Combining Fault Analysis Technologies for ISO26262 Functional Safety Verification

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

DOI
10.1109/ATS47505.2019.00024

Publication date
2020

Document Version
Accepted author manuscript

Published in
Proceedings - 2019 IEEE 28th Asian Test Symposium, ATS 2019

Combining Fault Analysis Technologies for ISO26262 Functional Safety Verification

Felipe Augusto da Silva[1,2], Ahmet Cagri Bagbaba[1], Said Hamdioui[2] and Christian Sauer[1]
[1]Cadence Design Systems, Feldkirchen, Germany - {dasilva, abagbaba, sauerc}@cadence.com
[2]Delft University of Technology, Delft, The Netherlands - {f.augustodasilva, s.hamdioui}@tudelft.nl

***Abstract*— The development of Integrated Circuits for the Automotive sector imposes in complex challenges. ISO26262 Functional Safety requirements entail extensive Fault Injection campaigns and complex analysis for the evaluation of deployed Software Tools. This paper proposes a methodology to improve Fault Analysis Tools Confidence Level (TCL) by detecting errors in the classification of faults. By combining the strengths of Automatic Test Pattern Generators (ATPG), Formal Methods and Fault Injection Simulators we are able to automatically generate a Test Environment that enables the validation of the tools and provide supplementary information about the design behavior. Our results showed fault detection rates above 99% including information to improve ISO26262 metrics calculation.**

Keywords - ISO26262; Fault Injection; Formal Methods; Simulation; Tool Confidence Level; Functional Safety; Verification; ATPG.

I. INTRODUCTION

Functional Safety Verification is one of the most challenging steps for Integrated Circuit (IC) compliance with ISO26262. Particularly for safety-critical applications such as autonomous driving, where in case of a failure, a life-threatening situation can happen. For such applications, the system must include Safety Mechanisms being able to detect up to 99% of the random faults susceptive of the design. At the IC Gate-Level representation, the number of faults can easily reach the millions figure, requiring huge efforts to analyze all of them. In addition, ISO26262 requires that all possible malfunctions of tools (used during fault analysis) have to be considered, meaning that developers have to assess the level of confidence on the outputs of a tool. The tool may require compliance with Tool Qualification requirements; this even increases the complexity of functional safety verification. Therefore, there is a high demand for effective Functional Safety Verification methodologies allowing the reduction of costs while maintaining the same levels of safety.

The commonly used method for Functional Safety Verification is Fault Injection (FI) Simulation [1][2][3][4]. The purpose is to show that fault effects can propagate to outputs and that Safety Mechanisms can detect them. Propagation of faults during simulation is key for achieving ISO26262 requirements. An injected fault that is not observed on the outputs, must be re-simulated or proven to be untestable. In order to provoke propagation of all faults, complex verification environments with numerous test inputs are required, resulting in long FI Campaigns. To address this challenge, we can deploy different verification technologies in a single methodology. Formal Methods can be employed to leverage the most appropriate setups for simulation campaigns. The ability of formal in analyzing design behavior to all test inputs can help to identify untestable faults and to determine test inputs for corner cases [5][6][7]. Anyhow, Formal Methods are not capable of analyzing all faults in an acceptable time frame. Therefore, another solution is still required to analyze a large portion of the faults. The application of automatically generated ATPG Testbenches can decrease the efforts on the development of simulation environments. ATPG tools are able to create test patterns that potentialize fault propagation. Simulation can be performed with the generated test vectors aiming to achieve better failure coverage with reduced simulation times [8][9]. Nonetheless, ATPG focuses on manufacturing test and is not optimal for determining untestable faults or covering faults on areas out of the scan chains reach. Even though Simulation, Formal Methods, and ATPG have complementary strengths, to the best of our knowledge, they were not previously combined in a single fault analysis flow that aims at fault propagation for compliance to ISO26262 requirements.

