: logic diagnosis for field returns. Indeed, the testing phase for devices intended for safety-critical sectors is extended to field life to ensure their reliability does not diminish over time. Companies have had many returns to the field, and analyzing failures without having information about what happened during the life of the device is complex. In fact, NTF situations often happen, where devices returned to the field do not manifest failures once returned to the manufacturer. Therefore, this thesis addresses this problem by proposing a methodology that includes two steps: collecting data during the operational life of devices in the field and exploiting this data to diagnose failures when returned to manufacturers in the field. The approach exploits the LBIST programmed by the firmware to store corrupted MISR signatures that manufacturers can use to diagnose potential faults. The methodology is validated using failed devices provided by STMicroelectronics, experimental results Section 4.5 show how the proposal achieves better diagnostic resolutions compared to other state-of-the-art methodologies.

In conclusion, the thesis proposes several methodologies, touching on all phases of testing safety-critical electronic devices. The proposals aim to address state-of-the-art issues, optimize the production testing process (from scan-based to system-level test), and logic diagnose field returns. Experimental results obtained on benchmark circuits and real industrial devices prove the methodologies' effectiveness and applicability in industry.

Impact and perspectives

The main goal of the research has been to deal with the increasing complexity of SoCs in the Automotive field, where the ever-increasing number of gates, stricter safety standards, and integration of different functionalities require innovative test

and diagnostic solutions. Such an issue became a key point in industries nowadays, and reasonably it will be always more important. As the complexity continues to rise, even state-of-the-art methodologies, deeply-rooted in the literature and in the knowledge of the field, could become less effective.

In fact, as shown in the various chapters, complexity is challenging the established techniques of the testing world, such as scan-based testing and the manufacturing test flow. The issues lie first in the costs involved in such testing for very large and complex modern devices. Therefore, the traditional scan methodology and the manufacturing test process also needs to be modernized to adapt to the new complex designs. In addition, new devices have intricate architectures that give rise to different categories of faults that are difficult to cover by traditional ATPG flows. All these situations then lead to devices escaping testing and turning into returns from the field, hard to diagnose.

The contributions in this thesis directly address these challenges. In detail, the thesis contributes to each manufacturing and in-field test phase to face the issue of modern devices' complexity, which industries must continuously to deal with.

Furthermore, the research has had a relevant impact for the industrial processes of the STMicroelectronics, providing data analysis on their corner lots, novel methodologies to generate scan-based patterns, and an optimized flow for stress testing for a large ADAS device currently in production. Through these efforts, the thesis demonstrates the potential for both improved reliability and cost-effectiveness for the testing field of increasingly complex automotive SoCs.

Acronyms

ADAS Advanced Driver Assistance Systems.

ATE Automatic Test Equipment.

ATPG Automatic Test Pattern Generator.

BI Burn-In.

DfT Design-for-Test.

DMA Direct Memory Access.

DUT Device Under Test.

FSM Finite State Machine.

LBIST Logic Built-In Self-Test.

LFSR Linear Feedback Shift Register.

MCU Microcontroller Unit.

MISR Multiple Input Signature Register.

NTF No Trouble Found.

ODE Output-Data-Evaluator.

PCS Pattern Count Stop.

PI Primary Inputs.

PO Primary Outputs.

PRNG Pseudo Random Number Generator.

SA Stuck-At.

SLM Silicon Lifecycle Management.

SLT System-Level Test.

SoC System-on-Chip.

TPG test pattern generator.

TRN Transition.

UUT Unit Under Test.

VCD Value Change Dump.

VLSI Very Large Scale Integration.

References

[1] Rohit Srivastava et al. Soc time to market improvement through device driver reuse: An industrial experience. In *2012 International Symposium on Electronic System Design*, 2012.

[2] Iso 26262-[1-10], road vehicles – functional safety. 2011.

[3] Vincenzo Tancorre et al. eflash mcus multi-temperature coverage maximization and test cost optimization. In *Proceedings International Test Conference 2015*, 2015.

[4] Chen He and Yanyao Yu. Wafer level stress: Enabling zero defect quality for automotive microcontrollers without package burn-in. *2020 IEEE International Test Conference (ITC)*, pages 1–10, 2020.

[5] Tai Song et al. Pattern reorder for test cost reduction through improved svmrank algorithm. *IEEE Access*, 8:147965–147972, 2020.

