Automatic Identification of Functionally Untestable Cell-Aware Faults in Microprocessors

Automatic Identification of Functionally Untestable Cell-Aware Faults in Microprocessors

Nikolaos I. Deligiannis†, Tobias Faller*, Iacopo Guglielminetti‡, Riccardo Cantoro†,
Bernd Becker*, Matteo Sonza Reorda†
† Politecnico di Torino, Department of Control and Computer Engineering (DAUIN) - Turin, Italy
‡ STMicroelectronics — Turin, Italy * University of Freiburg, Department of Computer Science — Freiburg, Germany

***Abstract*—In-field test of microprocessors is a major topic for the industry, especially in the safety-critical domain, where the respective standards mandate high test coverage thresholds. The dominant fault models used are the transition delay and the stuck-at fault model. However, the adoption of very advanced semiconductor technologies to manufacture devices used in safety-critical applications pushes toward considering new fault models that are better suited to catch subtle and age-related defects. Among the other phenomena, latent cell-internal defects emerged as relevant causes for several failures. Hence, the necessity for the Cell-Aware Test (CAT) was born, and the inclusion of the CAT fault model in the latest safety standards. Although CAT amends the issue of the numerous test escapes, it may suffer as well from the presence of functionally untestable faults that may pollute the overall test efficiency with their presence. In this paper, we propose a solution, based on formal methods, for the automatic identification of functionally untestable faults under the Cell-Aware fault model for the case where the DUT is a fully pipelined processor. As a case study, we used the RISC-V processor `RI5CY` for which we applied the minimum constraints required to ensure a functional behavior to demonstrate the effectiveness and impact of the approach. With the considered constraints, a significant percentage of functionally untestable faults was located in the several modules within the processor. Furthermore, the method allows to flexibly take into account any constraint stemming from the system configuration and the application. The obtained results have been validated by resorting to commercial EDA tools.**

***Index Terms*—Cell-Aware Test, Functionally Untestable Faults, Microprocessor, RISC-V, In-Field Test, Bounded Model Checking, Safety**

I. Introduction

While Design-for-Testability (DfT) infrastructures are the dominant solution for manufacturers, functional at-speed testing methods are widely used for testing products during their mission cycle. Functional safety standards (e.g., ISO26262 for automotive) strengthen the necessity for functional test procedures by not only rendering them popular but also mandating for functional test procedures to reach certain high coverage thresholds in the safety-critical domain of applications. For example, in the automotive industry, ISO26262 mandates a 98% functional fault coverage for every critical environment in the car (e.g. airbags). Therefore, in such systems, it is necessary for test engineers to know which faults are untestable (*safe*), i.e., cannot cause a failure in the considered operational scenario. The presence of untestable faults (if not identified) can negatively impact the achieved fault coverage, and thus, they can become a major concern for test engineers.

In recent years, statistics have reported an increasing number of failing devices returned to semiconductor suppliers, although their test flows have been sufficient in terms of achieved test coverage (thus complying with the thresholds mandated by the standards). The root cause of the failing devices was in some cases proved to be latent defects related to cell-internal faults [1]. These defects are not covered by the state-of-the-art fault models (e.g., stuck-at and transition delay faults) because in these models, the common assumption is that a fault can only be present on the connections between cells (i.e., the ports of the technology library cells). Other fault models were proposed in order to target these cases, such as N-detect [2] and Gate Exhaustive testing [3]. However, these fault models are not applicable for industrial-scale circuits due to the fact that they result in excessively high test costs in terms of time and tester memory footprint. On the other hand, the cell-aware fault model [4] has been proposed as a solution to systematically target all cell-internal defects of a technology library with great success. Researchers have presented results from high volume production tests that showed a significant reduction in the defective-parts-per-million (dppm) rate.

Although these aforementioned benefits of cell-aware testing (CAT) regard the end-of-manufacturing test, the in-field test can also benefit from it. Most dominant fault models share the common assumption that a fault is not occurring in the internal part of the cells. Thus, the same latent defects (if not caught early) could still be a serious threat in a safety-critical system. CAT, being able to amend these defects, has the potential to be included as a complementary fault model in the upcoming safety standards revisions. Furthermore, in the area of advanced semiconductor technology manufacturing, where silicon aging is a prime concern, the cell-aware test seems to be a valuable solution.