Our work takes advantage of three different technologies aiming to verify the correctness of fault classification while providing data to support traditional FI Campaigns. Initially, ATPG is used to generate a verification environment that provides high fault propagation rate. The outputs from ATPG are used by the FI Simulator, to verify the functional behavior of the design under each fault. In parallel, Formal Methods are applied to identify faults that are untestable and determine the behavior of faults that are not covered by ATPG. Finally, the outputs of each tool are verified against each other to identify malfunctions, increasing the confidence in the tool's outputs, as required by ISO26262 [10]. The main contributions of our methodology are:

- Increasing Tool Confidence Level according to ISO26262. By providing an automated flow for error detection in Fault Analysis tools, we can avoid the extensive ISO26262 Tool Qualification requirements.
- Identification of untestable faults. Formal Methods can prove that faults cannot be tested, and therefore can be ignored during safety metrics calculation, increasing compliance with ISO26262 fault metrics.
- Initial assessment of the fault propagation behavior by

the deployment of ATPG Test Environments and Formal results. The achieved fault detection rates, above 99% on tested designs, can be employed to support the ISO26262 Functional Safety Verification.

This paper is organized as follows. Section II investigates how fault analysis is implemented by different technologies. Section III describes the proposed methodology. Section IV presents the validation process and explain our results. And last, Section V presents our final conclusions.

II. FAULT ANALYSIS

This section investigates how fault analysis is implemented by different technologies. The examination aims to identify the strengths and weaknesses of each solution and determine how they comply with Functional Safety requirements. ISO26262 requires that any component that implements a safety-related functionality, reach a minimum level of tolerance to random hardware failures. Coverage for this type of failure is usually increased by the addition of Safety Mechanisms to the design. Safety Mechanisms, as defined by ISO26262, should be able to detect faults or control failures in order to achieve or maintain a safe state.

The effectiveness of the design to cope with random hardware failures should be quantitatively demonstrated by the calculation of metrics defined by the standard [11]. It is necessary to evaluate the efficiency of the Safety Mechanisms to handle critical faults, contributing to achieving targeted safety metrics. Fault Injection Simulation is a widely used technique to perform this analysis being the method recommended by ISO26262.

A. Fault Injection Simulation

Analysis of Fault Injection by Simulation is widely used and available in a variety of tools. These tools are able to analyze a Register Transfer Level (RTL) or Gate-Level (GTL) descriptions of an IC and, based on given test inputs, simulate their behavior. The effect that a fault produces in the design is determined by comparing the behavior of the design with and without faults. The flow implemented by Fault Injection Simulation Tools is described below:

1) Elaboration of RTL/GTL design description.
2) Fault List Generation: candidates for fault injection are defined for each available fault model. The user should define rules (e.g. all signals) to identify fault node candidates and fault models (e.g. Stuck-at-0 (SA0) and Stuck-at-1 (SA1)). Information is stored in a fault database.
3) Fault List Optimization: Faults list is analyzed to identify candidates for optimization. Based on the elaboration results, tools can estimate the behavior of some faults decreasing the number of faults to be simulated. Information is updated on the fault database.
4) Good Simulation: fault-free behavior of design is simulated. The user should define observation points in the design to identify: (1) Fault propagation to a functional output: functional strobes; (2) Activation of the Safety Mechanism: checker strobes. The values of the Strobes during good simulation are stored.
5) Fault Injection Simulation: For each fault in the fault database, the design faulty behavior is simulated, and the observation points compared against the reference values from the Good Simulation. The behavior of the design under each fault is analyzed and stored.

FI Simulation determines the behavior change provoked by a fault when the effect is observable in one of the outputs (strobes). Faults that don't produce changes in the strobes are classified as Undetected. This is considered a weak result of the simulation, as a different test may cause fault propagation. Fault propagation is required to assure correct classification. If there are no test stimulus that provokes the propagation of a fault, this should be proved by analysis. For that reason, FI Simulation demands the development of complex Testbenches and additional untestable fault analysis.

B. Formal Methods

Identification of untestable faults requires proof that the fault cannot be tested by ANY functional test stimulus. Formal analysis appears as a good alternative for this purpose since it is not limited to a specific time or state. Instead, the scope is global, and every evaluation context and test stimulus is considered. Consequently, formal analysis can exhaustively prove that a fault can never produce any failure. This class of faults can be considered untestable and don't require further fault simulation.

Different EDA vendors explore fault analysis capabilities in their formal solutions. Generally speaking, these solutions automatically generate properties, not requiring knowledge of formal languages. In addition, they allow integration with FI Simulators providing fault lists optimization and reducing simulation campaigns. Tools used for fault formal analysis usually apply two main fault analysis techniques, Standard Analysis, and Advanced Analysis.

