Fault Grading Techniques for Evaluating Software-Based Self-Test with Respect to Small Delay Defects

Fault Grading Techniques for Evaluating Software-Based Self-Test with Respect to Small Delay Defects

Michelangelo Bartolomucci, Nikolaos I. Deligiannis, Riccardo Cantoro, Matteo Sonza Reorda
Department of Control and Computer Engineering, Politecnico di Torino, Italy

***Abstract*—A widely adopted practice for in-field testing of electronic devices uses Software-Based Self-Test (SBST) in the form of Software Test Libraries (STLs). Typically, STLs target the stuck-at and Transition Delay Fault (TDF) models. However, to face the new defects introduced by the most recent semiconductor technologies, new fault models must be adopted. Small Delay Defects (SDDs) play an increasingly important role in this scenario. Unlike TDFs, SDDs slightly increase the paths' timing, whose size is not in the same order of magnitude of the clock period. These defects can cause failures during the operational phase if they affect the critical paths. Remarkably, in scan testing the propagation time of a fault is limited, as a fault effect has to reach the scan flip-flops to be detected. However, in functional testing, the fault effect may require several clock cycles before reaching an observable point. Thus, the delay due to the fault cannot be indefinitely long.**

As there will be the need to move to delay faults when developing STLs, it is important to use the timing information correctly in functional fault simulations. SDDs are the typical choice. In this paper, we implemented a fault grading process for STLs to show how the fault coverage they can achieve changes when the delay defect increases (from SDDs to the extreme case of TDFs). The work uses static timing analysis; although this is known to yield pessimistic results in some cases, it gives a very good indication of the trend in fault coverage as the SDDs approximate TDFs. Differences in fault coverages with respect to the TDF model are highlighted, while an assessment of the effects of multi-cycle delays is also provided.

***Index Terms*—Small Delay Defects, Delay Test, Safety-Critical Domain, Functional Safety, Processors**

I. Introduction

In the dynamic realm of digital technology, ensuring the reliability and effective testing of digital circuits emerges as a fundamental requirement. As our reliance on electronic systems grows, ensuring digital circuits' seamless and error-free operation becomes crucial for various applications, ranging from consumer electronics to safety-critical systems, e.g., in aerospace and healthcare.

In the safety-critical domain, from the manufacturers' standpoint meeting the high and stringent reliability thresholds, imposed by the respective safety standards (e.g., ISO-26262 for the automotive industry) is not enough. They must further guarantee that a fault will never produce a system failure that will put at danger a human life or endanger the environment.

A common test procedure that is widely in the end-of-manufacturing phase but most importantly during the mission phase of the systems (as in-field test) is functional testing. In the case of end-of-manufacturing, functional testing is performed to detect any defects that may have escaped the traditional structural test. When in-field testing is targeted, a functional test is performed first due to its flexibility. Moreover, functional test is guaranteed not to target any fault that cannot propagate and cause a system failure. Furthermore, functional testing is favored as a form of in-field testing because the system company (i.e., the customer) can also launch it, knowing the achieved fault coverage, which is assessed by the manufacturing company, which developed the STL.

In-field test is sometimes applied through Software-Based Self-Test (SBST) for the development of Software Test Libraries (STLs) [1]. STLs are a collection of test programs and are typically executed during the idle times of the system to ensure that the system functions correctly and safely. They represent a suitable solution due to their non-intrusiveness as they preserve the system state and do not alter it, are flexible, and can provide coverage for the widest possible number of faults.

Currently, the dominant fault models are the stuck-at (SAF) and transition delay (TDF) fault models as they adequately model the vast majority of defects that may manifest in an operational scenario. However, in the last few years, it has been observed that as new technological advancements yield denser and faster circuits, the number of test escapes increases. This negative trend is attributed primarily to defects not adequately modeled by the adopted fault models. As a solution to alleviate this problem, new fault models have been proposed, like those of the Cell-Aware Test [2] and the Small-Delay Defects (SDD) testing [3].

