Delft University of Technology

An automated formal-based approach for reducing undetected faults in ISO 26262 hardware compliant designs

Augusto da Silva, Felipe; Bagbaba, Ahmet Cagri; Hamdioui, Said; Sauer, Christian

DOI
10.1109/ITC50571.2021.00047

Publication date
2021

Document Version
Final published version

Published in
Proceedings - 2021 IEEE International Test Conference, ITC 2021

Citation (APA)
Augusto da Silva, F., Bagbaba, A. C., Hamdioui, S., & Sauer, C. (2021). An automated formal-based approach for reducing undetected faults in ISO 26262 hardware compliant designs. In R. S. Bilof (Ed.), *Proceedings - 2021 IEEE International Test Conference, ITC 2021: Proceedings* (pp. 329-333). Article 9611308 (Proceedings - International Test Conference). IEEE. https://doi.org/10.1109/ITC50571.2021.00047

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

An automated formal-based approach for reducing undetected faults in ISO 26262 hardware compliant designs

Felipe Augusto da Silva[*][†] Ahmet Cagri Bagbaba[*] Said Hamdioui[†] Christian Sauer[*]

[*]Cadence Design Systems
Munich, Germany

[†]Delft University of Technology
Delft, The Netherlands

2021 IEEE International Test Conference (ITC) | 978-1-6654-1695-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/ITC50571.2021.00047

Abstract—**The current demands for developing safe automotive applications require extensive analysis and evaluation of potential random hardware faults. In general, part of this analysis is manually performed by experts, resulting in an expensive, time-consuming, and error-prone process. This paper proposes an automated approach to classify faults overlooked by traditional methods. Our methodology deploys code coverage and formal to identify nodes that do not disrupt safety-critical functionalities, enabling the classification of additional faults. The approach is validated based on an Automotive CPU, according to ISO 26262 guidelines. The results show an improvement in Diagnostic Coverage of 1.15%, increasing the Single Point Fault Metric (SPFM) to 97.3%, enabling ASIL C compliance without any hardware redundancy.**

Keywords - ISO 26262; Safe Faults; Fault Injection; Formal Methods; Simulation; Functional Safety; Verification.

I. INTRODUCTION

The increasing complexity in automotive applications is causing an escalation in the demands for electronic devices. An Integrated Circuit (IC) that implements safety-critical applications, such as autonomous driving, must incorporate mechanisms to reduce the risk of failures resulting in life-threatening situations. For such applications, engineers must analyze a huge fault space and demonstrate they cannot affect safety-critical functionalities. In the most advanced automotive ICs, where the fault space is related to millions of design components, this process becomes challenging. Fault analysis and classification according to safety standards is an arduous task that requires extensive knowledge of the design functionalities. Therefore, there is a high demand for methodologies that can speed up fault classification reducing time to market and verification costs.

Fault Injection (FI) Simulation is the state-of-the-art method to evaluate the fault effects in the operation of Automotive ICs. The simulation of the design behavior during fault injection allows monitoring the impact of such fault on intended functionalities and the activation of Safety Mechanisms when needed; each fault is then classified depending on its influence on the design functionality. Simulation and optimization of FI campaigns are demanded research topics [1]. In general, it is not possible to evaluate the design to all possible inputs; therefore, not all faults can be classified. Hence, alternative methodologies are needed to prove whether these could disturb safety-critical functionalities. *Formal Methods* can be employed to leverage the classification of faults. The ability of formal techniques in analyzing the design behavior for all possible combinations of inputs contributes to the fault space analysis [2]. The combination of FI Simulation and Formal techniques was also examined [3]; such a hybrid approach makes use of the strengths of both technologies. Nevertheless, even with such an approach, there will be residual unclassified faults. In [4], the authors proposed a methodology to reduce the subset of unclassified faults further and identify, e.g., those that cannot affect safety-critical functionalities. Even though the presented results are promising, part of the process relies on *manual* analysis by experts. Consequently, an automated and reliable methodology that decreases manual efforts while fulfilling ISO 26262 requirements is needed.