However, the issue of the identification of the functionally untestable faults (i.e., those faults that will never be able to produce any failure in the considered operational scenario) under the cell-aware fault model remains an open topic. In this paper, we face this issue for the first time to the best of our knowledge and present a method based on Bounded Model Checking (BMC) to systematically identify the untestable static cell-aware faults in combinational cells in a microprocessor given a set of functional constraints. To demonstrate its effectiveness, our method was applied to a RISC-V processor with a minimum functional constraint set i.e., the minimum

constraints required to enable a functional behavior of the system. That is, no assumptions about the software executed on the core were made.

In Section II we present the state-of-the-art and previous works related to the identification of untestable faults in microprocessors, in Section III we present our formal method's approach to tackle the problem, in Section IV we present our experimental results and in Section V we draw some conclusions and provide some insight on our future work.

II. Background

A. Functionally Untestable Faults

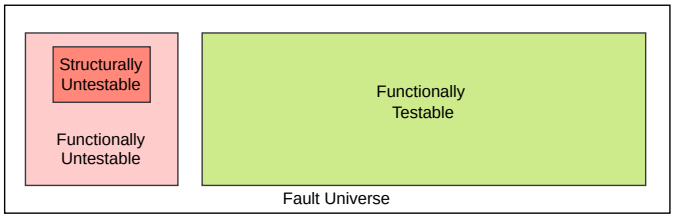

Fig. 1. Fault universe

Untestable faults are divided into two main categories, as depicted in fig. 1; *structurally* and *functionally* untestable faults. The former set, which is a subset of the latter, is connected to fault sites that cannot be forced to a specific logic value due to (i) being unconnected (redundant connection) or (ii) being tied to 0 or 1. Furthermore, the set contains faults that can be excited but whose effects cannot be propagated to an observable point. The research community has extensively studied the problem of structurally untestable fault identification [5, 6, 7]. The latter set is a superset of the previous one, which also contains faults that, although excitable and propagatable to an observable point of the circuit, are not able to produce any failure under the considered operational scenario. For example, assuming a processor as the underlying circuit, faults in the debugger module might be testable by DfT infrastructures (e.g., scan) yet, since the debug unit remains inactive when the system is in functional mode, it is not possible to excite and/or propagate them.

For in-field functional testing, the most used fault models currently employed by the industry are the stuck-at and the transition delay fault model. The problem of the identification of untestable faults has been extensively studied in the past for both fault models [8, 9]. As researchers have showcased, given the mission application profile of the system, it is possible to discover a large volume of functionally untestable faults (more than 10% of the total faults). In order to distinguish between functionally testable and untestable faults, an analysis must be performed on the application code and system configuration in order to identify which functional blocks for the circuit under test (CUT) are fully, partially used, or unused (e.g., the debugger module) by the application and to analyze the propagation chances of any fault. Currently, this analysis is mainly performed in a manual manner with reduced support by the EDA tools.

In a mission-critical system the circuit is expected to perform a well-defined set of functions in specific configurations (e.g., in terms of used instructions or memory address space). Among the several possible solutions, the Failure Mode, Effects and Criticality Analysis (FMECA) is in charge to identify which faults are not able to produce any failure (*safe* faults according to ISO26262). Since FMECA is barely automated at the moment, the issue of identifying such faults is a major concern for the industry.

Numerous solutions have been proposed in the past by the research community, considering various fault models. In [10], the authors propose a method based on model-checking in order to identify functionally untestable stuck-at faults in the registers of a circuit. In [11], the researchers propose techniques based on local multi-node conflicts derived from the circuits' sequential implications to systematically identify functionally untestable faults. Regarding the transition delay fault model, in [9, 12] the authors present a methods based on static-logic implications and implication graphs to identify functionally untestable transition delay faults.

The subject has also been extensively studied under other fault models. In [13], the authors study the problem of untestable bridging faults identification and propose a low-cost solution based on iterative logic arrays and symbolic simulation. In [14] the author proposes a methodology to identify untestable faults under the two-cycle gate-exhaustive model based on the launch-on-capture and launch-on-shift techniques.

Up to our knowledge, this is the first paper that proposes a complete methodology to identify functionally untestable faults under the static cell-aware fault model.

