2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Use of Formal Methods for verification and optimization of Fault Lists in the scope of ISO26262

Felipe Augusto da Silva, Cadence Design Systems GmbH, Feldkirchen, Germany (*dasilva@cadence.com*)

Ahmet Cagri Bagbaba, Cadence Design Systems GmbH, Feldkirchen, Germany (*abagbaba@cadence.com*)

Said Hamdioui, Delft University of Technology, Delft, The Netherlands (*s.hamdiouia@tudelft.nl*)

Christian Sauer, Cadence Design Systems GmbH, Feldkirchen, Germany (*sauerc@cadence.com*)

***Abstract*—This work aims at an alternative method to verify the correctness of Fault Lists generated by fault simulators tools in context of safety verification. The lists generated by simulation tools are verified against lists from formal tools. The consistency evaluation between the lists supports the Tool Confidence Level (TCL) assessment, defined in the ISO26262. In addition, formal tools have the potential of performing optimization in Fault Lists by annotation of the expected behavior of the design under fault. Our work demonstrates the feasibility of using Formal Methods to verify and optimize the fault list from simulators. Results indicate an average reduction of 29.5% on the number of faults to be simulated and demonstrate that it is possible to achieve TCL by verification of the fault lists.**

Keywords—ISO26262; Fault Injection; Formal; Simulation; Tool Qualification.

I. INTRODUCTION

With the increasing complexity in automotive applications such as autonomous driving, the use of electronics systems in this domain is growing exponentially. This is causing a shift in the traditional design flow and is pushing ISO26262 compliance down to the semiconductor chain. As a result, Functional Safety compliance becomes part of the requirements for the development of complex systems. During the development of an Integrated Circuit (IC) compliant with ISO26262, one of the critical tasks is the evaluation of the effectiveness of the design to cope with random hardware failures. This is usually done by execution of Fault Injection (FI) Simulations, where each possible fault candidate of the design is evaluated for robustness to random faults, and the behavior of the design under these faults is simulated. In complex designs, where millions of design components are susceptible for random faults, this process becomes challenging [1].

To facilitate FI Simulation campaigns, EDA tools may be used for automation of behavioral analysis of a design. By examining the description of a design, simulation tools are able to identify what design components should be considered for fault injection and simulate the behavior of the design under the effect of these faults. The provided automation increases the possibility of faults being introduced or masked by malfunction in the tool. Aiming to avoid these malfunctions, ISO26262 includes instructions for qualification of tools. Any tool that supports activities required by the standard, must be evaluated to show the minimum level of confidence necessary for the intend activities.

The level of confidence of a tool is determined by evaluating the possibility of a malfunctioning to introduce or fail to detect errors in the design under development. In the case of a tool used in FI Simulation, a malfunction could mask or introduce an error on the fault candidates and on the analysis of the behavior of the faulty design. To guarantee the confidence in the results generated by the tool, an evaluation methodology is required. The outputs of the tool should be verified to prevent or detect any malfunctions.

Considering the automation provided by EDA tools on FI Simulations, this work focuses on improving the Level of Confidence on the Fault Lists generated by simulators with Formal Methods. In addition, the results from formal analysis allowed us to optimize the Fault Lists and reduce the time of FI Simulation campaigns.

II. RELATED WORK

The challenges of tool qualification per ISO26262 are exemplified in [2]. The authors present a semi-automatic qualification method for a verification tool that can reduce costs in the qualification process. The work highlights

the importance of applying automatic verification on the outputs of a tool, to decrease the efforts of manual verification. In [3], formal methods are used to determine the behavior of a design under fault. The authors propose a fault injection model that allows the verification of a design by symbolic simulation. A mixed approach using formal methods with simulation to decrease the time of fault injections campaigns, is explored in [4]. Formal is used to show that a failure state is not achievable with the injected fault, thus the simulation can be stopped. Different works are employing combined fault injection analysis flows with simulation and formal methods, [5] [6][7]. The strength of formal methods, in analyzing the behavior of a design to all test stimulus, is applied to leverage the most appropriate setups for the simulation campaigns.

Our approach combines simulation and formal methods as a methodology for verification of the results generated by both tools. Formal analysis is used to verify Fault Lists generated by the simulator, thus increasing the confidence in the tool outputs, as required by ISO26262. To the best of our knowledge this approach was not previously used. In addition, as seen in other works, formal analysis can reduce the number of required simulations by pre-evaluating the fault propagation potentials.

III. FAULT INJECTION CAMPAIGNS

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 addition of Safety Mechanisms to the design. Safety Mechanisms should be able to guarantee that fault propagation cannot disturb a safety related functionality.

