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

Doctoral Dissertation
Doctoral Program in Computer Engineering (38^{th}cycle)

Manufacturing and In-Field Testing Techniques

By

Gabriele Filipponi

******

Supervisor(s):
Prof. Paolo Bernardi
Prof. Riccardo Cantoro, Co-Supervisor

Doctoral Examination Committee:
Prof. Alberto Bosio, Referee, Lyon Institute of Nanotechnology
Prof. Marcello Traiola, Referee, Inria Centre at Rennes University
Prof. Mario Barbareschi, Università Federico II Napoli

Politecnico di Torino
2026

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.

Gabriele Filipponi
2026

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

I would like to dedicate this thesis to my loving family, Mammina, Pelo, Felpa and JoJo.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Prof. Paolo Bernardi. Thank you for your guidance, and for giving me the opportunity to pursue this path. Your way of mentoring, often by throwing me straight into the ocean and letting me figure out how to swim, was challenging, but ultimately fundamental to my growth. Through this process, I learned how to face problems, make mistakes, and gain confidence in my own abilities. Although I still have nightmares about ETS's VAT.

I am incredibly grateful to my colleagues, lab mates, and now best friends: Giusa, Frankisko, Vittorio, Lorenzo, Giorgio, Nima, and Tommaso. Thank you for your daily support, the technical discussions that inevitably turned into chit-chat. You turned long hours, interrupted simulations, coffee breaks, stressful deadlines, exhausting table footballs matches, into something manageable and often even enjoyable. I couldn't have asked for a better group of people to share this journey with. I would also like to mention Nicola, whom I almost forgot, which somehow seems appropriate. As for Frankisko, his true identity remains unclear, but his contribution to both science and chaos is indisputable. The lab would not have been the same without you.

A special thank you goes to my, doppelganger and partner in gossip, Giusa, who was much more than a colleague: almost a second supervisor and, at times, a second mom during this journey. Your guidance, support, and honesty were invaluable. Thank you for letting this journey being so fun.

To my friends from my hometown, Giova, Doda, Leo, Serra, Ste, and Rangel thank you for being my constant, no matter how far this journey took me. You knew me long before the PhD, and throughout it, you mostly saw me on vacation, complaining about not working enough, and confidently claiming that I was doing absolutely nothing. And yet, every time I came back home, you welcomed me exactly the same. You may never have really seen me work, but you always gave

me what I needed most: normality, laughter, and a place where I could disconnect without guilt. That balance, whether intentional or not, helped me more than you know.

Finally, and most importantly, my deepest love and gratitude go to my family. To Mammina, Pelo, Felpa, my beloved Lepepe, and the ticket potato in this world, JoJi: you are my greatest motivation. You may not fully understand what my PhD is about, and you probably have rarely seen me working, but you always believed that in me and that I would find a solution in the end.

Even if technology and research feel very far from your world, you were always close to mine. Your support never depended on results, explanations, or achievements, it was constant, unconditional, and effortless. During moments of doubt and quiet anxiety, knowing that you believed in me, gave me the strength to keep going. This work is built on your patience, your love, and the certainty that no matter what, I always had a place to return to. This achievement is yours as much as it is mine.

Abstract

Over the past decade, integrated circuits (ICs) have become increasingly complex and are now integrated into almost all modern systems. In the automotive industry, innovations such as Advanced Driver Assistance Systems (ADAS) have significantly increased the functional and safety requirements of electronic components. As a result, ensuring reliability has become more important than ever, as malfunctions can cause catastrophic events.

Testing plays a pivotal role in ensuring product quality and reliability. This is especially true in safety-critical domains, such as automotive, where safety standards, such as ISO26262, mandates strict requirements to be compliant with.

However, the exponential growth in complexity, the huge increase in the number of transistors, the complexity of deep operational logic, and the integration of third-party "black-box" intellectual property (IP) have made traditional testing strategies obsolete. Furthermore, conventional structural testing techniques, originally developed for smaller circuits, fail to scale to modern System-on-Chips (SoCs). These limitations result in reduced fault coverage and potential gaps in testing, highlighting the need for new methodologies.

This thesis addresses such limitations by proposing novel methodologies in the automotive manufacturing testing flow, targeting Burn-In, System-Level Test, and In-Field phases.

The first contribution focuses on the Burn-In (BI) phase, addressing testability issues that arise in the absence of netlists for third-party IPs, as this prevents the generation of effective test patterns. A novel methodology is proposed to synthesize functional stress patterns for AI accelerators based on indirect, netlist-less measurements. Unlike standard functional programs which fail to excite deep arithmetic pipelines due to high data sparsity and correlation, this approach maximizes toggle activity through specialized data patterns. Experimental results demonstrate that the

generated functional suite achieves higher switching activity and a more uniform stress distribution when compared to structural patterns, even without the netlist.

The second contribution addresses System-Level Test (SLT), specifically the inability of standard loop-back tests to excite error detection and correction logic in communication peripherals. In this regard, this thesis introduces an FPGA-based companion module that interacts with the Device Under Test (DUT) to inject protocol-level errors and timing violations. Validated on an industrial automotive SoC, this suite improves stuck-at fault coverage by approximately 50% over undetected structural faults and increases transition delay fault coverage by about 15%, effectively closing the coverage gap left by scan-based techniques.

To address the manual effort and lack of portability in SLT development, this thesis introduces a framework capable of generating functional test automatically. The proposed framework, exploiting the Device Tree Specification (DTS), abstracts the device architecture into a graph-based model, enabling the creation of portable test programs. Thus manages to reduce development time and eliminate the implicit bound of tests and a specific instruction set architecture (ISA), allowing the test suite to be used on different architecture of SoCs.

Finally, the thesis addresses the problem of devices that fail during in-field operation but pass all tests when returned to manufacturers, known as No-Trouble-Found (NTF). The proposed method leverages the on-chip Logic Built-In Self-Test (LBIST), executed at key/on-off events, to collect diagnostic information. Through a binary search algorithm, the index of the first LBIST error pattern and its corresponding signature are stored in non-volatile memory (NVM). A tree-based fault dictionary is then used to perform logic diagnosis using only the collected in-field diagnostic data.

The proposed approach was validated experimentally on both ITC'99 circuits and automotive SoCs. In this regard, real failing devices in the field were used to demonstrate the effectiveness of the methodology. Compared to state-of-the-art techniques, it provides similar or better accuracy, while remaining compatible with existing LBIST architectures, ensuring scalability and applicability.

Contents

List of Figures xi

List of Tables xiv

1 Introduction 1

1.1 Optimized Burn-In Strategies for AI Accelerators 2

1.2 Advanced System-Level-Test Methodologies 4

1.2.1 Automated Generation of Functional Test Procedures 6

1.3 In-Field Logic Diagnosis via LBIST 7

1.4 Summary of Contributions 9

2 Background 11

2.1 Manufacturing Test Flow 11

2.1.1 Burn-In (BI) 12

2.2 Stress Metrics for Burn-In (BI) Analysis 13

2.2.1 AI HW Accelerators 14

2.2.2 Limitations of Current Stress Methodologies for AI Accelerators 16

2.2.3 System-Level Test (SLT) 16

2.3 System-Level Test State-Of-The-Art 18

2.3.1 Critical Gaps in SLT Automation and Portability 19

2.3.2 Stress induced by System-Level Test (SLT) 19
2.3.3 State-Of-The-Art for Communication Peripherals Testing . . 20
2.3.4 Limitations in Testing Error-Correction Logic 21
2.4 Built-In Self-Test (BIST) 21
2.5 Logic Diagnosis 24
2.6 Silicon Life-cycle Management (SLM) 25
2.7 State-of-the-art on Logic Diagnosis for LBIST-Caught Failures . . . 25
2.7.1 Limitations in Diagnostic Capabilities for Constrained Environments 26

3 The proposed methodologies 29
3.0.1 Burn-In Functional Stress Suite for AI HW Accelerators . . 29
3.0.2 Current Consumption Measurement Flow 31
3.0.3 Memory Toggle Metrics (MTMs) 33
3.0.4 Functional Stress Pattern Selection Algorithm 34
3.0.5 System-Level-Test for communication Peripherals 36
3.0.6 Structural Test Weaknesses Analysis 37
3.0.7 System-Level Test functional program suite generation . . . 39
3.0.8 Companion module 46
3.0.9 System-Level-Test test generation automation 47
3.0.10 Graph-based SoC Modeling 48
3.0.11 FSWGen Framework 48
3.0.12 In-Field Logic Diagnosis leveraging LBIST 49
3.0.13 In-field Data Collection through LBIST 52
3.0.14 Logic Diagnosis of Field Return Devices 58

4 Experimental Setup and Results 63
4.1 Case of Study 63

4.1.1 FPGA-based Tester 66
4.1.2 Burn-In Function Stress Suite for AI HW Accelerators . . . 68
4.1.3 Experiment Statistical Confidence and Parameters 69
4.1.4 Functional Stress Patterns Evaluation 70
4.1.5 An Additional Case Study 74
4.1.6 System-Level-Test for Communication Peripherals 82
4.1.7 System-Level-Test Test Generation Automation 86
4.1.8 In-Field Logic Diagnosis leveraging LBIST 87
4.1.9 FLASH Footprint 88
4.1.10 Signature collection time 89
4.1.11 Collected data from faulty devices 90
4.1.12 Diagnostic Expectation 91
4.1.13 Logic Diagnosis of faulty devices 95
4.1.14 Using another architecture scheme 97
4.1.15 High-Level Summary Comparisons 98

5 Conclusions **102**
5.0.1 Summary of Contributions 102
5.0.2 Expected Impact 104
5.1 Future Works 104

References **109**

List of Figures

1.1 Generic communication peripheral internals [1]. 5

2.1 Manufacturing testing flow for safety critical sectors. As it can be noted catching malfunctions at the early stages of manufacturing reduces costs, as next stages won't be applied to discarded dies. . . 12

2.2 Failure Rate of devices over Time. 13

2.3 Generic Block diagram of an AI Hardware Accelerator (AI HW) Accelerator [2]. 15

2.4 Standard Self-Test Using MISR/Parallel SRSG (STUMP) Built-in-Self-Test (BIST) architecture showing its primary functional blocks [3]. 22

2.5 Tree-based fault dictionary example with three applied test patterns [3]. 25

3.1 Proposed firmware execution flow used to grade the current consumption of a given functional pattern [2]. 31

3.2 Visual representation of the relationship between Memory Toggle Coverage (MTC), Memory Toggle Activity Variance (VAR(MTA)), and gate activity across two functional patterns [2]. 34

3.3 Example of Memory Toggle Coverage computation [2]. 35

3.4 Rank and Sift Selection Algorithm [2]. 36

3.5 Flow diagram of the proposed methodology [4] 37

3.6 Companion module view with a companion memory for error injection in the communication's protocol [4]. 38
3.7 Architecture of a generic communication peripheral [4]. 38
3.8 (a) DTS graph of a basic example SoC and a peripheral (b) [5]. . . . 49
3.9 Flow of the proposed approach [3] 51
3.10 Generation of N Logic BIST (LBIST) patterns, where the X_{th} propagates an erroneous values till self-test end in the MISR's signature [3]. 53
3.11 Finite-State-Machine (FSM) of the logic implemented of the proposed work [3]. 54
3.12 If the signature is good, test ends immediately [3]. 55
3.13 When the resulting signature is wrong, the dichotomic search algorithm is executed, which stops when two dichotomies cannot be distinguished [3]. 56
3.14 Memory layout required [3]. 58
3.15 Tree-based fault dictionary for LBIST with an example of three test patterns [3]. 59
3.16 Overview on the Logic Diagnosis steps [3]. 60

4.1 Complete architecture of the Case of Study. 64
4.2 Proposed experimental setup. 66
4.3 Experimental setup with Xilinx ZCU104 evaluation board and Controller Area Network (CAN) peripheral bus. 67
4.4 (a) Layout of the case study with AI HW Accelerator gates highlighted in green and (b) Measurement setup [2]. 69
4.5 Embedded thermometer's temperature profile [2]. 70
4.6 Toggle Coverage (TC) comparison [2]. 75
4.7 Toggle Activity (TA) comparison [2]. 75
4.8 Memory Toggle Activity (MTA) comparison [2]. 76

4.9 Stress of TC (a) TA (b) heatmap comparison [2]. 76
4.10 TC comparison for DUT2 [2]. 80
4.11 AVG(TA) comparison for DUT2 [2]. 80
4.12 VAR(TA) comparison for DUT2 [2]. 81
4.13 Stress TC heatmap comparison for DUT2 [2]. 81
4.14 Stress AVG(TA) heatmap comparison for DUT2 [2]. 82
4.15 Layout of the four CAN controllers [4]. 84
4.16 Laboratory experimental setup. 85
4.17 Venn diagrams of detected faults between various test approaches [4]. 86
4.18 (a) DTS graph for a complex SoC. (b) DTS graph of a CAN module [5]. 87
4.19 Failures information of failing devices based on the LBIST execution frequency [3]. 91
4.20 DE dependability, in the industrial case study, on the position of the first failing pattern for LBIST partition 5 [3]. 92

List of Tables

2.1 Comparison of different test approaches for communication peripherals [4]. 20
2.2 Logic diagnosis LBIST methodologies. 27
4.1 Faults summary report for different LBIST partitions in the industrial case of study [3]. 65
4.2 Population details of the functional patterns [2]. 70
4.3 Functional stress pattern selection results [2]. 72
4.4 Approaches results comparison [2]. 73
4.5 Functional patterns details for DUT2 [2]. 77
4.6 Functional stress pattern selection for DUT2 [2]. 78
4.7 Pattern approaches comparison for DUT2 [2]. 79
4.8 Properties of the SLT suite for the CAN peripheral. 84
4.9 Fault coverages for Stuck-at fault (435,967 faults) and Transition delay fault models (435,966 faults) [4]. 85
4.10 Incremental fault coverage for the CAN peripheral [5]. 86
4.11 Average DE for ITC'99 benchmarks [3]. 93
4.12 Average Diagnostic Expectation (DE) for each LBIST partition in the industrial case of study [3]. 93
4.13 Diagnostic Resolution (DR) for a batch of faulty devices field return [3]. 96

4.14 High-Level Summary among different LBIST diagnostic methodologies [3]. 99

Chapter 1

Introduction

Over the last decade, the automotive industry has experienced an impressive technological transformation [6]. The introduction of Advanced Driver-Assistance Systems (ADAS), Artificial Intelligence (AI), and enhanced connectivity has significantly increased the number and complexity of semiconductor components integrated into vehicles. System-on-Chips (SoCs) are the computational heart of these systems, hosting Central-Processing-Units (CPUs), Co-Processors, analog modules, many communication peripherals, and embedded memories within a single silicon die. As automotive electronics become responsible for safety-critical functions such as braking, steering, and collision avoidance, even a single malfunction can have catastrophic consequences. Such device complexity translates to the testing environment, hence conventional testing techniques, that were developed in an era of lower integration and larger technology nodes, fail to scale or provide the required level of reliability.

Ensuring such high level of reliability is a complex challenge, as safety-critical sectors are governed by strict standards, such as ISO 26262 [7], that mandate rigorous compliance with high reliability across all the device life-cycle.

From the manufacturers standpoint, the test flow of a semiconductor device in a safety critical sector is divided into distinct phases: Wafer Sort, Package Test, Burn-In (BI), SLT, Final Test, and finally the In-Field operation. Each stage contributes uniquely in assessing the reliability of devices. BI and SLT focus on latent defects that only appear under prolonged stress or complex functional conditions. Finally, In-

Field self-tests extend observability beyond the factory, providing valuable insights for manufacturers.

Hence, the goal of this thesis is to propose methodologies that optimize the reliability, reduce test escapes during manufacturing, and to provide better solutions to diagnose In-Field failing devices. The common thread of this work is the manufacturing testing flow: it aims to provide contributions to the testing phases starting from the earlier ones and lasting til the In-Field. The resulting work enables both improved screening during production and improve the diagnostic-ability of In-Field failures.

1.1 Optimized Burn-In Strategies for AI Accelerators

The first part of this work focuses on the BI, a key stage in safety critical sectors in which early-life failures are highlighted through combined electrical and thermal stress, through a mixture of structural and/or functional tests. Being the first major step for functional pattern execution testability is highly affected by the integration process of modules into the SoC. Module netlists are often supplied by specialized third-party vendors and subsequently integrated into a SoC by a separate system integrator. To protect Intellectual Property (IP), many modules are frequently delivered as hard macros and incorporated into the SoC as black-boxes. The absence of a detailed netlist or the limited accessibility of Design-for-Testability (DfT) impose substantial constraints on the generation of suitable test patterns, leading to testability issues.

Structural tests manage to achieve high test coverages and stress, however they lack functional awareness. As a result, stress distribution within the device can be very uneven, and some areas may not be excited properly. Recent studies have investigated the use of functional or mission-mode stimuli to replace or supplement it BI [8], aiming to filling the gap between structural tests and final mission-mode behavior.

Given that and the continuously increasing complexity of both SoC designs, where gate counts are growing exponentially [9], and DfT structures [10–12], the proposed methodology provides a way to synthesize a suite of functional stress programs without the need of the netlist.

These programs specifically target arithmetic and computational modules, with the primary focus on AI hardware accelerators embedded within SoCs. The development of functional stress methodologies for AI HW remains largely unexplored in the literature. Employing functional stress during BI enables the introduction of additional stress mechanisms that complement traditional scan-based stress. Functional workloads operate at the nominal frequency of the device, replicating real mission-mode behavior [13]. In contrast, scan-oriented techniques run at lower frequencies and require reconfiguring the circuit into non-functional states that differ from actual field operation. As a result, functional workloads can achieve greater and better stress since they can be scheduled and executed more frequently within the same BI duration. Furthermore, the electrical stress induced by functional workloads tends to be more uniformly distributed across the gates of the Device-Under-Test (DUT) compared to scan- or DfT-based approaches [11].

This work aims to create a set of functional stress patterns to maximize node switching activity and uniform toggle distribution in the Module-Under-Test (MUT). This enables highly effective BI stress without requiring access to the design's netlist. The proposed methodology relies on two key indirect measurement:

- *Current consumption* of the functional stress patterns, measured from real devices.
- *Memory Toggle Metrics (MTMs)*, novel metrics that are based on the accessible memory from the programmer's view of the AI accelerator's memory map.

In summary, the method evaluates a large population of functional stress patterns, either generated randomly or derived from existing validation and verification program suites . These patterns are first executed on-chip to measure current absorption. The resulting data are used to rank the patterns by current consumption, after which the highest-consuming patterns are filtered according to the MTMs.

It is important to emphasize that the proposed method does not rely on netlists for functional stress pattern selection. This feature is particularly valuable when the netlist is unavailable to the company integrating and testing SoCs containing AI hardware accelerators. The method operates purely from the programmer's perspective, using only memory maps, control registers, and status registers of the accelerator. Nevertheless, even when the netlist is accessible, the approach remains

advantageous due to its superior speed and efficiency compared to simulation-based techniques achieving similar objectives [14].

1.2 Advanced System-Level-Test Methodologies

Structural tests, such as those leveraging DfT hardware, like scan chains, and LBIST and Memory BIST (MBIST), are de-facto methods to achieve a high test coverage. However, despite their effectiveness, structural tests still exhibits many test escapes [15, 16], i.e., faults that remain undetected by the the test suite, but may manifest later during the device's operational lifetime. Several factors contribute to this limitation:

- **Generation constraints**, such as limited time budgets for pattern creation, impact the performance of Automatic Test Pattern Generation (ATPG) tools. As a result, ATPG-generated pattern sets are often a trade-off between fault coverage and computational cost and time.
- Automatic Test Equipment (ATE), which applies test patterns to the SoC during manufacturing, introduces additional constraints. These highly specialized and costly machines typically provide **limited pattern memory**, restricting the number of test vectors that can be stored and executed. Given that modern SoCs may feature scan chains containing hundreds of thousands to millions of flip-flops, a single scan pattern can occupy from several hundred kilobytes up to tens of megabytes. Consequently, a portion of the ATPG-generated patterns may need to be excluded from the final test flow to accommodate ATE memory constraints.
- Overall test cost considerations often lead test engineers to impose **strict limits on ATE usage time per device**, measured in milliseconds or seconds. Such restrictions may further reduce achievable fault coverage [17].

Moreover, although ATPG aims for comprehensive fault coverage, it may produce patterns that configure the circuit into states never encountered during normal functional operation [18]. This over-testing increases test time and resource usage [19] and may discard "good" devices marked as "bad", while still leaving some

faults undetected [15]. Thus, even extensive scan-based test sets are not always exhaustive. The combination of these factors can result in a significant number of faults to remain undetected, ultimately leading to manufacturing test escapes and reduced product reliability.

To address these and meet the increasingly stringent quality requirements imposed by safety standards, the semiconductor industry in safety critical sectors has introduced an additional step in the manufacturing test flow over the past decade: the SLT. This testing phase closely emulates the real application environment, workload, and operational conditions of the device. Unlike structural tests, SLT is functional in nature and exercises the SoC as a whole through realistic workloads and software/hardware interactions.

In this context, modern automotive SoCs may incorporate numerous communication peripherals to enable interaction between devices within the vehicle that the SoC is in. However, during manufacturing tests, communication with external systems is typically unavailable, which may leave parts of the logic untested. Moreover, ATPG limitations can result in untested regions near module boundaries.

Figure 1.1 illustrates a generic communication peripheral, highlighting its possible sub-modules. In this visualization, the arrow colors indicate the criticality of interactions between modules, and the alphabetic labels group sub-modules by functionality.

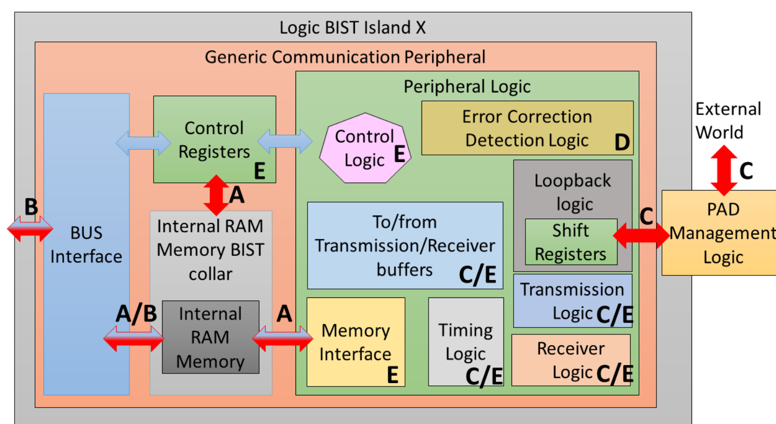


Fig. 1.1 Generic communication peripheral internals [1].

As such this thesis tackles the possible structural tests weaknesses that may arise in communication peripherals. This translates to areas that may be overlooked during structural tests:

A **Embedded memory access ports:** may not be fully covered during structural testing due to the presence of collars and memory DfT circuits such as MBIST.

B **Interfaces to other on-chip components:** can be distributed in different LBIST or scan-chain domains, thus introducing gaps in testability.

C **Transmission/Reception interfaces to chip top:** may comprehend signals and I/O pins that may remain not excited, thus untested, during manufacturing.

D **Detection and correction logic circuits:** such circuits are often implemented with deep logic functions that may be difficult to reach by means structural tests.

E **Complex hardware–software functions:** operations such as protocol handling and synchronization mechanisms are rarely exercised in structural tests.

Therefore, an effective SLT suite for communication peripherals should comprise a collection of programs designed to systematically address all the above considerations, ensuring comprehensive coverage of both hardware and functional aspects. Moreover the proposed methodology exploits a custom error injector, synthesized in a Field-Programmable-Gate-Array (FPGA), that interfaces through the ATE to provide stimuli to the error detection/correction and timing sub-modules of the MUT.

1.2.1 Automated Generation of Functional Test Procedures

Despite its innovation and significant advantages, SLT have several challenges. The creation of functional test procedures is labor-intensive and prone to human error, requiring expertise in both hardware and software knowledge and development. Each functional test must be carefully crafted to the specific DUT's architecture and Instruction Set Architecture (ISA) of the target device, thus strictly tied to them. Thus, test procedures are often difficult to reuse across different SoCs.

Although functional testing is well established in the State-Of-The-Art (SOTA), current literature lacks a unified framework capable of automatically decoupling

hardware descriptions and test flow to synthesize portable code. Current industry practices often rely on manual porting of legacy firmware, which cannot scale with the growing heterogeneity of automotive SoCs. Manual approaches do not address the fundamental need for a level of abstraction that separates the test from the underlying hardware specifications.

This lack of portability increases development time and costs, and limits scalability, as test engineers must usually rewrite or modify existing code to match the SoCs's architecture or hardware revisions. To overcome these limitations, this thesis introduces a framework that is capable of generating automatically functional test procedures leveraging the Device Tree Specification (DTS). DTS is a hierarchical and extensible data structure used to describe the hardware of an embedded system in a platform-independent format [20]. By exploiting the DTS, the proposed framework is able to automatically analyze the SoC architecture, identify master CPUs (or coprocessors), buses, peripherals, and finally generate functional programs. In essence, the DTS-based approach bridges the gap between SoC-level hardware descriptions and functional test generation, facilitating a scalable and maintainable methodology for SLT across different hardware platforms.

1.3 In-Field Logic Diagnosis via LBIST

In addition to the manufacturing testing phase, representing the first research branch of this thesis, ensuring the long-term reliability of SoCs throughout their operational lifetime, particularly in safety-critical automotive applications, constitutes the second major focus of this work. Guaranteeing dependable operation in the field is a growing challenge, as modern automotive SoCs are expected to operate continuously under environmental variations, progressive wear-out. As such failures can still manifest long after manufacturing [21]. For this reason, on-chip test are essential to support self diagnostic and health monitoring.

To address this, field testing has gained increasing relevance. This post deployment testing phase continuously monitors the health of devices, enabling the early detection of failures such as aging-induced degradation, soft errors that may evolve into functional failures over time [22, 23].

That is key to manufacturers: early detection of an In-Field failure enable manufacturers to understand the root-cause and provide appropriate actions either in the design SoCs or strengthen their test suites. Moreover, a known issue to manufacturers is the so called No Trouble Found (NTF) problem: that is when an In-Field device fail; however, when brought back to the factory it passes all tests, leaving the manufacturers with no clue on what went wrong, and no possibility to close the necessary feedback-loop.

A key element for in-field tests is LBIST [24, 25], which provides on-chip diagnostic capabilities. LBIST is capable of autonomously generating and applying scan-based test patterns to internal logic blocks, capturing shift-outs from scan chains into a resulting signature to detect anomalies. However, LBIST configurability during field operation is often constrained to avoid excessive power consumption, electrical stress, or thermal load that could damage the device. As a result, several restrictions, such as fixed seeds and limited reprogramming capabilities, make logic diagnosis much harder, particularly in scenarios such as NTF [26].

The SOTA presents several methodologies for diagnosing based on LBIST [27–32]. For example, the work in [28] introduces strategies that combine the execution of field self-tests with fault diagnosis, although their effectiveness depends on customized LBIST architectures. More recent contributions, such as [33], propose dedicated LBIST frameworks that improve conventional ATPG-based diagnosis by leveraging on-chip pattern generation techniques [34]. Other studies [31, 32] explore diagnostic approaches based solely on the analysis of faulty LBIST signatures, without requiring additional diagnostic hardware.

Starting on these foundations, the thesis introduces an innovative logic diagnosis methodology, designed specifically for constrained configurations. The proposed approach identifies the first failing pattern during LBIST execution, significantly narrowing down the set of potential fault candidates and enabling accurate diagnosis even under limited configurability conditions.

The experimental validation of these methodologies was carried out on real industrially relevant SoCs. Such are complex automotive SoCs featuring multiple cores, extensive LBIST infrastructure, and embedded telemetry capabilities. Moreover for the In-Field branch real failed devices from manufacturers were used a case of study.

