Collecting diagnostic information through dichotomic search from Logic BIST of failing in-field automotive SoCs with delay faults

Collecting diagnostic information through dichotomic search from Logic BIST of failing in-field automotive SoCs with delay faults

Paolo Bernardi, Gabriele Filipponi, Matteo Sonza Reorda
Politecnico di Torino, Italy
name.surname at polito.it

Davide Appello, Claudia Bertani, Vincenzo Tancorre
ST Microelectronics, Italy
name.surname at st.com

***Abstract*—Embedded nano-electronic devices have spread in daily life over the past ten years. Chip and embedded system manufacturing has thus become more challenging in recent years.**

When safety-critical sectors like the automobile are considered, addressing system anomalies and faults is crucial. Therefore, it is necessary to develop and research innovative ways to maintain high reliability in safety-critical sectors despite the complexity of present Systems-on-Chip (SoCs).

In order to ensure high reliability, and be compliant with reliability standards, designers started to add additional circuitry to perform on-device tests. Built-In-Self-Test (BIST) is a technology that allows to conduct exhaustive tests within devices and, most importantly, without the need for external equipment. BIST can detect faults by outputting a signature at test end, which can be compared with a known value. Thus such known signatures are key, and in case of a signature mismatch it is not trivial to understand the root cause of the failure.

This paper proposes a methodology to find the first failing pattern which causes the BIST's signature to deviate and a way to collect good signatures from in-field devices, at key on/off, where BISTs are programmed and executed by the firmware at maximum frequency for an industrial case study produced by STMicroelectronics.

The transition delay fault model is the primary target for the described work.

***Index Terms*—Logic BIST, MISR signature, Dichotomic search.**

I. Introduction

Embedded nano-electronic devices have grown more common in people's daily lives during the previous decade. Integrated Circuits (IC) are already ubiquitous in everyday objects, and the features required by ICs are becoming increasingly demanding. As a result, chip and embedded systems have become exceedingly sophisticated, broadening the surface of undesirable behaviors and potential defects during regular system operation.

When it comes to safety-critical sectors like automotive, it is evident that resolving system anomalies and faults become vital, as it is required to spot malfunctioning devices before they can cause damage to the environment or people. Thus, testing becomes required throughout the entire device's life cycle, beginning with manufacture and continuing during normal mission mode.

Automatic Test Equipment (ATE), which incorporates a command line interface and the ability to save patterns generated by Automatic Test Pattern Generation (ATPG) for usage on the device's multiple units, is now often used to undertake test pattern operations. To accomplish this, the ATE must fulfill memory and channel requirements capable of storing patterns and meeting the number of interconnections necessary to apply the patterns to the device's inputs, which becomes exceedingly costly. Because the rising complexity of today's Systems-on-Chip leads to a rise in the number of patterns and pins required for the ATE, another testing technique was designed to deal with the enormity of modern systems. Manufacturers began to include into devices modules capable of self-testing, formally named Built-In Self Test (BIST), without the need for any external equipment [1]. BIST manages to execute exhaustive testing using structural approaches and output a signature, through a Multiple Input Signature Register (MISR), of the test result after being programmed with meaningful parameters such as the starting seed, amount of generated patterns, and working frequency. The signature is computed on each BIST cycle and compacted so that only the final one is visible after the test, and thus to check for abnormalities, it is necessary to compare it to a known one, generally referred to as golden signature.

In modern SoCs, a module named Self-Test-Control-Unit (STCU) exposes registers that enable software to schedule and execute onboard BISTs, thus opening for in-field testing. In the automotive field, assessing the reliability of the processor is crucial, hence, in addition to the classic pre-sale test that regards the manufacturing test flow [2] [3], the device is checked while it is in the field to avoid the possible occurrence of abnormal behavior. Such events usually are key on/off, meaning when the vehicle is turned on and off.

Despite its benefits, BIST requires known golden values for every potential parameter combination to be compared, and determining the underlying cause of a signature mismatch is difficult. The compacted signature loses information, and thus determining the pattern that caused the signature to deviate requires a significant amount of BIST execution and time.

For this reason, we propose a methodology for collecting Logic BIST signatures extracted from the in-field devices at key on-off, where BISTs are executed at maximum frequency and programmed by firmware, and a methodology for identifying the first failing pattern, which holds key information. Such information can be later exploited to perform a diagnosis of the device [4]. The focus is primarily on detecting transition delay faults, but stuck-at faults can also be addressed with an

979-8-3503-3277-3/23/$31.00 ©2023 IEEE

expedient described later.

