Enhancing Logic Diagnosis of field returns through Logic BIST in Automotive SoCs

Enhancing Logic Diagnosis of field returns through Logic BIST in Automotive SoCs

Paolo Bernardi, Gabriele Filipponi, Giusy Iaria
Politecnico di Torino, Italy
name.surname@polito.it

Claudia Bertani, Vincenzo Tancorre
STMicroelectronics, Italy
name.surname@st.com

***Abstract*—Electronic devices intended for safety-critical industries, such as automotive, undergo extensive testing to ensure their reliability and compliance with safety standards. It is critical to constantly monitor, even in mission mode, that no failure due to wear and tear can compromise their reliability. Using self-testing circuitry such as Logic Built-In Self-Test (LBIST), it is possible to implement key-on and key-off tests to monitor the reliability status of the device also during its operational lifetime.**

This paper proposes a methodology based on LBIST data collection launched at key-on and key-off via CPU to improve the logic diagnosis of devices returned from the field. The proposed methodology exploits the reprogramming of pattern generation and signature compaction registers to retrieve information about all the failing and successful patterns instead of the sole generic fail information. As such, the proposal achieves better diagnostic effectiveness than other state-of-the-art methodologies.

Experimental results show that the diagnosis has a significant increase in accuracy for an industrial automotive device manufactured by STMicroelectronics.

***Index Terms*—Functional safety; Logic Built-In Self-Test (LBIST); Silicon Life-cycle Management (SLM); Logic Diagnosis;**

I. Introduction

The increasing complexity of electronic devices in safety-critical applications, such as the Automotive field, require rigorous testing phases throughout their entire life-cycle [1]–[3].

Although production testing is critical to ensure the initial quality of semiconductor devices, it does not guarantee defect-free operation throughout the products' lifetime. Despite the numerous tests to which devices are subjected, manufacturers face a persistent and significant challenge known as "No-Trouble-Found" (NTF) [4]. This term refers to cases in which devices returned to manufacturers from the field because of faulty behaviors have no defects when retested in fab.

Devices are subject to aging, environmental stress factors such as temperature fluctuations, and electromagnetic interference, which can cause failures during mission mode. As a result, monitoring and field testing have become indispensable components of Silicon Life-cycle Management (SLM) [5], [6]. Logic Built-In Self-Test (LBIST) is generally used to perform periodic testing at key-on/key-off, as it allows on-chip testing without needing external equipment. The standard LBIST scheme comprises on-the-fly pattern generation through an LFSR and an output compacted signature recorded in a Multiple Input Signature Register (MISR). Once one signature mismatches from the golden reference, all the subsequent ones mismatch as well. As such, LBIST approaches are typically designed to run a pre-defined number of patterns and signal possible failures through the final signature output.

In the state-of-the-art, various methodologies [7]–[14] have been proposed for diagnosing failures caught by LBIST. In [10], the authors suggest strategies that combine the in-field self-test execution and the diagnosis process. Their applicability relies on custom LBIST architectures even if they reach valuable diagnostic accuracy. Additionally, more recently, also ref. [15] proposed a specific LBIST scheme to enhance the standard ATPG diagnosis process by exploiting the on-chip pattern generation method introduced in [16]. There are, then, a few examples [13], [14] proposing logic diagnosis strategies that do not require any specific diagnostic feature and are solely based on the failing LBIST signatures analysis. Similarly to the latter, [17], [18] proposes a logic diagnosis strategy that relies entirely on the first failing signature. The advantage of such a method is that it requires very little space to store the information, and it can still give precious feedback to the manufacturers once it is returned to them. The method exploits dichotomic search to find the first failing pattern, and thanks to this, the diagnosis results are better than those of other state-of-the-art methodologies because it can remove faults covered by successful patterns. Even with this, the diagnostic accuracy could not be optimal since it strictly depends on the index of the first failing pattern.