The Standard Analysis aims to determine the testability of faults. It is applied as a pre-qualification flow for simulation, to reduce the fault list by identifying untestable faults. The testability of the faults is determined by verifying:

- if there is a physical connection between the fault location and the observation points (strobes).
- if the signals that drive the fault node allows activation of the fault.
- if the fault could be observable in at least one strobe of the design.

A fault that does not pass these verifications can be classified as untestable. In addition, the fault list may be optimized by Fault Relation Analysis. The tool analyzes the design to determine the relationship between fault pairs. Fault pairs are then included in the same Collapsing Group. The behavior of all Collapsing Group is predicted by simulation of only one representative of the group, called the Prime Fault.

The Advanced Analysis deploys formal techniques to analyze propagation and activation of the faults. Activation

TABLE I
FAULT ANALYSIS TECHNOLOGIES COMPARISON

Technology	Strengths	Weaknesses
FI Simulation	- Comprehensive behavior analysis - Recommended by ISO26262	- Single test input at a time - Multiple simulations to propagate all faults - High Testbench development efforts
Formal Methods	- Analysis of all possible test inputs - Analysis of untestable faults - Generates test inputs for corner cases	- Time-consuming - Not able to determine behavior of all faults
ATPG	- Automatically generated Testbenches - High fault propagation rate	- Focus on manufacturing tests - No analysis of untestable - Do not reach corner cases

Analysis indicates whether the fault can be functionally activated from any combination of inputs. Propagation Analysis verifies if there is a combination of inputs that provoke fault propagation. Advanced Analysis will classify the faults, which were not previously classified by the Standard Analysis, in three groups:

- Untestable: Faults that cannot be activated or propagated.
- Dangerous: The tool identified a combination of test inputs that results in fault propagation.
- Unknown: All the others.

Formal properties to perform the Advanced Analysis are automatically generated and verified with respect to all possible input stimulus. The Advanced Analysis relies on formal properties and analysis to prove the properties to be true. The analysis of formal properties is time-consuming and cannot find results for all faults in complex designs. For that reason, this analysis is often applied as a last resource, on the faults that were not classified after fault injection simulation.

The different strengths of Simulation and Formal can complement each other. An integrated fault analysis flow allows the deployment of the Standard Analysis before the start of the simulation. The analysis will reduce the number of faults to be simulated by leveraging results for untestable faults and collapsing groups. After the simulation, Advanced Analysis can be executed on the remaining undetected faults to verify if there is a combination of test inputs that would result in fault propagation.

Even with the combination of Formal and Simulation, the development of the test environments is challenging. Advanced Analysis from Formal tools, that can support the identification of test stimulus for fault propagation, are time-consuming and cannot find results for all fault list. In this context, ATPG appears as a possible solution for generating Testbenches and test inputs that can be used for the FI Simulation.

C. Automatic Test Pattern Generator

Test patterns can be generated to identify if an IC contains manufacturing induced defects. In other words, to distinguish between the correct circuit behavior and the faulty circuit behavior. When applying the test pattern to the inputs of a circuit, the values observed at the outputs should be monitored. A defect is detected if any of the outputs are different from the expected pattern. Nowadays, ATPG is a well-established technology being used on the development of almost all IC. ATPG tools can generate a minimal group of test vectors to achieve acceptable levels of manufacturing defects detection. In addition, tools can generate reports about the testability of each defect, allowing the generation of metrics to indicate test quality and test application time.

Usually, an ATPG flow receives as inputs a Gate-Level description of an IC and specification of the scan chains. Then, it verifies if the implemented scan chains can ensure the required levels of testability. If affirmative, it generates a fault model and test patterns, to assure propagation of fault effects to the design outputs. Typically, the test patterns and expected outputs are programmed in a Test Equipment that will be used in IC manufacturing tests. The Test Equipment applies the test patterns in the inputs of the circuit and monitors the outputs to verify if the values are the expected ones. We propose a similar approach using FI Simulation. Instead of using a Test Equipment, we apply the ATPG test patterns on the design simulation and use the strobe functionality to monitor the outputs of the design. During the Good Simulation, the Simulator stores the strobe values, defining the expected output pattern. Afterward, the simulation of each fault is executed using the same inputs and monitoring the outputs. This way, we can use the propagation capabilities of ATPG to identify behavioral changes caused by injected faults.