Our work advances state-of-the-art by setting steps toward fully automated fault space analysis for ISO 26262. Our focus is on identifying the nature of each fault in the fault space. Suppose the effect of a fault does not affect safety-related functionalities. In that case, the fault can be classified as a Safe fault, increasing compliance to safety standards. Initially, we deploy code coverage techniques to identify design elements that are not operated during functional verification. The code coverage reports are examined by an automated tool that generates formal properties to reproduce the observed behavior. Finally, we configure all the properties in a Formal analysis tool, improving the tool's efficiency. The additional classification causes an immediate increase in the Safety Metrics, enabling compliance with ISO 26262. The main contributions of this work are:

- An automated approach for identifying the nature of each fault in the fault space, i.e., faults not affecting safety-critical outputs;
- Validation of the proposed methodology following ISO 26262 Functional Safety requirements (including an FMEDA, Failure Rate analysis, and Safety Metrics calculation) in an Automotive CPU;
- Evidence of the methodology efficiency by detailing how the increase of 1.15% in the Diagnostic Coverage supports a Single Point Fault Metric (SPFM) of 97.3%, enabling ASIL C compliance without hardware redundancy.

The rest of the paper is organized as follows: Fault classification methodologies are introduced in Section II. Section III describes the proposed methodology. Section IV explains the validation process and results. Section V concludes.

Short Paper

Authorized licensed use limited to: TU Delft Library. Downloaded on November 25,2021 at 08:18:42 UTC from IEEE Xplore. Restrictions apply.

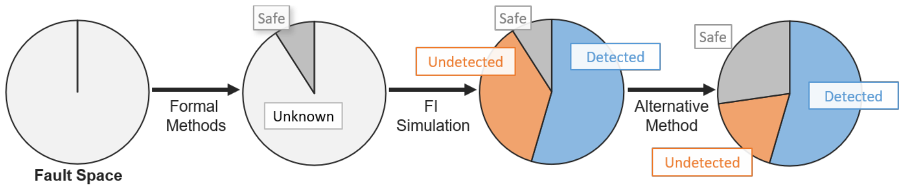

Fig. 1. Fault Classification Flow

II. FAULT CLASSIFICATION AND EXISTING METHODS

This section first defines the faults as seen by ISO 26262 and, after that, briefly describes the commonly known two methods for fault classification, namely Fault Injection, and Formal Methods.

A. Fault Classification

According to ISO 26262, the safety analysis of a Hardware device must consider the classification of the effect of faults on the circuit's functional behavior. The fault space for this analysis includes Stuck-At-0 (SA0) and Stuck-At-1 (SA1) faults at all inputs and outputs of the design gates. The philosophy behind the fault classification is to identify three fault sub-classes through the analysis of their behavior:

- Safe faults: these are faults that do not cause any disturbance of safety-critical functionalities.
- Detected faults: these are faults that can disturb the safety-critical functionalities; hence *Safety Mechanisms (SMs)* are deployed to correct them and ensure that they become innocent.
- Undetected: these are faults for which the effect is unknown; they can be either safe, detected, or even dangerous faults without associated safety mechanisms.

Figure 1 illustrates the process of fault classification and analysis for ISO 26262. The process starts with the definition of the fault space; all faults are classified then as Unknown. Next, Formal Methods are deployed for the identification of Safe faults. After that, FI Simulation is applied to the remaining faults to evaluate the efficiency of Safety Mechanisms (SM); the FI Simulation classifies the faults as Detected when SM covers them and as Undetected otherwise. This initial assessment allows the calculation of the Diagnostic Coverage (DC); it represents the efficiency of Safety Mechanisms, and it is essential for ISO 26262 compliance. If the desired DC is achieved, the process ends. Otherwise, an alternative methodology is necessary to classify the Undetected residual faults. The reduction in the number of Undetected faults contributes to improving the DC, hence, to ISO 26262 compliance. The DC is calculated according to the equation 1:

$$DC = (Detected)/(Total - Safe) \tag{1}$$

where $Detected$ presents the number of faults classified as Detected by FI Simulation, $Total$ is the size of the fault space, and $Safe$ the number of faults that cannot disturb safety-critical functionalities. The following sections detail the implementation of the lead technologies for fault classification.