In this context, this paper presents an enhanced CPU-launched LBIST methodology, extending the concepts published in [18] and addressing the mentioned limitations through two key proposals:

1) Collecting all diagnostic information produced by failing LBIST patterns, by **reseeding** the LFSR and resetting the MISR through CPU.
2) Performing logic diagnosis considering the **comprehensive diagnostic information** about fail and success patterns collected in-field.

The proposed methodology is validated through an industrial case study produced by STMicroelectronics. The experimental results show how the average diagnostic resolution coming from the enhanced logic diagnosis proposal signifi-

This work was supported by the Italian Ministry of Research and University under the DM1061.

978-1-6654-7763-5/25/$31.00 ©2025 IEEE

cantly improved, reaching more than a significant accuracy increase. The computed results are also compared to other state-of-the-art methodologies, showing the efficiency of the proposal. The work has an emphasis on transition delay (TRN) faults to monitor aging problems [5], [6].

The remainder of the paper is organized as follows. Section II gives the reader an essential technical background and describes related state-of-the-art work. Section III details the proposed methodology. Section IV reports the experimental results for the industrial automotive devices produced by STMicroelectronics, also providing a comparative analysis with other state-of-the-art methodologies. Finally, Section V draws some conclusions.

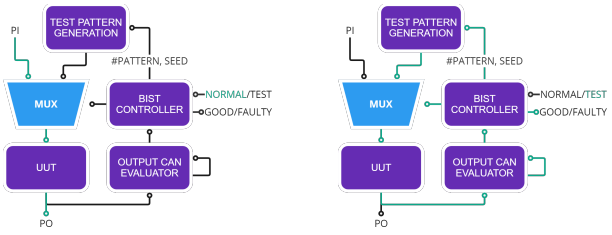
II. Background

This section details the basic notion and the related works about the use of LBIST to perform logic diagnosis.

A. Logic Built-In Self-Test

BIST technology enhances the devices' reliability. It offers a more cost-effective alternative to the more elaborate and expensive ATEs. BIST requires minimal additional hardware by repurposing unused scan chains post-manufacturing, enabling circuit testing without exposing sensitive design information to users.

Nonetheless, manufacturers must provide guidelines for executing the tests and interpreting their results. In BIST methodology, the Unit Under Test (UUT) can operate in functional or test mode. In test mode, the device's Primary Inputs (PI) and Primary Outputs (PO) are disconnected, and a BIST controller supplies test patterns generated by a Test-Pattern-Generator (TPG) to the inputs. The outputs are assessed by an Output-Data-Evaluator (ODE), which checks if the produced signature matches the expected one. A MISR typically updates the signature after each BIST cycle based on current and previous test outcomes, eliminating the need to compare each output with expected results and verifying that the signature's accuracy suffices. The Self-Testing Using MISR and Parallel SRSG (Shift register sequence generator) (STUMPS) architecture, shown in Figure 1, exemplifies this standard procedure.



(a) (b)

Fig. 1: Standard STUMPS BIST architecture composed of its major functional blocks [19], [20].(**a**) Normal functional paths are enabled, PIs reach the UUT. (**b**) Self-test paths are enabled, the BIST controller feeds test vectors to the UUT and captures its response in the ODE. Figure from [18].

When the self-test targets the combinational part of the device, the BIST is called Logic BIST (LBIST) [19]. LBIST can be executed and programmed externally by means of a test access port or, when an interface is made available by designers, through firmware. In the latter case LBIST execution is destructive to the current running software as a cleanup is necessary after its execution. Manufacturers usually suggests a restart of the SoC by performing a *functional reset* of the device, which is applied by the LBIST module.

B. Silicon Life-Cycle Management

In areas where safety is paramount, the end of the manufacturing process does not mean the end of testing procedures. Instead, it is critical to maintain continuous surveillance of the device throughout its life cycle to ensure continued reliability. Consequently, Silicon Life-Cycle Management (SLM) has become a crucial industry goal in recent decades [5].