The fault propagation potential of ATPG test environments is a powerful benefit for compliance with Functional Safety. However, ATPG focuses on manufacturing tests and the estimated results should be demonstrated via simulation. In addition, ATPG doesn't consider untestable faults and faults out of the scan chain reach. Formal Analysis can be deployed for addressing these cases.

Table I summarizes the strengths and weakness of each technology. Considering this examination, we propose a methodology that highlights the strengths of Simulation, Formal and ATPG for Functional Safety Verification.

III. PROPOSED METHODOLOGY

This section describes the application of three fault analysis technologies in an efficient methodology for ISO26262

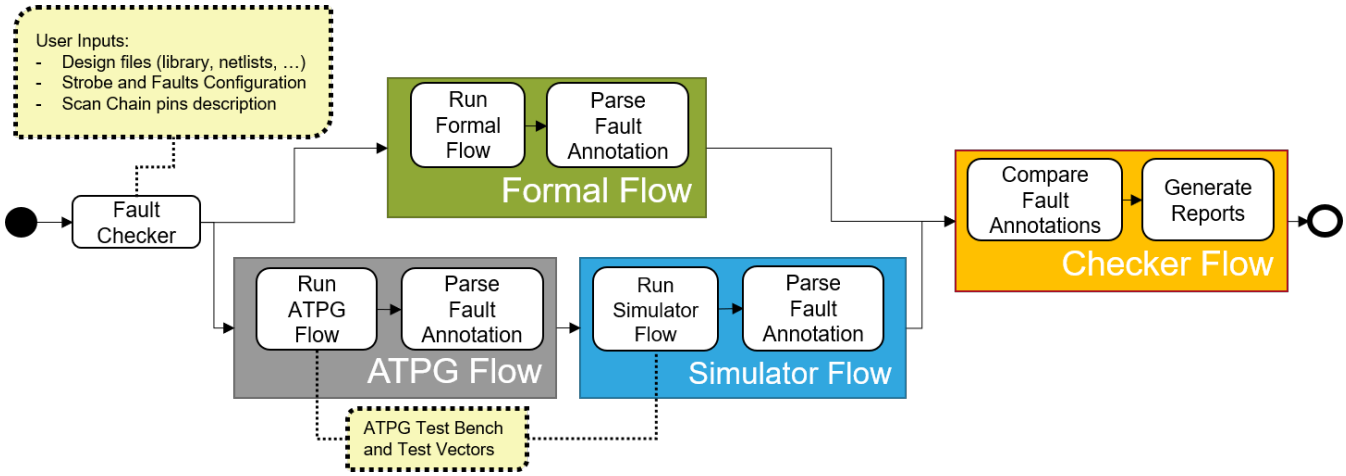

Fig. 1. Fault Checker Execution Flow.

Functional Safety Verification. The methodology highlights the strengths of Simulation, Formal and ATPG to generate a comprehensive fault analysis report. An application was developed aiming to automate the execution of the different tools. The Fault Checker application implements a generic control flow that is configurable with tools from different vendors. In the end, the reports of each tool are parsed and saved in a common format. The fault classification of each tool is combined in a final report that allows the identification of tool malfunctions and detailed analysis of faults behavior.

The Fault Checker application must be configured with scripts to control the execution of each tool and with the rules for parsing the reports. Also, the user must provide design-specific information, as fault targets and observation points (strobes). With all the required information, the application can start the execution of the ATPG and Formal flows. As these two flows are independent, they can be executed in parallel using different CPUs. Simulator flow requires the ATPG Testbench and test vectors to start. So, after the ATPG flow is finished, the Fault Checker will extract the generated Test Environment and will use it for the FI Simulation. At the end of each flow, the reports generated by the tools are parsed to a common format and saved. Finally, at the end of all flows, the relevant parsed data is retrieved and compared. The comparison is based on rules that associate the classifications used by each tool. In case a rule is not obeyed, the Fault Checker will include a Warning tag, informing that this fault requires attention from the designer. Fig. 1, illustrates the execution flow of the Fault Checker application.