B. Cell Aware Test and User Defined Fault Models

The quality requirements imposed on the industry in recent years are becoming increasingly tougher. This means that the manufacturers have to refine their test flow in order to improve the defect coverage of their products. However, as mentioned previously, an increasing number of failing devices has been reported by the consumers to their suppliers although sufficient coverage percentages have been achieved under the state-of-the-art fault models. These numerous test escapes led to cell-aware and other fault models to be developed. Instead of focusing only on defects outside the cell and only its interconnections with neighboring cells, the internal structure is also included in the fault model generation.

Typically, each cell's behavior is simulated via an electrical simulator (e.g., SPICE) under different defects and operating conditions. A cell-aware fault model is derived from the resulting defect injection campaign, called Cell Test Model (CTM) [15], or User Defined Fault Model (UDFM) [16]. During automatic test pattern generation (ATPG), the cell-aware fault model is applied to the cells in the circuit which approximates the faulty behavior of the defective cell. Experimental results have showcased the effectiveness of CAT in industrial scale circuits in terms of reduced dppm figures [4]. Ultimately, test escapes related to internal-cell defects can be effectively

targeted by CAT or even custom fault models [17]. Although it is true that the generation of the cell test models can be time consuming due to SPICE simulations being computationally expensive, this process needs to happen only once for each new technology library.

III. Proposed Approach

Given the gate-level description of a pipelined processor and a cell-aware fault model, we aim to identify the static cell-aware faults on combinational cells that are functionally untestable. In other words, we want to identify the faults for which no test stimuli can be generated for under the given functional constraints. The set of functional constraints varies according to the application. For instance, certain functionality limitations (e.g., subset of the ISA) or configurations (e.g., limited memory address space) can result in functionally untestable faults. For this reason, we incorporate the constraints in our BMC-based ATPG using a *validity checker module* (VCM) which is an efficient method to accurately define and enforce functional constraints using an auxiliary circuit observing the CUT.

In Section III-A we first present the process of transforming the cell-aware fault model into the BMC instance. In Section III-B we will then present how the functional constraints are applied to the BMC instance.

A. Cell-Aware Test via Bounded Model Checking

BMC is a well known technique that has been used extensively in the test and reliability domain for a variety of problems: From software-based self-test generation purposes under the stuck-at [18] and transition delay [19] fault models up to functional stress stimuli generation [20] and untestable fault identification [10]. The core concept of BMC is to extract the Boolean formula of the CUT and convert it into a conjunctive normal form (CNF). Then, custom requirements are encoded on top in the form of properties (*invariances*). For example, the textual definition of the property for identifying if a fault X is uncontrollable would be: "Can the fault site X be assigned to both logic values?". After the BMC instance is formulated, the BMC solver is started with the task of identifying a model (solution) for the BMC instance. The solver unrolls and solves the BMC problem incrementally by typically translating it to multiple satisfiability (SAT) instances that are solved by a SAT-solver, up until a predefined maximum unrolling depth k is reached or a timeout is reached.

For test generation the cell-aware fault model specifies a set of faults per cell of the technology library in the form of defect matrices. Each fault can have one or multiple test alternatives that excite the fault. Table I contains a simplified defect matrix representing a defect in an AND gate with two inputs `A` and `B`, and one output `O`. This fault model only contains the input combinations for which the cell's outputs differ from the fault-free outputs. In this example, the cell has two different entries, called *test alternatives* in the following, with input combinations, called *fault conditions*, for which the fault effect of the respective combination is made visible to the gate's output port.

Alternative No.	A	B	O
1	1	0	1
2	1	1	0

TABLE I
Example defect matrix

For each fault in the cell-aware fault model an ATPG process is started. This process builds a BMC instance for the defect matrix and solves the BMC instance generating a test pattern or reports it to be unreachable. In the case that a test pattern could be generated, the fault is marked testable and a fault dropping simulation is performed to check for other detectable faults. In case no test pattern could be generated the fault is marked untestable.