However, the problem of generating STLs, even for the dominant models, i.e., SAFs and TDFs, is an arduous task for the test engineers. Albeit certain automated solutions exist to partially assist in the generation of the test programs, typically a significant amount of manual effort is required for their development and evaluation.

In this paper we propose a novel technique for assessing and profiling STLs, written for the TDF model, characterizing them in terms of their SDD detection capabilities. We present the results of a fault grading process for STLs that shows how the fault coverage (FC) is impacted as the delay value at the fault site increases, approximating the extreme case of a TDF. The delay information is derived from a commercial static timing analysis tool. The method has been applied to

functional units of a single issue, scalar pipelined RISC-V processor.

The rest of the paper is organized as follows. In Section II, we delve into the background of SDDs, exploring motivations and elaborating upon relevant works. In Section III we present our method and in Section IV the experimental results. Lastly, in Section V we draw some conclusions and provide insight into our future works on this topic.

II. Background

A. Small Delay Defects

An SDD is a type of delay fault occurring in between interconnections that can make the circuit fail during at-speed operation. Such defects are typically caused by small voids in the interconnections, e.g., due to the deposition of particles on the silicon during fabrication, resulting in weak, resistive defects in the circuit.

When an SDD is manifested in the circuit's longest (critical) path, it can cause the total delay of the path to violate the global clock timing [4]. They are different from gross delay faults (covered by the TDF model), which cause the circuit to fail regardless of the path they affect. More specifically, the TDF model assumes that single circuit nodes, rather than paths, introduce large delay values ($\gg$ clock period) on both polarities (i.e., slow-to-rise and slow-to-fall); thus, all transitions passing through the node are going to be delayed past the clock period. Hence, when performing TDF-oriented ATPG, the algorithm will try to propagate the fault effect with minimal computational effort. This means that the critical paths are not explicitly enumerated for performance reasons since the time required for enumerating the total paths of a modern sequential circuit can be enormous [5], and thus potential SDDs are left untested. The timing-aware ATPG [6] has been proposed to amend this obstacle. Delay information in standard delay format (SDF) is incorporated through the test generation phase, which is utilized to identify critical circuit paths and target SDDs effectively and systematically.

However, it may be possible that an SDD cannot be propagated through the critical part of the circuit. This implies that even with timing-aware ATPG patterns, these defects will remain undetected at the operational frequency. This category of SDDs is called *hidden delay defects*. If they remain undetected, their latency may grow due to aging effects and may cause early life failure and reliability issues. To solve this problem, the post-ATPG method of faster-than-at-speed testing (FAST) [7] has been introduced. FAST utilizes the already computed test patterns and evaluates their fault detection capabilities under multiple, faster than the operating frequency timing checks.

However, in the context of functional testing (e.g., during in-field testing) and when STLs are utilized, there is no definitive method for effectively evaluating the fault detection capabilities of the programs concerning SDDs.

B. Relevant Works

The problem of identifying critical paths in a sequential circuit has bothered the research community in the past. In [8], the authors propose a versatile method for enumerating all or a user-specified number of the longest sensitizable paths of a circuit by encoding the path search as an instance of the Boolean Satisfiability (SAT) problem. The same methodology is employed in [9] for generating tests that sensitize paths of user-defined lengths while considering the complex timing behavior of manufacturing defects influenced by process variations. In [10], the authors present an innovative approach for analyzing reachable sensitizable paths in sequential circuits. This approach integrates a SAT-based path-enumeration algorithm with a model-checking solver to assess the reachability of circuit states. Specifically, the method identifies the longest sensitizable paths within a circuit and determines the minimal-length test sequences that can sensitize these paths from a given initial state.

Regarding the area of functional test generation, in [11] the authors, building on top of their previous works, propose a deterministic, timing-aware ATPG system for identifying and testing for SDDs in non-scan circuits. This methodology determines the longest sensitizable paths for fault activation and generates sub-sequences for each fault, starting from the circuit's initial state to avoid over-testing. The process incorporates various phases, including synchronization, path generation, fault propagation, and sequence connection, each contributing to formulating a comprehensive and efficient testing strategy.