1.4 Summary of Contributions

In summary, the contributions presented in this thesis are organized along the natural order of the manufacturing testing flow:

- **Burn-In function stress suite for AI HW Accelerators**: methodology to create a functional test suite for AI HW modules for the BI phase. The methodology focuses on toggle metrics and work in absence of netlist by resorting to indirect measurements to rank the functional programs in the final suite.
- **System-Level-Test for communication peripherals**: methodology that analyzes the potential weaknesses of structural tests such as scan-based and BIST. Then, it deliver guidelines on how to properly develop a functional SLT programs software suite to address such overlooked areas in structural tests. If the communication peripheral under test has detection/correction features, the methodology proposes the design of a hardware companion module, that may be added to the ATE, capable of interacting with the SoC communication module and purposely corrupting data frames, if necessary.
- **System-Level-Test test generation automation**: a novel framework that sets the base of portability and reusability of functional SLT methodologies across different SoC architectures. The proposed methodology leverages the DTS of the DUT to create functional tests automatically. The construction of a graph representation of the DTS, is the core of the approach which enables the creation of functional test programs across different SoC architectures through a simple graph traversal process. This abstraction makes it highly adaptable and efficient in simplifying SLT development.
- **In-Field Logic Diagnosis leveraging LBIST**: a comprehensive methodology that utilizes constrained In-Field LBIST engines for collecting field data and performing subsequent logic diagnosis based on the acquired information. The methodology effectively manages to reduce the number of candidate faults by leveraging the first failing pattern, thus enabling efficient logic diagnosis of field returns by manufacturers. This work is extremely valuable for industrial devices as it requires no architectural modifications. Accordingly, this study presents two main contributions:

1. A comprehensive methodology for **In-Field data collection** of failure information **using constrained LBIST engines** operating during the device's mission mode;
2. A **logic diagnosis approach** that supports failure analysis of field returns by **leveraging In-Field failure data** gathered on the chip.

Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 provides the necessary theoretical background and reviews the SOTA. Chapter 3 details the proposed methodologies, following the sequence of the manufacturing testing flow (from wafer to in-field). Chapter 4 describes the experimental setup and results obtained on the reported case of studies. Moreover the performance and scalability of the proposed approaches are discussed in this chapter. Finally, chapter 5 concludes the dissertation, summarizing the main outcomes and possible future research directions of the works.

Chapter 2

Background

Testing of Integrated Circuits (ICs) must be done throughout their entire life-cycle, from wafer manufacturing to operation in the field. This chapter presents the technical background necessary to understand the key principles, guiding the research projects and the SOTA related works with same and/or similar objectives

2.1 Manufacturing Test Flow

As transistor scaling advances, the number of transistors per unit area increases [35]. This progress has resulted in an exponential growth in the complexity of ICs, thereby increasing the challenges associated with testing [9, 36]. To manage this complexity, structural tests have been introduced in the manufacturers testing flow to simplify the testing process and facilitate the automation of test pattern generation for ICs [37].

The manufacturing testing flow's goal is to ensure that only faulty-free devices are shipped to customers while, also providing data to improve process yield and quality. The complete flow for safety critical sectors consists of different stages, as shown in fig. 2.1 [38, 39]:

- **Wafer Sort**: conducted at the wafer level before the dies are cut, this phase verifies the primary electrical functionalities of the chip by executing structural or scan-based test patterns [38].

- **Package Test**: performed after the dies identified as functional during the wafer test are cut, baked, and packaged. This stage ensures that the packaging process did not damage the dies, typically by reapplying the same set of structural test patterns to evaluate the electrical characteristics of the pins [40].
- **BI**: applied primarily to automotive and safety-critical devices, this phase highlights latent defects [41, 42] by applying a mixture of environmental and electrical distress to devices.
- **SLT**: introduced as an additional step for complex, safety-critical, or automotive devices [15], this test executes functional programs [43, 39] that emulate real operating conditions to verify the correct interaction among system components withing the SoC.
- **Final Test**: conducted before product shipment, this phase combines structural and functional tests to ensure that only fully operational devices reach the customer [44].

Fig. 2.1 Manufacturing testing flow for safety critical sectors. As it can be noted catching malfunctions at the early stages of manufacturing reduces costs, as next stages won't be applied to discarded dies.

2.1.1 Burn-In (BI)

BI is used to accelerate latent defects by applying electrical, thermal, and sometimes mechanical stress conditions for an extended duration.

The objective of BI is to smooth out "infant mortality" failures. The principle relies on the empirical observation that the failure rate of electronic devices follows a "bathtub curve" [21], as shown in fig. 2.2 . Three different phases can be observed: early-life, steady-state, and wear-out. BI specifically targets the steep initial decline in failure rate.

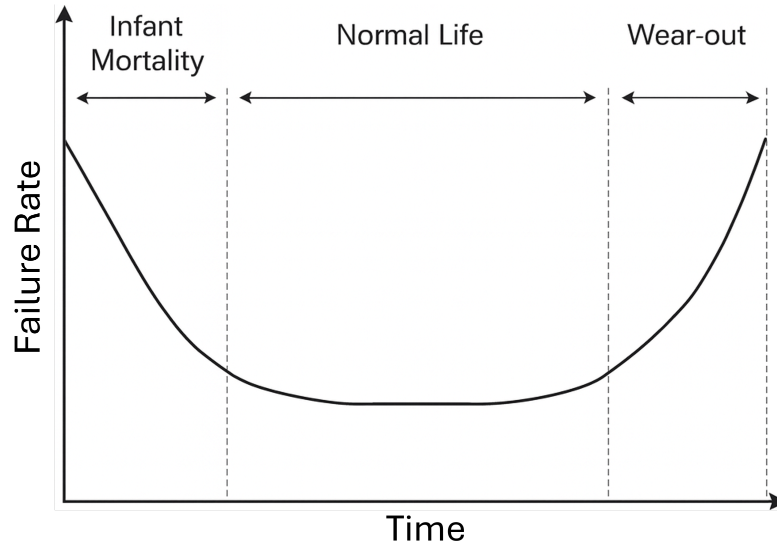


Fig. 2.2 Failure Rate of devices over Time.

Patterns applied in this phase are evaluated using specific metrics in order to determine the stress capabilities [42, 45]. Appropriate activity on emergent technologies, can be obtained only with complex methods (e.g., static electrical stress approaches Direct Drain Quiescent Current (IDDQ) [46] patterns).

2.2 Stress Metrics for Burn-In (BI) Analysis

This study evaluates specific stress metrics applicable to BI. Given a circuit composed of N nodes, let T_i be the number of transitions that occur in a specific node i during the entire duration of the test. By monitoring the number of transitions T_i in the entire set of nodes, two key metrics are obtained: TC and TA [47].

The TC metric offers a precise assessment of the percentage of gates that undergo toggling relative to the total number of gates (accounting for both rising and falling edges generated by the stress pattern). This relationship is defined in eq. (2.1):

$$\textbf{TC [\%]} = 100 \times \frac{\sum_i^{0 \to N} rising(T_i) + falling(T_i)}{2 \times N} \tag{2.1}$$

To assess the overall intensity of a stress pattern, TA is utilized, which accounts all toggles in the circuits. TA is usually broken down into two components: the

average toggle activity, denoted as $TA_{average}$ or $AVG(TA)$, and the toggle activity variance, $VAR(TA)$. These metrics must be normalized over a specific time interval (e.g., 1 us) and are calculated as shown in eq. (2.2) and eq. (2.3):

$$\mathbf{AVG}(\mathbf{TA})\left[\frac{\mathbf{T}}{\mathbf{us}}\right] = \frac{\sum_i^{0\to N} T_i}{time[us]} \tag{2.2}$$

$$\mathbf{VAR}(\mathbf{TA}) = \sqrt{\frac{\sum_i^{0\to N}(T_i - \hat{T})^2}{N}},\ \hat{T} = \frac{\sum_i^{0\to N} T_i}{N} \tag{2.3}$$

To satisfy the strict BI stress criteria required for high-quality silicon devices, optimization of TC and TA implies the following targets:

- **High** TC, to ensure the activation of the highest possible number of circuit nodes.
- **High** $AVG(TA)$, to guarantee a high degree of stress effectiveness on the gates.
- **Low** $VAR(TA)$, to achieve a uniform distribution of stress across the gates, thus avoiding scenarios where certain circuit areas are over-stressed while others remain under-stressed.

2.2.1 AI HW Accelerators

Workloads involving AI are commonly processed using CPUs, Graphics-Processing-Units (GPUs), and FPGAs architectures. Although such platforms excel in high-performance AI domains like Large-Language-Models (LLMs), they are not suited for safety-critical edge implementations, especially within the automotive industry, as strict real-time constraints demand low computational latency. Therefore, these constraints are predominantly met by employing specialized Application-Specific Integrated Circuits (ASICs) or AI-focused accelerators embedded in SoCs.

Figure 2.3 presents a schematic overview of a typical AI HW accelerator architecture. Such modules are usually integrated within the SoC, functioning as peripherals under the management of the host CPU. Given the repetitive nature of AI algorithms, the architectural design favors a data flow processing paradigm. This strategy reduces control costs and improves productivity through the use of deeply

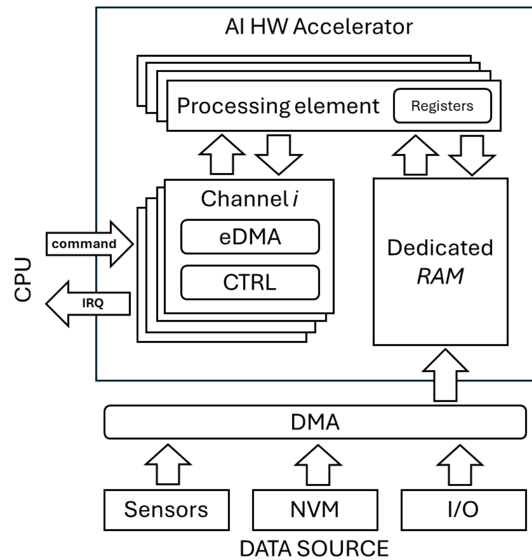


Fig. 2.3 Generic Block diagram of an AI HW Accelerator [2].

pipelined Processing Elements (PEs) [48]. Within these PEs, specialized Arithmetic Logic Units (ALUs) are employed to perform AI-related calculations.

Aside from the core computational logic and simplified control units (not explicitly shown in fig. 2.3), AI HW accelerators comprise several essential functional blocks:

- **Dedicated Random Access Memory (RAM) Memory**: Acting as the data source for the PEs, this memory is loaded from sensors, Non-Volatile Memories (NVMs), or I/O interfaces, usually facilitated by a system Direct Memory Access (DMA).
- **Bus Interface**: It handles multiple input channels and integrates a DMA to handle data migration to and from the accelerator. It contains control and configuration registers, which facilitate interaction with the CPU for receiving instructions or asserting Interrupt ReQuestss (IRQs) when tasks finish or arithmetic exceptions arise. Additionally, it provides accessibility to the internal dedicated RAM.

2.2.2 Limitations of Current Stress Methodologies for AI Accelerators

While general-purpose BI strategies are well-established, applying them to AI HW accelerators presents unique challenges that remain under-addressed in the literature. First, black-box IPs, where accelerator netlists are protected, renders traditional netlist-based stress generation infeasible for system integrators. Second, unlike general CPUs, AI accelerators rely on dataflow architectures; standard functional programs do not achieve high switching activity in these deeply pipelined structures without specialized data patterns. Existing methodologies either rely on structural scan-chains, which fail to replicate high-current mission-mode stress, or generic functional benchmarks that are unoptimized for the specific toggle-maximization required during BI. This thesis addresses this gap by proposing a netlist-independent functional stress generation methodology specifically tailored for AI dataflow architectures.

2.2.3 System-Level Test (SLT)

In the past decade, companies started to introduce an additional test phase into the manufacturing test flow, referred to as the SLT [43, 16, 49]. At the same time, ATE providers have begun supporting SLT [50, 51]. SLT has become a critical stage in the semiconductor production process, designed to further reduce test escapes from previous steps for several reasons:

- **Ever-increasing quality requirements**: With the growing integration of ICs into everyday object, reliability has become key, especially in safety-critical domains.
- **Pushing the limits of technology**: Compliancy to performance and complexity requirements may result in aggressive scaling and advanced packaging, which can introduce defects that are not easily detected by traditional tests.
- **High number of defects not detected by testing**: Even with approximately 95% coverage, the absolute number of undetected defects remains significant. For example, a SoC with 30 million possible defects and 95% coverage still leaves 1.5 million undetected defects, likely distributed throughout the layout.

SLT primarily involves executing functional applications that emulate the device's In-Field behavior, including workloads and environmental conditions. A common baseline benchmark consists of booting and running a Real-Time Operating System (RTOS) with workloads resembling operational scenarios. The SLT phase encompasses multiple aspects of modern safety-critical devices, as described in [43]:

- **Hardware/Software Interactions**: including dependencies among clock and power domains [52].
- **RTOS Bootstrap Interactions [15].**
- **Workload-Dependent Aging**: errors induced by effects such as activity-related supply voltage droop and cross-talk, which are exacerbated under high memory access rates and accelerate device aging [53].
- **Hardware-Implemented Protocols**: such protocols cannot be tested properly through structural testing [54].
- **Complex Hardware Resource Management**: including dynamic system reconfiguration [54].
- **Heavy Workload Exercise**: stressing processor pipelines through architecture-specific operations, floating-point coprocessors, concurrent multi-threaded operations, TLB misses/hits, cache accesses, and tag fields [54, 45].
- **Back-End Assembly Structure Damage**: induced by aging or thermo-mechanical stress [55].

For SLT to effectively identify faulty devices, it must encompass all these aspects, thereby complementing structural tests. This phase is crucial for modern, complex semiconductor devices, as scan- and BIST-based approaches focus primarily on individual component testing without emulating real operational conditions. Consequently, such structural methods fail to properly exercise interactions between the internal and external of the chip components or hardware and software. For instance, hardware implemented protocols such as CAN and Serial-Pheriperal-Interface (SPI) are not thoroughly tested by structural methods. Loopback strategies are often employed to minimize ATE interaction, which can cause coverage gaps in detection and correction logic [54]. Additionally, scan and LBIST techniques may introduce

untested logic when the architecture is partitioned into separate domains or islands, potentially leaving inter-domain logic structurally untested. SLT provides superior activation stimuli compared to LBIST, as it functionally exercises the entire device [10, 43, 16, 49, 39].

SLT primarily targets fault detection for test escapes in earlier manufacturing phases. SLT adopts a holistic approach by applying intensive functional workloads that stimulate complex hardware-software interactions across all on-chip and off-chip system components, including intricate protocol functions, while requiring additional ATE capabilities such as driving communication pins directly from the tester.

2.3 System-Level Test State-Of-The-Art

SLT has become a key element in the semiconductor testing flow, managing to address the limitations of traditional structural and functional test methodologies [56, 15]. As ICs continue to increase in complexity, SLT plays a crucial role in ensuring product quality, reliability, and performance under real-world operating conditions [49].

Furthermore, integration with other testing phases, such as BI testing and post-silicon validation, has become increasingly common to improve the efficiency of the overall test flow [57]. Despite its advantages, SLT faces several challenges, including high testing costs due to specialized hardware requirements, prolonged testing times, and portability of functional procedures across different architectures [39]. Furthermore, debugging and integration with other testing phases remain complex tasks [**?**].

SLT has achieved wide adoption in industries demanding high reliability and performance, such as automotive, mobile and consumer electronics, and data center applications [39, 58]. Case studies from major semiconductor companies have demonstrated its effectiveness in reducing Defective Parts Per Million (DPPM) and improving overall test quality [16, 49].

2.3.1 Critical Gaps in SLT Automation and Portability

Despite the widespread usage of SLT, key limitations are *scalability* and *reusability* of test content. Current practices often rely on manual development of functional tests, which are tightly coupled to a specific SoC's architecture. The literature lacks a framework for automated functional test generation that tries to abstracts hardware details. As such, migrating test content between different SoC architectures usually requires an enormous manual effort, thus possibly introducing human error and increasing time-to-market. This thesis addresses these limitations by proposing a DTS-driven framework that automates the abstraction and generation of SLT programs.

2.3.2 Stress induced by SLT

The increasing complexity of modern SoCs and the strict requirements for safety-critical systems have led to the development of advanced testing methodologies. In the automotive domain. among these, the BI process has become a fundamental step in identifying early latent defects in electronic devices. However, traditional BI procedures are characterized by high costs and long durations, thus novel methodologies to lower costs and improve stress are required.

The optimized Test-During-Burn-In (TDBI) process integrates functional testing into the BI process, enabling stress and test operations to be performed simultaneously, i.e. flash memory cycling, RAM stress, and CPU functional stress can occur simultaneously, thus significantly reducing the duration of the BI process and costs. Experimental results on automotive SoC have demonstrated a significant reduction in time and fewer recycled chips, confirming the effectiveness of TDBI in improving productivity and cost efficiency [13].

To further reduce BI costs, low-cost tester architectures have been introduced. These testers use general-purpose microcontroller units with embedded processors and limited memory resources to generate pseudo-random stress patterns. By relying on on-the-fly pattern generation rather than extensive memory storage, these architectures manage to achieve high toggle coverage and stress metrics comparable to traditional ATPG-based solutions. Their integration into SLT systems also enables

seamless functional testing under operational conditions, improving the reliability of automotive SoCs [59].

Recent developments have explored the combination of BI and SLT into a single step. This approach leverages SLT's functional testing capabilities to identify undetected faults by structural tests, while BI provides worst-case operational conditions during tests. The combination leads to significant cost reductions through minimized handling and equipment use, while maintaining high fault coverage. Industrial case studies confirm the viability of this approach, especially when coupled with high-voltage stress techniques to further decrease BI duration [57].

2.3.3 State-Of-The-Art for Communication Peripherals Testing

Since the introduction of SoCs, designers have increasingly integrated a growing number of peripherals into a single die. This evolution has expanded both the complexity and performance of SoCs, consequently raising the testing effort required. Several approaches have been proposed in the literature to effectively test communication peripherals. Within the scope of this work, table 2.1 compares various test methodologies for communication peripherals.

Test Nature	Test Approach	Pros	Cons	Target
Structural	ATPG	Automated High coverage capabilities	No functional interactions, No online testing, Low Speed Testing	All
	BIST [60, 61]	At-speed testing High coverage capabilities Limited online testing (startup)	Area overhead Limited capabilities for TDF No functional interactions	SPI, RS-232
Functional	SBST [62]	At-speed Online testing Deterministic and automated methodologies	Simple protocol operations No off-chip communications	UART, HDLC, ETHERNET
	SBST [63]	At-speed Online testing	No off-chip communications	CAN
	SBST [64]	At-speed Online testing Off-chip communications	Simple protocols No errors injector	UART, PIA

Table 2.1 Comparison of different test approaches for communication peripherals [4].

The ATPG is the most common approach for testing communication peripherals, as provides high fault coverage while being completely automated. However, ATPG lacks online, and at-speed testing capabilities. Moreover, scan-based tests fail

to properly activate and excite functional paths in communication peripherals, as discussed earlier.

Still under the scan-based test umbrella, BIST mechanisms are often used. Although BIST manage to achieve high at-speed coverage for stuck-at faults and limited online testing during SoC startup, it requires significant area, resulting in considerable overhead [60, 61].

2.3.4 Limitations in Testing Error-Correction Logic

While the table highlights the trade-offs between structural and functional tests, a significant gap remains in the ability to test **error detection and correction logic** under functional conditions. Standard Software-Based Self-Tests (SBST) or loopback tests typically exercise only the "happy path", valid transmission and reception, but fail to systematically provoke error conditions (e.g., CRC errors, timing violations) that trigger the peripheral's safety mechanisms. Structural tests scan this logic but do not validate its dynamic response to corrupted frames. There is currently no standardized methodology that integrates an external *error injector* into the SLT flow to purposively corrupt communication frames and validate the SoC's functional response to faults, which is a key contribution of this thesis.

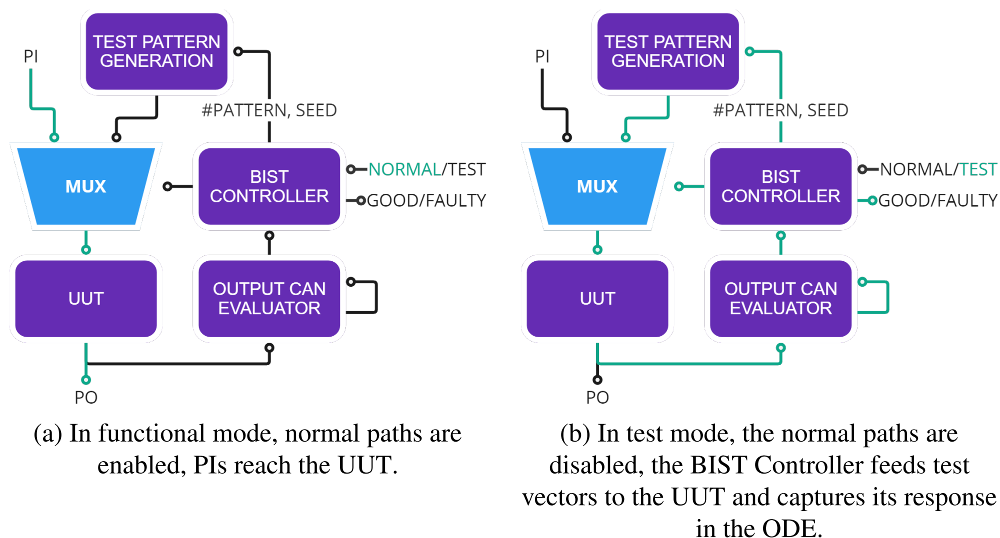
2.4 Built-In Self-Test (BIST)

Very-Large-Scale-Integration (VLSI) technology has fundamentally transformed testing methodologies. Boundary scan techniques have been adopted as a DfT strategy aimed at detecting as many faults as possible during the production phase. This approach stimulates the DUT through dedicated DfT hardware and scan chains. Test patterns are applied to the structure inputs, and the resulting outputs are monitored and compared with the expected fault-free behavior. However, the activation and control of these structures are complex and therefore require the use of ATE.

The continuous advancement of technology has made the testing environment increasingly demanding. Keeping pace with the growing complexity of modern devices presents significant challenges, particularly in safety-critical sectors such as automotive, where stringent reliability standards are applied.

As operating frequencies, pin counts, and associated costs increase, traditional ATE solutions have become obsolete. To address these challenges, manufacturers have started to introduce auxiliary modules capable of performing on-chip self-tests, thus reducing dependence on external equipment. This technique, known as BIST, exploits scan chains that would otherwise remain unused after manufacturing. It enables exhaustive testing with minimal hardware and without the need of companies to reveal confidential design information.

The SoC, the Unit-Under-Test (UUT) can operate in either functional or test mode. In test mode, Primary Inputs (PIs) and Primary Outputs (POs) are disconnected. Test patterns generated by a Test Pattern Generation (TPG) are applied to the inputs via the BIST controller, while the outputs are connected to an Output Data Evaluator (ODE), which verifies that the produced final signature matches the expected reference one. Typically, the ODE updates its internal signature after each BIST cycle using a Multiple Input Signature Register (MISR). Since the MISR signature depends on both previous and current test results, it is sufficient to compare the final generated signature with the reference one, rather than verifying each output individually. This concept corresponds to the standard STUMP architecture illustrated in fig. 2.4 .



(a) In functional mode, normal paths are enabled, PIs reach the UUT.

(b) In test mode, the normal paths are disabled, the BIST Controller feeds test vectors to the UUT and captures its response in the ODE.

Fig. 2.4 Standard STUMP BIST architecture showing its primary functional blocks [3].

LBIST is the application of BIST to logic modules. Its implementation typically includes the following components:

- **LBIST Controller**: its the module that chooses between test and functional modes and manages LBIST configuration.
- **Pseudo-Random Number Generator (PRNG)**: generates test patterns for the UUT.
- **MISR**: accumulates signatures for each LBIST cycles, producing a final signature that depends on all test outcomes.

The LBIST controller is programmable and supports several key configuration parameters:

- **Pattern-Count-Stop (PCS)**: defines the total number of patterns to be generated by the TPG.
- **Initial Seed**: the starting value for the Pseudo-Random Number Generator (PRNG) used to generate patterns.
- **Frequency of Application:** the frequency at which tests are applied to, which can be adjusted based on testing requirements.

Once initialized, the LBIST begins the self-testing procedure by performing the following iteratively sequence:

1. The Linear Feedback Shift Register (LFSR) generates a pseudo-random pattern from the current seed.
2. The pattern data is shifted in, and consequently data is shifted out of the scan chains, and the MISR is updated.
3. If the number of shifts is less than the total required, repeat step 1.
4. Apply the generated pattern by pulsing capture clock cycles.
5. If the number of patterns is less than the PCS, return to step 1.
6. Upon completion, the MISR signature is compared against a golden reference value.

2.5 Logic Diagnosis

Minimizing yield loss during IC production remains a major objective in the semiconductor industry [65, 66]. Consequently, the study of the relationship between physical defects and logical failures in digital circuits has gained significant attention.

Logic diagnosis is the process of identifying and localizing faults that cause malfunctions or operational failures in digital systems. Its goal is to determine the root causes of failures, thus possibly guiding improvements in next generation designs and test development. This process begins with the detection of abnormal circuit behavior. By analyzing the circuit responses to various test patterns and comparing them against expected references values, logic diagnosis can isolate defective components.

However, logical diagnosis is computationally demanding requiring a lot of memory. A modern area of research is the development of solutions capable of efficiently storing the data that help during fault localization. These systems, known as *fault dictionaries*, allow the identification of sets of potential candidates locations that are responsible for the observed faulty behavior.

Fault dictionary research has been extensively explored [67, 68]. Various methods have been proposed to construct these structures efficiently, as their size and performance are strongly influenced by the number of faults considered. As the fault space grows, balancing computational feasibility and memory efficiency becomes increasingly difficult.

Although powerful, the design of fault dictionaries requires a balance between storage size and diagnostic accuracy: higher accuracy of fault identification increases the memory requirement of its dictionary, while smaller dictionaries result in a worst diagnostic granularity. Therefore, it is essential to achieve an optimal balance between accuracy and resource efficiency.

Boppana *et al.* [69] introduced a tree-based fault representation method. This approach is advantageous because it preserves the classification of faults derived from the test pattern set. The faulty behavior of a circuit can be traced by traversing the tree from the root to a leaf node, where each leaf corresponds to an equivalence class of faults producing identical responses for all applied patterns. fig. 2.5 provides an illustrative example using three test patterns ($T1$, $T2$, and $T3$) and a fault universe represented by labels A through I.

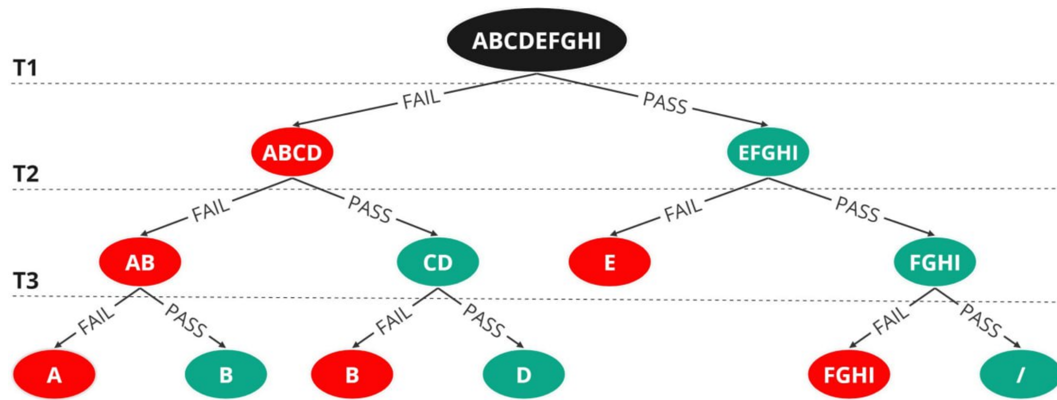


Fig. 2.5 Tree-based fault dictionary example with three applied test patterns [3].