SLM covers all phases of a device's life, from its initial design to its deployment in consumer systems. The main goal of SLM is to collect and interpret critical data during the operational phase of a device, thereby obtaining information that can be leveraged to improve performance and reliability.

C. State-of-the-art about logic diagnosis through LBIST

Diagnosing failures detected by LBIST is a topic discussed in the literature, and many works have been proposed, such as [7]–[14]. In [10], strategies integrating in-field self-test execution with the diagnosis process are suggested. This approach involves custom LBIST architectures that combine Built-In Self-Test with Built-In Self-Diagnosis (BISD) capabilities to improve fault localization during in-field testing. However, the requirement for custom architectures limits applicability due to cost and complexity constraints.

Reference [15] proposes an LBIST scheme enhancing the standard Automatic Test Pattern Generation (ATPG) diagnosis process by generating fully deterministic test patterns within the LBIST environment, as introduced in [16]. This allows for precise fault detection using traditional ATPG techniques but may require significant modifications to the existing LBIST infrastructure.

Methodologies focusing on logic diagnosis only exploiting failing signatures have been proposed in [13] and [14]. While attractive due to minimal resource requirements, they often offer limited diagnostic resolution because they rely on compressed signature data, which may need more detailed information about individual faults.

Similarly, [18] introduces a logic diagnosis strategy relying entirely on the first failing signature captured during LBIST execution. The main benefit is reduced storage requirements, as only the first failing signature is stored in the device's nonvolatile memory. However, because LBIST execution stops at the first failing pattern, subsequent test responses—which could provide additional insights into latent or intermittent faults—are not captured, limiting diagnostic effectiveness.

In summary, despite significant progress in LBIST logic diagnosis methods, existing methodologies often need to work

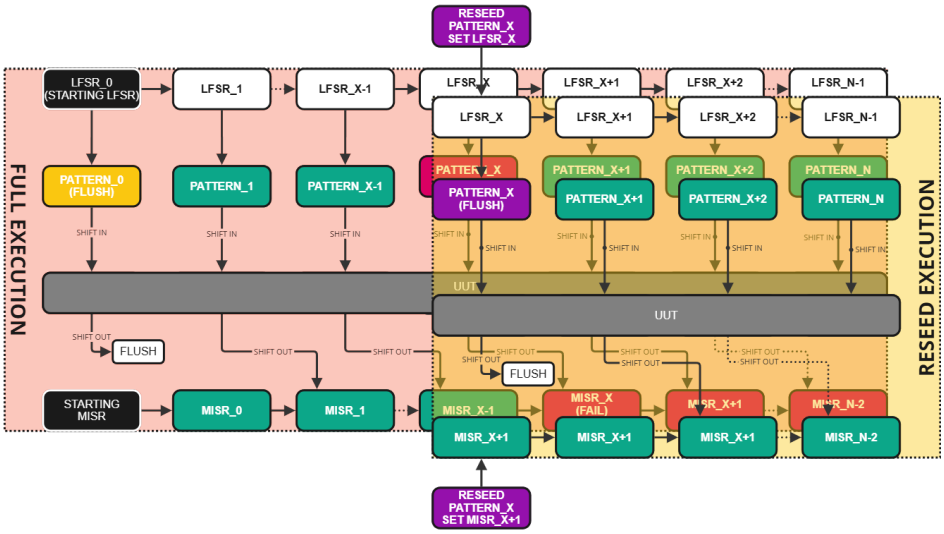


Fig. 2: LBIST execution started from the last failing X by means of reseeding: its interface is programmed with the LFSR at step $X-1$ and MISR at step X. This result in the final signature being the expected one.

on distinguishing between diagnostic resolution, hardware complexity, and applicability to in-field testing. Approaches requiring custom LBIST architectures or extensive hardware modifications may need to be more practical for general adoption. At the same time, methods minimizing resource requirements tend to offer limited diagnostic capabilities. Therefore, there is a need for a solution that enhances in-field logic diagnosis without needing architectural changes or incurring significant overhead.