In [12] the authors propose a novel approach for automatically generating functional microprocessor test sequences targeting SDDs by utilizing bounded model checking. This work is the first to automate functional test program generation for small-delay faults in processors, highlighting the significance of applying formal methods and constraint-based ATPG to enhance testing effectiveness while maintaining processor functionality and operational integrity.

In [13] the authors introduce PHAETON, an advanced tool designed for the automated generation of functional test sequences that target small-delay faults within microprocessors. The tool utilizes SAT-based approaches and the principles of bounded model checking. The tool is shown to identify the longest sensitizable paths for testing and to incorporate mechanisms to ensure that generated test sequences are functional, meaning they can be directly applied in a real processor environment without additional hardware modifications.

In [14], the authors introduce a novel algorithm for diagnosing SDDs in compressed test responses, a first in addressing timing issues amidst manufacturing variations and new FinFET defect behaviors. The approach combines variation-invariant structural analysis, GPU-accelerated simulation, and variation-tolerant matching, showing high accuracy and scalability on benchmark circuits. It's designed for enhancing fault candidate data for volume diagnosis rather than precision diagnosis, which deals with multiple faults and requires

uncompressed data. However, this work focuses on scan-based designs, diverging from the functional-based approach that we are focusing on in this paper.

In [15], the authors address the challenge of accurately assessing the fault coverage of STLs used for in-field testing of integrated circuits in safety-critical applications. Traditional fault simulation tools, designed for end-of-manufacturing tests, do not suffice for in-field scenarios where STLs operate, due to their different requirements and operational contexts. The paper differentiates between Sequential Circuit Fault Simulation (SC-FSIM) and Self-Test Procedures Fault Simulation fault simulation (STP-FSIM), highlighting the computational challenges of the latter and proposing optimized methodologies to balance accuracy and computational efficiency. These methodologies leverage commercial tools to facilitate adoption in professional settings. Experimental results validate the effectiveness of these techniques, especially in the context of the automotive sector's stringent safety standards, like ISO 26262.

Lastly, in [16], the authors present a method for enhancing digital peripheral testing in SoCs by using SBSTs for detecting delay and stuck-at faults. Their approach aims to complement or replace traditional scan-based testing, potentially lowering costs and improving fault coverage without additional power consumption. A case study on a digital communication peripheral showcases that SBSTs can uniquely identify faults not detectable by scan methods, particularly in transition delay faults. The study also explores SDDs and the role of functionally untestable faults, emphasizing the synergy between SBST and scan testing for a thorough testing strategy in SoCs.

III. Proposed Grading Method

Our goal is, given an STL, which has been written for a TDF fault model, to asses its fault detection capabilities for SDDs while utilizing commercial electronic design automation (EDA) tools.

After synthesizing the RT-level description of our sequential circuit, which acts as our circuit under test (CUT), we obtain the gate-level description and the circuit's delay information in SDF. To extract the timing information of each path of the CUT, we resort to static timing analysis (STA) [17]. Using a commercial STA, we can identify the exact delay and slack of all structural paths within the CUT.

As depicted in Figure 1, the paths within a sequential circuit belong to four categories according to their start and end points. These are:

1) A path beginning from a primary input (PI) and ending to a primary output (PO), colored in red.
2) A path beginning from a primary input (PI) and ending to an FF, colored in yellow.
3) A path beginning from an FF and ending to an FF, colored in green.
4) A path beginning from an FF and ending to a PO, colored in blue.