2.6 Silicon Life-cycle Management (SLM)

For safety-critical systems, testing does not conclude after manufacturing, as detailed in section 2.1. Instead, continuous monitoring throughout the device's lifetime is required to ensure long-term reliability. As a result, Silicon Life-cycle Management (SLM) has emerged as a vital research and industrial focus. SLM encompasses the entire device life-cycle, from design to In-Field deployment, aiming to collect and analyze operational data to enhance performance and reliability [22].

In SLM, field testing plays an essential role in continuously assessing device reliability during operational phases and detecting performance degradation. Meeting reliability standards across different application domains is especially important for safety-critical devices. Field tests can be executed during power-on, power-off, or normal operation phases [70].

In particular, on-chip LBIST mechanisms (as discussed in section 2.4) are widely utilized to detect In-Field failures and support LBIST implementations [71, 72].

2.7 State-of-the-art on Logic Diagnosis for LBIST-Caught Failures

LBIST-based logic diagnosis has become a prominent research topic due to the growing importance of In-Field testing methodologies. Several notable approaches have been presented in the literature.

Elm *et al.* [28] proposed a Built-In Self-Diagnosis (BISD) architecture that extends BIST for field diagnosis. The approach stores fault signatures in a dedicated memory and ranks faults within the complete fault universe based on bitwise similarity between collected and reference signatures.

Jayalakshmi et al. developed a methodology to minimize testing time and memory usage in high-volume tests. Their two-stage process employs GO/NOGO patterns in the first stage to determine the need to run diagnostic patterns in the second stage.

Ubar *et al.* [30] introduced a diagnostic weighting technique for BIST environments using deterministic patterns stored in dedicated memory. The method balances the number of patterns with available memory resources to achieve the desired diagnostic accuracy.

Cheng et al. proposed a method that uses the error propagation function that maps scan cell failures to MISR outputs, enabling the identification of candidate failures within intersecting logic cones. This approach does not require architectural modifications and relies solely on the MISR signature. A subsequent study [32] improved this methodology by incorporating information from scan cells corresponding to the correct MISR bits, further improving diagnostic resolution.

Based on on-chip test generation, Gopalsamy et al. extended the deterministic test vector permutation technique introduced in Pomeranz. The approach constructs deterministic test sets to reduce candidate fault sets and improve diagnostic accuracy, as demonstrated on reference circuits.

As discussed, the BISD approach [28] relies on custom architecture where failure signatures stored in memory are analyzed by ranking faults according to signature similarity with precomputed reference signatures.

Table 2.2 reports the most relevant LBIST-based logic diagnosis methodologies, including their reported case studies, and context of application.

2.7.1 Limitations in Diagnostic Capabilities for Constrained Environments

Despite the variety of methods presented in Table 2.2, a critical gap persists regarding diagnostic applicability in constrained automotive environments. Most existing

Table 2.2 Logic diagnosis LBIST methodologies.

Method	Brief Description	Reported Case of Study	Application
BISD: Scan-based Built-In self-diagnosis [28]	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 [31, 32]	Authors propose exploiting error propagation function that maps scan capture failures to MISR	Industrial Circuits $\approx$ 1.3, 1.9M gates	Logic Diagnosis
A Storage Based LBIST Scheme for Logic Diagnosis [33, 34]	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 [73], ITC'99 [74], 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 [74] and **Automotive SOC** 40*nm* $\approx$ **20M gates**	In-field Data Collection & Logic Diagnosis

approaches assume a degree of flexibility, such as reconfigurable BIST engines, and reprogrammable test vectors, that is often unavailable in safety-critical SoCs once deployed in the field. In real-world automotive scenarios, LBIST engines are often "locked" to fixed seeds and configurations to guarantee deterministic execution within a precise time window. The SOTA lacks a diagnostic workflow that relies *solely* on standard, minimal data (like the MISR signature and failing pattern index) without requiring architectural changes or flexible test reconfiguration. This thesis targets this specific limitation by developing a method to pinpoint faults using only the "First Failing Pattern" index, which is readily available in standard automotive LBIST controllers.

Chapter 3

The proposed methodologies

This chapter reports the methodologies that were the result of the research activities of this thesis. Following the order of the semiconductor manufacturing flow, each section introduces a technique designed to strengthen reliability and diagnostic observability at a specific stage of a SoC device's testing phase. The progression begins with early-life stress testing during manufacturing, proceeds through functional and SLT, and culminates in the field.

Section 3.0.1 details the proposed approach designed to optimize stress stimuli during the BI phase for AI HW accelerators when the netlist is not available. Section 3.0.5 addresses structural tests weaknesses for communication peripherals, those are then taken as the starting point for a SLT functional test suite generation. Section 3.0.9 examines the use of DTS to automate SLT test generation, specifically targeting peripherals. Finally, section 3.0.12 discusses the post-manufacturing phase, for In-Field data collection and logic diagnosis to specifically address NTF scenarios for manufacturers.

3.0.1 Burn-In Functional Stress Suite for AI HW Accelerators

Modern SoCs incorporate IPs from third-party vendors, often with limited visibility into their internals. The proposed methodology addresses such cases, resorting to the creation of functional workloads through indirect indicators, specifically, current consumption and memory overview, thus eliminating the necessity on netlists. This approach enables system integrators and manufacturers, to provide effective stress of

an AI accelerator based solely on functional execution and the specifications detailed in user manuals.

The procedure begins with the creation of a suite of functional patterns that will be executed on the AI HW under stress. In this regard, a functional stress pattern is defined as the data content that will be loaded by the AI HW into its dedicated RAM, which is then processed by the PEs. Such patterns may be sourced from existing verification suites or synthesized using algebraic rules, such as enforcing specific distributions of logic '0' and '1' values.

The initial pattern population can be generated randomly, to respect defined proportions of logic 1s and 0s within the memory space. Given the operational constraints of the AI HW Accelerator, this generation is facilitated by an offline tool that ensures functional validity, guaranteeing that patterns do not trigger exceptions or arithmetic overflows. In terms of structural diversity, the tool ensures a random distribution of logic values across the memory map. However, this does not necessarily reflect realistic workloads, as the primary objective during the BI phase is to functionally activate all logic cones associated with the accelerator's memory interface. This pattern generation process serves as a foundational step, which can be augmented by other tools or manual refinement.

Once the candidate population of functional stress patterns is created, the methodology proceeds to measure the current consumption of each individual pattern on real devices.

The current consumption observed for each functional stress pattern acts as an indirect proxy for the stress activity induced within the AI HW Accelerator [75]. Consequently, the method ranks patterns in descending order of current draw; higher consumption implies greater expected toggle activity across the circuit nodes.

Although current consumption is a key criteria during selection, relying solely on high-current does not guarantee good or uniform stress distribution. High demanding patterns may exhibit redundant stress patterns, repeatedly activating the same areas of the circuit while leaving others inactive.

To refine the selection, a heuristic algorithm is employed to sift the ranked list using MTMs. This process filters out redundant patterns by analyzing the activity within the AI HW Accelerator's memory elements (e.g., dedicated RAM or control

registers). The result is a compact set of functional stress patterns that, when combined, achieve high-quality stress coverage.

3.0.2 Current Consumption Measurement Flow

The measurement flow utilized to grade the efficacy of functional stress patterns requires a setup capable of synchronizing measurement instrumentation with specific firmware executed by the DUT.

In the SOTA there are various studies that have detailed current probing configurations, ranging from direct methods (e.g., inductive probes [76]) to indirect techniques (e.g., shunt resistors coupled with oscilloscopes [75]). A crucial requirement is that the sampling frequency must be comparable to the device's operating frequency; if sampling is too slow, measurements should be aggregated over multiple AI HW runs.

The flow begins by configuring the measurement system, then the control firmware is executed. A schematic overview of the bare-metal firmware architecture is provided in fig. 3.1.

The firmware runs on the CPU, while the accelerator AI HW do its tasks with the provided data. The measurement instrumentation is activated by the CPU via digital I/O pins (e.g., signaling the oscilloscope).

The firmware initializes the AI HW, which involves loading data into its internal RAM. Subsequently, N executions of the same functional stress pattern are programmed. During the execution of the AI HW accelerator, the CPU CPU enters IDLE mode to minimize its contribution to total power consumption. Upon completion of each execution, an IRQ interrupt reactivates the CPU CPU to trigger the next iteration.

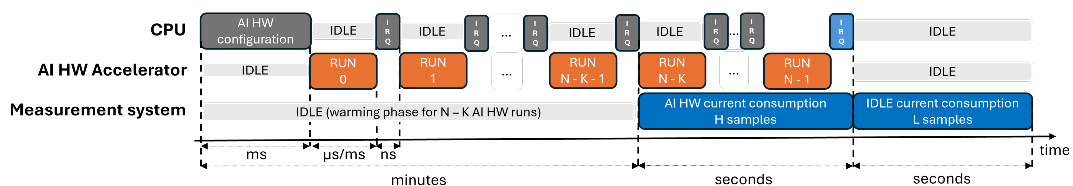


Fig. 3.1 Proposed firmware execution flow used to grade the current consumption of a given functional pattern [2].

As depicted in fig. 3.1, the measurement system remains inactive for an initial period of K runs. This warm-up phase allows the SoC to reach thermal and electrical stability [77]. Early measurements may yield low consumption values compared to the system at steady-state. Furthermore, evaluating patterns under transient conditions can lead to inconsistent data.

The CPU firmware tracks the warm-up runs and activates the measurement system at the appropriate time. In fig. 3.1, the IRQ following the K-th run enables measurement for the remaining $N-K$ cycles. Ideally, N should equal $K+1$ if the sampling rate matches the working frequency. However, due to instrumentation limits, multiple runs are often required. Thus, N is determined dynamically based on the ratio of sampling to working frequencies [78], and standard practices for noise reduction.

In details, the K parameter indicates the point at which the system reaches a steady current consumption. Only samples collected after this point, $\{I_K, I_{K+1}, ...I_{N-1}\}$, are used for analysis. The total count of N runs ensures that sufficient steady readings are collected to have a high degree of statistical confidence for the collected data. The assessment of measurement uncertainty, combines instrumental uncertainty (Type B) and repeatability (Type A). Instrumental uncertainty refers to systematic errors based on sensor specifications and calibration, usually given by the sensor's manufacturers. Repeatability is evaluated using the experimental standard deviation σ_K of measurements at steady state, taking into account random noise and environmental fluctuations.

To mitigate random noise, the final measurement is derived from the average of H samples under steady conditions, thus reducing noise to the inverse of the square root of H. The relative standard uncertainty (type A) is computed as:

$$\mu_{A,r}(\bar{i}) = \frac{s \times 100}{\sqrt{H} \times \bar{i}}$$

This analysis and the dynamic selection of K and N ensure that the collected values are accurate and experiments repeatable.

During the measurement phase, each execution generates H samples. Although at the end of each execution there is an IRQ handled by the CPU, which adds ε samples, its duration is negligible compared to the computation time of the AI HW accelerator, making its impact almost zero.

After completing the $N-K$ measurement runs, the power consumption is calculated as the average of the collected samples. In this context, current consumption (c^i) denotes the sample average ($\mu(x)$).

Next, the inactive consumption of the system is measured using L samples. Since SoC is already stable, L only needs to be large enough to account for measurement error.

The samples from this idle phase form a subtractive term [79], calculated as the average of L samples. Although optional in temperature-controlled environments (e.g., climate chambers), this step is highly recommended in standard laboratory environments to compensate for variations in ambient temperature.

Despite its usefulness, current consumption alone can introduce "aliasing," in which multiple patterns are classified as equally effective because they activate the same regions of the circuit, not providing uniform stress. To overcome this problem, metrics related to the accelerator's memory are integrated into the evaluation.

3.0.3 Memory Toggle Metrics (MTMs)

MTMs are introduced to enhance the evaluation of functional patterns. These metrics analyze memory elements visible from the programmer's perspective, such as operand memory banks or registers.

The idea of MTMs is to quantify the activity of memory elements during the execution of a pattern. In detail the used metrics for grading are:

- **Memory Toggle Coverage (MTC)**: Measures the percentage of bits in memory that transitioned (0 to 1 or 1 to 0) during pattern application, similar to TC in eq. (2.1).
- **Memory Toggle Activity (MTA)**: Counts the number of total transitions per each memory element. It is split into:
  - $MTA_{Average}$ **or** $AVG(MTA)$, analogous to eq. (2.2).
  - $MTA_{Variance}$ **or** $VAR(MTA)$, analogous to eq. (2.3).

The hypothesis driving MTMs is that high memory toggle coverage and activity correlate with increased toggle activity in the underlying circuit nodes. Conversely,

low MTC and MTA values suggest partial circuit activation. A static value in a memory bit often implies low testability for the connected logic cone, effectively feeding a constant value into the sea of gates. High gate coverage typically necessitates high gate activity. As shown in fig. 3.2, higher MTC generally corresponds to increased gate activity, while lower VAR(MTA) indicates a more uniform distribution. Priority is given to MTC, as low variance alone does not guarantee high activity (e.g., the 25% MTC case has lower variance but lower activity than the 50% case).

Figure 3.3 provides a visual example of MTC: memory elements are displayed in a grid where each cell represents an observable bit. The resulting map is computed by overlapping the memory transitions of functional patterns.

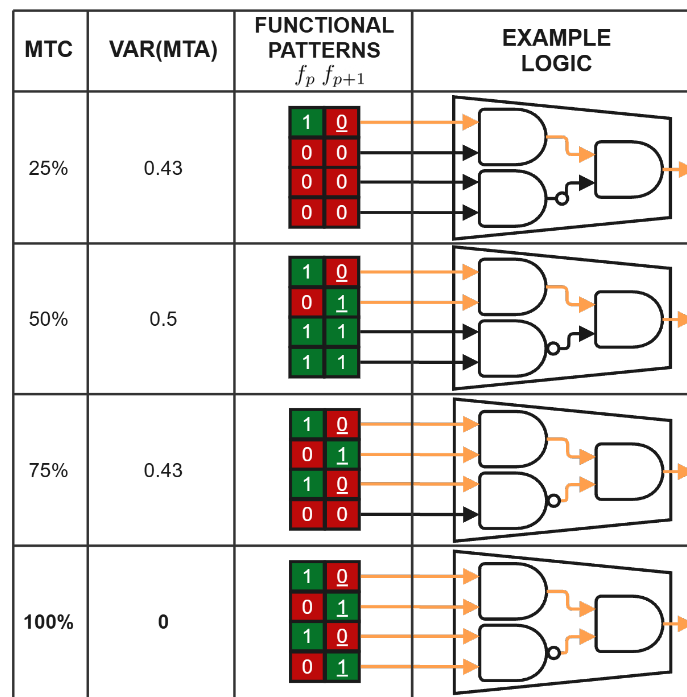


Fig. 3.2 Visual representation of the relationship between MTC, VAR(MTA), and gate activity across two functional patterns [2].

3.0.4 Functional Stress Pattern Selection Algorithm

The functional patterns are selected by means of an algorithm that consider collected indirect measurements.

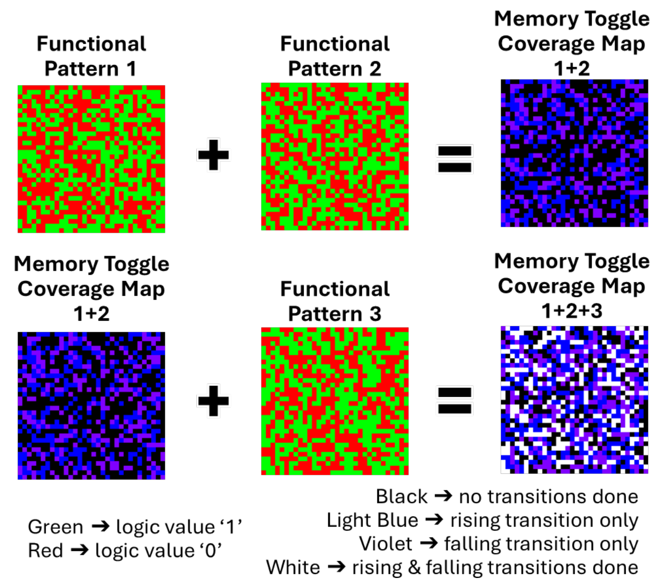


Fig. 3.3 Example of Memory Toggle Coverage computation [2].

Figure 3.4 illustrates the algorithm's flow. It begins with the list of functional patterns sorted by descending current consumption. The algorithm iteratively evaluates the top-ranked pattern to see if it increases the cumulative MTC of the selected suite. If it does, the pattern is added. The highest-ranked pattern is automatically included, while subsequent patterns undergo evaluation.

If a selected pattern does not improve MTC, the algorithm checks if it improves (reduces) VAR(MTA). In such cases, it is added to the final set, otherwise, it is discarded.

This process is repeated until MTC reaches 100%, ensuring that all memory elements accessible to the programmer are activated at least once.

The algorithm prioritizes high-current patterns as they are the most promising for circuit node activation [78]. However, to ensure diversity, MTMs are employed to prevent the selection of redundant programs that stress identical logic cones. Once the final set is selected via this heuristic rank-and-sift approach, the suite can be executed cyclically during BI for the required duration.

From a functional point of view, the rank-and-sift algorithm is simple and based on the selection of sorted lists. Its novelty lies in the combination of indirect measurements (current and MTMs) to classify patterns in two stages, sorting them

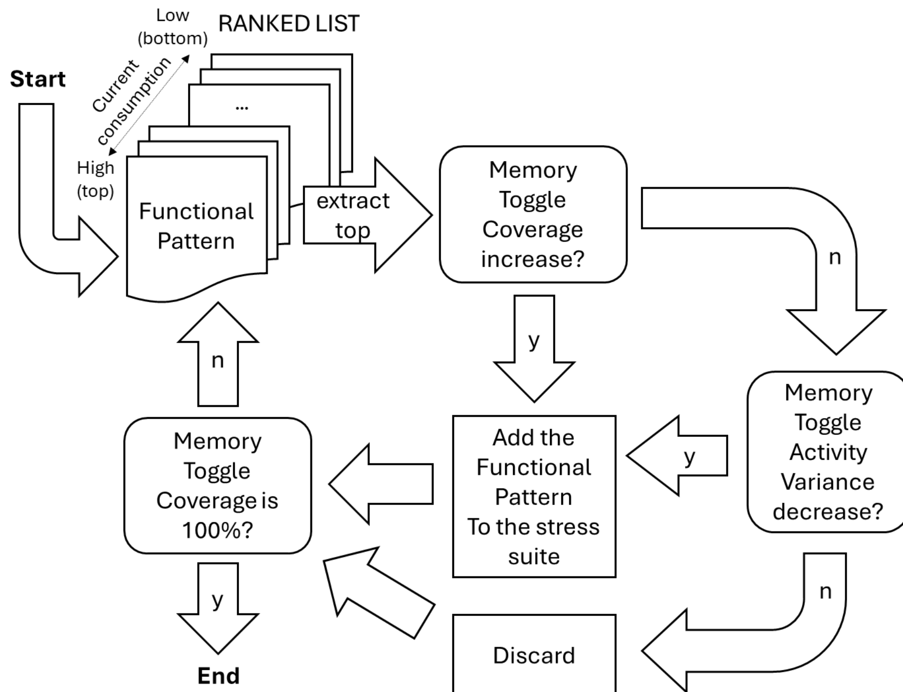


Fig. 3.4 Rank and Sift Selection Algorithm [2].

according to current consumption and refining them according to memory metrics, without requiring netlist-based measurements.

3.0.5 System-Level-Test for communication Peripherals

This thesis presents a methodology for generating an effective SLT suite of functional test programs designed to complement structural tests, with a specific focus on communication peripherals in SoCs.

The proposed methodology advances the current SOTA in SLT by eliminating the dependence on holistic assumptions, such as considering system boot as an adequate SLT strategy. In the SOTA, this approach is the first to explicitly analyze potential weaknesses in structural testing. The resulting analysis guide the development of an SLT suite tailored for communication peripherals, aiming to detect faults that structural tests may miss in the overlooked areas regions of the circuit. The overall methodology flow is illustrated in fig. 3.5.

The first step is to identify potential weaknesses of the structural tests by analyzing the list of undetected faults obtained after evaluating the ATPG, MBIST, and

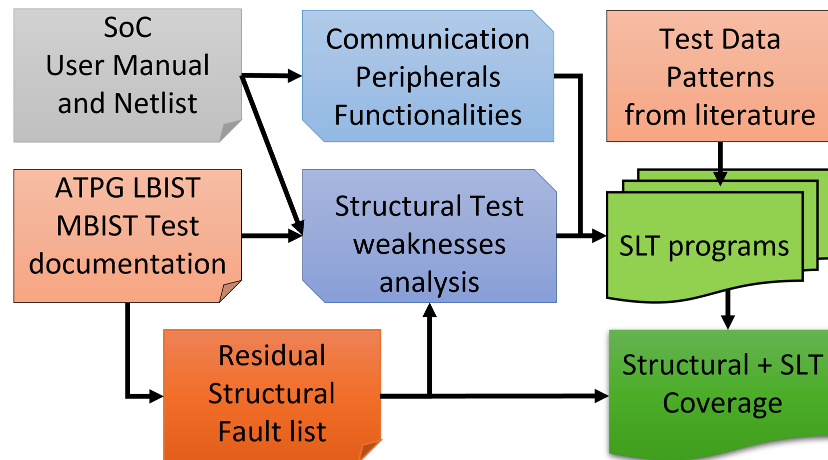


Fig. 3.5 Flow diagram of the proposed methodology [4]

LBIST approaches. This preliminary analysis guides following efforts in generating functional programs SLT that specifically target the circuit regions and fault that are overlooked in structual tests.

The generation of functional programs must encompass all operational modes and parameter configurations that an In-Field application may employ, as required by SLT. Additionally, particular emphasis is placed on functionalities that may be vulnerable due to structural testing limitations.

Stimulating off-chip interactions with the external environment through the communication interfaces exposed by the chip constitutes a key SLT point. Structural tests commonly constrain these interfaces to loop-back configurations. Consequently, the proposed methodology leverages the design of a flexible companion module capable of managing bidirectional communication between the ATE and the SoC under SLT, as illustrated in fig. 3.6. This companion module becomes essential when the MUT incorporates detection or correction capabilities, as such mechanisms cannot be effectively verified under loop-back conditions that always transmit uncorrupted bitstreams.

3.0.6 Structural Test Weaknesses Analysis

The analysis of structural test weaknesses is conducted manually with reference to the user manual, the netlist, and the residual structural fault list. The MUT is expanded as schematically illustrated in fig. 3.7. Arrows and labels are added to

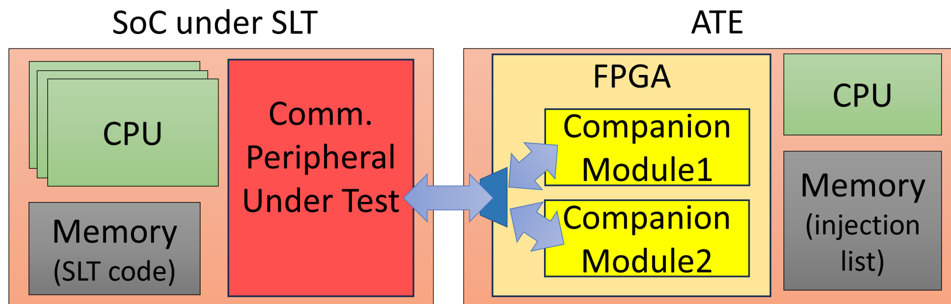


Fig. 3.6 Companion module view with a companion memory for error injection in the communication's protocol [4].

the sub-module indicating the criticality level of the interactions among modules, whereas the alphabetic labels group sub-modules according to their functionality (e.g., label "A" classifies all functionalities associated with the internal RAM).

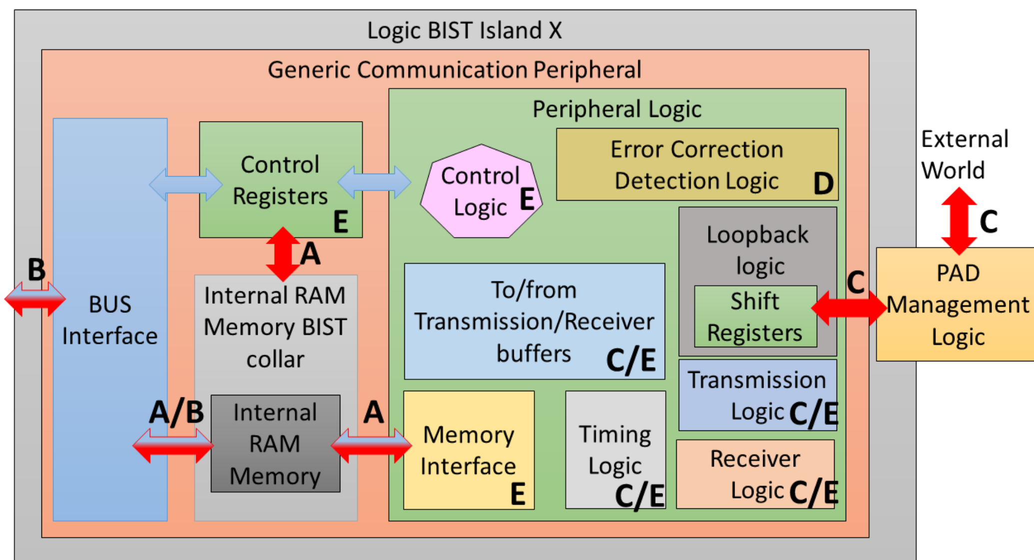


Fig. 3.7 Architecture of a generic communication peripheral [4].

The correlation between functional behavior and the limitations of structural testing becomes apparent in fig. 3.7. Consequently, when developing SLT functional programs, the following areas should be prioritized:

A **Embedded memory access ports**: may not be fully covered during structural testing due to the presence of collars and memory DfT circuitry such as MBIST [11].

B **Interfaces to other on-chip components**: may belong to different LBIST or Scan Chain domains, which can introduce testability issues [10, 80].

C **Transmission and reception interfaces towards the top of the chip**: some signals and pins connected to external SoC interfaces may not be tested during production.

D **Detection and correction logic circuits**: often consisting of large, deep logic functions that are difficult to target through structural testing [39].

E **Complex hardware–software functions**: complex protocol handling and synchronization mechanisms are typically not exercised during structural test [39].

An effective suite therefore comprises a series of programs designed to address all of the above considerations. The following subsections provide detailed guidelines for generating functional programs based on the identified criticality of areas neglected in structural testing.

3.0.7 System-Level Test functional program suite generation

The generation of the functional SLT suite addressing the identified critical elements builds upon previously established functional methodologies from the SOTA, as summarized in table 2.1. Existing techniques, such as those presented in [63, 81, 64], already cover several critical functionalities using standard test configurations. These include exploiting loop-back configurations between communication channels and resorting to normal transmission and reception operations. However, additional efforts are needed to cover structural coverage for modules that remain insufficiently explored in the literature, such as the Error-Management-Logic (EML), Bus-off sequence (if present), shared bus arbitration, and synchronization logic within communication peripherals.

The resulting test suite, combining both established and newly developed approaches, has been manually developed by test engineers using the SoC's documentation, the netlist, and the list of residual structural test fault lists. The proposed suite comprises 6 test programs carefully assembled to target circuit areas typically overlooked by structural tests. Among these, 2 programs require the involvement of

the companion module, which is essential for stimulating the communication with the external world.

The specifications of the functional programs are presented in the following sections using pseudo-code that employs generic communication peripheral functions, abstracting the underlying software implementation.

Regarding pattern selection, a call to the stub $get_pattern()$ is used in the pseudo-code. This enables the proposed algorithms to be executed multiple times with different data patterns, including those already established from SOTA methodologies such as [82].

Alternatively, patterns can be generated automatically using methodologies based on ATPG [82] or by resorting to random generation, provided that the cost of fault simulation remains acceptable. In these cases, a generation cycle can iteratively refine the set of patterns, while a deterministic approach requires only a single evaluation.