The experimental setup is based on an external tool that sends commands to program, execute, and acquire results from BISTs in the case of the study, an SPC58 automotive SoC produced by STMicroelectronics. BIST experiments are executed by firmware with variable parameters: the operating system clock frequency and the generated pattern number (PCS). The whole work can be applied to a real-world scenario as it can be ported to devices in firmware.

The paper continues as follows. In Section II, the reader is provided with the basics regarding Built-In Self Testing, focusing on Logic BIST technology, its signature, the MISR, and in-field testing. Then, the dichotomic search algorithm is briefly discussed. Section III describes the proposed signature collection approach, dichotomic search applied to MISR, and how the memory is used to hold the needed data. In Section IV, the experimental setup and results are shown. Section V draws some conclusions.

II. Background

This section briefly describes the foundation of what this work stands on top. Firstly, BIST's architecture is discussed, which is the piece of hardware that enables in-field testing, and in the case that the logic circuitry is tested reports the result in the form of a compacted signature. The way such signature is computed enables the dichotomic search algorithm to process it, and thus is discussed secondly.

A. Built-In Self Test

Testing has gotten more challenging with the emergence of Very-Large-Scale-Integration (VLSI) in the previous few decades, as a result test creation and testing cannot keep up with the rising complexity of devices. When considering safety-critical sectors such as automotive, it is straightforward how much more complicated the situation is since tight standards regulate reliability assessment. With increasing operating frequencies, pin-outs, and prices, standard test equipment (ATE) becomes unfeasible. To resolve these issues, manufacturers began to incorporate auxiliary modules within devices capable of self-testing from the inside without the use of external equipment, hence enhancing reliability, named Built-In-Self-Test (BIST).

The BIST approach initially addressed at [1], is frequently utilized since it substitutes more sophisticated and costly ATEs. Furthermore, BIST may evaluate complex devices with highly interconnected components efficiently. Moreover, BIST enables the user to test the circuit without device makers exposing any information about its design, although it is necessary for them to give the user instructions on how to run the test and understand the findings.

The Unit-Under-Test (UUT) can be programmed to operate in functional or test modes. When in test mode, the device's Primary Inputs (PI) and Outputs (PO) are disconnected, and the BIST controller begins feeding patterns produced by a Test-Pattern-Generator (TPG) into the device inputs. The outputs are attached to an Output-Data-Evaluator (ODE) that can detect if the intended signature matches the resulting one. The ODE, which is often implemented using a Multiple-Input-Signature Register (MISR), is updated on each BIST cycle, resulting in a signature that depends not only on the current test outputs but also on previous tests outcome. Thus there is no need to compare each output with what would be the expected one: it is enough to know the correct signature and verify that it matches the one produced by the ODE.

BIST is called Logic BIST when the circuits being tested are logic modules (LBIST). In general, its implementation consists of the following modules:

- LBIST Controller which selects between test or normal functional mode and handles the configuration of BIST;
- Pseudo Random Number Generator (PRNG) generates patterns to apply to Unit Under Test (UUT);
- Multiple Input Signature Register (MISR) collects the output signature obtained after the run of the BIST. The signature is updated on every BIST cycle and depends on previous and current test results.

The LBIST Controller can be further programmed with meaningful testing parameters such as:

1) Pattern-Count-Stop (PCS): it instructs the TPG on how many patterns need to be generated;
2) Initial seed of the PRPG for the BIST structure.

In order to detect failures in devices, it is possible to exploit On-Chip BISTs. As described in [5], and [6], On-Chip LBISTs run results can be collected for further usage in the field of Silicon Life-Cycle Management.

B. In-field testing

Device manufacturing is not the final stage in the testing cycle in safety-critical sectors. Devices must be checked during their entire life cycle to retain their reliability. Because of this, Silicon Life-cycle Management (SLM) has emerged as one of the fields in which industries have concentrated over the past few decades. Its primary goal is to gather as much valuable data as possible and analyze it during its operational lifetime to learn helpful information that will help to enhance device operations [7].

The stage of the SLM, known as in-field testing, ensures the reliability of SoC devices in their operating state. Meeting the reliability requirements of numerous application domains is becoming increasingly important. Rules and standards, such as ISO 26262 [8], may explicitly mandate that fault coverage estimates be provided for various degrees of safety, especially when the system is used in specific safety-critical sectors. In-field testing may be done during the device's key-on/off events or while the application is running, depending on the scope of the application [9].