[6] Guo-Yu Lin et al. A test-application-count based learning technique for test time reduction. In *VLSI Design, Automation and Test(VLSI-DAT)*, pages 1–4, 2015.

[7] S.K. Goel et al. On-chip test infrastructure design for optimal multi-site testing of system chips. In *Design, Automation and Test in Europe*, pages 44–49 Vol. 1, 2005.

[8] Sudarshan Bahukudumbi et al. Defect-oriented and time-constrained wafer-level test-length selection for core-based digital socs. In *2006 IEEE International Test Conference*, pages 1–10, 2006.

[9] Matt Grady et al. Adaptive testing - cost reduction through test pattern sampling. In *2013 IEEE International Test Conference*, pages 1–8, 2013.

[10] François-Fabien Ferhani et al. How many test patterns are useless? In *26th IEEE VLSI Test Symposium (vts 2008)*, pages 23–28, 2008.

[11] Harry Chen. Beyond structural test,the rising need for system-level test. In *International Symposium on VLSI Design, Automation and Test*, 2018.

[12] Dilip Kumar Reddy Tipparthi and Karthik Krishna Kumar. Concurrent system level test (cslt) methodology for complex system-on-chip. In *IEEE EPTC*, pages 196–199, 2014.

[13] Ilia Polian, Jens Anders, Steffen Becker, Paolo Bernardi, Krishnendu Chakrabarty, Nourhan ElHamawy, Matthias Sauer, Adit Singh, Matteo Sonza Reorda, and Stefan Wagner. Exploring the mysteries of system-level test. In *IEEE ATS*, pages 1–6, 2020.

[14] Francesco Angione, Davide Appello, Paolo Bernardi, Claudia Bertani, Giovambattista Gallo, Stefano Littardi, Giorgio Pollaccia, Walter Ruggeri, Matteo Sonza Reorda, Vincenzo Tancorre, and Roberto Ugioli. A low-cost burn-in tester architecture to supply effective electrical stress. *IEEE Transactions on Computers*, 72(5):1447–1459, 2023.

[15] Rajesh Kashyap. Silicon lifecycle management (slm) with in-chip monitoring. In *2021 IEEE International Reliability Physics Symposium (IRPS)*, pages 1–4, 2021.

[16] Byunghyun Jang, Jin Kyung Lee, Minsu Choi, and Kyung Ki Kim. On-chip aging prediction circuit in nanometer digital circuits. In *2014 International SoC Design Conference (ISOCC)*, pages 68–69, 2014.

[17] Paul H. Bardell, William H. McAnney, and Jacob Savir. *Built-in Test for VLSI: Pseudorandom Techniques*. Wiley-Interscience, USA, 1987.

[18] Miron Abramovici, Melvin A. Breuer, and Arthur D. Friedman. Digital systems testing and testable design. 1990.

[19] Thong Tran, Sudheer Reddy Gundala, Komal Soni, Aaron Baker, Adam Fogle, and Sandhya Chandrashekhar. No trouble found (ntf) customer return analysis. In *2020 IEEE International Reliability Physics Symposium (IRPS)*, pages 1–6, 2020.

[20] Ismet Bayraktaroglu and Alex Orailoğlu. Improved fault diagnosis in scan-based bist via superposition. In *Proceedings of the 37th Annual Design Automation Conference*, DAC '00, page 55–58, New York, NY, USA, 2000. Association for Computing Machinery.

[21] Melanie Elm and Hans-Joachim Wunderlich. Bisd: Scan-based built-in self-diagnosis. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pages 1243–1248, 2010.

[22] Andal Jayalakshmi and Tan Ewe Cheong. A methodology for lbist logic diagnosis in high volume manufacturing. In *2012 4th Asia Symposium on Quality Electronic Design (ASQED)*, pages 249–253, 2012.

[23] Raimund Ubar, Sergei Kostin, Jaan Raik, Teet Evartson, and Harri Lensen. Fault diagnosis in integrated circuits with bist. In *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007)*, pages 604–610, 2007.

[24] Wu-Tung Cheng, Manish Sharma, Thomas Rinderknecht, Liyang Lai, and Chris Hill. Signature based diagnosis for logic bist. In *2007 IEEE International Test Conference*, pages 1–9, 2007.

[25] Wu-tung CHENG, Manish SHARMA, and H. RINDERKNECHT, Thomas. Improving the performance of signature based diagnosis for logic bist, 2009.