The subsequent subsections describe both the communication software and the companion module from an algorithmic perspective, addressing each consideration depicted in fig. 3.7. All presented algorithms produce a signature that is compared against a golden reference at the conclusion of each test execution.

Embedded Memory access ports

In SoC testing, embedded memory access ports are often not exercised by structural DfT circuits due to the nature of the MBIST architecture [11]. When the MBIST is activated, it isolates the memory from the remaining SoC components, thereby testing only a subset of the connections between the memory and the rest of the circuit. Communication peripherals typically include a dedicated memory block used for message storage. Its functionality can be effectively verified by populating the memory with incoming messages. The proposed test procedure employs all available transmission and reception buffers, when multiple instances are present. The CPU executing the SLT program transmits a burst of messages large enough to occupy all buffers completely. The pseudo-code representation of the test program is shown in algorithm 1.

Algorithm 1 Embedded Memory Access Port test [4].

```
1: signature ← 0                                        ▷ Init signature var
2: while peripheral_rx_buffers_are_full() ≠ True do
3:     data ← get_pattern()
4:     signarure ← signature ⊕ data
5:     send_data(data)
6: end while
7: wait_reception()
8: while peripheral_rx_buffers_are_empty() ≠ True do
9:     data ← receive_data()
10:    signature ← signature ⊕ data
11: end while
```

Interfaces to other SoC components

The utilization of communication interfaces connecting the SoC to other components, such as CPUs and DMAs, is inherently limited during structural testing. These interfaces are complex and often not properly excited by ATPG techniques, as they typically comprise multiplexers that are difficult to test. The challenge becomes even more complex when the SoC modules involved in communication are physically distant within the layout and belong to different LBIST islands [10] or scan domains [80]. Moreover, because structural tests must usually account for power consumption constraints, test patterns are generally applied sequentially.

Furthermore, communication protocols requires both digital and analog circuits with are usually located in distinct domains, which can further reduce the coverage of structural tests. For example, analog IP blocks are often bypassed during scan-based tests, resulting in communication logic not being properly tested.

The register interface is a set of registers responsible for controlling peripheral behavior, issuing commands, storing data for transmission and reception, configuring additional peripheral units (e.g., timing management logic), and reporting information from the peripheral controller, including status and interrupt registers. Reading and writing these registers ensures the correct use of the on-chip bus system and verifies the functional interactions between different LBIST islands. In the proposed algorithm, these operations are performed by exercising the bus in an active but not fully initialized state.

Additionally, many modern micro-controllers feature dedicated hardware capable of resetting modules across the SoC. This test involves stopping the peripheral,

initiating a reset, verifying the values of the registers through read-back operations, and then restarting the peripheral.

Let **R** denote the configuration and control register file of the peripheral. The corresponding test procedure is shown in algorithm 2.

Algorithm 2 Interfaces to other SoC components test [4].

Require: R, the set of the peripheral register file.
1: $signature \leftarrow 0$ ▷ Init signature var
2: stop_peripheral()
3: assert_functional_reset_peripheral()
4: peripheral_enable_clock() ▷ Avoid init
5: **for each** $r \in \mathscr{R}$ **do**
6: **if** r is init_register **then**
7: $r \leftarrow r \wedge (1 \ll INIT_field_pos)$
8: $signature = signature \oplus r$
9: $r \leftarrow r \vee \sim (1 \ll INIT_field_pos)$
10: $signature = signature \oplus r$
11: **else**
12: $r \leftarrow 0$
13: $signature \leftarrow signature \oplus r$
14: $r \leftarrow UINT_MAX$
15: $signature = signature \oplus r$
16: **end if**
17: **end for**

Transmission/Reception Interfaces to Chip Top

The transmission and reception interfaces connecting the chip to the external world are critical for ensuring correct message exchange. Structural testing of these interfaces is highly constrained by limited test access, as loop-back strategies are commonly adopted to minimize the number of pins that must be contacted and controlled by the ATE. Consequently, to functionally exercise both the transmission and reception logic, the procedure described in algorithm 3 is executed in conjunction with a companion module responsible for message sequencing, whose implementation is detailed in Section section 3.0.8.

The proposed test program targets the transmission and reception modules by initializing the peripheral to operate under various configuration modes. The corresponding pseudo-code representation is shown in algorithm 3.

Algorithm 3 Transmission/Reception to Chip Top test [4].

Require: B_{tx}, the set of transmission configuration modes.
Require: B_{rx}, the set of reception configuration modes.
Require: *shared*, data shared between ATE and the DUT.

```
 1: signature ← 0                                            ▷ Init signature var
 2: enable_companion_module("message_sequencer")
 3: for each btx ∈ 𝓑tx do
 4:     for each brx ∈ 𝓑rx do
 5:         configure_peripheral(btx, brx)
 6:         if is_transmission_enabled(btx) then
 7:             data ← get_pattern()
 8:             signature ← signature ⊕ data
 9:             send_data(data)
10:         else
11:             signature ← signature ⊕ shared
12:         end if
13:         wait_reception()
14:         data ← receive_data()
15:         signature ← signature ⊕ data
16:     end for
17: end for
```

Detection and correction unit

Many peripheral protocols are inherently susceptible to faulty behaviors. To enhance their reliability and robustness, they incorporate an EML that provides both error detection and correction capabilities.

The detection functionality enables the peripheral to identify transmission or reception errors in data, typically by means of detection codes embedded within the transmitted information. In addition to detection, correction logic allows the recovery of erroneous data through the use of Error Correction Codes (ECCs) transmitted alongside the payload.

Moreover, bus error detection mechanisms ensure that failing nodes can be identified on the communication bus. This prevents faulty nodes from disrupting the network by detecting incorrect messages and such nodes can autonomously disconnect from the bus, entering a *bus-off* state until they are able to re-synchronize, without errors, and safely resume communication.

The pseudo-code representation of the corresponding test program is provided in algorithm 4. In this case, a companion module capable of injecting errors into mes-

sage frames is required. Error injection can be performed using different strategies, as described in Section section 3.0.8.

Algorithm 4 Detection and correction unit test [4].

```
1:  signature ← 0                                          ▷ Init signature var
2:  enable_companion_module("pattern_recognizer")
3:  data ← get_pattern()
4:  signature ← signature ⊕ data
5:  send_data(data)
6:  wait_reception_or_errors()
7:  if peripheral_has_errors() then
8:      data ← get_failed_transmitted_data()
9:  else
10:     data ← received_data()
11: end if
12: signature ← signature ⊕ data
13: if support_bus_off() then
14:     wait_bus_off()
15: end if
16: disable_companion_module()
17: data ← get_pattern()
18: signature ← signature ⊕ data
19: send_data(data)
20: wait_reception()
21: data ← receive_data()
22: signature ← signature ⊕ data
```

Complex hardware-software functions

The transmission logic, previously examined in subsection algorithm 3, plays a crucial role in managing message transmission, particularly when a priority scheme is incorporated into the peripheral's protocol. In such protocols, the transmission handler includes complex control logic that performs a *transmission scan* to evaluate pending transmission requests and identify the one with the highest priority. The pseudo-code of the corresponding test program is provided in algorithm 5.

Communication protocols may rely either on a dedicated bus, where each node is connected individually to the others, or on a shared bus that interconnects multiple nodes. The latter requires additional logic, as each node must include functionality to perform arbitration over the shared channel. When a node initiates a transmission,

Algorithm 5 Complex transmission functions test [4].

Require: ID_1, ID_2, ID_3, s.t. $ID_1 > ID_2, ID_3 > ID_2$.

```
 1: signature ← 0                                  ▷ Init signature var
 2: data ← get_pattern()
 3: signature ← signature ⊕ data
 4: send_data(ID1, data)                           ▷ Send data with ID1
 5: data ← get_pattern()
 6: send_data(ID2, data)                           ▷ Send data with ID2
 7: data ← get_pattern()
 8: send_data(ID3, data)                           ▷ Send data with ID3
 9: cancel_transmission(ID2)
10: cancel_transmission(ID3)
11: wait_reception()
12: if rx_contains(ID2) ∨ rx_contains(ID3) then
13:     ▷ Failure
14: end if
15: data ← receive_data()
16: signature ← signature ⊕ data
```

it may encounter two possible conditions: the bus is idle (i.e., no other nodes are transmitting), or another node is already using the bus. In the first case, the active node can immediately take control of the bus; however, bus sensing alone is insufficient when N nodes attempt to transmit simultaneously. To address this, modern protocols introduce an initial transmission stage known as the *arbitration phase*.

The arbitration logic can be functionally verified using the proposed companion module operating in *message sequencer* mode, which requires a shared identifier between the SLT and the ATE. Assuming that a lower identifier corresponds to a higher transmission priority, the pseudo-code of the test program is provided in algorithm 6, while the implementation details of the *message sequencer* mode are discussed in Section section 3.0.8.

Finally, the timing characteristics of communication protocols must also be verified. This verification requires strict synchronization, and the use of a companion module capable of introducing arbitrary delays into transmission frames is essential.

Algorithm 6 Synchronization functions test [4].

Require: ID, predefined ID between ATE and this program.
1: $signature \leftarrow 0$ ▷ Init signature var
2: enable_companion_module("*message_sequencer*")
3: $data \leftarrow$ get_pattern()
4: $signature \leftarrow signature \oplus (ID+1)$
5: send_data(ID, $data$) ▷ Send *data* with *ID*
6: ▷ Companion senses a frame and transmits ID+1.
7: $data_ID \leftarrow$ receive_data_ID()
8: $signature \leftarrow signature \oplus data_ID$
9: $data \leftarrow$ get_pattern()
10: $signature \leftarrow signature \oplus (ID-1)$
11: send_data(ID, $data$) ▷ Send *data* with *ID*
12: ▷ Companion senses a frame and transmits of ID-1.
13: $data_ID \leftarrow$ receive_data_ID()
14: $signature \leftarrow signature \oplus data_ID$

3.0.8 Companion module

Multiple companion modules can be implemented on the ATE FPGA. These modules can coexist on the ATE, with their inputs and outputs multiplexed. The appropriate functionality is activated, and communication is established, during the execution of the SLT functional programs that require external interaction with the ATE.

Several companion module functionalities are required to complement portions of the SLT firmware executed on the device under test:

- Message reception and transmission to and from the MUT, referred to as the message sequencer.
- Error injection into specific segments of message data transmitted to the MUT.
- Delay injection during message transmission.

In message sequencer mode, the companion module transmits pre-programmed messages to the SoC. These messages are stored in a dedicated memory within the companion module on the ATE. The module in this mode comprises a central control unit and a memory component. The transmitted messages can be manually crafted, extracted from verification stimuli, or generated automatically resorting to pseudo-random methods. Upon receiving an enable signal, i.e. a digital General

Purpose I/O (GPIO), the module transmits the message and returns to an idle state upon completion.

For error injection three different approach can be used:

- Injecting errors in pre-defined areas provided before test execution (e.g., a list of injection times).
- Employing a pseudo-random injector that introduces errors at random times and locations within the message data frame (e.g., based on pseudo-randomly generated values).
- Utilizing a protocol-aware injector that intentionally corrupts protocol-specific features (e.g., by extending or shortening the duration of start and stop bits).

For delay injection, the companion module is required to introduce arbitrary transmission delays. These delays cause de-synchronization, thereby stimulating the communication peripheral modules responsible for communication timing. Moreover, when such delays lead to uncorrectable misbehavior, the peripheral's error detection mechanisms are also effectively exercised.

3.0.9 System-Level-Test test generation automation

Although SLT offers clear advantages, its development is still a time-consuming and error-prone process. It often depends on specific hardware architectures and ISAs, which makes portability and reuse across different SoCs difficult.

To address these limitations, this thesis presents a framework that automates the generation of functional SLT workloads using DTS. DTS provides a unified and hierarchical view of SoC hardware, describing components, their interconnections, and their capabilities. This structured description enables the automatic production of portable and reusable test code. At the center of the framework is an algorithm that builds and traverses a graph-based representation of the SoC, capturing both its structural organization and behavioral constraints to guide functional test generation.

3.0.10 Graph-based SoC Modeling

The proposed methodology represents the SoC architecture as an undirected graph, where each hardware component corresponds to a vertex and the links between components form the edges. This abstraction captures both the hierarchy and the internal connectivity of the system.

Master components—typically CPU cores—are modeled with detailed architectural attributes, as illustrated in fig. 3.8(a). These include registers, architectural sets, and memory structures such as instruction and data caches. Slave components, shown in fig. 3.8(b), represent peripherals described with finer granularity, incorporating registers, register fields, and memory blocks.

To reflect device-specific behavioral rules, such as protected register access sequences or conditional unlock procedures, the framework supports the definition of custom procedures called functional actions. These actions allow the automated handling of complex peripheral behaviors.

The framework also takes advantage of the descriptive and portable *compatible* and *model* keywords defined in DTS. These attributes support incremental loading and matching of device descriptions, enabling component reuse and smooth integration into the SoC graph. By decoupling the SoC description from the individual component specifications, the methodology enhances modularity and improves portability, especially when integrating third-party IP. This modular design removes the requirement for a single monolithic DTS file and encourages reusable, component-based hardware descriptions.

3.0.11 FSWGen Framework

The proposed framework, called *FSWGen*, is based on the algorithm presented in algorithm 7. It starts by loading the SoC together with all master and slave DTS files, followed by an initial exploration of the master nodes.

For each master, a compatible DTS file is identified and incrementally integrated into the main graph. A subsequent traversal is then performed to identify all nodes reachable from the master components. For each reachable slave, if a compatible hardware description is found, it is similarly incorporated into the SoC graph.

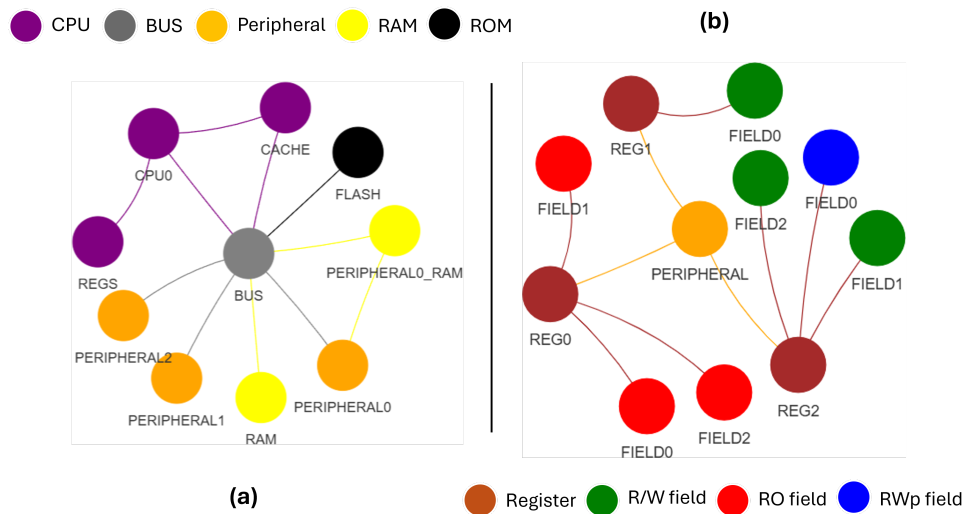


Fig. 3.8 (a) DTS graph of a basic example SoC and a peripheral (b) [5].

During this second traversal, functional SLT procedures are automatically generated for both master and peripheral nodes. These procedures, implemented in assembly code and executed by the master, perform read (R) and write (W) operations on the slave registers. When peripherals require specific unlock or access sequences, these actions are performed automatically based on the corresponding functional action descriptions.

3.0.12 In-Field Logic Diagnosis leveraging LBIST

This section presents the main contributions of the thesis concerning the post-sale operational phase of automotive devices. Portions of the work described have been published in [72] and [3].

The proposed methodology involves the In-Field collection of failure data and its subsequent use to perform logic diagnosis, relying exclusively on information gathered from constrained LBIST executions carried out by the embedded CPUs.

Figure 3.9 illustrates the flow of the proposed approach. The process begins with step (0), in which the FAB designs and markets the device. As detailed in this thesis, fault dictionaries are generated by the FAB prior to device shipment. Under operational constraints, LBIST is executed in step (1) during the vehicle's key-on and key-off events. Maintaining a fixed LFSR seed and MISR configuration ensures test consistency and prevents potential hazards to the DUT while operating in the

Algorithm 7 Pseudo code of the FSWGEN framework.

```
Require: input, SoC's DTS file.
Require: cpus_dir, cpus include directory.
Require: peripheral_dir, peripherals include directory.
 1: cpus ←
 2: peripherals ←
 3: dts ← load_DTS(input)
 4: graph ← create_graph(dts)
 5: for each fdts ∈ scandir(cpus_dir) do
 6:     cdts ← load_DTS(fdts)
 7:     cpus[cdts.compatible] ← cdts
 8: end for
 9: for each fdts ∈ scandir(peripheral_dir) do
10:     pdts ← load_DTS(fdts)
11:     peripherals[pdts.compatible] ← pdts
12: end for
13: for each dcpu ∈ dts.cpus do
14:     if dcpu.compatible ∉ cpus then
15:         break
16:     end if
17:     cpu ← cpus[dcpu.compatible]
18:     arch ← load_arch(cpu)
19:     reach ← graph.get_reach(cpu)
20:     for each dperipheral ∈ reach do
21:         if dperipheral.compatible ∉ peripherals then
22:             break
23:         end if
24:         peripheral ← peripherals[dperipheral.compatible]
25:         actions ← peripheral. load_actions(include, arch)
26:         for each register ∈ peripheral.registers do
27:             value ← 0
28:             for each field ∈ register.fields do
29:                 i ← (∼ field.rst_value) ≪ field.bit_pos
30:                 value ← value | i
31:             end for
32:             if register.access ∈ actions then
33:                 actions[register.access](arch, register, value)
34:             else
35:                 arch[register.access](register, value)
36:             end if
37:         end for
38:     end for
39: end for
```

field. This initial phase produces a final signature as output. If the obtained signature matches the expected reference one, the vehicle is considered safe, and the process continues on next events. Conversely, a mismatch indicates a failure within the UUT, rendering the vehicle unsafe and necessitating further investigation.

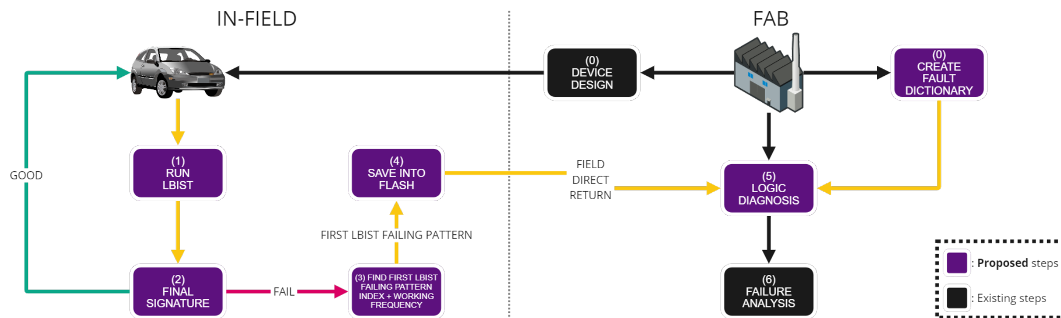


Fig. 3.9 Flow of the proposed approach [3]

The signature captured in step (2) provides only limited diagnostic insight, as it does not indicate which pattern caused the failure. Identifying the index of the first failing pattern, however, enhances the diagnostic value of the final signature drastically.

In this matter, a dichotomic search algorithm (step 3) is employed to determine the index of the first failing pattern. This step also extracts the operating frequency at which the tests succeed. Achieving this requires precise control over both the number of applied patterns and access to the resulting final signature.

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

The faults associated with each test pattern can be determined through fault simulation. Based on these results, candidate faults for a failing device are identified by comparing fault coverage sets. Specifically, the first failing pattern is expected to detect faults not covered by previous patterns; therefore, faults already detected in earlier steps can be removed, isolating only the unique fault coverage of the first failing pattern.

The overall methodology is thus divided into two main 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 test frequency is gradually reduced in subsequent runs to determine the threshold frequency that produces a correct result. The index of the first failing LBIST pattern is

then identified using a dichotomic search algorithm (as detailed in algorithm 8) and stored in flash memory together with the corresponding signature.

B. **Logic diagnosis of field returns**: when a device is returned from the field, the stored information is utilized to support failure diagnosis. Fault dictionaries enable the identification of the failing fault class, while the index of the first failing pattern ensures that earlier undetected faults do not compromise the diagnostic outcome.

3.0.13 In-field Data Collection through LBIST

This section describes the In-Field data collection process as presented in [3]. It introduces the application of a dichotomic search algorithm to identify the index of the first failing LBIST pattern, outlines the dynamic computation of the golden signature, and concludes with an analysis of the storage requirements necessary for implementing the proposed In-Field methodology.

First Failing LBIST Signature Collection

When an LBIST signature deviates from the expected (golden) value, determining the index of the first failing pattern is not straightforward. Since the LBIST signature is computed incrementally, each new signature depends also on data from all preceding patterns. As a result, any corrupted value introduced by a fault propagates through to the final signature, which becomes available only at the end of the test sequence, as illustrated in fig. 3.10. Therefore, the tests must be executed sequentially until the first failing pattern is identified.

To address this challenge, the proposed approach employs a well-known algorithm in computer science: the dichotomic search. The novelty of this work lies in applying the dichotomic search to identify the index of the first failing LBIST pattern in a faulty device. This strategy significantly reduces the number of required test executions, achieving a logarithmic scaling with respect to the total number of patterns. The pseudo-code of the dichotomic search algorithm is reported in algorithm 8:

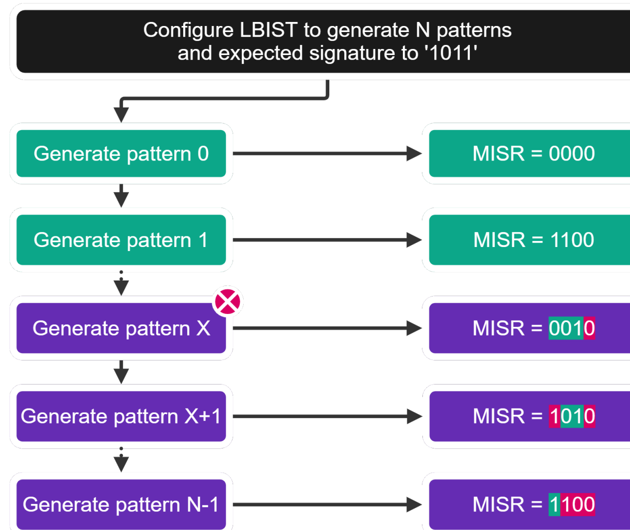


Fig. 3.10 Generation of N LBIST patterns, where the X_{th} propagates an erroneous values till self-test end in the MISR's signature [3].

Algorithm 8 Pseudo-code of dichotomic search [3].

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

The LBIST signature satisfies all the prerequisites for applying the dichotomic search algorithm presented in algorithm 8, rendering it well-suited for this purpose. These prerequisites include:

- The signature is computed incrementally, with each new value depending on the preceding ones, ensuring that any deviation propagates throughout the entire test sequence.
- The cardinality of the test corresponds to the number of generated patterns, which can be configured through the LBIST controller.
- A comparison function is available to evaluate the expected (golden) signature against the obtained one, producing a boolean outcome.

The process flow, illustrated in fig. 3.11, is designed to exploit the step-wise programmability of the LBIST. To improve the detection of TRN faults, each LBIST execution is initially carried out at the maximum operating frequency.

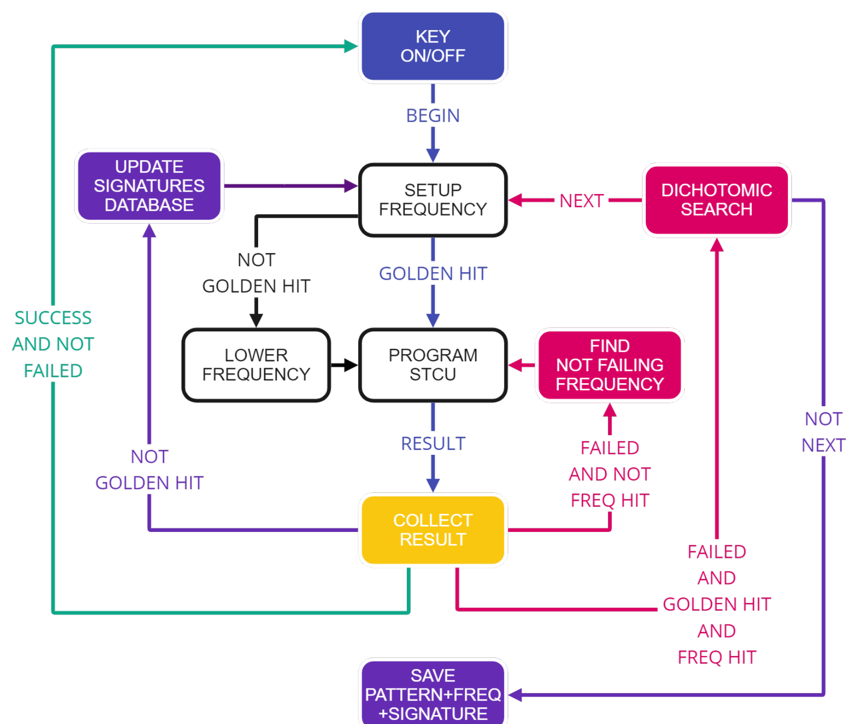


Fig. 3.11 FSM of the logic implemented of the proposed work [3].

The process proceeds as follows:

1. The procedure begins with executing LBIST from the firmware using the maximum number of patterns, $\#total$, at the highest operating frequency.

2. If the resulting signature matches the expected (golden) one, all patterns generated between 0 and $\#total$ are deemed successful (indicated in green). In this case, the device is confirmed to operate correctly, as illustrated in fig. 3.12, and the testing process terminates.

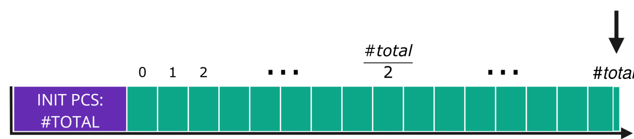


Fig. 3.12 If the signature is good, test ends immediately [3].

3. If a signature mismatch occurs, the proposed method first identifies the frequency at which the tests pass by progressively decreasing the operating frequency in steps of size Δ. The value of Δ depends on the architectural constraints of the register range and corresponds to the smallest frequency decrement at which device anomalies can be observed.

   Once the appropriate frequency is determined, the dichotomic algorithm is initiated by configuring the LBIST to execute half of the total number of patterns, $\frac{\#total\ patterns}{2}$. If the resulting signature remains incorrect, all signatures corresponding to pattern counts between $\frac{\#total\ patterns}{2}$ and $\#total\ patterns$ can be excluded, as the fault must have occurred at or before $\frac{\#total\ patterns}{2}$. Consequently, these signatures are marked in red without re-executing the LBIST, since the error is already detected by an earlier pattern.

4. The process continues iteratively by applying the logic described in algorithm 8. The procedure terminates when no further divisions are possible, leaving a single remaining pattern count, which corresponds to the first failing pattern. This final condition is illustrated in fig. 3.13.

The index of the first failing pattern, denoted as p_{ff}, obtained through the dichotomic search algorithm, carries essential diagnostic information. It enables a refined logical diagnosis by leveraging the coverage data from previously successful patterns.

Considering p_{ff} as the first failing pattern, the set of patterns within the range $[p_0, p_{ff})$ does not cover the observed fault, whereas p_{ff} must necessarily detect it.