In the case of the example from Table I, a BMC instance encoding both test alternatives is generated. The faulty gate is encoded according to the formula shown in Figure 2, where the first two lines encode the faulty behavior according to the defect matrix; the third line encodes the behavior of the gate when none of the test alternatives is applied to the gate's inputs and is equal to the fault-free gate behavior.

$$
\begin{aligned}
A \wedge \neg B &\rightarrow \quad O \\
A \wedge \quad B &\rightarrow \neg O \\
\neg(A \wedge \neg B) \wedge \neg(A \wedge B) &\rightarrow \underbrace{(A \wedge B \leftrightarrow O)}_{\text{fault-free behavior}}
\end{aligned}
$$

Fig. 2. Encoding of faulty AND gate for example model

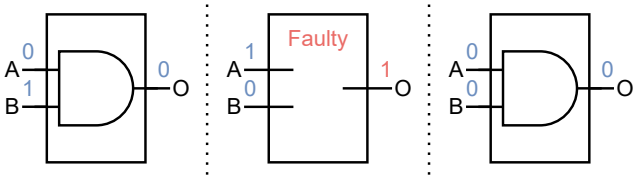

Fig. 3. Fault injection on AND gate for example model

Additionally, the BMC instance is extended to enforce a fault activation in at least one timeframe. This is shown in Figure 3, where the middle timeframe is sensitized, while the first and last timeframes show no fault behavior. The propagation of the fault effect is then enforced to at least one primary output.

B. Validity Checking

Functional constraints are applied to the CUT via a so-called *validity checker module* (VCM). The VCM is a circuit only used for ATPG that contains the functional constraints. Similarly to the CUT, the VCM is synthesized and encoded into the CUT's CNF. As shown in Figure 4, the VCM observes selected inputs, outputs, and internal signals of the CUT's

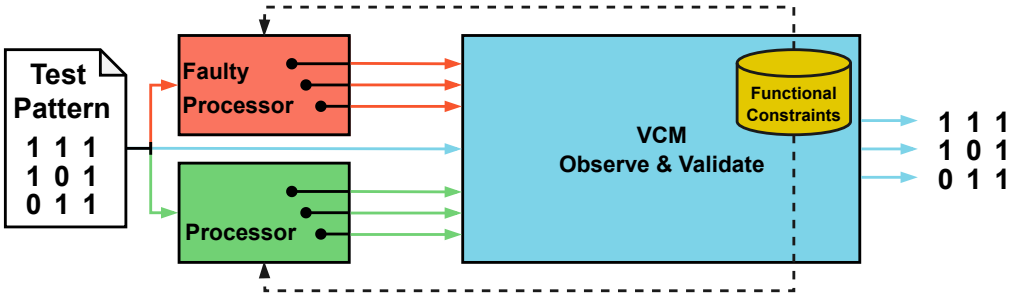

Fig. 4. Validity Checker Module

miter circuit and validates the correct functional behavior of the CUT according to the functional constraints. The computed validation results in the form of multiple so-called *constraint-valid* signals are connected to the VCM's outputs. During the creation of the BMC instance, the VCM's outputs are forced to have a high (constraint valid) value which leads to the encoded constraints being enforced. During fault dropping simulation, the constraints are validated by observing the VCM's outputs and only regarding a fault as detected if all constraint outputs are high (constraint valid) until the fault has fully been propagated i.e., it has reached a primary output.

IV. Results

The described approach has been implemented as workflow via the FreiTest ATPG framework actively developed by the University of Freiburg. This required changes that included the addition of the new cell-aware fault model and a workflow to orchestrate the BMC and the fault dropping simulation. In total roughly 600 lines of `C++` code have been written for implementing the cell-aware fault injection and roughly 350 lines for the workflow itself.

According to the application running on the CUT, a different set of functional constraints may apply. We consider the general case where the set of minimal constraints is chosen that ensures the functional behavior of the processor. The minimal constraint set includes the following constraints and ensures that the ATPG only allows valid functional behavior.

- Valid processor control (reset, run, etc.)
- Valid pipeline control (stall, IF misses)
- Valid executed instructions (ISA)
- Valid control and status registers (CSRs)
- Disable interrupts
- Disable auxiliary processing unit (APU)
- Disable debug interface

The executed instructions of the processor are constrained on the instruction bus itself by encoding the RISC-V ISA opcodes into the VCM by automatically transforming them from the official RISC-V opcode specifications into Verilog code [21]. The interrupts are disabled assuming that typically the interrupts are disabled at the start of an in-field test.