[26] S. Gopalsamy and I. Pomeranz. A storage based lbist scheme for logic diagnosis. In *2024 IEEE 42nd VLSI Test Symposium (VTS)*, pages 1–7, 2024.

[27] S. Gopalsamy and I. Pomeranz. Fully deterministic storage based logic built-in self-test. In *2023 IEEE 41st VLSI Test Symposium (VTS)*, pages 1–7, 2023.

[28] T.W. Williams et al. Defect level as a function of fault coverage. *IEEE Transactions on Computers*, C-30(12):987–988, 1981.

[29] F. Corsi et al. Defect level as a function of fault coverage and yield. In *Proceedings ETC 93 Third European Test Conference*, pages 507–508, 1993.

[30] Michael Campbell. Plenary presentations: Keynote: The product complexity and test — how product complexity impacts test industry. In *2010 15th IEEE European Test Symposium*, pages 9–9, 2010.

[31] Alfredo Benso, Alberto Bosio, Stefano Di Carlo, Giorgio Di Natale, and Paolo Prinetto. Atpg for dynamic burn-in test in full-scan circuits. In *IEEE ATS*, 2006.

[32] Francesco Angione, Davide Appello, Paolo Bernardi, Andrea Calabrese, Stefano Quer, Matteo Sonza Reorda, Vincenzo Tancorre, and Roberto Ugioli. A toolchain to quantify burn-in stress effectiveness on large automotive system-on-chips. *IEEE Access*, pages 1–1, 2023.

[33] Alessandro Birolini. Reliability engineering theory and practice. *Springer*, 2017.

[34] OpenRISC, 2024. https://openrisc.io/.

[35] Walter Ruggeri et al. Innovative methods for burn-in related stress metrics computation. In *International Conference on Design Technology of Integrated Systems in Nanoscale Era*, 2021.

[36] D. Appello, P. Bernardi, G. Giacopelli, A. Motta, A. Pagani, G. Pollaccia, C. Rabbi, M. Restifo, P. Ruberg, E. Sanchez, C.M. Villa, and F. Venini. A comprehensive methodology for stress procedures evaluation and comparison for burn-in of automotive soc. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 646–649, 2017.

[37] Xinfei Guo, Wayne Burleson, and Mircea Stan. Modeling and experimental demonstration of accelerated self-healing techniques. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2014.

[38] Masahiro Fujita, Naoki Taguchi, Kentaro Iwata, and Alan Mishchenko. Incremental atpg methods for multiple faults under multiple fault models. In *ISQED*, 2015.

[39] Christelle Hobeika, Claude Thibeault, and Jean François Boland. Illegal state extraction from register transfer level. In *Proceedings of the 8th IEEE International NEWCAS Conference 2010*, pages 245–248, 2010.

[40] Irith Pomeranz. Built-in generation of functional broadside tests using a fixed hardware structure. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(1):124–132, 2013.

[41] Daniel Tille et al. Towards an automated flow for implementation ofdedicated lbist scan chains for functional safety. *TUZ*, 2020.

[42] Naghmeh Karimi, Krishnendu Chakrabarty, Pallav Gupta, and Srinivas Patil. Test generation for clock-domain crossing faults in integrated circuits. In *IEEE DATE*, pages 406–411, 2012.

[43] M. Bushnell and Vishwani Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Springer Publishing Company, Incorporated, 2013.

[44] Fujiwara and Shimono. On the acceleration of test generation algorithms. *IEEE Transactions on Computers*, 1983.

[45] D. Appello et al. System-level test: State of the art and challenges. In *IEEE International Symposium on On-Line Testing and Robust System Design*, 2021.

[46] Peter Maxwell et al. Bridge over troubled waters: Critical area based pattern generation. In *IEEE European Test Symposium*, 2017.

[47] F. Hapke et al. Total critical area based testing. In *2018 IEEE International Test Conference (ITC)*, pages 1–10, 2018.

[48] F. Hapke et al. Defect-oriented test: Effectiveness in high volume manufacturing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(3):584–597, 2021.

[49] Janusz Rajski, Vivek Chickermane, Jean-François Côté, Stephan Eggersglüß, Nilanjan Mukherjee, and Jerzy Tyszer. The future of design for test and silicon lifecycle management. *IEEE Design & Test*, 41(4):35–49, 2024.