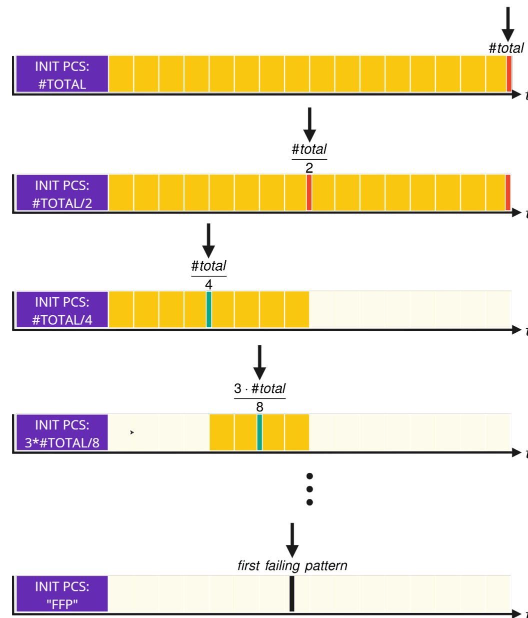


Fig. 3.13 When the resulting signature is wrong, the dichotomic search algorithm is executed, which stops when two dichotomies cannot be distinguished [3].

Golden signatures are retrieved from a database stored in FLASH memory for comparison during testing. If a required signature is not available in the database, a procedure is executed to extract and store it. The process of obtaining a golden signature and the organization of the FLASH memory are detailed in the following sections.

Golden Signature Computation

Golden signatures play a key role in LBIST for erroneous behavior during signature comparison, as discussed in section 2.4. The tests, illustrated in fig. 3.11, are executed at high frequency to effectively detect TRN faults. However, collecting golden signatures at high frequencies increases the likelihood of corrupting the final signature. When a golden signature is not already available in FLASH memory, it can be safely generated and stored by following the steps below:

- Suspend the ongoing test.
- Reduce the operating frequency to a safe LBIST execution range.
- Collect the golden signature and store it in FLASH upon self-test completion.

- Resume the initially suspended test.

The purpose of frequency reduction is to minimize the probability of capturing transient or frequency-dependent faults that could compromise the integrity of the stored signature and subsequent comparisons.

Although this study primarily targets TRN faults, the proposed dichotomic search methodology can also be applied to LBIST signatures for diagnosing Stuck-At (SA) faults. In such cases, however, a precompiled signature database should be used instead of dynamically extracting golden signatures from In-Field devices.

It is important to note that, based on the bathtub curve [83], which models the failure rate of devices throughout their production and operational lifetime, and considering Burn-In (BI) testing [84] for detecting early-life defects, it is reasonable to assume that failures initially manifest as TRN faults before evolving into SA faults. Consequently, the primary goal of this study is the early detection of faults, with an emphasis on maximizing TRN fault coverage.

Data Storage

The structure of the NVM required for the proposed approach, illustrated in fig. 3.14, is organized into three distinct regions:

1. The *frequency region*, which stores the operating frequency at which the tests pass.
2. The *golden region*, dedicated to the storage of golden signatures.
3. The *failure region*, which contains information about the first failing pattern identified through the dichotomic search method.

The second and third regions share a common internal structure composed of the following elements:

- The index representing the number of applied patterns during the test.
- The resulting LBIST signature.

The required memory footprint depends on the considered fault model.

For TRN fault modeling during device shipment, the memory structure is initially empty and progressively populated during normal device operation. At first, only the valid signature occupies the first entry in the *golden region*, since no failures have been detected. If LBIST identifies a malfunction, the *frequency region* records the safe operating frequency at which the tests pass. The *golden region* is then populated using the dichotomic search algorithm (see algorithm 8), while the *failure region* stores the index of the first failing pattern. The number of entries required for each region is determined by the number of possible divisions of the test sequence and is computed as $\#entries = log_2(\#total\ patterns) + 1$.

For SA fault modeling, the *golden region* must be precomputed, as no In-Field computation of golden signatures is performed. In this case, the total number of entries in the region is given by $\#entries = \#total\ patterns + 1$.

Furthermore, for SA fault modeling, the *frequency region* and the indices of the golden signatures can be omitted, since frequency adjustments are irrelevant and indices can be derived as offsets from a base address within the FLASH memory.

Regardless of the chosen fault model, the described memory structure must be replicated for each LBIST partition within the device to ensure complete and reliable operation.

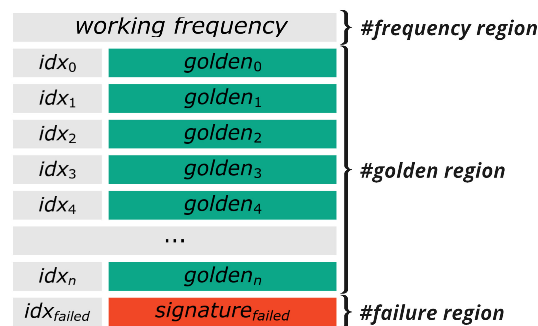


Fig. 3.14 Memory layout required [3].

3.0.14 Logic Diagnosis of Field Return Devices

The growing number of devices returned from the field poses a significant challenge for manufacturing companies. As modern SoCs continue to increase in complexity,

particularly within safety-critical domains, identifying the root cause of failures has become progressively more difficult.

To mitigate this challenge, the proposed In-Field data collection mechanism improves the diagnostic capabilities available to manufacturers by recording and storing both the index and corresponding signature of the first failing pattern during key-on and key-off test events. This information is essential because the operation of LBIST inherently causes corruption in data derived from patterns that fail after the initial one, as MISR signatures are computed cumulatively from preceding ones. Consequently, all signatures obtained following the first failure lose diagnostic value and may even obscure accurate fault interpretation (see fig. 3.15).

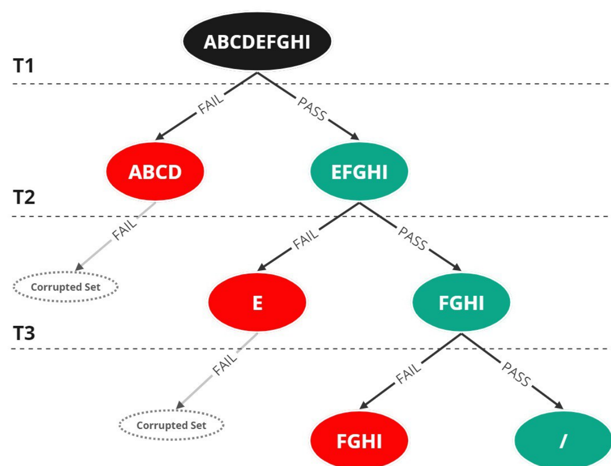


Fig. 3.15 Tree-based fault dictionary for LBIST with an example of three test patterns [3].

As illustrated in fig. 3.15, the fault dictionary generated for diagnosing failures using the proposed methodology is inherently incomplete. This occurs because, for each applied pattern, the analysis focuses exclusively on the first two *pass/fail* outcomes, continuing the evaluation only through *pass* nodes. Consequently, each *fail* node becomes a terminal point, as it is not possible to determine subsequent outcomes due to the inherent limitations of the MISR, which accumulates data across patterns.

The overall diagnostic process is depicted in fig. 3.16. It consists of two main phases:

- **Fault Dictionary Calculation**: a tree-based fault dictionary is generated through fault simulations, capturing the unique fault coverage associated with

each pattern produced by the LFSR. This information is stored for future use during field-return analysis.

- **Logic Diagnosis of In-Field Returns**: when a device is returned due to a detected malfunction, the precomputed fault dictionary is employed to identify potential fault candidates, using the data stored in the device's FLASH memory through the In-Field data collection mechanism. By locating the index of the first failing pattern, the corresponding candidate fault set can be retrieved directly from the tree-based fault dictionary.

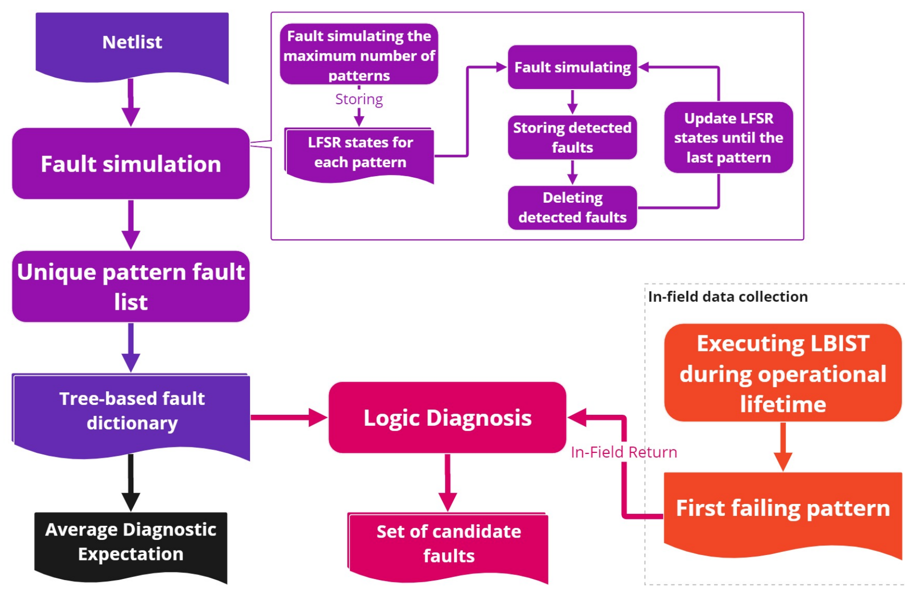


Fig. 3.16 Overview on the Logic Diagnosis steps [3].

To compute the fault dictionary for the LBIST, a preliminary fault simulation campaign must be performed to determine the unique coverage of each pattern generated by the LFSR. This information is later used during the device's In-Field data collection.

The algorithmic procedure to derive the unique coverage can be summarized as follows:

1. Perform a fault simulation using the maximum number of applicable patterns while tracking the state of the LFSR for each applied pattern.

2. Use the recorded LFSR states to apply the corresponding patterns on silicon.

3. For each pattern:

   (a) Initialize the LFSR with the corresponding seed.

   (b) Execute fault simulation based on the defined fault list.

   (c) Update the fault list by removing detected faults.

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

This process enables the extraction of all relevant diagnostic information from failures observed during In-Field data collection. By varying the seed of the LFSR, it becomes possible to simulate each pattern independently and extract its unique fault coverage. Once these unique coverages have been identified, the tree-based fault dictionary for LBIST can be constructed following the algorithm described in algorithm 9. The proposed methodology employs a tree-based structure [69, 85] to build a fault dictionary that encompasses all potential patterns the LBIST can generate.

Figure 2.5 illustrates an example implementation in a case where the maximum number of LBIST-applied patterns 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 representing a distinct fault. For each applied pattern, only two new nodes are created, differing from traditional non-LBIST diagnostic approaches described in section 2.5 and illustrated in fig. 2.5. In the LBIST-specific tree-based fault dictionary, each *pass* node branches into two nodes (*pass* and *fail*) for the subsequent pattern, whereas each *fail* node becomes a terminal leaf representing an equivalence class in the fault dictionary.

This structural limitation arises because information from the MISR becomes useless after the first failing pattern corrupts its internal state. Consequently, it becomes not possible to determine which additional test patterns would have also failed. Therefore, data captured from the very first failing pattern during In-Field operation is crucial for post-return diagnosis. Identification of candidate faults using the precomputed fault dictionary relies exclusively on this information, which serves as the only reliable reference for effective diagnostic reasoning.

Algorithm 9 Tree-based fault dictionary for LBIST [3].

Require: $fault_universe$, set containing all faults.
Require: $unique_coverages$, list of sets containing the unique covered faults for each pattern.

1: $prev \leftarrow fault_universe$
2: $fault_dictionary \leftarrow [\emptyset]$
3: $i \leftarrow 0$
4: $N \leftarrow$ length($unique_coverages$)
5: **while** $i < N$ **do** ▷ for each pattern
6: $pass_node \leftarrow prev - unique_coverages[i]$
7: $fail_node \leftarrow prev \bigcap unique_coverages[i]$
8: $fault_dictionary[i].pass \leftarrow pass_node$
9: $fault_dictionary[i].fail \leftarrow fail_node$
10: $prev \leftarrow pass_node$
11: $i \leftarrow i+1$
12: **end while**

Referring to the tree-based fault dictionary shown in fig. 3.15, when a device is returned to the manufacturer and the first failing pattern is identified as the third one ($T3$), the corresponding set of potential faults is the *fail* node associated with that pattern, which in this case is F, G, H, I. Once the fault dictionary has been established, the most direct method to complete the logic diagnosis process is to access the *fail* node at the index corresponding to the first observed failing pattern, as detailed in algorithm 10.

Algorithm 10 Logic diagnosis exploiting both tree-based fault dictionary and In-Field collected information [3].

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 Setup and Results

This chapter details the experimental validation performed to assess the methodologies presented in the previous chapter. It focuses exclusively on the setup, and observed results obtained.

In detail each section describes the experimental setup used to assess the respective methodology, including configuration parameters, measurement conditions, and quantitative observations. All results are reported as observed data, expressed through representative metrics such as coverage, signature correlation, and diagnostic granularity. Where appropriate, figure and table placeholders are included to denote visual summaries of key outcomes.

4.1 Case of Study

The industrial case study adopted to validate the proposals presented in this thesis is a large Automotive SoC developed by STMicroelectronics belonging to the SPC58 family, characterized by the following specifications:

- 40 nm technology.
- Approximately 20 million logic gates.
- Around 700 thousand flip-flops.
- Multi-core architecture.

- PowerPC ISA.
- ASIL-D compliance.
- Many communication peripherals, including CAN.
- On-chip co-processor for AI computations with 32 KB of dedicated RAM.
- 7 LBIST islands.
- 92 MBIST.

The SoC features a multi-core architecture comprising three 32-bit cores implementing the PowerPC Variable-Length Encoding (VLE) instruction set. It integrates 6,MB of Flash memory and 128,KB of general-purpose SRAM. Several peripheral bridges provide access to a variety of on-chip and off-chip peripherals. its full architecture is detailed in fig. 4.1.

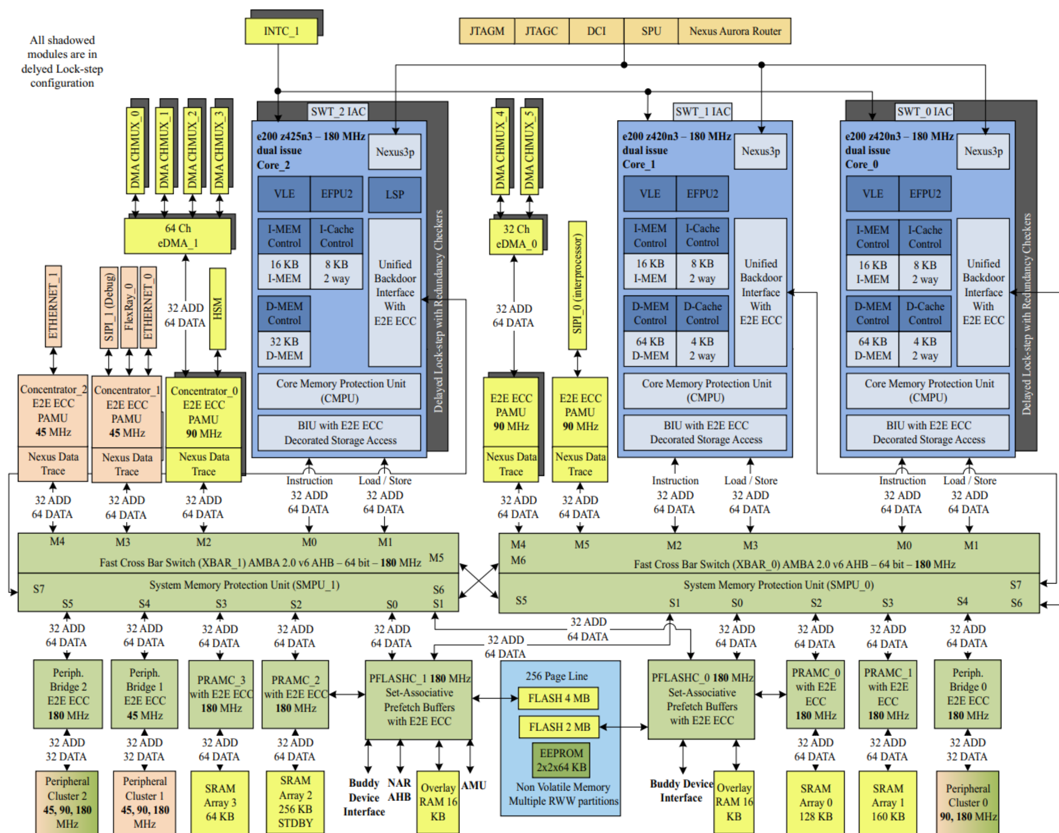


Fig. 4.1 Complete architecture of the Case of Study.

The complete fault list for the case study consists of approximately 80 million Stuck-At Faults (SAFs) and Transition Delay Faults (TDFs). Table 4.1 summarizes the seven LBIST partitions of the device, reporting for each the number of scan cells, scan chains, and total faults.

Table 4.1 Faults summary report for different LBIST partitions in the industrial case of study [3].

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

In order to develop effective SLT workloads and test strategies, information from the Register-Transfer-Level (RTL), gate-level netlist, and physical layout of the manufactured automotive SoCs are combined, as illustrated in fig. 4.2.

In fig. 4.2, the logic simulation is performed using the *Incisive Suite* from *Cadence*. Functional fault simulations are executed with *Z01X* from *Synopsys*. Toggle analysis is carried out using a custom toolchain developed at Politecnico di Torino [42].

The debug environment for developing and validating the functional test programs, including stress tests and SLT workloads, was implemented on an evaluation board for the DUT. A *Trace 32 PowerDebug* unit from *Lauterbach* was used as the main debugging interface. All functional test programs were compiled using a GNU-based toolchain specifically designed for the PowerPC VLE instruction set.

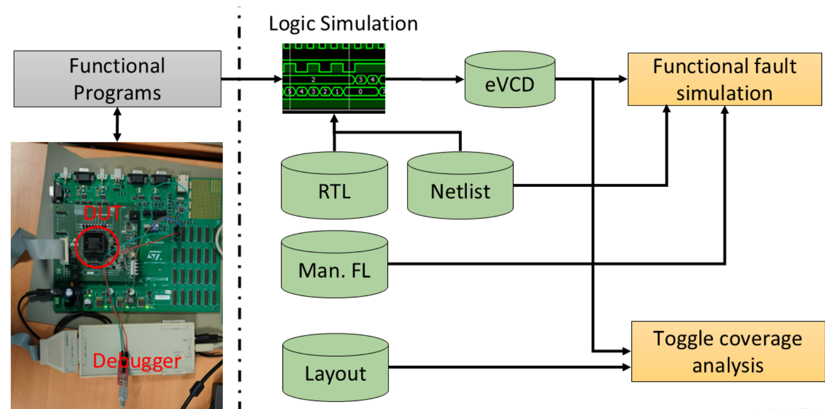


Fig. 4.2 Proposed experimental setup.

4.1.1 FPGA-based Tester

This subsection introduces the FPGA-based tester, which serves as the experimental platform for several of the obtained results. The tester is a flexible, research-oriented setup designed to enhance the testing capabilities of automotive SoCs [86].

The proposed tester integrates both structural tests and functional tests into a single unified platform. This integration enables efficient screening of faulty devices, reduces dependence on costly industrial ATE, and provides a modular architecture adaptable to diverse testing requirements.

The tester is synthesized for the AMD-Xilinx Zynq Ultrascale+ MPSoC ZCU104 Evaluation Kit [87], which includes a ZU7EV device featuring a quad-core ARM Cortex-A53 application processor, a dual-core Cortex-R5 real-time processor, and a Mali-400 MP2 graphics processing unit. The FPGA fabric comprises 504k system logic cells and 38 Mb of memory. In fig. 4.3, the tester is connected to the Test Access Port (TAP) controller of the DUT for debugging purposes.

On top of the MPSoC hardware, a Linux-based software stack provides networking capabilities and a custom control framework. This software layer communicates with the hardware modules, enabling the tester to perform the following tasks:

- Access the on-chip TAP controller to retrieve basic device information.
- Extract register and memory contents.
- Set on-chip breakpoints.

- Modify registers and memories.
- Perform flash erase, programming, and verification.
- Enter test modes for structural testing across single or multiple scan chains.

All these functionalities are device-specific and implemented per DUT, whereas the generic Hardware Abstraction Layers (HALs) used to interface with the debugger and hardware modules remain unchanged unless specific customization is required. The use of the Python-based PYNQ framework [88] allows for straightforward scripting, automation, and integration of test operations.

As an example, consider a scenario where the CAN module of the DUT must be functionally tested. During the SLT phase, peripherals, being critical components in safety-oriented systems, must undergo functional validation. In this case, a generic communication peripheral can be implemented within the programmable logic to communicate with the DUT. If protocol-level fault injection is required, a custom module can be integrated into the tester to introduce controlled errors into communication frames, as shown in fig. 4.3.

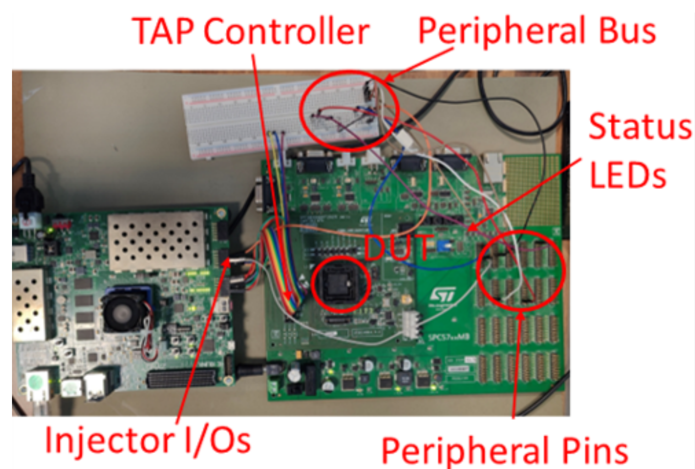


Fig. 4.3 Experimental setup with Xilinx ZCU104 evaluation board and CAN peripheral bus.

The FPGA-based tester thus serves as an enabling platform for the experimental validation of the proposed methodologies. Although equivalent results could be obtained using a commercial debugger, the FPGA-based solution was selected for its capability to support multiple experimental setups in parallel. This design also supports parallel remote access to different setups, providing a scalable, cost-efficient, and practical solution for laboratory-based research environments.

4.1.2 Burn-In Function Stress Suite for AI HW Accelerators

The presented experimental analysis focuses on a case study featuring an AI HW Accelerator [89] designed to execute floating-point Multiply and Accumulate (MAC) operations and exponential calculations. The architecture of the AI HW accelerator employs a time-multiplexed processing scheme, where a single ALU supports two independent calculation channels. Figure 4.4.a highlights the accelerator in green against the chip's surface plot, contrasting with the "sea of gates" in gray and memory cores in white. This module fetures approximately $80K$ gates and a dedicated 32 KB RAM for operand storage, which serves as the basis for computing MTMs.

The experimental setup is shown in fig. 4.4.b. The setup includes the physical chip, a single-site FPGA-based debugger [90] responsible for flashing and executing the firmware that applies functional patterns, and a current sensor. The sensor communicates with the debugger to transmit measurements to a host PC (connected via Ethernet, not pictured). The power monitoring hardware consists of a low-cost discrete module featuring an INA219 power monitor IC coupled with a 100 mΩ shunt resistor. This resistor is placed in series with the DUT power supply, enabling the calculation of current and power via voltage drop. The INA219 integrates a Programmable Gain Amplifier (PGA), which is configured for a full-scale range of $\pm 80\,\mathrm{mV}$. With a 12 bit ADC, the system achieves a voltage resolution of $\frac{80\,\mathrm{mV}}{2^{12}} = 0.0195\,\mathrm{mV}$. Given the 100 mΩ shunt resistance, this translates to a theoretical current resolution of 0.2 mA. To enhance stability and mitigate noise, the module averages 128 samples per reading over approximately 68 ms, resulting in a sampling rate of $\approx 2\,\mathrm{Ksamples/s}$.

The host PC executes the selection algorithm using the collected current data and the Memory Toggle Metrics.

For this specific setup, the gate-level netlist is available at both the gate and layout levels. Although the methodology proposed in this work operates independently of netlist data, this information is utilized here to validate the results through layout analysis and gate-level simulations of the identified stress patterns.

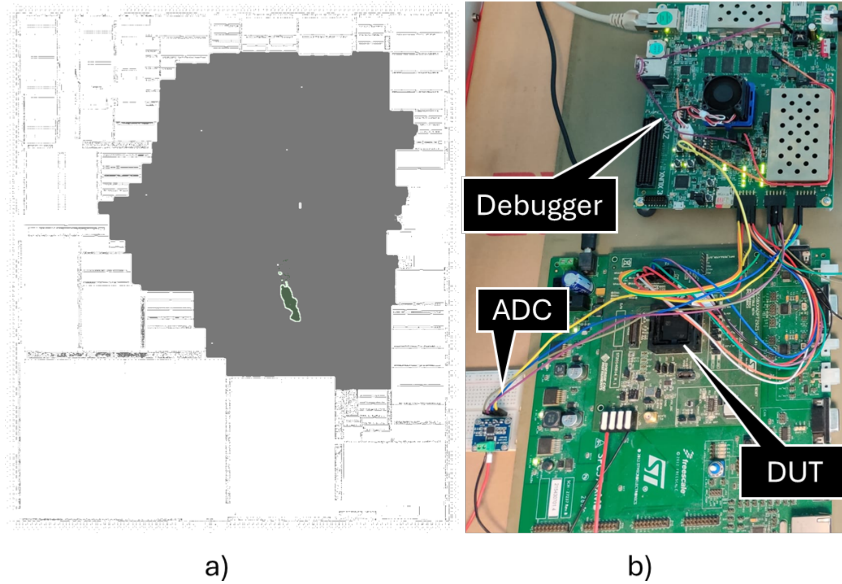


Fig. 4.4 (a) Layout of the case study with AI HW Accelerator gates highlighted in green and (b) Measurement setup [2].

4.1.3 Experiment Statistical Confidence and Parameters

Regarding measurement uncertainty, Type B uncertainty (related to the intrinsic accuracy of the sensor) is estimated at 2˘3% based on manufacturer specifications and calibration data. Type A uncertainty is derived from the standard deviation, σ_K, of current measurements during steady-state operation. Each reported measurement is an average of $H = 140$ steady-state samples. Experimental data indicates that σ_K consistently remains below 0.5 mA after 120 s of execution, resulting in a Type A standard uncertainty of 0.083%.

The combined uncertainty (Type A and Type B) falls within the 3% to 4% range, ensuring statistically robust results with repeatability sufficient to compare currents within tens of micro-amperes. Thermal data from the on-chip sensor, illustrated in fig. 4.5, confirms temperature stabilization around the 120 s mark. Since a single execution run (t_{run}) takes up to 150us, the stabilization is made of $K = \frac{120\,\mathrm{s}}{t_{run}} = 800,000$ samples. A transition phase of 10 s is allocated, corresponding to approximately $N - K = \frac{10\,\mathrm{s}}{t_{run}} \approx 80,000$ runs.

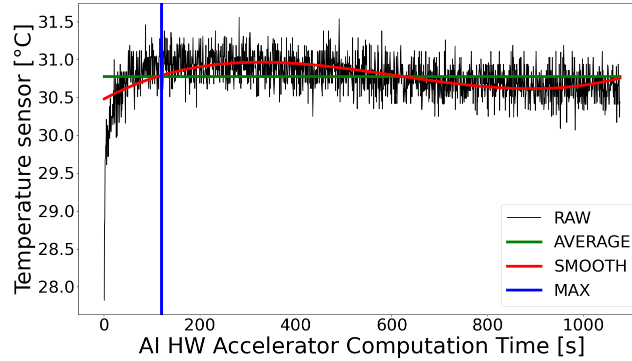


Fig. 4.5 Embedded thermometer's temperature profile [2].

4.1.4 Functional Stress Patterns Evaluation

The initial phase of the experiment involves assembling a population of functional stress patterns. In this instance, a pool of 700 functional programs is generated, each applying patterns corresponding to the content of the AI HW Accelerator's dedicated RAM. These patterns are created by randomly distributing logic '1' values according to specific percentage thresholds. Specifically, M patterns are synthesized by populating $N\%$ of the memory with logic '1's.