The timing information concerning PIs and POs is typically tailored to the specific application context. For instance, if the

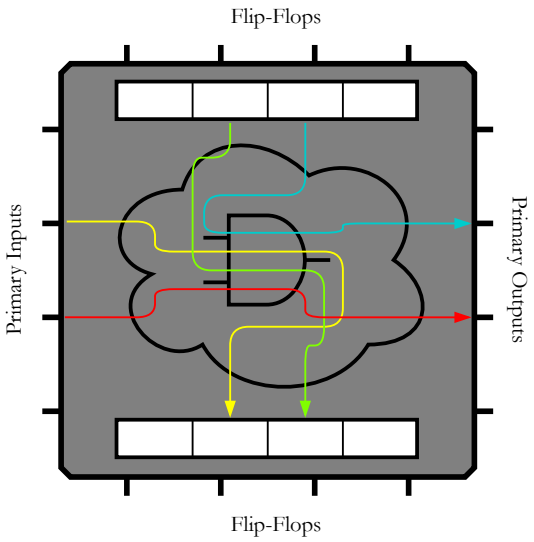


Fig. 1: Path types in a sequential circuit.

CUT is integrated into a System-on-Chip (SoC), this timing data relies on the interactions with peripherals. It cannot be directly inferred from the logic synthesis of the standalone CUT unless these application-specific timings are considered constraints on PIs and POs, typically through SDC files (in Synopsys terminology) during synthesis. Therefore, without loss of generality, this paper focuses on SDDs affecting the paths from FFs to FFs. These timing concerns are strictly determined by the structural characteristics of the CUT and are independent of external factors. However, considering that the PI/PO timing information is available, the proposed grading method can be equally applied.

These paths are considered the fault sites for SDDs. Considering an arbitrary path, then the corresponding SDD latency is computed as:

$$\text{SDD}_{\text{latency}} = \begin{cases} \text{slack}_{\text{min}} + K \times (\text{slack}_{\text{max}} - \text{slack}_{\text{min}}) & 0 \leqslant K \leqslant 1.0 \\ K \times \text{clock period} & K > 1.0 \end{cases}$$

Let us consider the case depicted in Figure 2. The $\text{slack}_{\text{min}}$ value refers to the slack of the longest (critical) path, whereas $\text{slack}_{\text{max}}$ refers to the slack of the shortest path of the circuit. Hence, for K values less than 1.0, the SDD latency equals values in the range of $[\text{slack}_{\text{min}}, \text{slack}_{\text{max}}]$. These delay values are added on top of the minimum slack of the critical path of the CUT. However, as K exceeds the value 1.0, the SDD latency can induce multi-clock cycle violations on the CUT, approximating the effects of a TDF on the circuit.

Circuit paths are ranked from top to bottom, with the longest (critical) paths having small slack values, while shorter paths exhibit more significant slack values. When an SDD occurs in the critical path with a relatively small latency (denoted by a small K value), it activates only that critical path. This activation does not impact longer, less critical paths, as they would not experience timing violations. However, as fault

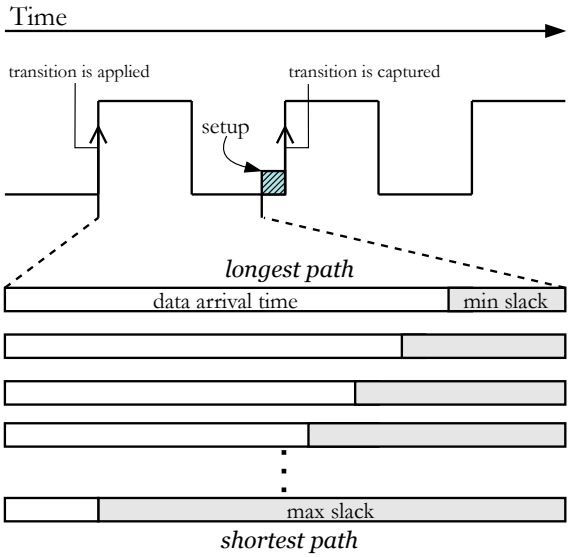


Fig. 2: Sequential circuit path ranking

latency increases, more timing violations occur, activating additional paths for propagating the violation to observable points. In the context of functional test, this means that as the latency of an SDD increases, inducing more path violations on the CUT, the probability of fault detection increases too in a proportional manner.