III. The Proposed Approach

This paper presents a CPU-launched LBIST methodology that extends the study proposed in [18] to address the issue of NTF for field returns. In particular, it is based on an in-field data collection of diagnostic information and a subsequent logic diagnosis phase that leverages the collected data to help the failure analysis in case of field returns.

The in-field data collection of LBIST failures usually refers to the general information of failure or the first failing pattern, as proposed in [18]. This work enhances such a collection by proposing a novel methodology that leverages the programmable feature of the LBIST module by reseeding the LFSR seed and resetting the MISR. By doing so, the diagnostic information collected results are much more valuable since they contain failing and successful data from more patterns.

Thus, the key contributions of the work are the following:

1) **Comprehensive diagnostic data collection**: reseeding the LFSR and resetting the MISR through CPU to enlarge the diagnostic information collected in-field.
2) **Enhanced logic diagnosis**: exploiting a tree-based fault dictionary [21] built upon the whole pattern set applied by the LBIST.

A. Comprehensive diagnostic data collection

The execution of LBIST presents notable challenges, particularly concerning signature compaction. When a failure pattern emerges, it produces an erroneous response within the MISR that propagates until the self-test is completed. To trace back to the specific index of the pattern responsible for the error, techniques such as those outlined in [22] can be employed. This information is key, potentially guiding the identification of a candidate set as discussed in [18]. However, a significant limitation arises when the failing pattern's index resides among the initial patterns; in such cases, the resulting candidate set becomes excessively broad, diminishing the effectiveness of the logic diagnosis.

As described in Subsection II-A, LBIST can be configured with an initial LFSR seed and MISR signature value. This capability allows for programming specific starting points within the LBIST, thus bypassing previously failed patterns. Specifically, supposing the pattern that is failing is at index X, in order to start the LBIST from this pattern and skip it, it is necessary to program the LBIST with the value of the LFSR at pattern X and MISR $X+1$. The dis-alignment is due to the first pattern used as a *flush* pattern. This is done to unload scan chains with unknown initial values. Moreover, while a pattern is being used this way, the MISR is disconnected; hence, it does not update. This is shown in Figure 2.

The proposed algorithm used to reconfigure the LBIST parameters is shown in Algorithm 2. The FLASH memory is used to store both the data structure for the LBIST parameters, used as temporary space between functional reset, shown in Algorithm 1, and failure LBIST information. The Finite-State-Machine implemented by the algorithm is shown in Figure 3. The flow begins when the vehicle is turned on or off. The process first checks whether LBISTs have been executed. If not, it initializes the parameters and reattempts LBIST execution. If LBIST has been executed, the system evaluates whether the first run was successful. If no failure data exists, the device is deemed good, and the process can restart at the next key-on and key-off. However, if it fails,

the system starts by searching for any failing patterns. If a failing pattern is identified, the failure data is saved in flash memory; otherwise, parameters are adjusted using a binary search, and the LBIST is restarted from the following pattern, as described in Figure 2. The system then checks if the pattern set is exhausted or a stop condition has been reached. If so, the process ends. For every iteration, an initial LBIST run at minimum capture frequency is performed to extract the golden reference signature. After that, the flow is resumed, and LBIST is started at maximum capture frequency to better target transition delay faults.