Table 4.2 Population details of the functional patterns [2].

Class	Amount of randomly distributed values '1'	Number of individuals	HIGHEST Current consumption [mA]	AVERAGE Current consumption [mA]
A	50%	200	50.90	46.63
B	40%	200	50.02	45.72
C	30%	100	49.96	43.67
D	20%	100	47.94	40.74
E	10%	100	42.19	37.52
F	0%-100%	5	37.00	34.25
G	provoking errors	20	9.99	7.36
B'	60%	10	48.21	45.57
C'	70%	2	45.56	44.45

Table 4.2 provides a detailed breakdown of the initial population, categorized by the density of random logic '1's or the induction of error conditions.

Patterns in classes A through F avoid triggering exceptions such as overflows or underflows. The parameter N is kept below 50% because higher densities of logical '1's increase the magnitude of operands processed by the ALU, hence raising the probability of arithmetic exceptions. Furthermore, as depicted in fig. 3.2, power consumption is expected to follow a Gaussian distribution relative to the percentage of '1's, peaking near 50%. Therefore, targeting approximately 50% allows for representative power profiling while minimizing overflow risks.

While power measurements were feasible at 60% and 70% densities, program synthesis at 70% proved inefficient, requiring over six hours to yield a few valid individuals. No valid programs were obtained at 80% within the same timeframe. Additionally, despite the increased synthesis effort, the power profiles at 60% and 70% were comparable to those at 30% and 40%.

Table 4.2 also lists the peak and average current consumption for each class. The measurement flow requires approximately 6 minutes per pattern: 3 minutes for firmware flashing, 2 minutes for warm-up, 10 seconds for measurement, and the remaining time for data transmission and metric computation. The automated evaluation of the full 700-pattern pool spanned approximately 4 days.

Table 4.3 presents the outcomes of the selection algorithm. The process identified 30 valid functional stress patterns (i.e., those not generating exceptions) for the final suite. The table lists the class of each selected pattern, its current consumption, and the incremental improvements in MTC and MTA (average and variance). Patterns that induce errors are also valuable, as such they are integrated into the suite using the same algorithm. Two patterns triggering overflow and underflow conditions are sufficient to maximize the MTMs.

To further validate the methodology, the identified patterns are evaluated via gate-level simulation to extract toggle metrics, which are then compared against alternative approaches. Table 4.4 compares the results with random selection, a netlist-based generation method from [91], and a scan-stress suite [59]. The table details the number of patterns, their typology (functional vs. structural), and for random selection reported TC and TA values are averaging the results for each of the ten pattern set per line.

The proposed method achieves a Toggle Coverage (TC) of 94.73%, which is comparable to the 94.94% average obtained by random selection with 40 patterns.

Table 4.3 Functional stress pattern selection results [2].

Ranked Functional Patterns	Class	Current Consumption [mA]	Incremental		
			MTC [%]	MTA Average	MTA Variance
1	A	50.9	-	-	-
2	A	50.84	25.0183	0.50	0.50
3	A	50.63	50.0429	1.00	0.71
4	A	50.56	68.7063	1.50	0.86
5	A	50.2	81.2610	2.00	1.00
6	A	50.16	89.0542	2.50	1.12
7	B	50.02	93.7459	3.00	1.22
8	C	49.96	96.1398	3.44	1.32
9	B	49.95	97.5783	3.87	1.42
10	B	49.94	98.4885	4.34	1.52
11	B	49.9	99.0370	4.81	1.61
12	A	49.71	99.4692	5.31	1.69
13	B	49.67	99.6708	5.81	1.77
14	A	49.52	99.8214	6.31	1.84
15	A	49.49	99.9073	6.81	1.91
16	B	49.48	99.9435	7.31	1.97
17	B	49.38	99.9665	7.77	2.03
18	A	49.36	99.9825	8.27	2.09
19	B	49.29	99.9896	8.77	2.16
20	A	49.26	99.9949	9.27	2.22
21	B	49.23	99.9968	9.77	2.28
22	B	49.22	99.9978	10.23	2.33
23	A	49.19	99.9978	10.51	2.17
24	B	49.11	99.9980	10.99	2.22
25	C	49.04	99.9986	11.43	2.28
26	A	49.03	99.9988	11.93	2.34
27	A	49.02	99.9994	12.43	2.39
28	A	49.01	99.9996	12.93	2.44
29	A	48.99	99.9998	13.43	2.49
discarded	A	48.98	-	-	-
discarded	B	48.94	-	-	-
30	B	48.93	**100**	**13.93**	**2.54**
31	G	9.99	100	14.17	2.38
32	G	9.42	100	14.41	2.24

Table 4.4 Approaches results comparison [2].

Number of patterns	Type of patterns	Generation Methodology	TC[%]*	AVG(TA) [T/us]	VAR(MTA)
40	Functional	Random Selection	94.94	3009505	234014
39			94.69	2903340	227795
38			94.69	3015525	222603
37			94.90	2957831	219646
36			94.82	3011621	212613
35			94.79	2940629	205545
34			94.88	2994593	201015
33			94.84	3036523	195981
32			94.62	2939122	193504
32	*Structural*	*Scan-stress [59]*	*95.09*	*1497837*	*829*
32	*Functional*	*Netlist-based [91]*	*89.62*	*1763422*	*1639*
32	**Functional**	**Proposed**	**94.73**	**4545416**	**170932**
31	Functional	Random Selection	94.52	2994104	182261
30			94.63	2931979	176218
29			94.38	3070885	175612
28			94.41	2974546	165449
27			94.64	2857780	158744
26			94.51	2954290	156581
25			94.42	2854391	150557
24			94.44	2884231	136099
23			94.30	2936833	135296
22			94.44	2962143	131748
21			94.16	2988284	125653
20			94.48	2895112	115462
512	Structural	*Scan-stress [59]*	*98.03*	*1508718*	*11581*
32+**32**	Structural + Functional	Scan-stress [59] + **Proposed**	**96.57**	**4455521**	**171629**
512+**32**			**99.05**	**3774564**	**180789**

*for Random Selections it reports the average value over ten pattern sets.

However, the TA values achieved by the proposed selection significantly outperform those of random selection.

A comparison with the netlist-based approach in [91] is also presented. This method employs a genetic algorithm to generate 32 functional patterns, grading them directly via netlist simulations. For a fair comparison, the netlist-based approach was restricted to the same 4-day timeframe used for the electrical grading of the proposed method. Due to simulation complexity, only 200 individuals could be evaluated in this period, resulting in metrics substantially lower than those achieved by the proposed electrical grading approach.

Additionally, scan-based ATPG stress sets comprising 32 and 512 patterns [59] were evaluated. The 32-pattern scan-stress set reaches a TC of 95.09%, slightly exceeding the proposed method, but exhibiting lower TA, but better uniformity (VAR(MTA)). The 512-pattern scan suite achieves the highest single-method TC at 98.02% but imposes significant ATE memory requirements.

The final rows of table 4.4 highlight the benefits of combining approaches. Adding the 32 selected functional patterns to the 32-pattern scan suite raises TC to 96.57%, with a 1.48% contribution solely from the functional patterns. Similarly, combining them with the 512-pattern set achieves a TC of 99.05%. In both scenarios, the functional patterns provide a robust increase in TC while maintaining high TA.

Figures fig. 4.6, fig. 4.7, and fig. 4.8 illustrate the performance relative to random selection.

While the TC of the proposed approach is acceptable, the method demonstrates superior AVG(TA) compared to the mediocre values of random selection. Regarding VAR(MTA), the proposed strategy yields significantly better uniformity than random selection, particularly as the number of patterns increases.

Figure 4.9a and fig. 4.9b display heatmap visualizations of node coverage and activity for (a) the 32-pattern scan-stress suite, (b) random selection with 32 patterns, and (c) the generated functional stress suite.

4.1.5 An Additional Case Study

To further demonstrate the robustness of the methodology, a second case study (DUT2) is introduced involving another automotive SoC from STMicroelectronics.

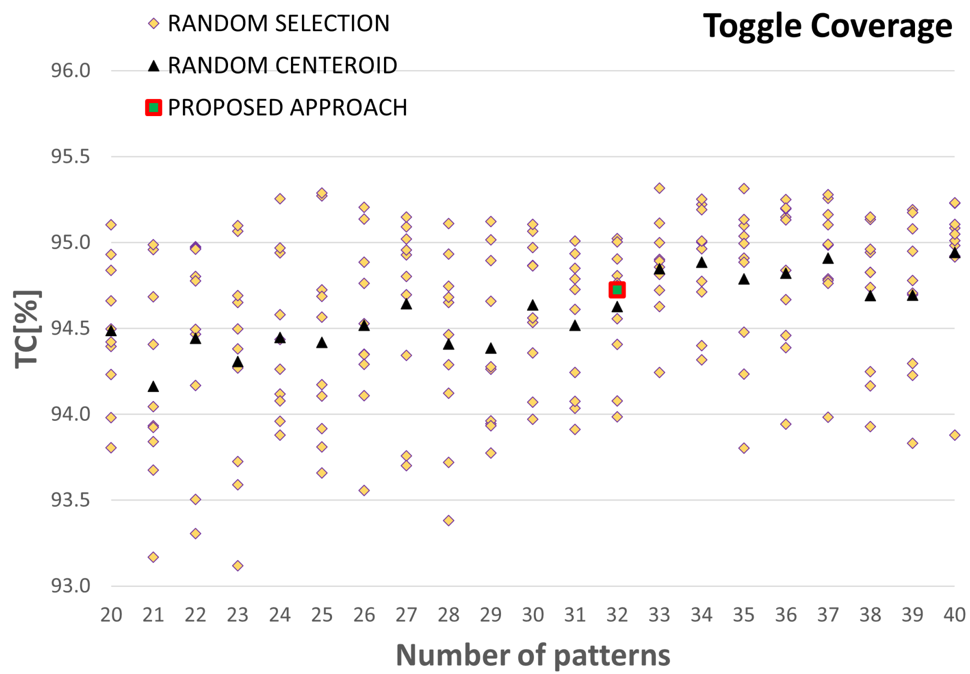


Fig. 4.6 TC comparison [2].

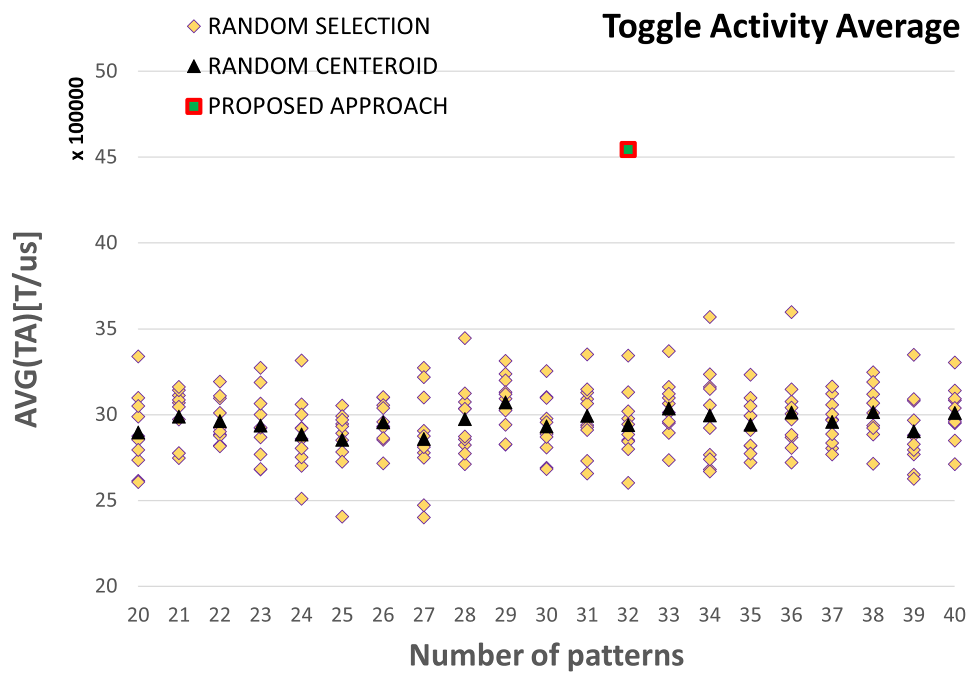


Fig. 4.7 TA comparison [2].

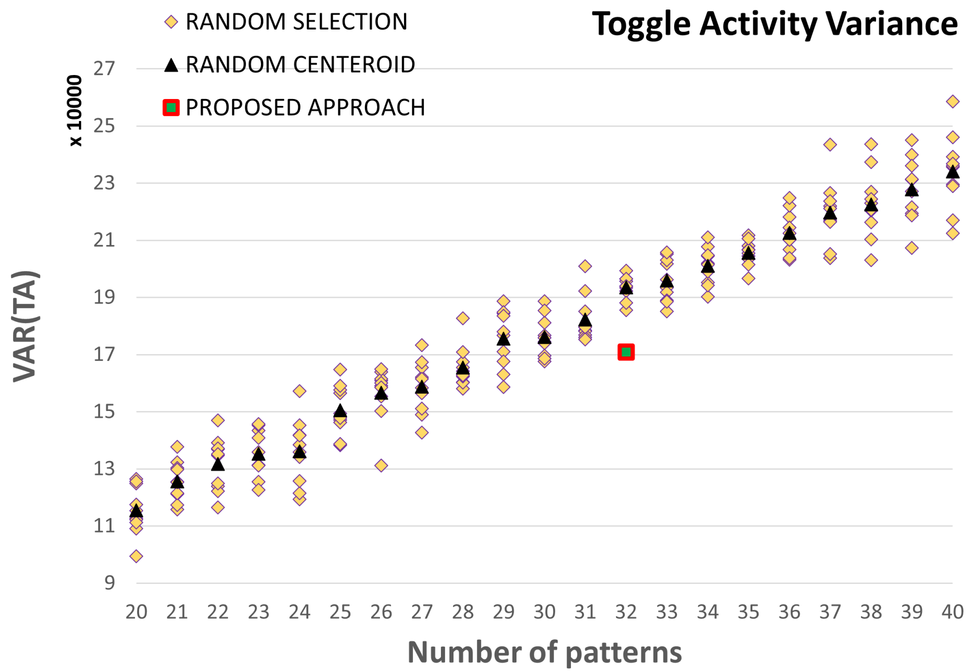
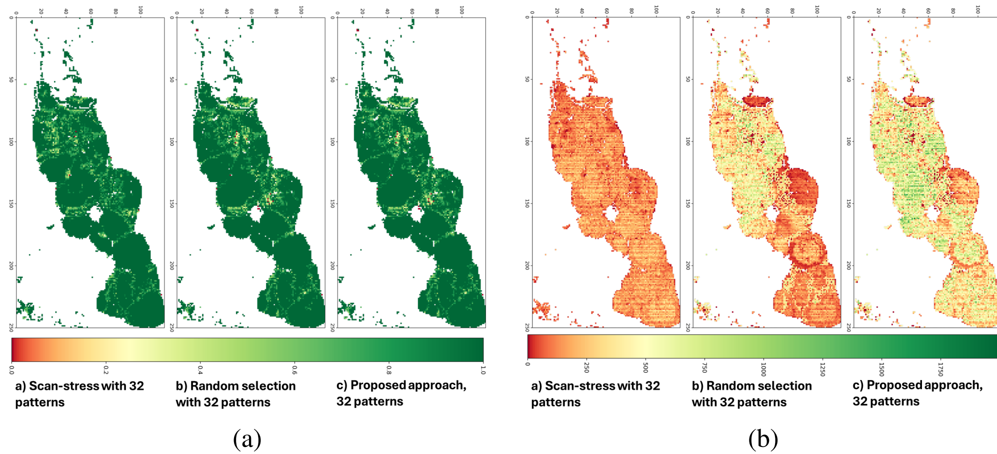


Fig. 4.8 MTA comparison [2].



(a) (b)

Fig. 4.9 Stress of TC (a) TA (b) heatmap comparison [2].

This device, targeting low-power applications, features a more complex architecture with additional CPUs and peripherals but shares the same AI HW accelerator as the first case study. However, the design underwent different synthesis, and DfT optimization processes. Since both devices share the same package form factor, DUT2 utilizes the same evaluation board shown in fig. 4.4.b. The debugger is configured to access the device via its JTAG/TAP controller, highlighting the method's portability across the chip family.

Experimental parameters remain consistent ($H = 140$), with a steady-state standard deviation $\sigma_K = 0.22\,\text{mA}$ after 120 s. The maximum current observed is 26.01 mA, and the relative Type A uncertainty is approximately 0.072%, maintaining high statistical confidence.

Table 4.5 Functional patterns details for DUT2 [2].

Class	Amount of randomly distributed values '1'	Number of individuals	HIGHEST Current consumption [mA]	AVERAGE Current consumption [mA]
A	50%	200	26.01	19.33
B	40%	200	25.00	18.35
C	30%	100	22.55	16.08
D	20%	100	21.00	12.55
E	10%	100	13.07	8.48
F	0%-100%	5	6.03	5.18
G	provoking errors	20	2.02	0.91

Table 4.5 details the initial population for DUT2, categorized by '1' density or error conditions.

The selection results in Table 4.6 indicate the identification of 30 valid functional stress patterns. Error-provoking patterns are included via a similar optimization process.

Table 4.7 compares the proposed method against random selections (averaged over ten sets). The proposed method yields a TC of 96.26%, comparable to the 96.79% average of 32 random patterns, but delivers significantly superior TA performance.

Figures fig. 4.13 and fig. 4.14 present heatmap comparisons of node coverage and activity for (a) the 32-pattern random selection and (c) the proposed functional stress suite.

Table 4.6 Functional stress pattern selection for DUT2 [2].

Ranked Functional Patterns	Class	Current Consumption [mA]	Incremental		
			MTC [%]	MTA Average	MTA Variance
1	A	26.01	-	-	-
2	B	25.00	25.0077	0.50	0.50
3	A	24.06	49.9685	1.00	0.71
4	A	24.01	68.6957	1.50	0.87
5	B	24.00	81.1838	2.00	1.00
6	B	24.00	88.4701	2.46	1.12
7	B	24.00	92.8063	2.93	1.24
8	A	23.96	95.8177	3.43	1.34
9	A	23.68	97.5934	3.93	1.43
10	B	23.26	98.6066	4.43	1.51
11	A	23.22	99.2075	4.92	1.59
12	A	23.00	99.5468	5.42	1.67
13	A	22.98	99.7334	5.92	1.74
14	C	22.55	99.8486	6.42	1.81
15	A	22.01	99.9130	6.92	1.88
16	A	22.00	99.9569	7.42	1.94
17	C	22.00	99.9783	7.92	2.01
18	C	22.00	99.9864	8.31	2.07
19	C	22.00	99.9898	8.69	2.14
20	B	21.99	99.9933	9.13	2.21
21	A	21.97	99.9951	9.63	2.27
22	A	21.95	99.9965	10.13	2.32
23	A	21.90	99.9978	10.63	2.38
24	A	21.83	99.9984	11.13	2.43
25	B	21.48	99.9992	11.63	2.48
discarded	B	21.23	-	-	-
26	C	21.08	99.9994	12.06	2.53
27	B	21.06	99.9994	12.34	2.40
28	B	21.00	99.9996	12.80	2.43
discarded	C	21.00	-	-	-
29	A	21.00	99.9998	13.30	2.48
30	A	21.00	**100**	**13.80**	**2.53**
31	G	1.86	100	14.03	2.39
32	G	1.13	100	14.29	2.27

Table 4.7 Pattern approaches comparison for DUT2 [2].

Number of patterns	Generation Methodology	TC[%]*	AVG(TA) [T/us]	VAR(MTA)
40	Random Selection	97.01	2679626	296403
39		96.95	2634621	289057
38		97.24	2600884	278947
37		96.77	2597179	276297
36		96.94	2567170	263147
35		96.79	2554242	258912
34		96.81	2636314	251348
33		96.88	2593437	241087
32		96.79	2650194	240249
32	**Proposed**	**96.26**	**4228201**	**224234**
31	Random Selection	96.71	2598327	237056
30		96.93	2720859	223581
29		96.17	2638669	223196
28		96.60	2646928	215920
27		96.92	2583732	202084
26		96.70	2650416	195918
25		96.57	2733406	191445
24		96.63	2612034	181865
23		96.27	2656088	173847
22		96.37	2651801	171152
21		96.30	2709068	169113
20		96.01	2631578	157602

*for Random Selections it reports the average value over ten pattern sets.

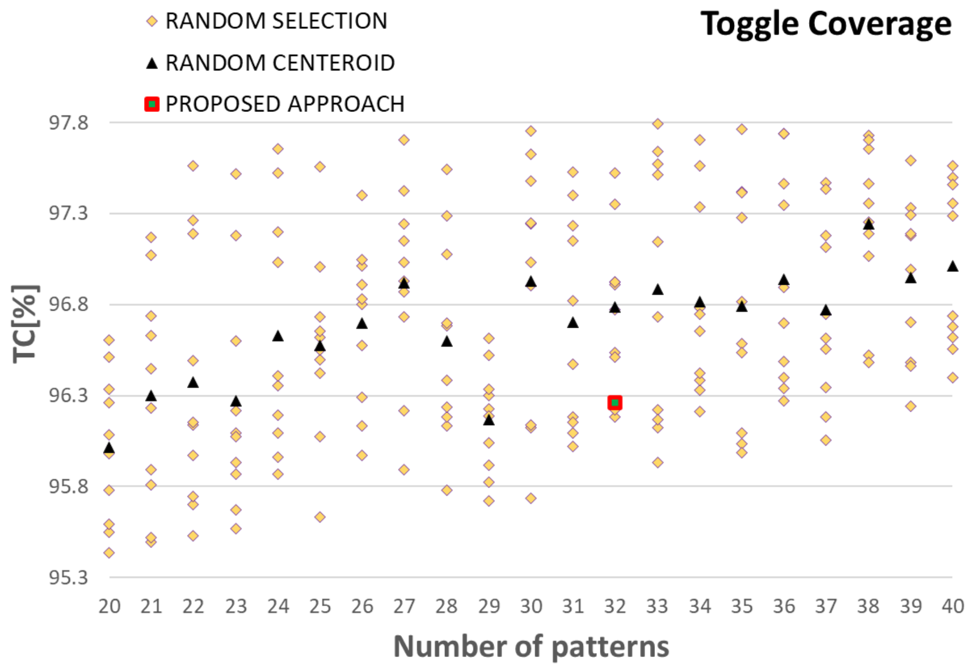


Fig. 4.10 TC comparison for DUT2 [2].

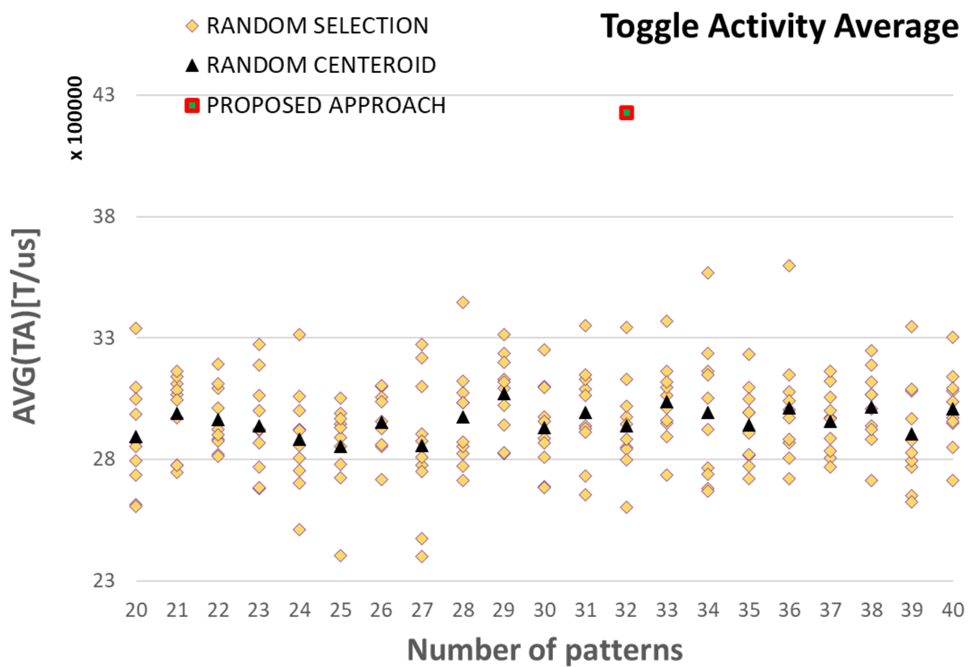


Fig. 4.11 AVG(TA) comparison for DUT2 [2].

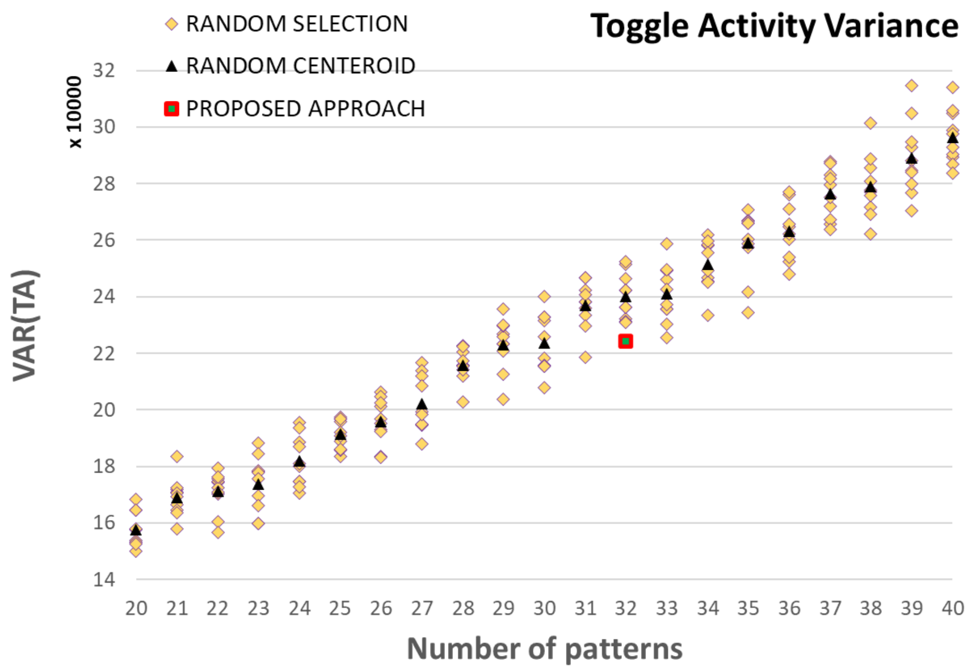


Fig. 4.12 VAR(TA) comparison for DUT2 [2].

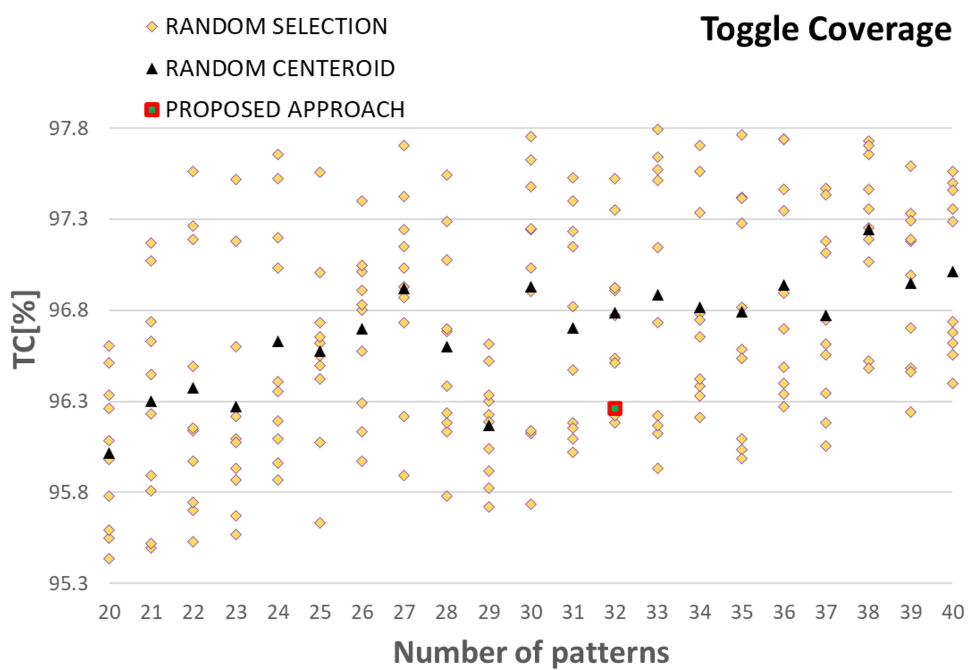


Fig. 4.13 Stress TC heatmap comparison for DUT2 [2].