Results can be analyzed in a CSV report that details the classification of each fault by each tool. An error caused by a malfunction in one of the tools will be indicated by a Warning in the report. For example, if the Simulator classifies a fault as Detected and Formal classifies the same fault as Safe, this would indicate a malfunction in one of the tools. A sample of the detailed report is demonstrated in Table II.

In addition to malfunction indication, the report provides supplementary information for fault analysis. For example, signal ”dut.u0.sig2” in Table II, is classified as Undetected by the Simulator and Ignored by ATPG. However, the fault is listed as Dangerous by Formal, meaning that formal analysis identified at least one test stimulus that can propagate the fault to a strobe. This information can be used on a new FI Simulation to achieve detection of this fault. Another example to highlight is ”dut.u0.sig1”, where Formal classified the fault as Safe, while the other tools classified as Undetected and Ignored. Results from the formal analysis can be used to demonstrate that the fault cannot propagate to a strobe, and therefore can be considered untestable, contributing to achieving ISO26262 metrics. Any other discrepancy between the faults is indicated in the report, as illustrated by signal ”dut.u0.INsT0.0”.

IV. VALIDATION

This section describes the validation process of the proposed methodology. First, we describe the adopted setup, the configuration of the tools and the tested designs. Then, we demonstrate our results and describe the benefits and limitations of our solution. The following validation aspects were considered: Detection of malfunction in the tools via detailed report; Application of fault analysis results to support Functional Safety verification of the design.

A. Validation Setup

The methodology was validated by deploying the Fault Checker application on example designs. First, the Fault Checker must be configured with the tools to execute each flow. Our work has adopted Cadence® Xcelium™ Fault

TABLE II
FAULT CHECKER REPORT EXAMPLE.

Signal Name	Fault Type	Formal Classification	Simulator Classification	ATPG Classification	Checker Results
dut.u0.rst	SA0	Dangerous	Detected	Tested	PASS
dut.u0.sig1	SA1	Safe	Undetected	Ignored	WARNING
dut.u0.sig2	SA0	Dangerous	Undetected	Ignored	WARNING
dut.u0.sig3	SA1	Dangerous	Detected	Tested	PASS
dut.u0.iNsT0.0	SA1	not_listed	not_listed	Tested	WARNING

TABLE III
FAULT CHECKER RESULTS.

Design	Faults (SA0/SA1)	Detection Rate	PASS	WARNING
Up Down Counter	162	100%	162	0
Memories	2782	99.78%	2776	6
AC97	57226	99.77%	57108	118
Conmax	153454	99.80%	153191	263

Simulator (XFS), Cadence® JasperGold (JG) Formal Verification Platform Functional Safety Verification (FSV) and Cadence® Modus DFT Software Solution ATPG component, as the representatives of each technology.

The selection of the designs contemplated different levels of complexity and the availability of Functional Testbenches. Complexity was determined by the number of fault targets in each design. The ISO26262 defines that all cell ports in the IC Gate-Level representation should be analyzed for faults. The selected designs were synthesized using the standard cell reference libraries provided with Cadence 45nm Generic Process Design Kit (GPDK) [12]. The selected designs are available on the IWLS 2005 benchmark [13]. The designs are: (1) Up-Down Counter: 4 bits adder containing 81 cell ports; (2) Memories: Two memories with CRC, containing 1391 cell ports; (3) AC97: An Audio Codec Controller compatible with Wishbone bus, containing 28610 cell ports; and (4) Conmax: An interconnect matrix IP core featuring parameterized priority-based arbiter, with 76727 cell ports.

Designs (1) and (2) were initially deployed to verify that the Fault Checker application was working properly. As the designs are smaller, it was possible to manually check the classification of each fault to ensure the correctness of the final report. The other designs were deployed to verify the behavior of the Fault Checker application when analyzing larger designs. In addition, for designs (3) and (4), the achieved results were compared with fault injection results using Functional Testbenches only. The achieved results are described in the following sections.

The experiments were executed on two Intel Xeon E5-2680 CPUs with 16 Cores and 252 GB of memory each. Being the Formal flow executed on CPU1 and ATPG followed by Simulation Flow in CPU2. Parallel fault injection simulations were performed to improve the overall time of the Simulation Flow.