B. Fault Injection Simulation

Fault Injection (FI) Simulation is widely used and available in a variety of tools. These tools can analyze a Register Transfer Level (RTL) or Gate-Level (GTL) descriptions of an IC by simulating the IC behavior for a given test input. The effect of a fault is determined by comparing the behavior of the design with and without faults. The flow implemented by FI Simulation Tools is as follows:

1) Elaboration of RTL/GTL design description.
2) *Fault list generation and optimization*.
3) *Fault-free simulation*: fault-free behavior of design is simulated for recording the observation points reference values. The observation points, defined by the user, are (1) functional strobes, which store information related to functional outputs, and (2) checker strobes, which indicates how the Safety Mechanisms react.
4) *Fault injection simulation*: For each fault, the faulty design behavior is simulated. The observation points are then compared to the reference values; the differences in the values determine the design behavior under fault.
5) *Fault classification*: If the fault effect is perceived in a checker strobe, then the fault is classified as Detected. Otherwise, the fault is classified as Undetected.

C. Formal Methods

Formal tools can verify a circuit in the global scope, considering every evaluation context and test stimuli. Consequently, these tools can exhaustively prove that a fault can never produce any failure.

The method consists of creating a representation of the boolean function implemented by the design under test, where formal proves can be deployed. Formal tools can automatically generate properties to determine the Activation and Propagation of faults. Activation analysis indicates whether any combination of inputs can functionally activate a fault. Propagation analysis verifies if there is a combination of input values that provoke fault propagation to an output. In addition, the tools can analyze the physical characteristics of the design to identify faults that cannot reach the functional strobes. Faults that are recognized by these analyses cannot cause failures; consequently, they are safe.

As formal analysis is resource hungry and limited due to the state explosion problem, the Undetected residual faults still require an alternative classification methodology.

Authorized licensed use limited to: TU Delft Library. Downloaded on November 25,2021 at 08:18:42 UTC from IEEE Xplore. Restrictions apply.

III. TESTABLE SAFE FAULTS IDENTIFICATION

The ISO 26262 Hardware Architectural Metrics determines the effectiveness of designs to cope with random hardware failures [5]. The failures addressed by these metrics are limited to elements that can contribute to the violation of safety goals; these define the required mitigation of hazardous events to avoid unreasonable risks caused by malfunctions. During the system development phase, safety goals are decomposed into a Functional Safety Concept that defines the requirements for the hardware architecture. However, the development of a hardware design demands additional components that are not related to the safety concept; these components will decrease the compliance to Hardware Architectural Metrics, even though they may not violate safety goals in case of faults. If that is the case, these components can be identified by their potential to disrupt safety goals, increasing Safe faults classification. Safe faults can be divided into two categories:

- Untestable: there is no combination of test stimuli that induce the fault to affect the functionality of the design. Also know as redundant in the DfT community.
- Testable: faults that can affect the output of the design with a suitable test stimulus. Nevertheless, they cannot affect safety-critical functionalities.

Testable Safe faults are faults that do not cause any deviation of the safety-related operational mode. Identifying the nature of faults (being safe or not) typically requires the judgment of hardware design experts; this is time-consuming and prone to errors.

Our approach deploys the code coverage to understand the design operational behavior; this behavior is automatically translated into formal properties. By reproducing the design operational behavior in a Formal tool, we decrease the space exploration, allowing the classification of additional faults. Next, the main steps of the method will be explained.

A. Code Coverage

Code coverage is a method of assessing to what extent test cases exercise a design under test. It is essential to notice that the test cases must be a valid representation of the design functionalities; at the safety analysis stage, a functional verification environment is already available and should be deployed for the code coverage. The flow starts with the design simulation considering all the available test cases. Next, verifying the variations in all internal signals makes it possible to generate reports for block and toggle coverage. Block coverage determines whether test scenarios exercise the statements in a block. A block is a series of sequential statements without delays or control flow statements (if, case, wait, while, among others). In other words, a block is a specific state in a state machine. Toggle coverage measures the activity of the signals in the design during the simulation. It provides information on untoggled signals or signals that remain constant during the simulation.

The metrics from the code coverage provide information regarding design parts that may not be safety-related. For instance, by recognizing states that are never activated due to block coverage, we can identify design modes that are not related to safety functionalities. Similarly, untoggled signals can highlight important details of the design, like invalid configurations, not utilized functions, status monitors, among others. The combination of toggle and block coverage provides additional information about specific functionalities. For example, the missing toggle in a control signal may be responsible for never activating a block in a state machine. Also, by bypassing a specific state, another signal may not be toggled.

B. Automated Code Coverage Analysis

The automation process aims to translate the code coverage behavior into formal properties. Suppose we can replicate the functional behavior of the design in the formal environment; this would reduce the space exploration, allowing the identification of not safety-relevant nodes. As the formal environment includes operational constraints, the newly identified faults are Testable Safe fault.