By utilizing the STA tool, we can identify the slack time for each SDD fault site. In an iterative manner, by considering different K values, we can generate a set of fault lists that differ in the latency of each fault. The next step is to run a fault simulation campaign of the CUT, considering all of the generated fault lists.

However, it is known that STA is pessimistic, and the longest paths may be false paths (non-sensitizable). This will result in a low coverage for low K values. Furthermore, by increasing K by a constant value for each fault site, we are not considering the different distributions of paths i.e., some fault sites may activate a higher percentage of new paths than others when increasing K. However, this grading method gives us a good indication of the FC trend.

Lastly, the proposed method is generic and can be applied to any sequential circuit used as a CUT.

IV. Experimental Results

A. Experimental Setup

As a case study, we used five different STLs that have been developed for the in-order 4-stage RISC-V processor CV32E40P [18]. Three STLs were written for the processor's load and store unit (LSU) and two for the multiplier unit. The processor's RT-level description was synthesized via Synopsys Design Compiler using the Silvaco 45nm technology library [19]. Table I reports the TDF coverage achieved by the test programs on the functional units we employ as targets within our CUT.

STL	TDF Coverage [%]	TAT [CCs]
MULT 1	89.01	48,209
MULT 2	49.72	22,215
LSU 1	71.91	1,910
LSU 2	71.28	1,185
LSU 3	65.29	1,305

TABLE I: TDF coverage of STLs

The STA tool we used to compute the slack values of each SDD fault site was PrimeTime by Synopsys. Each slack value reported for every SDD fault was then quantized in the following manner. For $K \leqslant 1$, we use an increment 0.05 and begin with minimum $K = 0.00$ and maximum $K = 1.0$. For $K > 1$, we use multiples of the clock period (CCs). Namely, 2, 3, and 4 times the clock period. The total number of SDD fault lists generated are:

- 1.0 / 0.05 + 1 = 21 fault lists for $K \leqslant 1$
- 3 fault lists for $K > 1$

In total, 24 SDD fault lists were generated, with a Test Generation Time of $1m25s$, each. The generated fault lists enumerate 5,760 faults for the LSU and 15,728 for the multiplier. For the fault simulation campaigns, we used Z01X by Synopsys. The STLs were used as a stimulus source in the form of eVCD. The latter was generated by performing a logic simulation of each STL on the processor's gate-level description in QuestaSIM by Siemens. Our experiments were performed on a machine using x2 Intel(R) Xeon(R) Gold 6238R processors and 256GB of RAM.

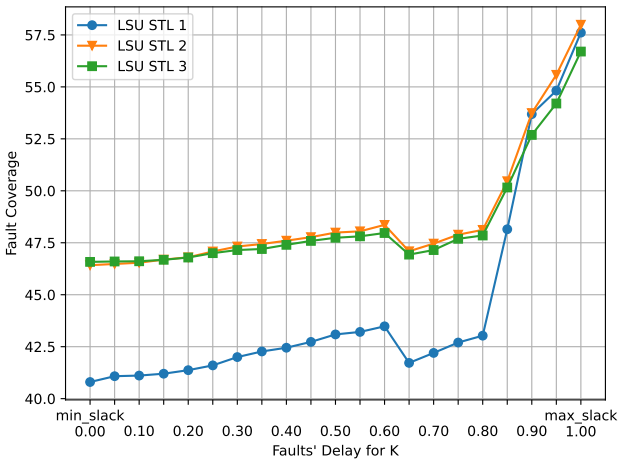
B. Results

The results of the SDD fault simulation campaigns are shown in Figure 3. The total runtime of all fault simulations for the LSU per STL was 38.68, 36.43, and 35.19 minutes for STL1, STL2, and STL3 respectively. For the case of the multiplier, the total runtime was 121.12 and 149.41 minutes for the respective STLs.