A total of 1600 `SystemVerilog` code lines have been used for specifying the constraints via the VCM. 1500 of them are automatically generated wire declarations and connections for observing decoded RISC-V instructions and operands. The VCM is synthesized with `Yosys` and the resulting gate-level VCM has a size of 405 gates.

In order to validate the proposed approach we evaluated it resorting to the processor `RI5CY` synthesized for the Nangate 45nm PDK [22]. The BMC depth has been set to 50 timeframes and the timeout to 5 minutes. The experiments have been run on an AMD Threadripper 3970X system (32 cores, 64 threads) with 256 GB of RAM and a dual AMD EPYC 7343 system (2x16 cores, 2x32 threads) with 2 TB of RAM in parallel.

The `RI5CY` core is a 4 stage 32-bit RISC-V in-order processor core. The ISA of `RI5CY` was extended to support multiple additional instructions, including hardware loops, post-increment load and store instructions, and additional ALU instructions that are not part of the standard RV ISA. `RI5CY` has become a popular core for a wide variety of applications, especially for IoT designs.

For evaluation two fault models in UDFM format are used. The first fault model is created for comparison with the conventional stuck-at fault model. It is created by a custom tool and the resulting cell-aware fault model is equivalent to a simple stuck-at model that only contains stuck-at faults at all the input and output ports of the cells. Note that this fault list contains equivalent faults. This UDFM allows for simplified comparison with conventional stuck-at simulation results. The UDFM has 534 faults with a total of 2,410 test alternatives. The second fault model is a cell-aware fault model generated by a commercial tool for the Nangate 45nm PDK which is converted into the UDFM format. It contains fault models for testable open, short, and transistor open and short defects derived from Nangate 45nm PDK SPICE cell models that have been extended with parasitic elements from the layout information. This UDFM has 1,244 faults with a total of 10,546 test alternatives.

Table II shows the results of our method. The first half of the table concerns the untestability analysis under the stuck-at fault model whereas the second half under the cell-aware respectively. The reported percentages for the two fault models are not correlated i.e., one is not a subset of the other. As mentioned earlier, the constraint set we have applied is the minimum functional one. That is, no further constraints derived from assumptions about the executed program(s) or

Functional Unit	Combinational Cells	Sequential Cells	Stuck-At Total Faults	Stuck-At Structurally Untestable	Stuck-At Functionally Untestable	Cell-Aware Total Faults	Cell-Aware Structurally Untestable	Cell-Aware Functionally Untestable
if_stage	2,013	304	10,979	123 (1.12 %)	179 (1.63 %)	22,387	161 (0.72 %)	218 (0.97 %)
id_stage	2,944	590	16,240	226 (1.39 %)	196 (1.21 %)	31,336	426 (1.36 %)	247 (0.79 %)
id_stage/regs	3,561	992	27,079	0 (0.00 %)	0 (0.00 %)	61,878	26 (0.04 %)	0 (0.00 %)
ex_stage	168	6	593	26 (4.38 %)	2 (0.34 %)	904	50 (5.53 %)	1 (0.11 %)
ex_stage/alu	4,583	107	24,948	18 (0.07 %)	4 (0.02 %)	48,782	201 (0.41 %)	7 (0.01 %)
ex_stage/mult	3,814	3	24,381	2 (0.01 %)	19 (0.08 %)	83,360	69 (0.08 %)	69 (0.08 %)
ls_unit	712	40	4,511	0 (0.00 %)	15 (0.33 %)	9,121	14 (0.15 %)	21 (0.23 %)
cs_registers	2,088	974	15,730	188 (1.20 %)	3,679 (23.39 %)	36,479	363 (1.00 %)	6,378 (17.48 %)
pmp_unit	11,983	1	50,591	4 (0.01 %)	22,744 (44.96 %)	98,792	144 (0.15 %)	31,790 (32.18 %)
top	25	1	118	2 (1.69 %)	8 (6.78 %)	150	2 (1.33 %)	11 (7.33 %)
total	31,891	3,018	175,170	589 (0.34 %)	26,846 (15.33 %)	393,189	1,456 (0.37 %)	38,742 (9.85 %)
		CPU Time (hours)	128.96 h (2.42 s per fault on EPYC system)			285.27 h (2.50 s per fault on EPYC system)		