B. Results

Table III demonstrates the results of the methodology for the selected designs. It details, for each design, the total number of faults, the fault detection rate, and the Pass/Warning indication resulting from the Fault Checker verification.

During the Up Down Counter design verification, the Fault Checker confirmed that all faults have equivalent classifications. As the example is relatively simple, the different technologies can determine that all faults can propagate to observation points (strobes).

TABLE IV
FAULT DETECTION COMPARISON.

Design	Faults (SA0/SA1)	Functional Testbench		Fault Checker	
		Detected	Undetected	Detected	Undetected
AC97	57220	71,50%	28,48%	99,77%	0,21%
Conmax	153454	81,66%	18,34%	99,80%	0,20%

For the Memories design, the application detected 6 faults with discrepant classifications. In this example, the Warnings were due to classifications of Safe Faults by Formal and Undetected by the Simulator. For these 6 faults, the Formal analysis proves that the faults are untestable, and can be disregarded, improving results for ISO26262 metrics calculation.

On the AC97 design, the Fault Checker was able to detect 118 faults with distinctive classifications. From these, 49 faults were classified as Safe by Formal and Undetected by the Simulator, and can be declared as untestable; 23 were classified as Dangerous by Formal and Undetected by the Simulator, meaning that these faults can be Detected in Simulation by applying the results from Formal as test inputs; 46 faults were considered Undetected by Simulation and ATPG and Unknown by Formal, indicating that none of the tools was able to define the possible behavior of these faults, and they require manual analysis; 6 faults were in cell ports related to power that are not relevant for Functional Safety Verification.

During the analysis of the Conmax design, the methodology detected 263 discrepancies between the tools. From these, 7 faults were classified as Dangerous by Formal and Undetected by Simulation. Meaning that results from Formal can be applied for detecting these faults during simulation. The other 256 faults were classified as Redundant by ATPG, Undetected by Simulation and Unknown by Formal. As the classifications are not conclusive, these faults should be manually analyzed.

To analyze the capability of the methodology for fault classification, we compared the Fault Checker results with results from fault injection when using a Functional Testbenches. The AC97 and Conmax designs include simulation environments for verification of their functionalities. Table IV demonstrate results of the FI simulation of the AC97 and the Conmax designs when deploying the Functional Testbenches and when using the Fault Checker. Due to the characteristics of fault propagation provided by the ATPG Testbenches, after one execution of the Fault Injection campaign, the Fault Checker achieves a fault Detection Rate improvement of 28,2% for the AC97 and 18,2% for the Conmax.

The Undetected classification is inconclusive for fault analysis. Undetected faults must be proven Untestable to collaborate to ISO26262 metrics and are more likely to mask a malfunction in a tool. For these reasons, we want to achieve as many detected faults as possible. If we have applied Functional Testbenches to achieve the same level of fault detection from the Fault Checker, we would need to repeat

the Fault Injection Campaign with new test inputs, until all faults get propagated to outputs, demanding the development of new Test Environments and longer FI Campaigns.

C. Discussion

The results demonstrated above corroborate with the selected evaluation criteria. First, the deployment of multiple fault analysis technologies enables the detection of erroneous fault classifications. The proposed methodology allows a high degree of confidence in tool error detection, resulting in a Tool Confidence Level (TCL) of one. A methodology with TCL1 doesn't require Tool Qualification, avoiding big efforts on documentation and analysis for compliance with ISO26262 [10]. Second, identification of Safe faults collaborates with ISO26262 compliance. By proving that a fault is untestable, we are able to disregard it, decreasing the total number of faults to be simulated and improving ISO26262 metrics [11]. Third, the proposed methodology achieved substantial fault detection rates. The use of ATPG test vectors during simulation and identification of Dangerous faults by Formal, provide extra information about the design behavior. In summary, our results can be applied to support the following aspects of ISO26262 Functional Safety Verification:

- Avoid efforts with Tool Qualification by automating tool error detection.
- Identification of Untestable Faults allows improvement of ISO26262 metrics and reduction of the number of faults to be simulated.
- Fault supplementary data can be used to support further fault injection campaigns.