C. Dichotomic Search Algorithm

Dycothomic search is a well-known and researched Computer Science algorithm [10] that finds an element in a set by recursively choosing between two possible selections (dichotomies) at each stage. A dichotomic search may be thought of as following the edges of an implicit binary tree structure until it hits a leaf, which can be seen as a goal or final state. Given $\mathbb{K}$, the set of all possible objects of cardinality k, and a function $f : \mathbb{K} \longrightarrow \{0, 1\}$ which takes an object of the set returns a boolean value of 0 or 1. Elements in the sets must satisfy the rule described at Equation 1.

$$\forall x, y \in \mathbb{K},\ x < y,\ f(x) = 0 \Longrightarrow f(y) = 0 \tag{1}$$

Given the prerequisites, finding a specific element within the set is possible without the need to test every member in the set but by applying dichotomic search. The algorithm is described in Listing 1:

```
low = 0, high = k-1
while (high > low):
    current = (high + low) >> 1
    if f(current) == true: /* range from bottom */
        low  = current - 1
    else:                  /* range from top    */
        high = current - 1
```

Listing 1. Pseudo code of dichotomy search algorithm.

The complexity of the algorithm relies on $O(log2(k))$ as the total number of tested objects is logarithmic relative to the cardinality of the given set.

In the proposed work dichotomy search is used for identifying the first failing BIST pattern.

III. The Proposed Approach

The proposed approach seeks to provide a mechanism for finding the first failing pattern when an abnormal Logic BIST's signature, if compared to a known golden one, in malfunctioning devices is reported. The goal is to propose a mechanism to partially execute BIST runs to identify the last BIST cycle executed correctly or the first failing BIST cycle, thus adding meaningful information to the reported signature and enabling better diagnosis. Moreover, if a golden signature is missing, a way to collect it safely and restart the flow is presented.

A mismatched Logic BIST's signature at key on/off, means a failure in the UUT's logic during field operations and thus needs further investigation to understand the root of the failure.

The flow, shown in Figure 1, starts with the vehicle executing Logic BIST at key on/off (1), which at the test end a signature is reported (2). If the signature is the expected one, the vehicle can be safely driven and the loop repeats at the next key on/off. Otherwise, if the signature mismatches, it indicates that a pattern failed thus the vehicle is not safe and further investigation is needed. Since the signature is computed by compressing previous ones with new data, there are two possibilities:

1) The last applied pattern made the UUT fail, which has a probability of $\frac{1}{\#tested\ patterns}$;
2) A pattern applied prior to the last one made the UUT fail, propagating a wrong signature until the end.

In the latter case, multiple patterns can fail the UUT, resulting in multiple signature corruption. Thus, the signature reported at (2) lacks information, as it is impossible to assess which pattern made the UUT fail, hence enabling limited logic diagnosis investigations. In this scenario, the only possibility for the diagnosis is to use the final signature.

Our approach enriches information about the signature by finding the first failing pattern through dichotomic search and saving it into the flash of the device (3, 4). This opens up possible future logic diagnosis, which can reduce the candidates list through information given by the first failing pattern and thus be used to identify faults (5).

Tests are conducted at the maximum working frequency to put more strain on the UUT and to better focus on delay misbehaviors. Nevertheless, the proposed method must control the speed to collect good BIST signatures at low frequencies.

A. Test flow

In the event of a Logic BIST's signature mismatch, it is not trivial to identify the first failing pattern. Since Logic BIST's signature is computed so that the previous data is used to generate a new signature, thus propagating until the end, tests need to be executed for every pattern until the searched one is found.

In this work dichotomic search is used to obtain the first failing pattern in a failed device, which allows reducing the number of required tests in the domain of $\log_2(\#total\ patterns)$.