TABLE II
UNTESTABLE STUCK-AT AND CELL-AWARE FAULTS IDENTIFIED BY FREITEST

the system configuration are made. For instance, combinatorial logic blocks in the fetch and decode stages that are driven by the reset and the clock gating enabling signals are marked as safe due to the fact that they are always expected to be forced to fixed values in order to ensure functional behavior. Furthermore, extra functionally untestable faults arise for logic related to the program counter, which is bound to a specific start and end address as is typical to happen in an embedded system in a mission-critical system with limited or shared resources with other peripheral devices that share the same memory which is divided into distinctive regions. In total, 1.12 % / 1.63 % of structurally / functionally untestable faults have been identified for the fetch stage for the stuck-at fault model and 0.72 % / 0.97 % for the cell-aware respectively.

Since the solver is completely agnostic of the architecture's ISA, we enforce all valid instructions with valid operands and operand ranges. Hence, any kind of interrupt stemming from an invalid instruction during decode is not triggered, and thus, the logic blocks related to handling such cases, nest functionally untestable faults. Furthermore, certain control and status registers related to the generation and the handling of interrupts also contain functionally untestable faults. This is because, in the in-field test scenario, the test is scheduled during idle periods of the system; hence, the interrupts are being disabled during this time to avoid any unpredictable scenario occurring which could potentially have catastrophic consequences to the system and finally to the user(s). For the decode stage, we have identified 1.39 % / 1.21 % of structurally / functionally untestable faults for the stuck-at and 1.36 % / 0.79 % for the cell-aware.

The highest amount of untestable faults is detected in the *physical memory protection* (PMP) unit and the CSRs. The PMP unit provides machine-mode control and status registers per hardware thread to allow memory access privileges to be specified for each physical memory region. The unit is performing checks on both the instruction and data memory of the system and is accessible via dedicated CSRs. Certain registers in our configuration were disallowed, which is something that leads to a large number of faults inside the PMP unit to be identified as safe. Furthermore, regarding the CSR unit, the number of safe faults identified is a result of only user level [23] being considered in the processor configuration. For the CSRs, we have identified 1.20 % / 23.39% of structurally / functionally untestable faults for the stuck-at and 1.00 % / 17.48 % respectively for the cell-aware. For the PMP unit, the percentages are 0.01 % / 44.96 % and 0.15 % / 32.18 % respectively.

Overall, a total of 9.85% of functionally untestable faults have been identified on the whole processor by our method for the static, combinational cell-aware fault model and 15.33% for the stuck-at model. To verify the results of our method, we have performed a fault simulation of a manually written Software Test Library (STL) that has been developed for the stuck-at fault model, reaching $\approx 60\%$ of fault coverage and compared with our findings. Two flows were developed in `Z01X`. One for the stuck-at and one for the cell-aware model. The results showed that the proven untestable faults in `Z01X` which correspond to structurally untestable faults were a subset of the faults we have detected with our method. Lastly, the untested faults in `Z01X` were proven to be functionally untestable faults by our method with a minute percentage (less than 0.1 %) of mismatches due to configuration differences.

V. CONCLUSIONS

Functional at-speed test is widely adopted for both end-of-manufacturing test from the suppliers and also in-field test from the customers. The generation of such test libraries, which is primarily done with limited automation, requires a lot of time and is further complicated by the presence of functionally untestable faults that negatively impact the computation of the test coverage. In the safety critical domain specifically, the identification of these faults is of utmost importance in order to meet the coverage threshold imposed by the respective standards.

Notwithstanding the success of stuck-at and transition delay fault models, which are the dominant fault models used especially for stimuli destined for safety critical systems, a

significant amount of failing devices has been reported in recent years due to internal cell defects, that the aforementioned fault models did not account for. CAT was developed to amend this issue with great success.

In this paper, for the first time to our knowledge, we present a BMC-based method to identify functionally untestable faults in microprocessor circuits, also considering static CAT faults. Our method was applied to the RISC-V processor `RI5CY` for the stuck-at and static cell-aware fault models and was able to identify 15.33 % (stuck-at) and 9.85 % (cell-aware) of functionally untestable faults, starting from a minimum set of functional constraints. Clearly, the reported results are strongly dependent on the considered set of constraints but show the ability of the method to effectively solve the faced problem. The results have been compared with a stuck-at test program reaching $\approx 60\%$ of functional test coverage that has been fault simulated for both fault models. As a future work, we plan to extend our method to further identify dynamic untestable cell-aware faults.