The automated code coverage analysis tool will translate design elements from the code coverage report into *assume statements* or *fault-propagation barriers*. *Assume statements* enable constraints configuration for formal analysis. When an expression is assumed, the formal verification tool constrains the design inputs accordingly. The role of the assume construct is useful in the confirmation of the design functional configuration. Also, by configuring the expected behavior of the design, we increase the capacity of Safe faults identification by limiting the test stimuli space. *Fault-propagation barriers* are design elements that can block the propagation of a fault. Faults that propagate only to certain elements may not affect safety-critical functionalities. Consequently, these can be Testable Safe faults.

The automated code coverage analysis must also examine the design element types and values. Input ports of the design instances are suitable candidates to *assume statements*. Output ports, on the other hand, are better candidates for *fault-propagation barriers*. The automated code coverage analysis tool automatically retrieves the signal types and values by analyzing the coverage report and source code. The generated formal properties are compiled in a text report. The output includes the *assume statements*, *fault-propagation barriers*, and supplementary information that lead to the creation of the properties.

The only manual step of the process is revising the formal properties generated by the automated code coverage analysis tool. The coverage result by itself is not enough to distinguish the potential Testable Safe faults. To assure the formal property does not conflict with the expected behavior, engineers must review the output of the automated code coverage analysis tool. For that reason, the tool output includes supplementary information to support the review process. An over-constrained formal environment would cause false positives, invalidating the results.

Authorized licensed use limited to: TU Delft Library. Downloaded on November 25,2021 at 08:18:42 UTC from IEEE Xplore. Restrictions apply.

C. *Formal Analysis of Testable Safe Faults*

The identification of Testable Safe faults will deploy the same techniques described in Section II to identify Safe faults. The difference is that the formal environment will incorporate the results of the automated code coverage analysis tool. By constraining the environment, we enable the tool to evaluate the design in a well-specified configuration, increasing the potential for the identification of Safe faults. Additional Safe faults will be classified as Testable Safe, as they are Safe considering the functional constraints included in the environment.

IV. RESULTS

A. *Test Case*

To validate the proposed methodology, we targeted a design that represents the challenges of the automotive industry. The AutoSoC is an open-source initiative for an automotive SoC benchmark suite, based on an OpenRISC implementation [6]. The chosen configuration of the AutoSoC includes Error-Detection-Correction Codes (ECC) protection on the internal memories and a Software-Based Self-Test (SBST) approach in the form of a Software Test Libraries (STL). As definite evidence of our contribution, we intend to improve the Automotive Safety Integrity Level (ASIL) compliance of the target CPU without hardware redundancy.

As described in [6], the internal memories occupy the highest area of the AutoSoC physical device, representing 91.3% of the fault space. Based on ISO 26262 standard recommendations, we can assume the ECC provides a Diagnostic Coverage of 99% for random hardware faults, representing a satisfying coverage for the overall CPU.

For verifying the efficiency of the STL, we adopt the Cadence® Xcelium™ Fault Simulator (XFS). The XFS was configured to inject SA0 and SA1 faults at every cell port of the GTL representation of the AutoSoC deploying the STL as workload. Considering the digital area, the AutoSoC CPU contains 96,354 fault targets for Simulation. During the FI Simulation, the STL presents coverage of 52.32% by detecting 50,412 of the faults. Also, by deploying Formal methods, we can identify 8,020 Untestable Safe faults, resulting in a Diagnostic Coverage of 57.07%.

In a preliminary analysis of the Safety Mechanisms listed above, we can conclude that the memory area has sufficient coverage with the DC provided by ECC. However, the digital area of the CPU may still require further coverage for achieving the required Safety Metrics. Even though the deployed STL achieves significant DC, over 37,000 faults are still Undetected. These faults must be classified to allow compliance with the requirements of ISO 26262. The following section shows how the proposed methodology impacts the Diagnostic Coverage of the STL by classifying Undetected faults as Testable Safe.