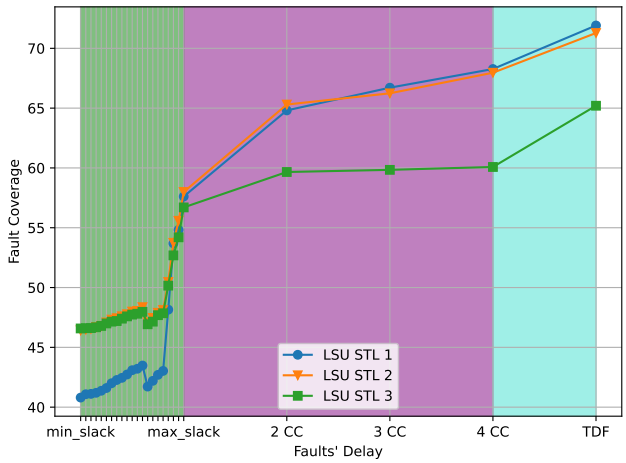
1) $K \leqslant 1$: In Figure 3a and Figure 3c, the results concerning smaller-than-the-clock period SDD latencies are presented for the LSU and the multiplier respectively. Both plots' min and max slack ticks represent each fault's minimum and maximum latency, computed with $K = 0.00$ and $K = 1.0$, respectively.

For the LSU, we observe that STL2 and STL3 showcase a similar trend in terms of FC, whereas STL1 reaches analogous FC percentages for K values greater than 0.85. This is attributed to the fact that the STL1 contains instructions with fewer variations of the memory-related instructions of the supported instruction set architecture. The baseline FC achieved for SDD fault lists computed the minimum fault latencies were 40.80%, 46.42%, and 46.58% for the STL1, STL2, and STL3, respectively. The maximum FC achieved for the SDD fault lists computed with the maximum fault latencies were found to be 57.61%, 57.99%, and 56.70%.

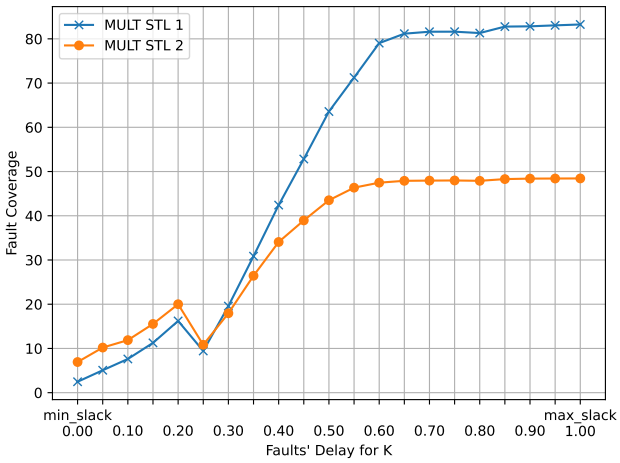
For the multiplier, we observe that albeit the two STLs initially showcase similar behavior, after a certain point (for



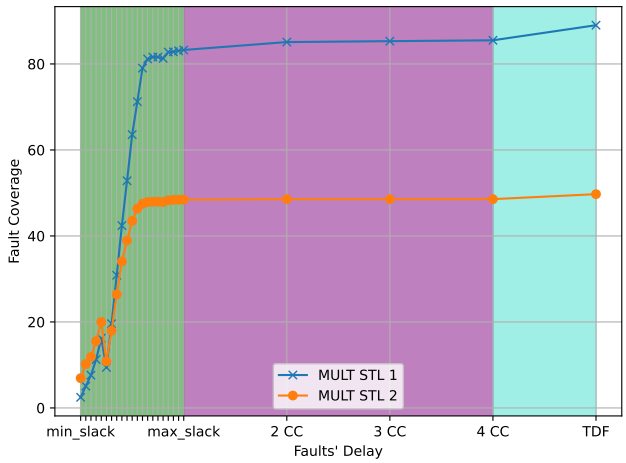
(a) Fault coverage behavior for LSU and $K \leqslant 1$.



(b) Fault coverage behavior for LSU for all K values.



(c) Fault coverage behavior for Multiplier and $K \leqslant 1$.



(d) Fault coverage behavior for Multiplier for all K values.