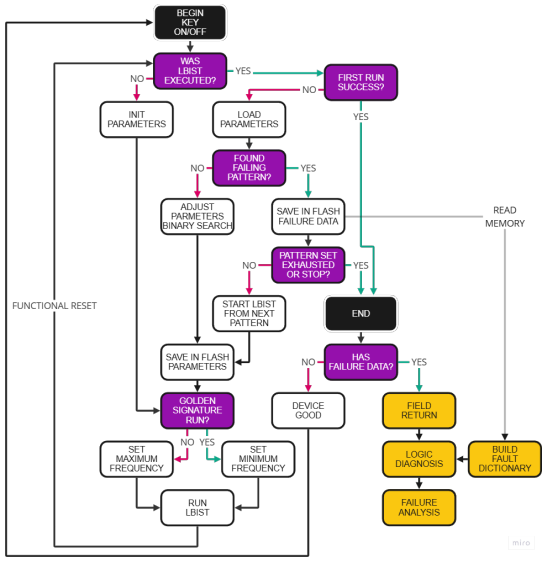


Fig. 3: FSM of the proposed approach.

Algorithm 1 Structure of the LBIST DATA.

```
1:  struct LBIST_DATA {
2:      gold;                          ▷ Golden run flag
3:      golden;                        ▷ Current golden signature
4:      spn;                           ▷ Starting pattern number
5:      epn;                           ▷ Ending pattern number
6:      depth;                         ▷ Binary search current depth
7:      fsig;                          ▷ Last erroneous signature
8:      gsig;                          ▷ Last golden signature
9:      seed;                          ▷ Starting LFSR seed
10:     misr;                          ▷ Starting MISR
11:     nfail;                         ▷ Number of saved failed signatures
12: }
```

Algorithm 2 Pseudo code of the proposed approach.

Require: $SEED$, starting value of the LBIST's LFSR.
Require: $MISR$, starting value of the LBIST's MISR.
Require: $MAXP$, maximum number of patterns.
Require: $MINF$, minimum LBIST frequency parameters.
Require: $MAXF$, maximum LBIST frequency parameters.
Require: **STP**, maximum number of failures to be saved.
Require: $data$, LBIST_DATA structure as defined in 1.

if $lbist\_was\_executed()$ **then**
  $flash\_load\_bist(\&data)$
  if not $data.gold$ **then**
    $data.golden \leftarrow bist\_get\_sig()$
  else
    $rsig \leftarrow bist\_get\_sig()$
    if $rsig = data.golden$ **and** $data.spn = 0$ **and** $data.epn = MAXP$ **then**
      return $BIST\_OK$
    else
      if $data.spn \neq data.epn$ **then**
        if $data.golden = rsig$ **then**
          $data.spn \leftarrow data.epn$
          $data.epn \leftarrow data.epn + \frac{MAXP}{2^{data.depth}}$
        else
          $data.epn \leftarrow data.epn - \frac{MAXP}{2^{data.depth}}$
          $data.gsig \leftarrow data.golden$
          $data.fsig \leftarrow rsig$
        end if
        $data.depth \leftarrow data.depth + 1$
      else
        if $data.golden = rsig$ **then**
          $flash\_save\_fail(data.spn - 1, data.gsig], data.fsig)$
        else
          $flash\_save\_fail(data.spn, data.golden, rsig)$
        end if
        $data.spn \leftarrow data.spn + 1$
        $data.nfail \leftarrow data.nfail + 1$
        if $data.spn = MAXP$ **or** $data.nfail \geq STP$ **then**
          return $BIST\_FAIL$
        end if
        $data.seed \leftarrow get\_lfsr(data.spn - 1)$
        $data.misr \leftarrow data.golden$
        $data.epn \leftarrow MAXP$
      end if
    end if
  end if
else
  $data.gold \leftarrow True$
  $data.spn \leftarrow 0$
  $data.epn \leftarrow MAXP$
  $data.depth \leftarrow 1$
  $data.seed \leftarrow= SEED$
  $data.misr \leftarrow MISR$
end if
if $data.gold$ **then**
  $set\_capture\_frequency(MINF)$
else
  $set\_capture\_frequency(MAXF)$
end if
$data.gold \leftarrow$ **not** $data.gold$
$flash\_save\_data(\&data)$
$run\_lbist(data.spn, data.epn, data.seed, data.misr)$

B. Memory management

The memory layout of the proposed approach consists of two different regions in non-volatile FLASH: the *parameters region* used to save the LBIST DATA structure shown in Algorithm 1 between functional resets; the *failure region* is used to store the failure LBIST information and is composed of the failing pattern index, the expected golden signature and the resulting one.