B. *Classification of Testable Safe Faults*

The first step for the identification of the Testable Safe faults is the code coverage analysis. The AutoSoC code coverage analysis consisted of the simulation of the 100 workloads

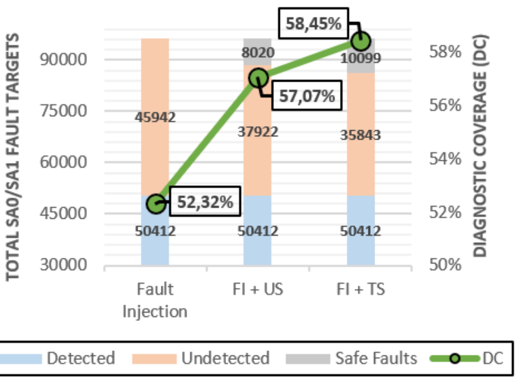

Fig. 2. STL Diagnostic Coverage Analysis

available in the benchmark. The workloads cover a variety of applications, producing a representative baseline of the CPU functionalities. Next, the automated code coverage analysis tool is deployed on the code coverage report to generate the next step's formal properties. We investigated the properties file to avoid over-constraining the formal analysis. The revision process included inspection of the RTL code and monitoring of the signals during the simulation. Also, some RTL internal signals needed to be traced to wires in the Gate level representation of the hardware. The final formal properties file consisted of 3,884 *assume statements* and 63 *fault-propagation barriers*.

Our work applies Cadence® Integrated Metrics Center (IMC) for code coverage and Cadence® JasperGold (JG) Formal Verification Platform Functional Safety Verification (FSV) for Formal Analysis. The identification of Safe faults consisted of two steps. First, we deploy JG FSV formal analysis for the identification of Untestable Safe faults. Next, we load the formal properties file into the Formal Analysis tool and repeat step one. The additional Safe faults identified in step two will be listed as Testable Safe.

Figure 2 details the results of the various analysis steps. The graph illustrates the faults classification contribution achieved during Fault Injection, Untestable Safe (US), and Testable Safe (TS) analysis. The process is incremental, always focusing on faults that were previously Undetected. Also, Figure 2 displays the calculated Diagnostic Coverage at each step. The proposed methodology can identify 2,079 additional Safe faults, resulting in a 1.38% increase in the Diagnostic Coverage.

C. *Functional Safety Analysis*

The calculation of the Diagnostic Coverage is an indication of the efficiency of each Safety Mechanism. However, to assure compliance with ISO 26262, a comprehensive Functional Safety Analysis must be performed. The analysis intends to confirm that the probability of failures in a safety-relevant system is reduced to acceptable levels. The primary methodology for the Functional Safety Analysis of hardware devices is the Failure Modes Effects and Diagnostic Analysis (FMEDA). The FMEDA correlates IC components (Gates, Flops, and

Authorized licensed use limited to: TU Delft Library. Downloaded on November 25,2021 at 08:18:42 UTC from IEEE Xplore. Restrictions apply.

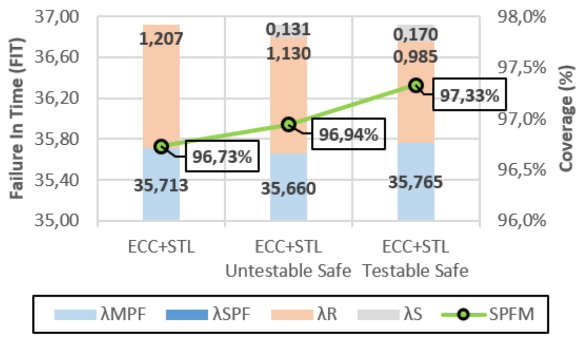

Fig. 3. Safety Metrics Analysis

Memory cells) to Failure Modes (FM). Also, considering the base Failure In Time (FIT) rate of individual IC components, the Diagnostic Coverage of Safety Mechanisms, and the Safe faults, we can determine the Residual FIT contribution of each FM.

The FMEDA starts with the definition of the FMs and the mapping of design components. For the AutoSoC, we considered ten subparts, one for each sub-block of the CPU. Each subpart includes the FMs for Data Corruption, Deadlock, Modified Execution, Exceptions, and Performance. After mapping each FM to the appropriate design components, we can incorporate the percentage of Safe faults and coverage of Safety Mechanisms (Diagnostic Coverage). The AutoSoC FMEDA comprises 75 Failure Modes mapped to 28,956 Gates, 1,983 Flops, and 316,672 Memory cells.

The Safety Metrics calculation is based on the contribution of each FM to the FIT Rate (λ). The λ of each FM can be classified according to the ISO 26262 definition of fault classes. The λSPF represents Single-Point faults that SMs do not cover. Residual (λR) describes faults Undetected by SMs. Detected faults, which could only violate a safety goal combined with a second fault, are called Multi-Point faults (λMPF). λS represents the contribution of Safe faults. The sum of the fault classes is equal to the total λ.