Acknowledgment

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project Scale4Edge under contract no. 16ME0132.

References

[1] F. Hapke et al. “Defect-oriented cell-aware ATPG and fault simulation for industrial cell libraries and designs”. In: *International Test Conference*. 2009. DOI: 10.1109/TEST.2009.5355741.

[2] I. Pomeranz and S.M. Reddy. “On n-detection test sets and variable n-detection test sets for transition faults”. In: *VLSI Test Symposium*. 1999. DOI: 10.1109/VTEST.1999.766662.

[3] Y. C. Kyoung, S. Mitra, and E.J. McCluskey. “Gate exhaustive testing”. In: *IEEE International Conference on Test*. 2005. DOI: 10.1109/TEST.2005.1584040.

[4] F. Hapke et al. “Cell-Aware Test”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33 (2014), pp. 1396–1409. DOI: 10.1109/TCAD.2014.2323216.

[5] H-C. Liang et al. “A sequential redundant fault identification scheme and its application to test generation”. In: *Asian Test Symposium (ATS)*. 1994. DOI: 10.1109/ATS.1994.367253.

[6] D. E. Long et al. “FILL and FUNI: Algorithms to Identify Illegal States and Sequentially Untestable Faults”. In: *ACM Transactions on Design Automation of Electronic Systems* 5 (2000), pp. 631–657. DOI: 10.1145/348019.348311.

[7] N. I. Deligiannis et al. “New Techniques for the Automatic Identification of Uncontrollable Lines in a CPU Core”. In: *IEEE VLSI Test Symposium*. 2021. DOI: 10.1109/VTS50974.2021.9441040.

[8] P. Bernardi et al. “On-line functionally untestable fault identification in embedded processor cores”. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2013. DOI: 10.7873/DATE.2013.298.

[9] X. Liu and M.S. Hsiao. “On identifying functionally untestable transition faults”. In: *Ninth IEEE International High-Level Design Validation and Test Workshop*. 2004. DOI: 10.1109/HLDVT.2004.1431252.

[10] J. Raik et al. “Untestable Fault Identification in Sequential Circuits Using Model-Checking”. In: *Asian Test Symposium*. 2008. DOI: 10.1109/ATS.2008.22.

[11] M. Syal and M.S. Hsiao. “New techniques for untestable fault identification in sequential circuits”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25 (2006), pp. 1117–1131. DOI: 10.1109/TCAD.2005.855967.

[12] K. Heragu et al. “Fast identification of untestable delay faults using implications”. In: *IEEE International Conference on Computer Aided Design (ICCAD)*. 1997. DOI: 10.1109/ICCAD.1997.643606.

[13] M. Syal et al. “Efficient implication-based untestable bridge fault identifier”. In: *VLSI Test Symposium*. 2003. DOI: 10.1109/VTEST.2003.1197680.

[14] I. Pomeranz. “Efficient Identification of Undetectable Two-Cycle Gate-Exhaustive Faults”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41 (2022), pp. 776–783. DOI: 10.1109/TCAD.2021.3062767.

[15] Synopsys. *CMGen User Guide*. 2022.

[16] Siemens EDA. *Tessent Shell Reference Manual*. 2019.

[17] S. Kundu et al. “Using Custom Fault Models to Improve Understanding of Silicon Failures”. In: *IEEE International Test Conference*. 2022. DOI: 10.1109/ITC50671.2022.00043.

[18] T. Faller et al. “Constraint-Based Automatic SBST Generation for RISC-V Processor Families”. In: *European Test Symposium*. 2023.

[19] Y. Zhang et al. “BMC-Based Temperature-Aware SBST for Worst-Case Delay Fault Testing Under High Temperature”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30 (2022), pp. 1677–1690. DOI: 10.1109/TVLSI.2022.3186946.

[20] N. I. Deligiannis et al. “Automating the Generation of Functional Stress Inducing Stimuli for Burn-In Testing”. In: *European Test Symposium*. 2023.

[21] RISC-V International. *RISC-V Opcodes*. https://github.com/riscv/riscv-opcodes. 2022.

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

[23] RISC-V International. *Privileged Specification version 20211203*. https://riscv.org/technical/specifications/.