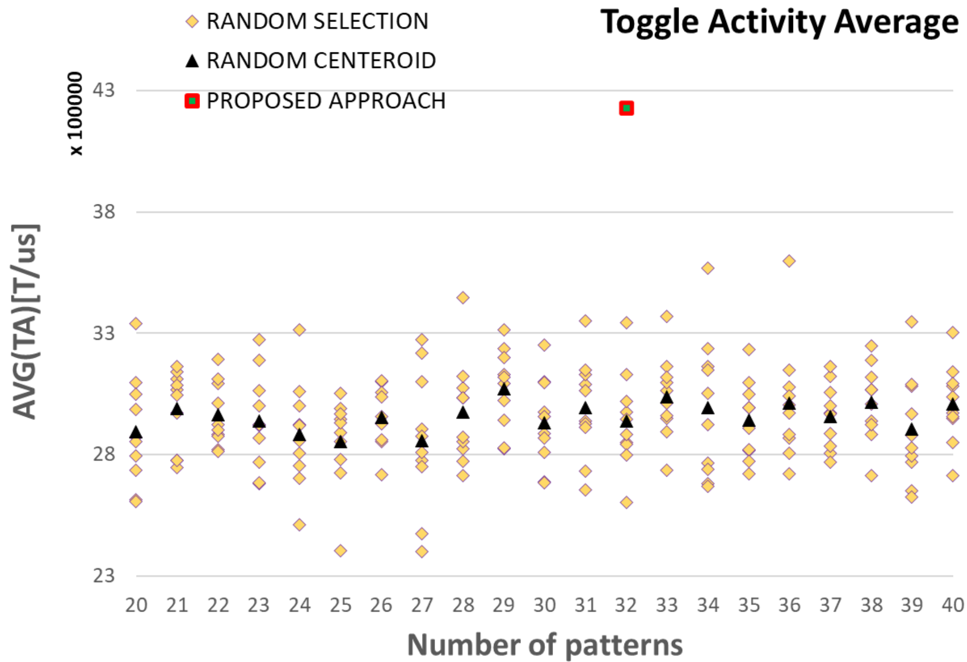


Fig. 4.14 Stress AVG(TA) heatmap comparison for DUT2 [2].

4.1.6 System-Level-Test for Communication Peripherals

The algorithms presented in the previous chapter were implemented for the subsystem of CAN modules in the SoC. Test patterns were generated following the deterministic approach described in [82]. The resulting suite, consisting of 6 functional test programs, is summarized in table 4.8. For each program, the table reports the targeted weakness, execution time, memory footprint, fault simulation grading time, and the approximate development effort required by a single engineer for both program and companion module design, implementation, and verification.

The complete functional SLT suite requires a total of $11,622,737$ clock cycles, corresponding to approximately 145 ms at an operating frequency of 80 MHz (the maximum achievable clock frequency for the CAN peripherals in the DUT). The suite was developed over a total of 136 hours, equivalent to approximately 8.5 working days, by two test engineers.

The companion modules were synthesized and implemented on the debugger described in section 4.1.1. The corresponding laboratory setup is illustrated in fig. 4.2. This configuration enables an external CAN node to be connected to

the DUT, thereby allowing off-chip communication, including full error injection capabilities.

The instantiated companion modules occupy 13,071 Look-up-Tables (LUTs), 6,441 flip-flops, two block RAMs, and three external I/O pins of the programmable logic. The design was synthesized and implemented for a clock frequency of 2 MHz.

To evaluate the quality of the developed SLT functional programs, functional fault simulations [92] were conducted using the commercial fault simulator *Z01X (Synopsys)*. Both SAF and TDF models were considered. Table 4.9 summarizes the resulting SLT fault coverage, reporting the “Single,” “Incremental,” and “Delta” (Δ) coverages, which respectively indicate the individual program coverage, the incremental coverage relative to previous tests, and the coverage increase between consecutive programs.

The structural tests, comprising scan-based, LBIST, and MBIST, achieved a base fault coverage of 97.89% for 435,967 SAFs and 89.38% for 435,966 TDFs. By integrating the proposed SLT, the overall coverage increased to 99.01% for SAFs and 90.89% for TDFs. Figure 4.17a and fig. 4.17b illustrate that the functional SLT programs contribute an additional 1.12% and 1.51% of coverage, respectively.

Scan-based testing achieves the broadest coverage overall, while functional SLT demonstrates superior fault detection in specific regions of the CAN subsystem compared to LBIST. However, LBIST still detects a subset of unique faults not covered by SLT. Since MBIST contributes no unique coverage, it is omitted from the diagrams.

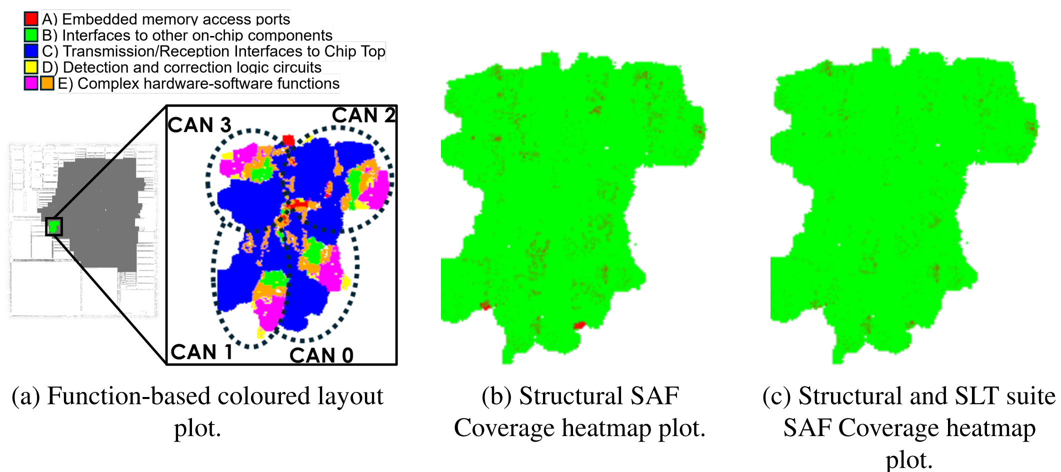
Figure 4.15 presents a layout heatmap extracted from the actual physical implementation of the CAN peripheral within the DUT, where the four distinct controllers can be clearly identified. Figure 4.15a illustrates a color-coded layout map of the CAN peripheral logic, highlighting the critical regions identified through the structural test weakness analysis discussed in section 3.0.6. In this visualization, at least two highly localized hot spots can be observed one corresponding to the error detection and correction logic, and another located in the Transmission/Reception interface connected to the chip top, situated in the southern portion of the heatmap.

These experimental findings corroborate the hypothesis formulated in the previous chapter regarding the structural test weaknesses. As a preview of the results discussed later, fig. 4.15b displays the layout heatmap of the CAN subsystem, where

Test Name	Algorithm Number	Mode[1]	Targeted Weakness	Execution Time [cc]	Mem. Foot-print [KB]	Fault Sim. Time[2] [h]	Develop. Time [h]
Embedded Memory Access Port	Algorithm 1	LPBK	**A**	2,522,475	9.96	13.96	20
Interfaces to other SoC components	Algorithm 2	NA	**B**	23,870	12.55	0.13	26
Transmission/Reception to Chip Top	Algorithm 3	LPBK, CMP	**C**	332,096	6.42	1.84	22
Detection and correction unit	Algorithm 4	CMP	**D**	7,294,466	6.82	40.39	34
Complex transmission functions	Algorithm 5	LPBK	**E**	1,002,251	6.92	5.55	10
Synchronization functions	Algorithm 6	CMP	**E**	447,579	6.77	2.48	34
			Total	**11,622,737**	**49.44**	**64.35**	**136**

Table 4.8 Properties of the SLT suite for the CAN peripheral.

[1] LPBK = Loopback, CMP = Companion Module. [2] Stuck-at + Transition Delay fault models for a total of ca. 900k faults [4].



(a) Function-based coloured layout plot.

(b) Structural SAF Coverage heatmap plot.

(c) Structural and SLT suite SAF Coverage heatmap plot.

Fig. 4.15 Layout of the four CAN controllers [4].

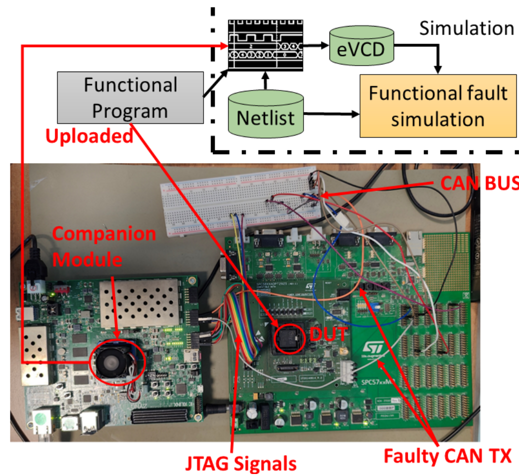


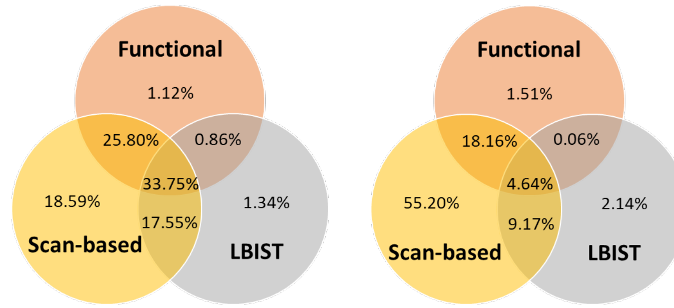
Fig. 4.16 Laboratory experimental setup.

untested SAFs are shown in red and tested faults (detected by all structural tests) are shown in green. Figure 4.15c then depicts the corresponding heatmap after applying the proposed SLT suite in conjunction with structural tests. It is evident that numerous red regions disappear or become significantly lighter, demonstrating the complementary effectiveness of the SLT methodology.

Test Nature	Test Name	Algorithm Number	Fault coverage SAF [%]			Fault coverage TDF [%]		
			Single	Incremental	Δ	Single	Incremental	Δ
Structural	Scan-based	NA	95.69	95.69	NA	87.17	87.17	NA
	LBIST	NA	53.46	97.89	2.2	16	89.38	2.15
	MBIST	NA	1.13	**97.89**	0	0.03	**89.38**	0
Functional	Transmission/Reception to Chip Top	Algorithm 3	40.30	98.45	0.56	12.55	89.65	0.27
	Embedded Memory Access Port	Algorithm 1	41.79	98.65	0.2	12.77	89.73	0.08
	Complex transmission functions	Algorithm 5	25.01	98.66	0.01	5.79	89.74	0.01
	Interfaces to other SoC components	Algorithm 2	13.36	98.96	0.3	5.11	90.63	0.89
	Detection and correction unit	Algorithm 4	39.94	98.99	0.03	11.82	90.67	0.04
	Synchronization functions	Algorithm 6	35.39	99.01	0.02	14.24	90.89	0.22
			Total	**99.01**	**1.12**	**Total**	**90.89**	**1.51**

Table 4.9 Fault coverages for Stuck-at fault (435,967 faults) and Transition delay fault models (435,966 faults) [4].

The relative improvement achieved by the proposed SLT suite for SAFs is approximately 50% with respect to the faults that remained untested after structural



(a) Stuck-At fault model. (b) Transition Delay fault model.

Fig. 4.17 Venn diagrams of detected faults between various test approaches [4].

testing. In the case of TDFs, although the incremental improvement is proportionally higher than that observed for SAFs, the overall gain with respect to untested faults is around 15%. Considering the comparatively low coverage obtained for TDFs through structural methods, it can be inferred that a significant portion of these faults are likely functionally untestable.

4.1.7 System-Level-Test Test Generation Automation

The proposed framework has been successfully applied to the generation of functional procedures for a complex SoC, as shown in fig. 4.18(a). table 4.10 presents preliminary results on incremental fault coverage for both Stuck-At (SAF) and Transition Delay (TDF) faults for the modular CAN, shown in fig. 4.18(b), obtained using the proposed approach compared to scan-based approaches. Despite their simplicity, the generated functional procedures lead to a substantial increase in fault coverage, highlighting the effectiveness of the approach. Furthermore, the test time represents only a small fraction of that required for scan-based tests and can therefore be easily included in manufacturers' test suites.

Test Nature	Test Name	Test Time [ms]	SAF [%]		TDF [%]	
			Incr.	Δ	Incr.	Δ
Structural	SCAN	7×10^9	97.89	-	89.38	-
Functional	FSWGEN	1.1	98.61	**0.72**	90.35	**0.97**

Table 4.10 Incremental fault coverage for the CAN peripheral [5].

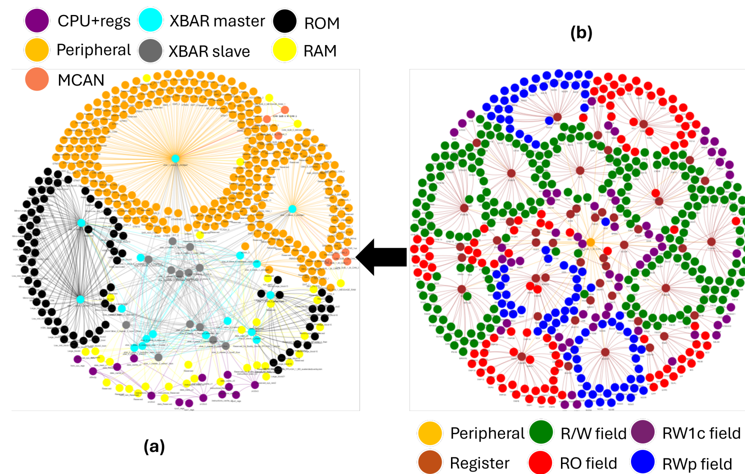


Fig. 4.18 (a) DTS graph for a complex SoC. (b) DTS graph of a CAN module [5].

4.1.8 In-Field Logic Diagnosis leveraging LBIST

This section presents the experimental results obtained using the methodology described in section 3.0.12, applied to both simulated circuits from the academic benchmark suite ITC'99 [74] and the industrial case study introduced in section 4.1. A portion of these results has been published in [3].

The DE values were derived by executing the complete diagnostic flow on batch faulty devices, that failed In-Field operations, throughout their operational lifetime. The outcomes from real defective devices analyzed using the proposed approach were compared against those obtained through the methodologies described in [31, 32], as discussed in section 2.7. These reference methods were chosen for comparison because, like the proposed approach, they rely exclusively on LBIST signatures without requiring architectural modifications to determine the root cause of failures. Both reference methodologies were also applied to real failed industrial devices to evaluate their respective DR performances.

In addition, the proposed methodology was compared with the simulation approach presented in [33], which employs a storage-based LBIST architecture. The comparison focused on academic benchmark circuits and simulated DE values for the industrial case study.

Furthermore, simulation results are provided to illustrate how the methodology described in [28] would perform if applied to the same industrial SoC, assuming the necessary architectural modifications were implemented. The section concludes

with a comparative analysis of the main state-of-the-art methodologies, highlighting their respective strengths and the specific scenarios in which the proposed approach demonstrates particular advantages.

The core component of the external software tool used in the experimental setup is responsible for orchestrating LBIST experiments, executing the dichotomic search algorithm associated with the proposed methodology, and managing communication with both the circuit board and the power supply.

It is worth noting that environmental conditions, such as temperature variations and operating parameters, can significantly affect the diagnostic data collected and, consequently, the resulting analysis. To mitigate these influences, preliminary experiments were performed on a fault-free device over several hours of continuous BIST execution under varying environmental and operational conditions. This characterization phase enabled the identification of optimal parameter configurations that ensure correct test execution, while also determining those that should be avoided, as excessive thermal stress or prolonged BIST activity can cause even a non-faulty device to produce incorrect signatures.

4.1.9 FLASH Footprint

Each of the 7 LBIST partitions on the board is equipped with 16 bits programmable Pattern-Count-Stop (PCS) registers, initially configured with a starting PCS value of $0xFFFF$. This value is progressively halved or otherwise adjusted during each iteration using the dichotomic search algorithm described in algorithm 8.

Each entry in the *frequency region* is represented by a 4 byte floating-point number. For both the *golden* and *failure* regions, each entry occupies 80 bits: 16 bits for the index and 64 bits for the corresponding signature. The total FLASH memory footprint depends on the selected fault model.

For TRN fault modeling, the total memory requirement is

$$32 + 7 \times 80 \times ((\log_2 (0xFFFF) + 1)) = 9,552$$

, which corresponds to approximately 9 Kb. Conversely, in the SA fault model, where the *frequency region* and the indexes of the *golden region* can be omitted, the

structure size increases to

$$7 \times 64 \times (0xFFFF + 1) + 16 = 29,360,144$$

, equivalent to roughly 29 Mb.

To reduce the memory footprint associated with SA fault modeling, a *skip-step* technique can be applied to the rows of the golden signature table. However, this optimization introduces a trade-off, where a signature mismatch may result in a reduced DR. Nevertheless, if golden signatures are strategically stored (e.g., following the fault coverage curve) the impact on diagnostic accuracy can be effectively minimized.

It is worth emphasizing that the proposed methodology focuses primarily on capturing In-Field diagnostic data related to TRN fault detection, which remains the primary focus of this work.

4.1.10 Signature collection time

In the considered case study, the self-test process requires approximately 100 ms to complete when all LBIST partitions are configured to test the maximum number of patterns, corresponding to $0xFFFF$ in parallel. Upon completion, the device performs a functional reset that reinitializes the cores while maintaining the activity of the STCU to collect signatures. Assuming the device operates correctly, only a single LBIST execution is required during each power-on or power-off event, resulting in a total self-test duration of 100 ms. Meanwhile, the detection of a signature mismatch, initiates the first step of estimating the operational working frequency with no failures. Beginning at the maximum frequency, the proposed method progressively reduces it by a fixed step Δ, executing LBIST with the full number of patterns at each step until a passing condition is reached. Considering a starting frequency of 200 MHz and $\Delta = 10$ MHz, the optimal case occurs when the passing frequency is $max_f - \Delta$ requiring a single iteration and an overall execution time of 100 ms. In the worst-case scenario, where the process must iterate through all frequency steps untilΔ is reached, the total execution time becomes $100 \times \frac{200}{\Delta} = 2,000$ ms. At this point, the dichotomic search algorithm described in algorithm 8 is initiated, involving a total of 16 LBIST executions with varying numbers of patterns. Under the worst-case condition, where the dichotomic search concludes after $0xFFFF - 1$ patterns, the

estimated total execution time is $100+\sum_{i=1}^{15}\frac{100\cdot(2^i-1)}{2^i} \sim 1,500$ ms, which accounts for the case where the upper dichotomy branch is always selected. In contrast, in the best-case scenario, where the index to be identified is 1, meaning the lower dichotomy branch is consistently chosen, the total time required is approximately: $\sum_{i=0}^{15}\frac{100}{2^i}$ ms.

It is worth noting that if golden signatures are not already available in the database, each missing signature must be computed and stored, effectively doubling the total execution time for that iteration.

The reported timing data include only the LBIST execution duration and exclude the time required for FLASH memory erase and programming operations. Although these operations occur at each iteration, their respective durations, 0.0192 ms and 0.00475 ms for programming, are negligible compared to the overall LBIST execution time.

4.1.11 Collected data from faulty devices

To determine an optimal frequency range for targeting TRN faults, LBIST signatures were collected in the experimental setup by varying the operating frequency at the core reference voltage. The frequency adjustment is done according to by the parameter Δ, which represents the decrement applied to the frequency at each step, set to 10 MHz. In terms of register configuration, the minimum permissible and valid frequency for the studied scenario is 2 MHz.

The first objective of the proposed methodology is to identify the lowest frequency at which the DUT no doesn't fail. Based on experimental observations from the industrial case study, the most efficient approach consisted of initially decreasing the frequency in 10 MHz steps until the DUT successfully passed the test. Once a stable operating point was reached, a dichotomic search was then applied to fine-tune the exact passing frequency by adjusting it in smaller increments or decrements of 2 MHz.

The results obtained from twenty-three defective devices using this procedure are presented in fig. 4.19.

DEVICE\CLK	150	160	170	180	190	200
GOOD						
FAIL#1						
FAIL#2						
FAIL#3						
FAIL#4						
FAIL#5						
FAIL#6						
FAIL#7						
FAIL#8						
FAIL#9						
FAIL#10						
FAIL#11						
FAIL#12						
FAIL#13						
FAIL#14						
FAIL#15						
FAIL#16						
FAIL#17						
FAIL#18						
FAIL#19						
FAIL#20						
FAIL#21						
FAIL#22						
FAIL#23						

Fig. 4.19 Failures information of failing devices based on the LBIST execution frequency [3].

4.1.12 Diagnostic Expectation

The average DE was calculated following the construction of the tree-based fault dictionary described in section 3.0.14. The resulting average DE values for each LBIST partition are reported in table 4.12. These values represent the mean number of candidate faults identified after a misbehavior is detected. The average DE is obtained from the tree-based fault dictionary by dividing the total number of candidate faults contained in the leaf nodes by the total number of executed patterns.

When constructing the fault dictionary for LBIST, the failure branches of previously *failed* nodes are pruned, thereby restricting diagnostic information to the first failing pattern only, as discussed in section 2.5. This limitation implies that, in the worst-case scenario, where the first failing pattern occurs among the earliest test patterns, the average DE tends to be relatively high.

For instance, in the industrial case study, LBIST partition 5 exhibits an average of approximately 109 potential faults per failure event. Considering that the total fault list for this partition contains roughly 6 million faults, this diagnostic resolution remains acceptable despite the comparatively large DE value.

However, when the first failing pattern occurs beyond the 64th pattern position, the average DE for LBIST partition 5 decreases significantly, reaching an average

of approximately 37. This trend indicates a clear dependence between DE and the index of the first failing pattern.

This relationship is illustrated in fig. 4.20, which depicts how the position of the first failing pattern directly influences the corresponding DE value.

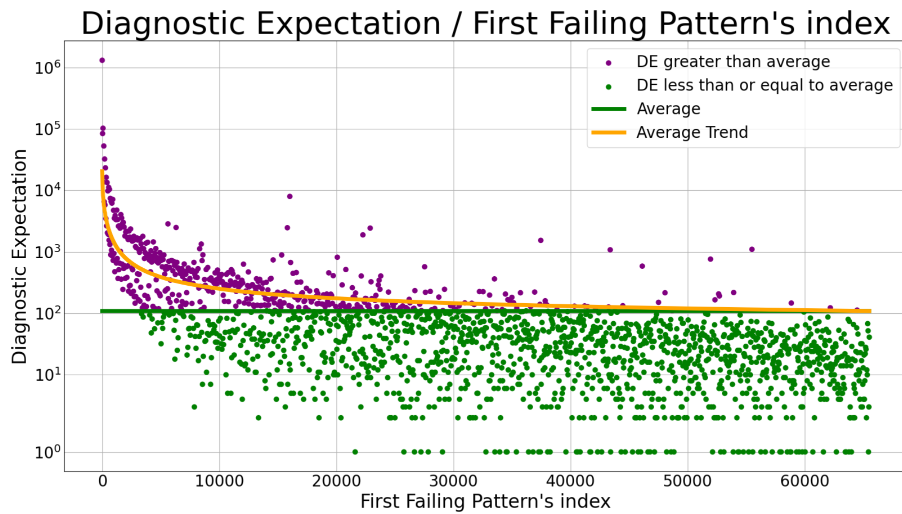


Fig. 4.20 DE dependability, in the industrial case study, on the position of the first failing pattern for LBIST partition 5 [3].

As illustrated in fig. 4.20, the DE exhibits a clear trend as a function of the index of the first failing pattern. The y-axis, plotted on a logarithmic scale, represents the DE values, while the x-axis denotes the index of the first failing pattern. The green horizontal line indicates the average DE, which is approximately 109, whereas the orange curve represents the average trend of the DE.

Data points corresponding to DE values greater than the average are shown in purple, while those below the average DE are displayed in green. This graphical representation clearly highlights the overall inverse correlation between the DE and the index of the first failing pattern. Specifically, as the index of the first failing pattern increases (moving right along the x-axis), the DE tends to decrease (moving downward along the y-axis). This reduction in DE indicates an improvement in diagnostic accuracy, as fewer candidate faults remain.

Table 4.11 reports the average DE results obtained using the proposed logic diagnosis methodology, compared against other state-of-the-art techniques for the ITC'99 benchmark circuits [74].

Among the compared approaches, the method presented in [32] achieves the best performance for most of the evaluated benchmark circuits. Nevertheless, the proposed methodology demonstrates comparable and competitive results, achieving similar levels of diagnostic accuracy while maintaining the advantages of low architectural overhead and In-Field applicability.

Table 4.11 Average DE for ITC'99 benchmarks [3].

Circuit	Avg. DE	Avg. DE [33]	Avg. DE [31]	Avg. DE [32]
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 each LBIST partition, table 4.12 provides a summary of the average DE obtained for the industrial case study, compared against the results achieved using the methodologies proposed in [33], [31], and [32]. The reported average DE values offer valuable insight into the diagnostic accuracy associated with specific partitions.

For example, in LBIST partition 1, which includes a total fault list of approximately 15 million faults, the proposed method achieves an average DE of 207, indicating a relatively high DR. In contrast, LBIST partition 0, comprising roughly 4 million faults, shows an average DE of 92, suggesting a more effective diagnostic performance due to the lower number of potential fault candidates.

Table 4.12 Average DE for each LBIST partition in the industrial case of study [3].

LBIST Partition	Avg. DE	Avg. DE [33]	Avg. DE [31]	Avg. DE [32]
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

The storage-based LBIST scheme proposed in [33] was also employed for comparison in the context of the industrial case study. The experimental procedure was reproduced to obtain comparable results, including both the software procedure for diagnostic pattern generation and the subsequent logic diagnosis methodology applied to a fault-injection campaign consisting of $2,000$ randomly selected faults.

The approach described in [31] leverages the error propagation function of the MISR to identify potential candidates responsible for faulty behavior, as well as the logic cones associated with scan cells that capture erroneous values. To compute the DE values corresponding to this method, the same procedure reported in the original study was followed. Specifically, $2,000$ random faults were injected into the combinational logic of each LBIST partition in the case study, and the resulting faulty signatures were diagnosed using the precomputed MISR error propagation functions.

A similar methodology was adopted for comparison with [32], which represents an extension of the work in [31]. In this enhanced approach, fault candidates are first derived from the logic cones of the scan cells capturing faulty values, following the same random fault injection procedure involving $2,000$ injected faults and MISR-based propagation analysis. The primary improvement introduced in [32] lies in an additional pruning mechanism that removes candidates whose associated errors would have propagated to MISR bits observed to be correct, thus improving diagnostic precision relative to [31].

For each LBIST partition, table 4.12 reports the average number of fault candidates obtained using the proposed methodology, as well as those derived from [31] and [32]. In the industrial case study, the latter two approaches yield comparatively higher DE values, indicating lower diagnostic resolution. This can be attributed to the fact that [31] relies solely on the MISR signature obtained from a single LBIST execution corresponding to the failing pattern, without exploiting the additional information from previously passed patterns as done in the proposed approach.