[50] M. Casarsa, G. Harutyunyan, and Y. Zorian. Test and diagnosis solution for functional safety. In *2020 IEEE International Test Conference (ITC)*, pages 1–5, 2020.

[51] Irith Pomeranz and Sudhakar M. Reddy. A same/different fault dictionary: An extended pass/fail fault dictionary with improved diagnostic resolution. In *2008 Design, Automation and Test in Europe*, pages 1474–1479, 2008.

[52] P. Bernardi et al. An adaptive tester architecture for volume diagnosis. In *2010 15th IEEE European Test Symposium*, pages 227–232, 2010.

[53] V. Boppana, I. Hartanto, and W.K. Fuchs. Full fault dictionary storage based on labeled tree encoding. In *Proceedings of 14th VLSI Test Symposium*, pages 174–179, 1996.

[54] Jacob A. Abraham et al. Special session 8b — panel: In-field testing of soc devices: Which solutions by which players? In *2014 IEEE 32nd VLSI Test Symposium (VTS)*, pages 1–2, 2014.

[55] A. Manzone, P. Bernardi, M. Grosso, M. Rebaudengo, E. Sanchez, and M.S. Reorda. Integrating bist techniques for on-line soc testing. In *11th IEEE International On-Line Testing Symposium*, pages 235–240, 2005.

[56] G. Filipponi, G. Iaria, M. Sonza Reorda, D. Appello, G. Garozzo, and V. Tancorre. In-field data collection system through logic bist for large automotive systems-on-chip. In *2022 IEEE International Test Conference (ITC)*, pages 646–649, 2022.

[57] Peikun Wang et al. An automatic test pattern generation method for multiple stuck-at faults by incrementally extending the test patterns. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2990–2999, 2020.

[58] Masahiro Fujita et al. Incremental atpg methods for multiple faults under multiple fault models. In *Sixteenth International Symposium on Quality Electronic Design*, pages 177–180, 2015.

[59] E.J. Marinissen et al. A set of benchmarks for modular testing of socs. In *Proceedings. International Test Conference*, pages 519–528, 2002.

[60] F. Brglez et al. Combinational profiles of sequential benchmark circuits. In *1989 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1929–1934 vol.3, 1989.

[61] F. Corno, M.S. Reorda, and G. Squillero. Rt-level itc'99 benchmarks and first atpg results. *IEEE Design & Test of Computers*, 17(3):44–53, 2000.

[62] Giusy Iaria, Paolo Bernardi, Claudia Bertani, Lorenzo Cardone, Giuseppe Garozzo, and Vincenzo Tancorre. A comprehensive scan test cost model to optimize the production of very large socs. *IEEE Transactions on Computers*, 2024.

[63] G. Iaria et al. A novel pattern selection algorithm to reduce the test cost of large automotive systems-on-chip. In *2022 IEEE 23rd Latin American Test Symposium (LATS)*, pages 1–6, 2022.

[64] P. Bernardi et al. About the correlation between logical identified faulty gates and their layout characteristics. In *2023 29th IEEE International Symposium on On-Line Testing and Robust System Design*, 2023.

[65] Christian P. Robert et al. *Monte Carlo Statistical Methods*. Springer Texts in Statistics. Springer New York, NY, 2 edition, 2004.

[66] P. Bernardi, M. Bonazza, E. Sanchez, M. Sonza Reorda, and O. Ballan. On-line functionally untestable fault identification in embedded processor cores. In *IEEE DATE*, pages 1462–1467, 2013.

[67] Paolo Bernardi et al. Logic diagnosis based on logic built-in self-test signatures collected in-field from failing system-on-chips. *Electronics*, 13(21), 2024.

[68] G.A. Klutke, P.C. Kiessler, and M.A. Wortman. A critical look at the bathtub curve. *IEEE Transactions on Reliability*, 52(1):125–129, 2003.

[69] Chen He et al. Wafer level stress: Enabling zero defect quality for automotive microcontrollers without package burn-in. In *IEEE International Test Conference (ITC)*, 2020.

[70] P. Bernardi, M. Grosso, M. Rebaudengo, and M. Sonza Reorda. A pattern ordering algorithm for reducing the size of fault dictionaries. In *24th IEEE VLSI Test Symposium*, pages 6 pp.–391, 2006.

[71] Arm. Amba ahb protocol specification. https://developer.arm.com/documentation/ihi0033/latest/.

[72] Siemens Digital Industries Software. Tessent testkompress, version 2022, Siemens 2022.