Fig. 3: Performance of TDF STLs for SDDs in the Multiplier and LSU for various K values.

$K = 0.40$) there exists a notable difference between the two curves which results in a deviation of approximately 30% (for the maximum fault latencies). This deviation is attributed to the fact that STL2 comprises fewer variations of operands for the multiplier unit than STL1. The baseline FC achieved in this case is 2.47% and 6.93%, whereas the FC achieved for the maximum fault latency considered is 83.25% and 48.44% for the STL1 and STL2, respectively.

For both cases, as we initially expected, the FC remains relatively low since smaller latencies induce path timing violations in fewer critical paths of the CUTs. However, as the SDD latency approaches the respective path's slack value, we observe a drastic increase in the FC. This is justified as the more severe the SDD becomes (in terms of latency), the more paths are violated in the CUT; hence, the easier it is to detect the fault.

Interestingly, we observe that the monotonicity of the FC plots for both cases is violated at a certain point. For the LSU, this happens when moving from $K = 0.60$ to $K = 0.65$ and for the multiplier when moving from $K = 0.20$ to $K = 0.25$. As mentioned previously, as the latency of the SDD increases, so do the path violations that occur in the CUT, such as slower, shorter paths in the presence of the fault become critical. This drop in the FC can be attributed to path reconvergencies that cancel the fault effects.

2) $K > 1$: In Figure 3b and Figure 3d, on top of the cases considered for $K \leqslant 1.0$ (colored in green) we further incorporate the results concerning larger-than-the-clock period SDD latencies (colored in purple) as well as the case of the TDF (colored in cyan).

For the LSU, we observe that for all STLs, the FC plots show a notable increase when moving for the case of $K = 1.0$ to $K = 2\times$CC. The magnitude of this increment showcases that we are still far from the case of gross delay faults. In general, for the fault latencies of $2\times$CC up to $4\times$CC we observe small increments in the FC. The maximum FC reached, for $K = 4\times$CC is 68.27%, 67.96%, and 60.08% for STL1, STL2, and STL3 respectively. In the last section of the

plot (colored in cyan), we report the FC values of Table I, for the case of TDF. In this final step, we observe an average increase of 4.06% in the FC for the three STLs. Noteworthily, STL1, which achieved lower FC values for low K values, and thus low SDD latencies, reaches the highest FC when considering SDDs with high latencies and TDFs. On the contrary, STL3, which outperformed STL1 when targeting SDDs with low latencies, achieved the lowest FC when considering higher SDD latencies and TDFs. This observation underscores the critical need for the refinement of grading techniques tailored specifically to accurately assess STLs. It compellingly demonstrates that relying solely on test programs designed for alternative delay fault models, such as TDF, is insufficient for comprehensive coverage of SDDs.

However, for the multiplier, we observe that for the case of $K = 1.0$ to $K = 2\times$CC, but also for the whole multi-cycle latency window (in purple) there are very small increments in the FC. This indicates that the case of gross delay faults is probably not far off (in terms of FC). For $K = 2\times$CC up to $4\times$CC we observe a saturation in the FC as for STL1 there is an increase of 0.41% and for STL2 the FC remains stable. Regarding the TDF, we observe an average increase of 2.34% in the FC for the rightmost part of the plot.

Overall, we observe that in some cases the TDF coverage is much higher than the one achieved when considering SDDs with $K = 1.0$ (max slack). This is evident in the case of the LSU where we observe a deviation of the three STLs between the TDF and the $K = 1.0$ of the order of 14.3%, 13.29%, and 8.59% for STL1, STL2, and STL3 respectively. In the case of the multiplier, these effects are less evident.

V. Conclusions

Functional test in the form of STLs is a well-known practice. However, the dominant models, namely the SAF and TDF do not fully suffice for modeling all possible defects in systems of new technology nodes. SDDs are of critical importance in industries since they have been found to be the source of most test escapes and reliability problems.