Even though we have achieved high fault detection rates, we need to consider that the examples used were of average complexity. One of the next steps of our work is to apply our methodology to more complex designs. We need to explore how the fault detection provided by ATPG in complex designs can leverage the Safe and Dangerous classifications from Formal for the achievement of ISO26262 requirements.

Another aspect to acknowledge is the possibility of changes in the fault propagation patterns when ATPG scan chains are disabled. The application of our technique in more complex designs, for instance, an Automotive CPU, should consider this effect and employ formal results to assess differences in the classification of the faults.

V. CONCLUSIONS

Due to the harsh requirements for random hardware failures tolerance, Functional Safety verification is a challenging step for ISO26262 compliance. Fault analysis, as part of this process, becomes a extensive procedure, that is usually repeated numerous times until the metrics for fault detection are achieved. Furthermore, ISO26262 requires specific criteria to determine the level of confidence in the adopted software tool, increasing the efforts even further. We propose a methodology that deploys ATPG and Formal to support Simulation results and to decrease the overall efforts of ISO26262 compliance. Our methodology enables the use of test environments created with ATPG tools for the simulation of faults, and the use of Formal for identification of untestable faults. Formal results allow the optimization of the Fault List, reducing the number of faults to be simulated, and the generation of test vectors for the detection of corner cases. In addition, the results of the tools are compared to identify potential malfunctions. The inclusion of redundancy as a method to detect malfunctions in tools is a suggested method for achieving ISO26262 Tool Confidence [10]. Our results have shown high fault detection rates, achieving more than 99% of detected faults. In addition, detailed fault information provided contributes to achieving ISO26262 metrics.

ACKNOWLEDGMENT

This research was supported by project RESCUE funded from the European Unions Horizon 2020 research and innovation programme under the Marie Sklodowaska-Curie grant agreement No 722325.

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] S. Pateras and T.-P. Tai, "Automotive semiconductor test," in *2017 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. IEEE, apr 2017.

[3] D. Alexandrescu, A. Evans, M. Glorieux, and I. Nofal, "EDA support for functional safety — How static and dynamic failure analysis can improve productivity in the assessment of functional safety," in *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, jul 2017.

[4] Y.-C. Chang, L.-R. Huang, H.-C. Liu, C.-J. Yang, and C.-T. Chiu, "Assessing automotive functional safety microprocessor with ISO 26262 hardware requirements," in *Technical Papers of 2014 International Symposium on VLSI Design, Automation and Test*. IEEE, 2014.

[5] K. Devarajegowda and J. Vliegen, "Deploying formal and simulation in mutual-exclusive manner using jaspergolds proofcore technology," in *Cadence User Conference CDNLive EMEA*, 2017.

[6] S. Marchese and J. Grosse, "Formal fault propagation analysis that scales to modern automotive SoCs," in *2017 Design and Verification Conference and Exhibition DVCON Europe*, 2017.

[7] A. Traskov, T. Ehrenberg, and S. Loitz, "Fault proof: Using formal techniques for safety verification and fault analysis," in *2016 Design and Verification Conference and Exhibition DVCON Europe*. DVCON, 2016, pp. 27–32.

[8] S. Praveen, S. Yellampalli, and A. Kothari, "Optimization of test time and fault grading of functional test vectors using fault simulation flow," in *2014 International Conference on Electronics, Communication and Computational Engineering (ICECCE)*. IEEE, nov 2014.

[9] S. Arekapudi, F. Xin, J. Peng, and I. G. Harris, "ATPG for timing-induced functional errors on trigger events in hardware-software systems," in *Proceedings The Seventh IEEE European Test Workshop*. IEEE Comput. Soc, 2002.

[10] ISO, *ISO 26262 - Road Veichles - Functional Safety - Part 8: Supporting processes*, International Standardization Organization Std., Nov. 2011.

[11] ISO, *ISO 26262 - Road Veichles - Functional Safety - Part 5: Product development at the hardware level*, International Standardization Organization Std., Nov. 2011.

[12] *GPDK045 Reference Manual*, Revision 5.0 ed., Cadence Design Systems , Inc., 2016.

[13] C. R. Berkeley, "International workshop on logic and synthesis (IWLS) 2005 benchmarks," Tech. Rep., 2005.