Although the improved method from [32] achieves better pruning efficiency by considering correct MISR bits, both methodologies face limitations when applied to large and complex SoCs. The industrial case study includes multiple scan chains, several LBIST partitions, and highly compacted MISR outputs, which lead to substantial information loss during signature compaction. Relying solely on MISR error propagation analysis is insufficient. Moreover, when scan chains of varying lengths

are present, the number of required shift operations can exceed the chain length, further complicating the computation of accurate error propagation functions.

These limitations become evident in the comparison results. For instance, in LBIST partition 5, a significant discrepancy is observed between the proposed approach and those presented in [31] and [32]. This partition features 64-bit MISRs, $1,000$ scan chains, and a total of approximately 6 million faults (see table 4.1). In this context, the reduced observability of error propagation significantly reduce the accuracy of both reference methods. While LBIST partition 0 also exhibits a notable difference between the approaches, both [32] and [31] perform relatively better for this partition due to its smaller fault population and lower number of scan chains, which results in simpler test compaction.

The best results for the industrial case study are achieved by the method proposed in [33]. This outcome is reasonable, as that approach employs predictable diagnostic patterns rather than pseudo-random ones and relies on more detailed diagnostic information. However, as previously discussed, In-Field LBIST execution is subject to several practical constraints. The methodology proposed in this thesis focuses on recovering diagnostic information from the first failing pattern only, thereby maximizing the data available under In-Field conditions. In contrast, [33] assumes the ability to re-test the device and access complete failure information, which is typically infeasible in real operational environments.

A final comparative summary of all considered methodologies, emphasizing their respective advantages and suitable application contexts, is presented in section 4.1.15.

4.1.13 Logic Diagnosis of faulty devices

A set of failed devices collected from the field was used to validate the proposed methodology, as shown in fig. 4.19. The methodology produces, for each failed device, the total number of candidate faults by leveraging the index of the first failing pattern during diagnosis. This index is used to locate the corresponding node within the tree-based fault dictionary, thereby enabling the retrieval and evaluation of the associated set of candidates.

As illustrated in fig. 4.19, a total of twenty-three devices were identified as faulty. Among these, eight were conclusively diagnosed as exhibiting combinational logic failures (see table 4.13), each associated with a specific set of candidate

faults. Additionally, fifteen devices were identified as affected by scan chain failures. This distinction is made possible because the LBIST can be configured to execute a predefined number of patterns as scan chain integrity tests, performed without capture, prior to the logic tests (with capture clocks). By using the index of the first failing pattern, the proposed method can differentiate between devices exhibiting scan chain failures and those affected by combinational logic issues.

The logic diagnosis results are summarized in table 4.13, where each row corresponds to a faulty device and reports the achieved DR, expressed as the number of candidate faults. The DR provides a direct measure of the diagnostic precision achieved for each failed chip. For example, device *FAIL#2* exhibits a DR of 2, indicating very high diagnostic accuracy, whereas device *FAIL#5* presents a comparatively larger set of candidate faults, an expected outcome given the limited diagnostic information retrievable from In-Field data.

Table 4.13 DR for a batch of faulty devices field return [3].

Faulty chip #	DR	DR [31]	DR [32]
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 report the DR values obtained using the methodologies described in [31] and [32], respectively. The proposed method demonstrates a clear advantage in diagnostic accuracy for the majority of the analyzed devices. While the latter approach [32] generally improves upon the DR achieved by [31], it still falls short of the results obtained with the proposed methodology.

The only case in which the proposed approach exhibits lower accuracy compared with the two reference methods is device *FAIL#5*. This deviation can be explained by the position of the first failing pattern, which in this case corresponds to the

16th pattern. As illustrated in fig. 4.20, the index of the first failing pattern directly influences the resulting DR.

Additionally, for device *FAIL#20*, the approach from [32] achieves a DR value nearly comparable to that of the proposed method, while also improving upon the outcome of [31]. Nevertheless, when considering the overall average performance reported in table 4.12, the proposed methodology consistently achieves far superior diagnostic accuracy across all devices.

4.1.14 Using another architecture scheme

Several works in the SOTA, as summarized in table 2.2, propose modifications or extensions to the standard STUMP BIST architecture to enhance diagnostic efficiency and resolution. A direct comparison with such approaches would not be appropriate, as the objective of the present work is to provide a methodology that can be seamlessly integrated into any existing architectural framework without requiring specific hardware adaptations.

Nevertheless, to evaluate the potential impact of architectural modifications, a simulation based on an alternative architecture was conducted using the industrial case study. In particular, results were computed for the BISD architecture [28], selected for its conceptual alignment with the goals of the proposed methodology, as discussed in section 2.7.

The initial step in this evaluation involved analyzing the memory requirements imposed by the BISD approach:

- 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.

Analyzing all LBIST partitions reported in table 4.1, the total required memory amounts to $3,146,528$ bits, corresponding to approximately 3 Mb. Considering that the target fault model for this study is the TDF, only a fraction of this memory is actually required. The reported value remains lower than that presented in section 4.1.9 for the SA fault model. Furthermore, this memory must be included in the area

overhead estimation, as the BISD approach introduces additional dedicated memory local to the BISD structure, rather than reusing existing on-chip memory resources.

Regarding the diagnostic phase, the method proposed in [28] begins by simulating all possible faulty signatures for each fault in the DUT, thereby constructing a comprehensive fault dictionary. However, this process introduces significant scalability limitations, as its complexity directly depends on the size of the FU. When the In-Field BISD collects incorrect signatures, if present, proceeds to rank all possible faults according to their likelihood. As described in the cited work, the fault appearing first in the ranking is assumed to be the most probable root cause. Based on 800 injected faults, the authors of [28] report experimental results in which DR is determined by counting how frequently the highest-ranked fault matches the injected one.

To adapt this approach to the industrial case study, a campaign of approximately $1,000$ fault injections was performed for each LBIST partition of the DUT. The results show that in 23% of the cases, the top-ranked fault corresponded to the actual injected fault. In the remaining cases, where the highest-ranked fault did not match the injected one, the number of candidate faults reached its maximum, corresponding to the full fault list of the LBIST partition.

Moreover, unlike [28], which provides a complete ranking of all faults, the proposed methodology explicitly defines a bounded set of candidate faults for each diagnosis instance. Therefore, while the ranking mechanism of [28] may be followed during post-failure analysis, it does not effectively reduce the number of candidate faults, whereas the proposed approach directly constrains and minimizes this set.

4.1.15 High-Level Summary Comparisons

A comprehensive comparison of multiple SOTA logic diagnosis techniques, including the proposed methodology, is provided in table 4.14. This table summarizes the various requirements for data acquisition from defective devices, the pre-compilation activities necessary for each approach, the type of information generated in the context of NTF devices, and the final diagnostic outcomes achieved.

When applied to complex real-world devices, the proposed methodology demonstrates superior efficiency, as evidenced by the experimental results, compared to the approaches presented in [28], [31], and [32]. In particular, the BISD approach [28]

Table 4.14 High-Level Summary among different LBIST diagnostic methodologies [3].

Method	Required data of the failed device	Pre-computation	Insights in case of NTF	Limitations	Outcome
BISD: Scan-based Built-In self-diagnosis [28]	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 [33]	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 [31]	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 [32]	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

depends on obtaining a complete set of faulty signatures, requiring extensive pre-computation and fault injection campaigns across the entire FU. This process is computationally demanding and ultimately produces a ranked list of all potential faults rather than a concise subset of plausible candidates, thus offering limited reduction of the fault search space.

Conversely, the methodologies described in [31] and [32] show satisfactory performance for benchmark circuits but exhibit reduced accuracy and scalability when applied to large and complex industrial systems.

Although the approach in [33] demonstrates high efficiency in industrial contexts, it does not provide diagnostic insights for NTF scenarios, as summarized in table 4.14. This method relies on generating deterministic diagnostic patterns and considers all failure patterns during analysis, thereby enhancing accuracy. However, the requirement to access all failure patterns renders it impractical in many real-world situations, particularly when devices cannot be re-tested post-field deployment.

Selecting an appropriate diagnostic strategy is therefore essential and must account for factors such as data availability, the possibility of re-testing the faulty device in debug mode, available computational resources, and the desired granularity of diagnostic information. In this context, the proposed approach offers a highly efficient solution for cases where limited data can be extracted from the device. Moreover, its compatibility with any LBIST scheme, achieved by primarily relying on the index of the first failing pattern, ensures broad applicability without architectural modification.

Notably, the proposed methodology can still generate a meaningful set of candidate faults even when the device operates correctly upon return to the manufacturer. In such circumstances, the proposed strategy stands out as the most practical and effective among the existing alternatives.

Scalability discussion

The proposed methodology for diagnosing devices returned from the field operates through two primary stages: data collection and logic diagnosis.

In the data collection phase, scaling to larger devices may introduce challenges related to data storage capacity. Nevertheless, since larger devices typically include

proportionally greater memory resources, the required diagnostic data can generally be accommodated without issue. In cases where stringent memory limitations exist, optimization strategies can be employed, such as storing only the index of the first failing pattern instead of the full LBIST signatures, or limiting the number of failing entries to retain.

Regarding the logic diagnosis phase, the main scalability concern arises from the time required to construct the fault dictionary. However, in industrial practice, such data are usually generated during the design and characterization stages of the device. Therefore, no additional computational effort is introduced by the proposed approach.

Finally, the proposed methodology demonstrates strong versatility, as it does not depend on a specific LBIST scheme. It can be readily applied to virtually any industrial device that supports LBIST execution, requiring only minimal additional resources. This distinguishes it from other approaches in the literature, such as those presented in [28] and [33], which impose stricter architectural and implementation constraints.

Chapter 5

Conclusions

This thesis has investigated advanced testing and diagnostic methodologies for automotive SoC devices across their entire life-cycle, from production testing to In-Field operation. The rapid growth in device complexity and the stringent safety standards in the automotive sector have made reliability assessment an increasingly critical challenge. Traditional structural testing, while well-established, struggles to maintain sufficient coverage and scalability for modern devices, resulting in test escapes and diagnostic uncertainty. To address these issues, this work proposed complementary functional and diagnostic solutions for both manufacturing and In-Field phases, aiming to improve fault coverage, diagnostic accuracy.

5.0.1 Summary of Contributions

The contributions of this thesis span two major domains of the device life cycle: the manufacturing test flow and the In-Field diagnostic phase.

Manufacturing Test Flow: Burn-In and System-Level Test

The first part of the thesis focuses on improving the effectiveness of the BI and SLT phases. A novel BI methodology has been introduced to apply high-quality stress using functional workloads derived from indirect, netlist-less measurements, rather than relying solely on structural test patterns. This approach enhances early-life failure screening by enabling meaningful stress conditions even when the design's

internal netlist or fault models are not directly accessible. Building upon this, the thesis presents a functional SLT methodology designed to complement structural testing, particularly addressing testability limitations in communication peripherals. The proposed SLT suite introduces FPGA-based companion modules capable of stimulating the DUT, injecting protocol-level errors, and emulating realistic communication scenarios. This setup allows for targeted testing of on-chip peripherals and interconnects that are otherwise inaccessible through traditional ATPG loop-back tests. Experimental fault grading performed on an industrial automotive SoC shows that the proposed SLT suite improves stuck-at fault coverage by approximately 50% for previously untested faults and increases transition delay fault coverage by about 15%, proving its effectiveness as a complementary layer to existing scan-based and BIST approaches.

To streamline the SLT development process, the thesis introduces the FSWGEN framework, a fully automated tool for generating functional test workloads. FSWGEN exploits the DTS to construct a graph-based model of the SoC's architecture, including component relationships, interfaces, and dependencies. From this model, the framework automatically produces portable test programs. While further extensive validation across diverse architectures is required, preliminary results suggest this approach has the potential to reduce manual development effort, decouple test logic from specific ISAs, and facilitate faster test deployment across product variants.

In-Field Diagnosis of Returned Devices

The second part of the thesis addresses In-Field diagnostic challenges, particularly the NTF phenomenon, where devices that fail during operation appear functional when tested in the lab. A lightweight diagnostic methodology has been developed based on LBIST, which executes during key-on and key-off events in the vehicle. By applying a dichotomic search algorithm, the method identifies the index of the first failing test pattern and records it, together with its LBIST signature, in NVM. A tree-based fault dictionary, precomputed from fault simulations, is then used to perform post-return logic diagnosis using only the stored data.

Validation on both ITC'99 benchmark circuits and an industrial automotive SoC confirmed that the proposed In-Field diagnosis achieves high diagnostic resolution while maintaining low memory and computational overhead. When compared to SOTA approaches, the methodology achieves comparable or superior accuracy,

demonstrating scalability, efficiency, and full compatibility with existing industrial LBIST frameworks.

5.0.2 Expected Impact

The methodologies presented in this thesis are expected to have significant industrial and scientific impact. From an industrial perspective, the proposed approaches offer practical solutions that enhance reliability assessment and reduce diagnostic uncertainty across the entire life-cycle of automotive SoCs:

- The BI enhancement improves early-life screening effectiveness, reducing latent defects.
- The SLT for communication peripherals and companion FPGA modules provide functional stimuli under realistic stress conditions, identifying faults undetectable by traditional scan or loop-back testing.
- The FSWGEN automation tool offers a promising pathway to reduce engineering effort and improve test portability, addressing the scalability challenges of next-generation SoC families.
- The In-Field diagnosis methodology bridges the gap between manufacturing and field reliability analysis, enabling data-driven fault tracing and reducing the rate of NTF cases.

From a research perspective, these contributions advance the SOTA in DfT, SLM, and post-silicon analysis, addressing the growing challenges of testing and diagnosing complex automotive SoCs. The results demonstrate that improved test coverage, faster test development, and efficient post-sale diagnostics can be achieved without additional architectural cost, paving the way toward more reliable, safe, and maintainable next-generation automotive integrated systems.

5.1 Future Works

While this thesis addresses several critical challenges in automotive SoC testing, the rapid evolution of semiconductor technologies opens new directions for further

research. Future work will focus on expanding the scope and capabilities of the proposed methodologies in the following directions:

- **Enhancement of FSWGEN capabilities:** The current framework relies on static parsing of Device Tree structures. Future iterations could integrate AI-assisted code generation techniques (e.g., LLMs) to synthesize more complex driver logic and handle dynamic peripheral dependencies that are not fully captured by static descriptors. Additionally, extending support to emerging open architectures, such as RISC-V, would further validate the portability claims.
- **Refine the "Netlist-less" Stress Methodology:** Future work will concentrate on optimizing the underlying algorithmic to maximize stress metrics and pattern quality. Additionally, a major objective is to reduce the memory footprint of the generated patterns. Minimizing these storage requirements is essential for scalability, enabling the methodology to be effectively applied to significantly larger and more complex ASICs without hitting resource constraints.

Acronyms

ADAS Advanced Driver-Assistance Systems.

AI Artificial Intelligence.

AI HW AI Hardware Accelerator.

ALU Arithmetic Logic Unit.

ASIC Application-Specific Integrated Circuit.

ATE Automatic Test Equipment.

ATPG Automatic Test Pattern Generation.

BI Burn-In.

BISD Built-In Self-Diagnosis.

BIST Built-in-Self-Test.

CAN Controller Area Network.

CPU Central-Processing-Unit.

DE Diagnostic Expectation.

DfT Design-for-Testability.

DMA Direct Memory Access.

DPPM Defective Parts Per Million.

DR Diagnostic Resolution.

DTS Device Tree Specification.

DUT Device-Under-Test.

ECC Error Correction Code.

EML Error-Management-Logic.

FPGA Field-Programmable-Gate-Array.

FSM Finite-State-Machine.

FU Fault Universe.

GPIO General Purpose I/O.

GPU Graphics-Processing-Unit.

HAL Hardware Abstraction Layer.

IC Integrated Circuit.

IDDQ Direct Drain Quiescent Current.

IP Intellectual Property.

IRQ Interrupt ReQuests.

ISA Instruction Set Architecture.

LBIST Logic BIST.

LFSR Linear Feedback Shift Register.

LLM Large-Language-Model.

LUT Look-up-Table.

MAC Multiply and Accumulate.

MBIST Memory BIST.

MISR Multiple Input Signature Register.

MTA Memory Toggle Activity.

MTC Memory Toggle Coverage.

MTM Memory Toggle Metric.

MUT Module-Under-Test.

NTF No Trouble Found.

NVM Non-Volatile Memorie.

ODE Output Data Evaluator.

PCS Pattern-Count-Stop.

PE Processing Element.

PGA Programmable Gain Amplifier.

PI Primary Input.

PO Primary Output.

PRNG Pseudo-Random Number Generator.

RAM Random Access Memory.

RTL Register-Transfer-Level.

RTOS Real-Time Operating System.

SA Stuck-At.

SAF Stuck-At Fault.

SBST Software-Based Self-Tests.

SISR Single-Input-Signature-Register.

SLM Silicon Life-cycle Management.

SLT System-Level Test.

SoC System-on-Chip.

SOTA State-Of-The-Art.

SPI Serial-Pheriperal-Interface.

SRSG Shift Register Sequence Generator.

STUMP Self-Test Using MISR/Parallel SRSG.

TA Toggle Activity.

TAP Test Access Port.

TC Toggle Coverage.

TDBI Test-During-Burn-In.

TDF Transition Delay Fault.

TPG Test Pattern Generation.

TRN Transition.

UUT Unit-Under-Test.

VAR(MTA) Memory Toggle Activity Variance.

VAR(TA) Toggle Activity Variance.

VLE Variable-Length Encoding.

VLSI Very-Large-Scale-Integration.

References

[1] Francesco Angione, Paolo Bernardi, Nicola di Gruttola Giardino, Gabriele Filipponi, Claudia Bertani, and Vincenzo Tancorre. A system-level test methodology for communication peripherals in system-on-chips. *IEEE Transactions on Computers*, pages 1–8, 2024.

[2] Gabriele Filipponi, Denis Schwachhofer, Francesco Angione, Claudia Bertani, Simone Corbellini, Nicola di Gruttola Giardino, Giuseppe Garozzo, Giorgio Insinga, Vincenzo Tancorre, and Paolo Bernardi. Netlist-independent functional stress pattern generation strategy for ai hw accelerators embedded into socs. *IEEE Transactions on Computers*, 75(2):554–566, 2026.

[3] 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.

[4] Francesco Angione, Paolo Bernardi, Nicola di Gruttola Giardino, Gabriele Filipponi, Claudia Bertani, and Vincenzo Tancorre. A system-level test methodology for communication peripherals in system-on-chips. *IEEE Transactions on Computers*, 74(2):731–739, 2025.

[5] Gabriele Filipponi. Fswgen: a device-tree specification driven system-level test workload generator. In *2025 IEEE International Test Conference (ITC)*, pages 558–559, 2025.

[6] 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.

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

[8] Davide Appello, Conrad Bugeja, Giorgio Pollaccia, Paolo Bernardi, Riccardo Cantoro, Marco Restifo, Ernesto Sanchez, and Federico Venini. An optimized test during burn-in for automotive soc. *IEEE Design Test*, 35(3):46–53, 2018.

[9] 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.

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

[11] F. Angione, P. Bernardi, G. Filipponi, M. Sonza Reorda, D. Appello, V. Tancorre, and R. Ugioli. An optimized burn-in stress flow targeting interconnections logic to embedded memories in automotive systems-on-chip. In *IEEE ETS*, pages 1–6, May 2022.

[12] Francesco Angione et al. Automatic Generation of System-Level Test for un-core logic of large Automotive SoC . *IEEE Transactions on Computers*, (01):1–14, July 5555.

[13] Paolo Bernardi, Alberto Bosio, Giorgio Di Natale, Andrea Guerriero, Ernesto Sanchez, and Federico Venini. Improving Stress Quality for SoC Using Faster-than-At-Speed Execution of Functional Programs. In Thomas Hollstein, Jaan Raik, Sergei Kostin, Anton Tšertov, Ian O'Connor, and Ricardo Reis, editors, *VLSI-SoC: System-on-Chip in the Nanoscale Era – Design, Verification and Reliability*, volume AICT-508 of *IFIP Advances in Information and Communication Technology*, pages 130–151, Tallinn, Estonia, September 2016. Springer International Publishing.

[14] F. Corno, G. Cumani, M. Sonza Reorda, and G. Squillero. Efficient machine-code test-program induction. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, volume 2, pages 1486–1491 vol.2, May 2002.

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

[16] 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.

[17] G. Iaria, F. Angione, P. Bernardi, M. Sonza Reorda, D. Appello, G. Garozzo, and V. Tancorre. A novel pattern selection algorithm to reduce the test cost of large automotive systems-on-chip. In *IEEE LATS*, pages 1–6, Sep. 2022.

[18] 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.

[19] 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.

[20] Devicetree specification - release v0.3. https://github.com/devicetree-org/devicetree-specification/releases/download/v0.3/devicetree-specification-v0.3.pdf. Accessed: 20 October 2023.

[21] 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.

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

[23] 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.

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

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

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

[33] 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.

[34] 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.

[35] Gordon E. Moore. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff. *IEEE Solid-State Circuits Society Newsletter*, 11(3):33–35, 2006.

[36] Laung-Terng Wang, Cheng-Wen Wu, and Xiaoqing Wen. *VLSI Test Principles and Architectures: Design for Testability*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[37] Ieee draft standard test access port and boundary scan architecture. *IEEE P1149.1/D2012.e27, September 2012*, pages 1–434, 2012.

[38] 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.

[39] 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.

[40] D. Appello et al. System-in-package testing: problems and solutions. *IEEE Design & Test of Computers*, 23(3):203–211, 2006.

[41] 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.

[42] 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.

[43] 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.

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

[45] 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.

[46] R. Kawahara, O. Nakayama, and T. Kurasawa. The effectiveness of iddq and high voltage stress for burn-in elimination [cmos production]. In *Digest of Papers 1996 IEEE International Workshop on IDDQ Testing*, pages 9–13, 1996.

[47] M. de Carvalho et al. An enhanced strategy for functional stress pattern generation for system-on-chip reliability characterization. In *MTV*, pages 29–34, 2010.

[48] Norman P. Jouppi et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, page 1–12, 2017.

[49] Sounil Biswas and Bruce Cory. An industrial study of system-level test. *IEEE Design Test of Computers*, 29(1):19–27, 2012.

[50] Peter Reichert. System Level Test. *Teradyne*.

[51] Dave Armstrong. Shifting Left = More Wafer Probe Real-World Implications. *SWTEST*, 2022.

[52] Hussam Amrouch, Victor M. van Santen, and Jörg Henkel. Interdependencies of degradation effects and their impact on computing. *IEEE Design & Test*, 34(3):59–67, June 2017.

[53] Bill Eklow. *On Microprocessor Reliability*. IEEE Computer Society, Computing Now, 1 edition, 2013.

[54] Parmod Aggarwal. Cost effective manufacturing test using mission mode tests. In *2007 IEEE International Test Conference*, pages 1–8, Oct 2007.

[55] F. X. Che, Jong-Kai Lin, K. Y. Au, and Xiaowu Zhang. Comprehensive study on reliability of chip-package interaction using cu pillar joint onto low k chip. In *2014 IEEE 16th Electronics Packaging Technology Conference (EPTC)*, pages 288–293, Dec 2014.

[56] Artur Jutman, Matteo Sonza Reorda, and Hans-Joachim Wunderlich. High quality system level test and diagnosis. In *2014 IEEE 23rd Asian Test Symposium*, pages 298–305, Nov 2014.

[57] F. Almeida et al. Effective screening of automotive socs by combining burn-in and system level test. In *IEEE International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2019.

[58] David P. Lerner, Benson Inkley, Shubhada H. Sahasrabudhe, Ethan Hansen, Luis D. Rojas Munoz, and Arjan van de Ven. Optimization of tests for managing silicon defects in data centers. In *2022 IEEE International Test Conference (ITC)*, pages 578–582, 2022.

[59] F. Angione, D. Appello, P. Bernardi, C. Bertani, G. Gallo, S. Littardi, G. Pollaccia, W. Ruggeri, M. Sonza Reorda, V. Tancorre, and R. Ugioli. A low-cost burn-in tester architecture to supply effective electrical stress. *IEEE Transactions on Computers*, pages 1–14, 2022.

[60] Bitty Jose and J. Samson Immanuel. Design of bist(built-in-self-test)embedded master-slave communication using spi protocol. In *2021 3rd International Conference on Signal Processing and Communication (ICPSC)*, pages 581–585, 2021.

[61] Shumit Saha, Md. Ashikur Rahman, and Amit Thakur. Design and implementation of a bist embedded high speed rs-422 utilized uart over fpga. In *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, pages 1–5, 2013.

[62] Andreas Apostolakis, Dimitris Gizopoulos, Mihalis Psarakis, Danilo Ravotto, and Matteo Sonza Reorda. Test program generation for communication peripherals in processor-based soc devices. *IEEE Design & Test of Computers*, 26(2):52–63, 2009.

[63] Felipe Augusto da Silva, Riccardo Cantoro, Said Hamdioui, Sandro Sartoni, Christian Sauer, and Matteo Sonza Reorda. A systematic method to generate effective stls for the in-field test of can bus controllers. *Electronics*, 11(16), 2022.

[64] P. Bernardi, L. Bolzani, A. Manzone, M. Osella, M. Violante, and M. Sonza Reorda. Software-based on-line test of communication peripherals in processor-based systems for automotive applications. In *Seventh International Workshop on Microprocessor Test and Verification (MTV'06)*, pages 3–8, 2006.

[65] 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.

[66] 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.

[67] 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.

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

[69] 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.

[70] 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.

[71] 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.

[72] 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.

[73] 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.

[74] 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.

[75] Denis Schwachhofer et al. Optimizing system-level test program generation via genetic programming. In *IEEE ETS*, pages 1–4, 2024.

[76] L. Degli Abbati et al. Industrial best practice: cases of study by automotive chip- makers. In *IEEE DFT*, pages 1–6, 2021.

[77] Karel DeVogeleer, Gerard Memmi, Pierre Jouvelot, and Fabien Coelho. Modeling the temperature bias of power consumption for nanometer-scale cpus in application processors. In *SAMOS XIV*, pages 172–180, 2014.

[78] Chen Guo et al. A survey of energy consumption measurement in embedded systems. *IEEE Access*, 9:60516–60530, 2021.

[79] Denis Schwachhofer et al. Automating greybox system-level test generation. In *IEEE ETS*, 2023.

[80] 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.

[81] M. Grosso, W. J. H. Perez, D. Ravotto, E. Sanchez, M. Sonza Reorda, and J. Velasco Medina. A software-based self-test methodology for system peripherals. In *2010 15th IEEE European Test Symposium*, pages 195–200, 2010.

[82] P. Bernardi et al. On the in-field testing of spare modules in automotive microprocessors. In *IEEE VLSI-SoC*, 2017.

[83] 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.

[84] 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.

[85] 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.

[86] Nicola di Gruttola Giardino, Francesco Angione, Paolo Bernardi, Tommaso Foscale, Claudia Bertani, and Vincenzo Tancorre. A flexible fpga-based test equipment for enabling out-of-production manufacturing test flow of digital systems. In *2024 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6, 2024.

[87] Xilinx. Zynq ultrascale+ mpsoc zcu104 evaluation kit. https://www.xilinx.com/products/boards-and-kits/zcu104.html/.

[88] Xilinx. Pynq. https://github.com/xilinx/pynq.

[89] STMicrolectronics. *RM0391 Reference manual*, pages 3925–4015. STMicrolectronics, 2024.

[90] Nicola di Gruttola Giardino et al. A flexible fpga-based test equipment for enabling out-of-production manufacturing test flow of digital systems. In *IEEE DFT*, pages 1–6, 2024.

[91] V. Turco et al. Uncovering hidden vulnerabilities in cnns through evolutionary-based image test libraries. In *IEEE DFT*, pages 1–6, 2023.

[92] P. Bernardi, M. Grosso, E. Sanchez, and O. Ballan. Fault grading of software-based self-test procedures for dependable automotive applications. In *IEEE DATE*, pages 1–2, March 2011.