Logic BIST's signature satisfies all the dichotomic search algorithm requirements previously described, subsection II-C, and thus can be applied:

- Signature depends on the past ones, and thus a deviation will propagate until the end of the test;
- Cardinality is the number of patterns generated, which can be programmed through Logic BIST Controller;

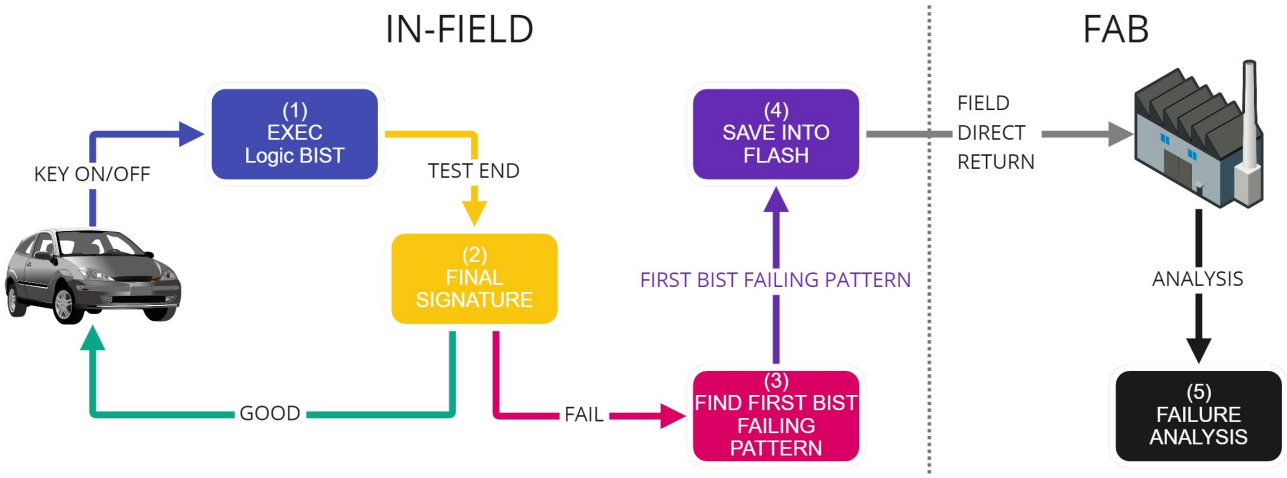

Fig. 1. Flow of the proposed approach.

- The function needed, which takes a signature and outputs a boolean value, is just the comparison between the expected signature and the obtained one.

The test flow, pictured in Figure 2, is based on the capability of BISTs to be programmed in steps, and in order to better target delay faults, every BIST execution is done at maximum working frequency.

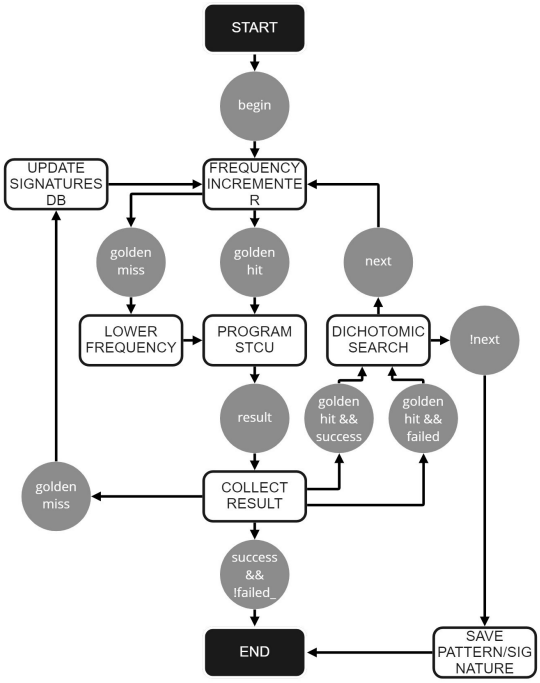

Fig. 2. Simplified Finite State Machine (FSM) version of the logic to be implemented for the proposed work.

The flow follows:

1) Start by executing Logic BISTs from firmware with the maximum number of patterns $\#total$ at the maximum frequency;
2) If the signature matches the expected one since the maximum number of patterns have been generated, every pattern generated between $0x0$ and $\#total\ patterns$ can be marked as green, thus the test ends, and the device is marked as good, as shown in Figure 3.

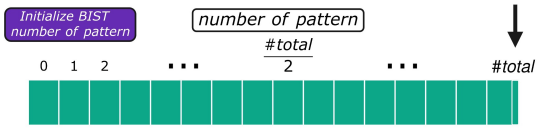

Fig. 3. In case the signature is good, test ends immediately.