In this paper, we present a methodology, based on commercial EDA tools, to grade the effectiveness of STLs, written for the TDF model, in terms of SDD fault detection capabilities. We use an STA tool to derive timing information of paths and model the latency of each SDD as a function of each path's slack.

As a case study, we used a RISC-V processor and assessed the effectiveness of STLs targeting TDFs in the core's LSU and multiplier unit. We considered a wide variety of SDD latency values, beginning from very small to multi-cycle latencies approximating the case of TDFs. From the experiments performed on the two functional units, we observed that relying solely on test programs designed for alternative delay fault models, such as TDF, is not guaranteed to adequately coverage of SDDs.

As future work, we plan to emphasize improving the accuracy of the proposed grading method (e.g., by excluding false paths). Furthermore, we plan to extend the considered STLs both in covering different functional units and also considering STLs developed for other fault models. For instance, to investigate the correlation between the Path Delay Faults and SDDs. Furthermore, in our grading method, we plan to incorporate parameters such as aging and process variations.

Acknowledgment

This publication is part of the project PNRR-NGEU which has received funding from the MUR – DM 117/2023

References

[1] M. Psarakis et al. "Microprocessor Software-Based Self-Testing". In: *IEEE Design & Test of Computers* 27 (2010), pp. 4–19.

[2] F. Hapke et al. "Cell-Aware Test". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33 (2014), pp. 1396–1409.

[3] Y. Sato et al. "Invisible Delay Quality - SDQM Model Model Lights Up What Could Not Be Seen". In: *IEEE International Conference on Test (ITC)*. 2005.

[4] P. Muthukrishnan and S. Sathasivam. "A Technical Survey on Delay Defects in Nanoscale Digital VLSI Circuits". In: *Applied Sciences* 12 (2022).

[5] K. Christou et al. "Identification of Critical Primitive Path Delay Faults Without any Path Enumeration". In: *IEEE VLSI Test Symposium (VTS)*. 2010.

[6] X. Lin et al. "Timing-Aware ATPG for High Quality At-speed Testing of Small Delay Defects". In: *IEEE Asian Test Symposium (ATS)*. 2006.

[7] T. Yoneda et al. "Faster-Than-At-Speed Test for Increased Test Quality and In-Field Reliability". In: *IEEE International Test Conference (ITC)*. 2011.

[8] M. Sauer et al. "Efficient SAT-Based Search for Longest Sensitisable Paths". In: *IEEE Asian Test Symposium (ATS)*. 2011.

[9] M. Sauer et al. "SAT-based Analysis of Sensitisable Paths". In: *IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. 2011.

[10] M. Sauer et al. "Analysis of Reachable Sensitisable Paths in Sequential Circuits with SAT and Craig Interpolation". In: *IEEE International Conference on VLSI Design (VLSI)*. 2012.

[11] M. Sauer et al. "Functional Test of Small-Delay Faults Using SAT and Craig Interpolation". In: *IEEE International Test Conference (ITC)*. 2012.

[12] A. Riefert. "An Effective Approach to Automatic Functional Processor Test Generation for Small-Delay Faults". In: *IEEE/ACM Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2014.

[13] A. Riefert et al. "A Flexible Framework for the Automatic Generation of SBST Programs". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24 (2016), pp. 3055–3066.

[14] S. Holst et al. "Variation-Aware Small Delay Fault Diagnosis on Compressed Test Responses". In: *2019 IEEE International Test Conference (ITC)*. 2019.

[15] A. Floridia et al. "Fault Grading Techniques of Software Test Libraries for Safety-Critical Applications". In: *IEEE Access* 7 (2019), pp. 63578–63587.

[16] M. Grosso et al. "Software-Based Self-Test for Delay Faults". In: *VLSI-SoC: New Technology Enabler*. 2020.

[17] Kukimoto, Y. and others. "Static Timing Analysis". In: *Logic Synthesis and Verification*. Springer US, 2002, pp. 373–401.

[18] M. Gautschi et al. *Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices*. 2017.

[19] *Silvaco 45nm Open Cell Library*. https://si2.org/open-cell-library.