The effectiveness of the design to cope with random hardware failures should be quantitatively demonstrated as defined by the standard. To accomplish this, it is necessary to assess the efficiency of the Safety Mechanisms to handle critical faults thus allowing to achieve 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 a design and, based on given test inputs, simulate their behavior. The effect that a fault causes in the design is determined by comparing the behavior of the design with and without faults. The selection of the tool must consider the available features and aspects of performance, as FI Simulation campaigns can become long as design complexity increases. Our work deploys Cadence® Xcelium™ Fault Simulator (XFS) to perform the Fault Injection Simulation [8]. The flow implemented by XFS for Fault Injection Simulation is as follow:

1. Elaboration of RTL/GTL design description.
2. Fault List Generation: fault node candidates found in the design are listed for each available fault type. User should define rules (e.g. all signals) to identify fault node candidates and fault types (e.g. Stuck-at-0 (SA0) and Stuck-at-1 (SA1)). Information is stored in a Plain Fault List.
3. Fault List Optimization: Plain Fault List is analyzed to identify faults that do not need to be simulated as the behavior of the design in presence of these faults can be predicted. Information is stored in an Optimized Fault List.
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. Strobe values during good simulation are stored.
5. Fault Injection Simulation: For each fault, listed in the Optimized Fault List, the design faulty behavior is simulated, and the observation points compared against the reference values generated during Good Simulation. Results of FI are stored in the Annotated Fault List.

Looking on the perspective of Tool Qualification, there are three main outputs of the Simulation tool that should be verified: The Plain Fault List, the Optimized Fault List and the Annotated Fault List. Figure 1 illustrates the FI Simulation flow with the required user inputs and described outputs.

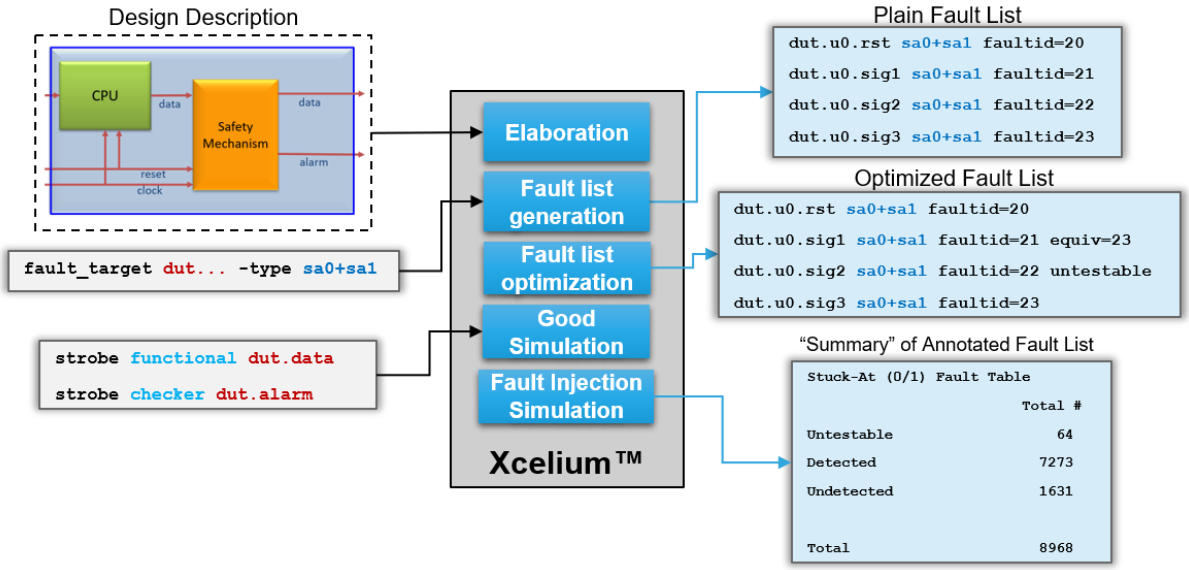

Figure 1. Xcelium Fault Injection Simulation Flow.

Although, FI simulation is the recommended method for FI analysis, the process of simulating each single Fault is highly costly. As the behavior of the design is simulated with single stimulus at a time, there is a considerable chance that faults will not propagate to a strobe, in other words will be Undetected for this specific stimulus. Defining the required group of stimuli to assure that every single fault will propagate to a strobe is nearly impossible in complex designs. To address these challenges, different technologies are being analyzed to decrease the efforts of FI analysis. The use of Formal Methods is a promising solution, being already deployed by different vendors in their Formal Solutions.