3) If the signature mismatches, the dichotomic algorithm starts by bisecting the range of patterns by programming Logic BIST with half of the total patterns $\frac{\#total\ patterns}{2}$. If the signature is wrong again, every signature generated with a number of patterns between $\frac{\#total\ patterns}{2}$ and $\#total\ pattern$ is wrong and thus can be marked as red without executing Logic BIST.
4) The flow continues by iterating the same logic as described in Listing 1. It ends when no further dichotomies can be distinguished, meaning, in this case, a single number of pattern has been reached. Figure 4 describes it.

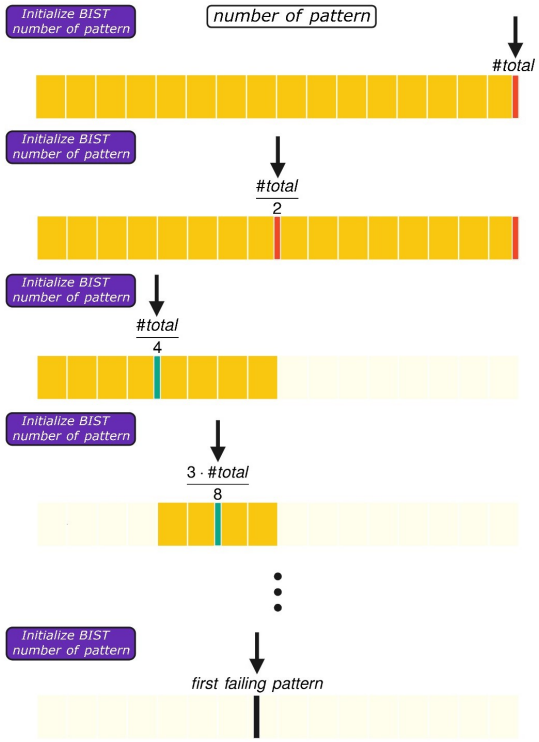

Fig. 4. When the signature is wrong, dichotomic search can be applied. The algorithm ends when two distinct dichotomies cannot be distinguished.

Golden signatures are loaded, if available, from a database saved into FLASH memory. In the event of a signature database miss a mechanism to extract the required signature safely is executed. The methodology to capture a golden signature and the flash memory layout is explained in the following sections.

B. Golden signature computation

Logic BIST requires golden signatures for the comparison to detect anomalies, as described in subsection II-A. As reported in Figure 2, tests are conducted at high frequency to better target delay faults, thus most likely corrupting the final signature. In the event a golden signature is missing from FLASH, it is possible to store it with the following steps safely:

- Current test is suspended;
- Frequency lowered to a safer working BIST range;
- At self-test end the resulting signature is collected and saved into FLASH and marked as golden;
- Initial test is finally restarted.

The frequency switch has the duty to lower the risks of capturing into the signature a possible failure, thus corrupting the result and later the comparison for the test.

This work focuses primarily on delay faults, yet dichotomy search can still be applied to BIST's signature in case stuck-at

faults are addressed: the mechanism previously proposed to capture golden signatures from an in-field device needs to be replaced by a pre-filled database of signatures because a stuck-at will corrupt the signature no matter the working frequency.

However it is essential to underline that by observing the bath curve [11] for device failure/aging, which states that faults are more likely to occur at device manufacturing and after heavy usage, and following Burn-In [3] to highlight production failures, it is logic to assume that a failure shall first manifest as a delay and then transform to a stuck-at. Nevertheless, this work has the objective is to identify misbehaviours as soon as they appear, hence delay coverage is a primary objective.

C. Data storage

The scheme of the memory layout, shown in Figure 5, is divided into two regions: the *golden region*, which holds golden signatures, and the *failure region* in which the first failing pattern obtained through dichotomy search is saved. The base structure of the regions comprehends:

- The index representing the number of patterns for the test;
- The resulting Logic BIST's signature.

Depending on the fault model being considered, the footprint changes.

For delay fault modeling, at device shipping, the structure is empty and filled during device operation. Only the first entry of the *golden region* is occupied with good devices, as no anomalies have been detected. If Logic BIST detects a fault, then all entries will be filled by dichotomy search, and the entry in the *failure region* will hold the first failing pattern or the last good. The number of entries depends on how many dichotomies the number of patterns can be divided $\#entries = log_2(\#total\ patterns) + 1$.

Meanwhile, for stuck-at modeling, the *golden region* must be pre-filled as no computation on golden signatures is done, hence the number of entries is $\#entries = \#total\ patterns + 1$. Moreover, the index for the golden signatures is omitted, as can derived as an offset from a FLASH base address.

Independently on the fault model being considered the scheme must be replicated for every BIST partition.