The classification of the λ classes is necessary for determining ASIL compliance. The ASIL requirements are expressed as target values in the form of metrics. These metrics are calculated based on the fault classes' contribution to the total λ. The metrics defined by ISO 26262 are the Single-Point Fault Metric (SPFM), the Latent Fault Metric (LFM), and the Probabilistic Metric for Random Hardware Faults (PMHF).

Figure 3 details the result of the Functional Safety Analysis of the AutoSoC considering three FMEDAs: Safety Mechanisms only (ECC+STL); Safety Mechanisms and Untestable Safe results; and Safety Mechanisms and Testable Safe results. The left axis presents the classification of the different λ classes in FIT. As the AutoSoC includes SMs in all FM, the λSPF is 0. The sum of the other λ classes represents the Total λ of the AutoSoC.

The right axis highlights the SPFM coverage. The graph illustrates how the increase in the λS, achieved by the proposed methodology, directly impacts the SPFM. The additional λS causes a decrease in λR, and therefore, an increase in the SPFM.

The additional coverage provided by the Testable Safe faults enabled an SPFM of 97.3%, achieving the minimum requirement for an ASIL C design. It is crucial noting that, as seen in Figure 3, our results allowed an immediate increase from ASIL B to C on the AutoSoC design.

Even with the increased fault classification, there is still a considerable contribution from λR (Undetected faults), which decreases the coverage restricting ASIL D compliance. The classification of the Undetected faults could be achieved by improving the STL coverage, including additional Safety Mechanisms, or adjusting the design analysis to increase the number of Safe faults. In complex designs, it is challenging to achieve 99% of SPFM without hardware redundancies.

V. CONCLUSIONS

The severe demands for tolerance to random faults are a hurdle for ICs targeting ISO 26262 safety-critical applications. As part of this process, fault analysis methods are still driven by experts, requiring manual analysis that are very expensive, time-consuming, and prone to errors. We propose an automated methodology that combines code coverage and Formal verification techniques for Safe fault identification. First, we identify design elements where a fault cannot disturb safety-critical functionalities. Next, those elements are automatically translated into formal rules, enabling the identification of Testable Safe faults. We validate our methodology on an Automotive CPU, including an FMEDA, Failure Rate analysis, and Safety Metrics calculation, according to ISO 26262 guidelines. Our approach improves the Diagnostic Coverage by 1.15%; such a coverage enables a SPFM of 97.3%, permitting ASIL C compliance. Our automated approach reduces the constraints of manual expert-based analysis, reducing the verification costs and time to market; also, it enables an accurate safety evaluation, allowing compliance to ISO 26262 without hardware redundancy.

REFERENCES

[1] A. Nardi and A. Armato, "Functional safety methodologies for automotive applications," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, nov 2017.
[2] M. Syal and M. Hsiao, "New techniques for untestable fault identification in sequential circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 6, pp. 1117–1131, jun 2006.
[3] F. Augusto da Silva, A. C. Bagbaba, S. Hamdioui, and C. Sauer, "Combining fault analysis technologies for ISO26262 functional safety verification," in *2019 IEEE 28th Asian Test Symposium (ATS)*. IEEE, dec 2019.
[4] F. Augusto da Silva, A. C. Bagbaba, S. Sartoni, R. Cantoro, M. S. Reorda, S. Hamdioui, and C. Sauer, "Determined-safe faults identification: A step towards ISO26262 hardware compliant designs," in *2020 IEEE European Test Symposium (ETS)*. IEEE, may 2020.
[5] ISO, *ISO 26262 Road Vehicles - Function Safety - Part 5: Product development at the hardware level*, International Standardization Organization Std., Dec. 2018.
[6] F. Augusto da Silva, A. C. Bagbaba, A. Ruospo, R. Mariani, G. Kanawati, E. Sanchez, M. Sonza Reorda, M. Jenihhin, S. Hamdioui, and C. Sauer, "Special session: AutoSoC - a suite of open-source automotive SoC benchmarks," in *2020 IEEE 38th VLSI Test Symposium (VTS)*. IEEE, apr 2020.

Authorized licensed use limited to: TU Delft Library. Downloaded on November 25,2021 at 08:18:42 UTC from IEEE Xplore. Restrictions apply.