B. Fault Analysis by Formal Methods

While FI simulation is limited to a single context, applying a single stimulus using a single fault model, formal fault analysis is not limited to a specific time or state. Instead, the context is global, and every evaluation context, stimulus and faults, are considered. Consequently, formal analysis can exhaustively prove that an Untestable fault can never propagate to a strobe. If there is no possibility of propagation, the fault can be considered Safe and do not need to be simulated.

Different vendors are implementing FI Analysis capabilities in their Formal Solutions, this work uses the Functional Safety Verification (FSV) application from Cadence JasperGold® (JG) Formal Verification Platform. JG FSV requires no formal languages knowledge, as all required properties are automatically generated by the tool. Fault Analysis is available in a standalone mode, but also includes build-in support for integration with XFS, allowing the deployment of both tools in a unified FI Analysis flow. JG FSV includes two main fault analysis techniques, Standard Analysis and Advanced Analysis [9].

The Standard Analysis verify the testability of faults. It is an automated pre-qualification flow for simulation that improve the results of the Optimized Fault List. FSV applies structural fault analysis techniques to verify if the injected faults could affect the results on one of the strobes. In addition, the fault list is optimized by Fault Relations Analysis. The tool analyzes the design for relationship between fault pairs in which the result of one fault can be predicted by the behavior of the other. 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 propagability and activation analysis. Activation Analysis checks whether the fault can be functionally activated from the inputs. Propagation Analysis checks whether the fault can propagate to a strobe. If it cannot, then it is determined to be Safe. If it can, this analysis will identify the necessary stimulus for propagation. The strobes can be functional or checker. Propagation of the fault to a functional strobe can lead to a functional safety violation, while propagation to a checker strobe indicates that the Safe Mechanism detected the fault. Figure 2 illustrates JG FSV Fault Injection Advanced Analysis flow. Properties to verify the propagation effects from faults and strobes detection are automatically generated, and then verified to all possible input stimulus. Results are compared against a copy (Bad Machine) of the design were the fault is injected.

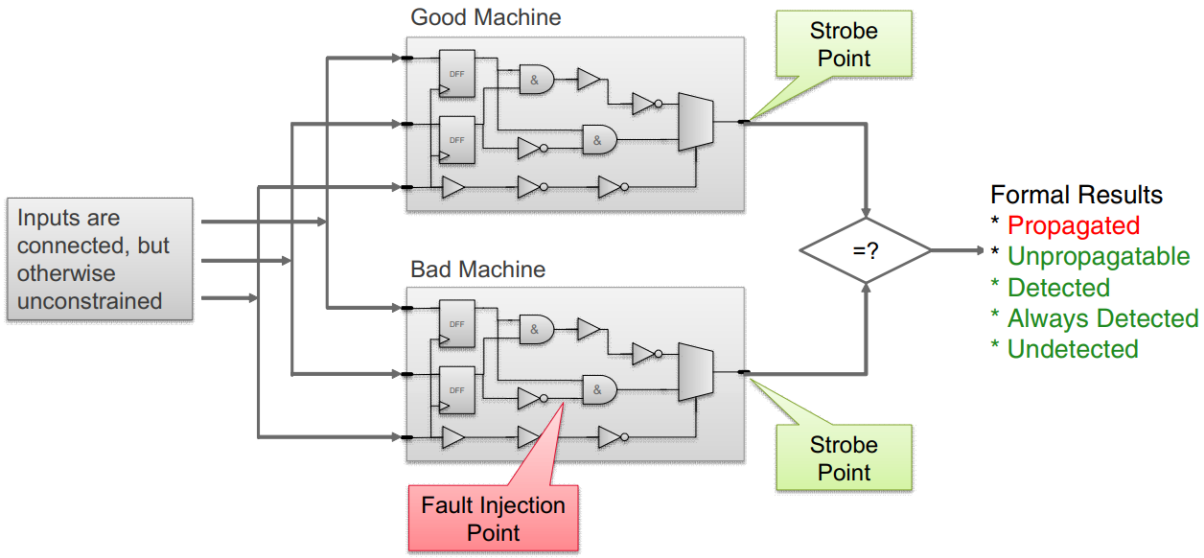

Figure 2. Jasper Gold FSV Fault Injection Analysis Flow.

The different strengths of Simulation and Formal can complement each other for FI Analysis. The combined flow allows reduction of simulation efforts by increasing fault optimization, identifying propagation potentials and by identifying stimulus that will cause fault propagation during simulation.