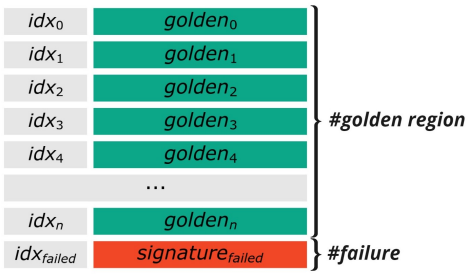

Fig. 5. Scheme of the memory layout for the approach.

IV. An Industrial Case of Study

The case study used for validating the methodology is an industrial automotive System-on-Chip belonging to the SPC58 family produced by STMicroelectronics. The Design under Test (DUT) has the following characteristics:

- 20 million gates;
- About 700,000 flip-flops;
- ASIL-D compliant;
- Multi-core architecture;
- Powerpc-VLE compliant;
- Double cascade PLL system;
- 7 LBIST partitions;
- 92 MBIST partitions.

A. FLASH footprint

Each of the 7 Logic BIST partitions onboard offers 16 bits of programmable PCS, thus Logic BISTs are initially programmed with a starting PCS of $0xFFFF$, which is then halved/adjusted by the dichotomic search on every iteration.

Every entry in the scheme described in subsection III-C is 80 bits wide: 16 bits for the index, 64 bits for the signature. Depending on the fault model used, the footprint of the FLASH changes. For delay modeling the whole structure occupies $7 \cdot 80 \cdot (\log_2(0xFFFF) + 1) = 9520bits$ or $\sim 9kb$. Meanwhile for stuck-at model, as the index can be omitted for the *golden region*, the structure spikes to $7 \cdot 64 \cdot (0xFFFF + 1) + 16 = 29360144bits$ or $\sim 29Mb$.

Please notice that the methodology is very effective to collect information about delay faults detection, which is the primary objective of the paper. In case stuck-at faulty behaviours needs to be considered, then a larger amount of memory need to be assigned.

B. Code size and complexity

The presented work can be ported in firmware to devices. Table I shows incremental code sizes starting from the base system, adding the library needed to handle BIST from firmware, and finally, implementing the proposed approach. The logic of the work can be written in a high-level language, like C, within $\sim$ 100 lines of code, and its complexity depends on the total number of patterns as described by $O(\log_2(\#total\ patterns))$.

	System	+ BIST library	+ Implementation
Code size	127,568	136,304	136,912
Delta	-	8,736	608

Table I. Incremental code size of the firmware implementing the proposed approach.

C. Signature collection time

In the case of study, Logic BISTs programmed to test concurrently the maximum number of patterns $0xFFFF$ take up to $\sim$ 100ms to complete the self-test procedure, and at self-test completion, the device asserts a functional reset of itself resetting the cores but the STCU, which enables signature collection. Thus at key on/off, if the device is not faulty, only one Logic BIST execution is necessary, hence, the duration of the procedure is100ms. If the initial signature mismatches, dichotomy search starts by executing Logic BISTs for a total of 16 times with decreasing/raising number of patterns. At worst, if the number of patterns that the dichotomy search finishes at is $0xFFFF - 1$, the time needed for the whole execution is $100 + \sum_{i=1}^{15} \frac{100 \cdot (2^i - 1)}{2^i} \sim 1500$ms as the selected dichotomy is always the upper one. While at best, the pattern to be found is $0x1$, meaning that the lower dichotomy is always

chosen ,then the time required is $\sum_{i=0}^{15} \frac{100}{2^i} \sim 200$ms. If golden signatures are not present in the database, the execution time will double, as for every execution a golden signature needs to be extracted. Presented measurements considers BIST time execution only, as FLASH erase and program operation, even if done on every iteration, take respectively 0.0192ms and 0.00475ms, thus begin negligible if compared to BIST execution time.

D. Experimental setup

To validate the proposed approach, an experimental setup, shown in Figure 6, was assembled, which includes:

1) The actual board with the SoC installed;
2) A programmable voltage regulator which powers the SoC's core;
3) An external tool that manages BIST parameters and handles the serial communication with the board and the voltage regulator.

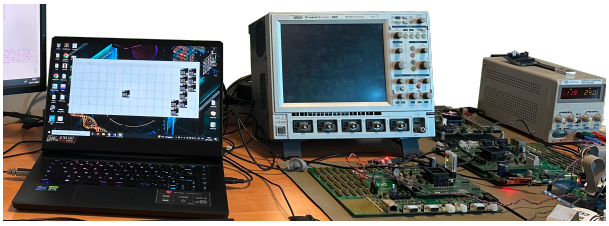

Fig. 6. Full experimental setup.