C. Enhanced Logic Diagnosis

As the number of field returns is a growing problem for companies, the in-field data collection is a key way to try to address the issue. [18] proposed a system capable of performing logic diagnosis for field returns exploiting the information of only the first failing pattern. The logic diagnosis process of that method is based on an *incomplete* tree-based fault dictionary, resulting into a not always optimal diagnosis accuracy.

The methodology for building of the tree-based fault dictionary is the same as described in [18], the key difference in this work is that the logic diagnosis is performed through a *complete* tree-based fault dictionary. Indeed, the novel comprehensive diagnostic data collection leverages on the reseeding of the LFSR to collect more information from patterns and

allowing to build a complete the tree-based fault dictionary, as shown in Figure 4.

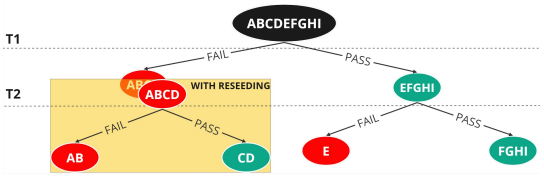


Fig. 4: Tree-based fault dictionary for LBIST with reseeding.

IV. Experimental Results

A. Experimental setup

The case study used to validate the methodology is an industrial automotive SoC manufactured by STMicroelectronics. The industrial DUT has the following specifics:

- About 20 million gates;
- About 700 thousand flip-flops;
- ASIL-D compliant;
- 7 LBIST partitions;
- 92 Memory BIST partitions.

Moreover, to further prove the effectiveness of the proposal for the industrial case study,a similar experimental setup presented in [18], [22] was used, composed of the following:

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

The firmware is responsible for programming and executing LBISTs during regular device operation. The main features of the external software tool are coordinating the LBIST experiments, implementing the proposed algorithm, and managing communication with the board.

Table I reports the 7 LBIST partitions with the corresponding number of scan cells and scan chains, and faults summary, including stuck-at (SA) and transition delay (TRN) faults.

TABLE I: LBIST partitions faults summary of the DUT.

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

B. Code size and complexity

The incremental code sizes are shown in Table II, starting with the base system, adding the library necessary for handling BIST functionalities, and finally, the proposed methodology. The logic reported in Algorithm 2 can be represented in a high-level programming language, such as C, with approximately 70 lines of code. The complexity of the proposed approach depends on the total number of patterns, as well as the maximum failing pattern that must be identified and stored, as described by $O(STP \times \log_2(MAXP))$.

TABLE II: Incremental code size of the proposed firmware.

	System	+ LBIST library	+ Proposed approach
Code size	17,418	26,054	26,118
Δ	-	8,636	134

In the case of the study, LBISTs programmed to test the maximum number of patterns $MAXP = 65,536$ take up to $\sim$ 100 ms to complete the self-test procedure. Thus, at key-on and key-off, if the device is not faulty, only one LBIST execution is necessary. Hence, the duration of the procedure is 100 ms. However, when the initial signature mismatches, the proposed algorithm searches for the first failing pattern. This is done by executing LBISTs following a binary search approach for a total of $\log_2(65,536) = 16$ times with decreasing number of patterns at each iteration, requiring $\sum_{i=0}^{15} \frac{100}{2^i} \sim 200$ ms for the first failing pattern. At worst, $STP = 73$ failing patterns are in the beginning, requiring a search on the complete set of patterns, hence 73×200 ms. Meanwhile, the best case is when failing patterns are at the end of the pattern set, meaning that LBIST will execute the first search and then very few patterns, thus ≈ 200 ms.