BISTs are programmed and executed by firmware during regular device operation. In order to do so, a Command Line Interface (CLI) is programmed into FLASH of the SoC to expose utilities for BIST operations.

The external software tool is the primary component that coordinates Logic BIST experiments, implements dichotomy search for the presented approach, and handles the communication with the board and the power supply. It is written in C++17 for the back-end and exploits Microsoft Foundation Classes (MFC) for the Graphical User Interface (GUI).

The programmable power supply manages the voltage to the device's core, and thus it is possible to raise/lower the voltage of the core, mirroring a real in-field scenario with voltage drops.

E. Experimental Results

Early results on the experimental setup show that the proposed approach indeed works, and within a batch of devices marked as failed with delay faults, three failures were observed through the proposed approach, and the respective failing pattern was extracted among the corresponding signature.

The data collection of Logic BISTs signatures in the experimental setup was done at the core reference voltage with raising frequencies to find a better range to target delay faults. The number of frequencies used depends upon a parameter named Δ, which indicates the quantity by which frequency is increased each time, equal to $10MHz$.

Results are arranged in a matrix that reports on the ordinate the device type, good or failed, and on the coordinate, the frequency used to conduct tests. Every cell in the matrix is colorized to visualize results better:

- GREEN: Logic BISTs reported a good signature at maximum programmed number of patterns;
- RED: Logic BISTs reported a wrong signature for every programmed PCS by the dichotomic search;
- PURPLE: Dichotomic search found the first failing pattern.

Devices marked as good reported no failure. Meanwhile, three devices marked as failed reported failures at $200MHz$ domain, and dichotomic search managed to retrieve the first failing pattern for those tests.

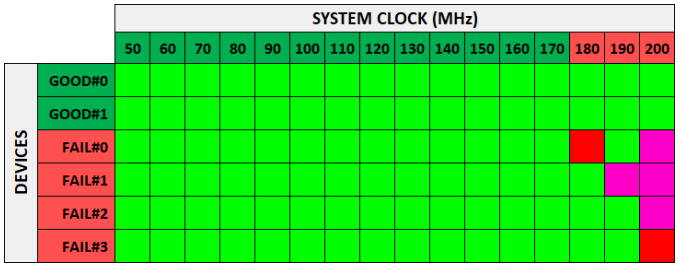

Fig. 7. Matrix of the results captured from a batch of devices.

V. Conclusion

A methodology is proposed to collect Logic BIST signatures from in-field devices at key-on/off. In particular, a way to safely collect golden signatures and find the first failing pattern when the initial signature mismatches through dichotomy search. This data can be exploited to improve the diagnosis of failures as test results are saved into the FLASH memory and can be accessed anytime by manufacturers.

Future work will explore the programmable voltage regulator to further stress the UUT, thus possibly capturing faults not detected in normal device conditions. Running tests with different pairs of frequencies/voltage are intended to better reflect real-world events in which the vehicle may be involved.

References

[1] N. Benowitz *et al.*, "An advanced fault isolation system for digital logic," *IEEE Transactions on Computers*, vol. C-24, no. 5, pp. 489–497, 1975.

[2] I. Polian *et al.*, "Exploring the mysteries of system-level test," in *2020 IEEE 29th Asian Test Symposium (ATS)*, 2020, pp. 1–6.

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

[4] W.-T. Cheng, M. Sharma, T. Rinderknecht, L. Lai, and C. Hill, "Signature based diagnosis for logic bist," in *2007 IEEE International Test Conference*, 2007, pp. 1–9.

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

[6] G. Filipponi, G. Iaria, M. S. 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)*, 2022, pp. 646–649.

[7] R. Kashyap, "Silicon lifecycle management (slm) with in-chip monitoring," in *2021 IEEE International Reliability Physics Symposium (IRPS)*, 2021, pp. 1–4.

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

[9] J. 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)*, 2014, pp. 1–2.

[10] D. Knuth, "Sorting and searching: The art of computer programming, vol. iii," 1973.

[11] G. Klutke, P. Kiessler, and M. Wortman, "A critical look at the bathtub curve," *IEEE Transactions on Reliability*, vol. 52, no. 1, pp. 125–129, 2003.