C. Memory requirements

Each of the 7 LBIST partitions on the case of study have 16 bits of programmable register to program the number of patterns to generate and execute, thus LBISTs are initially configured to run $65,536$ patterns.

Every entry in the *failure region* described in Subsection III-B is 144 bits wide: 16 bits for the index, 64 bits for the golden signature, and 64 bits for the failed signature. Meanwhile the *parameters region* occupies 341 bits: 1 bits for the golden flag, 64 for the current golden signature, 16 bits for the starting pattern, 16 bits for the ending pattern, 4 bits for the depth, 64 bits for the last failing signature, 64 bits for the last golden signature, 32 bits for the starting LFSR seed, 64 bits for the starting MISR seed, 16 bits for the counter of the number of fails. The whole memory footprint is $144 \times STP + 341$ bits.

D. Diagnostic Resolution

This work discusses logic diagnosis through LBIST data collection targeting transition delay faults (TRN). The collected in-field data is extremely valuable in this sense. The presented approach introduces a STP parameter as the number of failing patterns considered and saved into the failure region for the possible logic diagnosis. Such parameter offer a trade-off between DR, Execution Time, and memory requirements. Graphs 5-6 show the relation between the number of failing

patterns, and the resulting average DR and the average Execution Time. This work focuses on the maximum optimization of the DR; as such, experimentally it is observed that the number of patterns (STP) that optimize the DR while maintaining a reasonable Execution Time and Memory requirement is 73.

The DR values for each LBIST partition are shown in Table III, highlighting a significant improvement compared to the other methods. The values have been computed through $2,000$ random transition fault injections performed by commercial ATPG tool for the proposed method and [15], meanwhile for [18], the value refer to the average size of undistinguished fault classes over all faults.

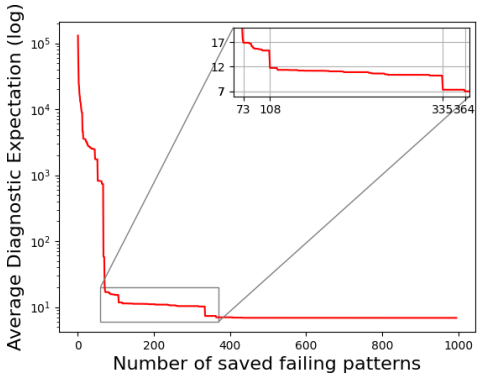


Fig. 5: Average Diagnostic Resolution over the number of pattern being saved (STP parameter) for LBIST partition 5.

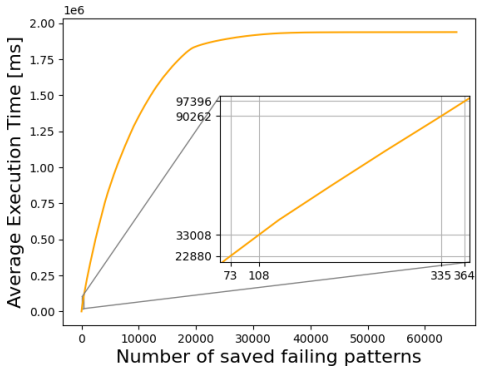


Fig. 6: Average Execution Time in millisecond over the number of pattern being saved (STP parameter) for LBIST partition 5.

TABLE III: Average DR for each LBIST partition of the DUT.

LBIST Partition	Avg. DR	Avg. DR [18]	Avg. DR [15]
0	8	92	44
1	34	207	81
2	15	103	52
3	14	100	65
4	14	101	58
5	17	109	56
6	27	123	77

V. Conclusion

In conclusion, the paper introduced a firmware-based methodology for LBIST to improve the diagnostic accuracy of devices returned from the field to manufacturers.

The proposed approach provides a practical and efficient way to improve diagnostic capabilities while avoiding architectural changes and keeping memory requirements low. It thus meets the need for in-depth diagnostics, even in the case of NTFs, while considering the practical constraints of the in-field. Experimental results on reference circuits and an industrial automotive device produced by STMicroelectronics validate the proposal's effectiveness compared to other state-of-the-art methodologies. The proposed solution effectively addresses the limitations of other methodologies, which either require custom LBIST architectures or offer limited diagnostic resolution due to the use of little information. Future research could focus on developing optimized adaptive reseeding strategies to further improve diagnosis accuracy.

References

[1] A. Vassighi *et al.*, "Cmos ic technology scaling and its impact on burn-in," *IEEE Transactions on Device and Materials Reliability*, 2004.

[2] M. Agrawal *et al.*, "Test-cost modeling and optimal test-flow selection of 3-d-stacked ics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 9, pp. 1523–1536, 2015.

[3] G. Iaria *et al.*, "A novel SEU injection setup for Automotive SoC," in *International Symposium on Industrial Electronics*, 2022.

[4] T. Tran *et al.*, "No trouble found (ntf) customer return analysis," in *2020 IEEE International Reliability Physics Symposium (IRPS)*, 2020.

[5] R. Kashyap, "Silicon lifecycle management (slm) with in-chip monitoring," in *2021 IRPS*, 2021, pp. 1–4.

[6] B. Jang *et al.*, "On-chip aging prediction circuit in nanometer digital circuits," in *2014 International SoC Design Conference*, 2014.

[7] I. Bayraktaroglu *et al.*, "Improved fault diagnosis in scan-based bist via superposition," in *Proceedings of the 37th DAC*, 2000.

[8] ——, "Deterministic partitioning techniques for fault diagnosis in scan-based bist," in *Proceedings International Test Conference 2000*, 2000.

[9] ——, "Diagnosis for scan-based bist: reaching deep into the signatures," in *Proceedings DATE 2001*, 2001, pp. 102–109.

[10] M. Elm *et al.*, "Bisd: Scan-based built-in self-diagnosis," in *DATE*, 2010.

[11] A. Jayalakshmi *et al.*, "A methodology for lbist logic diagnosis in high volume manufacturing," in *2012 ASQED*, 2012.

[12] R. Ubar *et al.*, "Fault diagnosis in integrated circuits with bist," in *DSD 2007*, 2007.

[13] W.-T. Cheng *et al.*, "Signature based diagnosis for logic bist," in *2007 IEEE International Test Conference*, 2007, pp. 1–9.

[14] W.-t. CHENG *et al.*, "Improving the performance of signature based diagnosis for logic bist," Patent PCT/US2009/034 933, 2009.

[15] S. Gopalsamy *et al.*, "A storage based lbist scheme for logic diagnosis," in *2024 IEEE 42nd VLSI Test Symposium (VTS)*, 2024, pp. 1–7.

[16] ——, "Fully deterministic storage based logic built-in self-test," in *2023 VTS*, 2023, pp. 1–7.

[17] G. Filipponi *et al.*, "In-field data collection system through logic bist for large automotive systems-on-chip," in *IEEE ITC*, 2022.

[18] P. Bernardi *et al.*, "Logic diagnosis based on logic built-in self-test signatures collected in-field from failing system-on-chips," *Electronics*, 2024.

[19] P. H. Bardell *et al.*, *Built-in Test for VLSI: Pseudorandom Techniques*. USA: Wiley-Interscience, 1987.

[20] ——, "Self-testing of multichip logic modules," in *International Test Conference*, 1982.

[21] V. Boppana *et al.*, "Full fault dictionary storage based on labeled tree encoding," in *Proceedings of 14th VLSI Test Symposium*, 1996.

[22] P. Bernardi *et al.*, "Collecting diagnostic information through dichotomic search from logic bist of failing in-field automotive socs with delay faults," in *2023 DDECS*, 2023, pp. 21–26.