The build-in integrated flow allows deployment of JG FSV Standard Analysis on the Optimized Fault List from XFS. The formal analysis will reduce the number of faults to be simulated by leveraging formal results for Safe Faults and Collapsing Groups. After simulation, JG FSV Advanced Analysis can be executed on the remaining Undetected faults to verify if they can propagate to a strobe and what is the required stimulus.

The FI Analysis from JG FSV can generate the same outputs that are generated by XFS. JG FSV flow starts by Analyzing and Elaborating a design description. Next, user should set the fault type and design candidates for fault injection, generating a Plain Fault List. After, the Standard Analysis will generate an Optimized Fault List, including Safe and Collapsing information. And last, the Advanced Analysis will generate an Annotated Fault List by including information about propagation and detection of faults. By using formal to generate the same outputs from the simulator, it is possible to automatically verify the consistence between the results. As stated by the ISO26262, prevention or detection of tool malfunctioning can be accomplished through redundancy in software tools [10].

IV. Verification and Optimization Flow

Even though the build-in integrated flow between Xcelium and JG facilitates the FI Analysis, from the perspective of tool qualification, it is preferable to run both tools in standalone mode. To use the outputs from formal to verify the outputs from the simulator, it is necessary to show that there is no interference between the tools. As during the integrated flow, the tools share the same fault database, we have decided to separate the flows.

To automate the evaluation of the outputs generated by both tools, a Build Manager application was developed. For each given design, the application automates the elaboration and analysis of the design, on both tools, and controls the execution flows, including the formal analysis in JG FSV and FI simulations on XFS. Finally, the

relevant data is retrieved from both tools and compared. The comparison between the lists is based on rules that associate the annotations used by the tools. For example, faults classified as Untestable by XFS are equivalent to faults classified as Safe by JG FSV.

A detailed report is generated to allow review of the results. An error in an output caused by a malfunction in one of the tools, can be detected by the annotation association rules and could be verified in the detailed report. For example, if XFS simulation annotates a fault as Detected and JG FSV annotates the same fault as Safe, this would indicate a malfunction in one of the tools. A sample of the detailed report, including an example of a tool malfunction, is illustrated in Table I.

Table I. Detailed Report Example

	XFS				JG FSV				Result
Fault ID	**Signal Name**	**Fault Type**	**Annotation**	**Collapsing**	**Signal Name**	**Fault Type**	**Annotation**	**Collapsing**	
0	dut.u0.rst	sa0	Dangerous		dut.u0.rst	SA0	Propagated		PASS
1	dut.u0.rst	sa1	Untestable		dut.u0.rst	SA1	Safe		PASS
2	dut.u0.sig1	sa0	Detected		dut.u0.sig1	SA0	Detected		PASS
3	**dut.u0.sig1**	**sa1**	**Detected**		**dut.u0.sig1**	**SA1**	**Safe**		**WARNING**
4	dut.u0.sig2	sa0	Dangerous	equiv=2	dut.u0.sig2	SA0	Propagated	2	PASS
5	dut.u0.sig2	sa1	Detected		dut.u0.sig2	SA1	Detected		PASS

V. RESULTS

Results were collected by executing the Build Manager application on a set of selected designs from the IWLS 2005 benchmarks [11]. The benchmark contains a collection of RTL and Gate Level description of 84 different designs, varying from small cores to complete System-on-Chip.

Initially, the RTL description of 16 designs were analyzed for Stuck-at-0 and Stuck-at-1 faults. Fault Lists generated by both tools are analyzed to verify: (1) The tools generated the same faults, (2) Which faults are annotated as “Safe”, (3) Which faults are collapsed and (4) All annotations respected the association rules.

Fault Injection analysis with simulation requires that different test stimuli is applied to assure propagation of each fault to a determined strobe. If a fault does not propagate, it is considered Undetected. For the scope of this work, as the idea is not to achieve full verification of the example designs, test vectors were not further developed and complete FI Simulation was not executed. Therefore, some faults are annotated as “Not Injected” by XFS and as “Unknown” by JG FSV. These faults are not considered as a tool malfunction, as they should be revaluated after full verification environment is set-up.

The results of the benchmark evaluation are shown in Table II. For each tested design, the total number of faults and Safe annotations for each tool are illustrated. The column Fault List Reduction highlights the fault reduction percentage per design, when including the JG annotation to the XFS Fault List. JG FSV Standard formal analysis run time in seconds is demonstrated in the corresponding column.

Table II. Summary of Results

Design	**XFS**		**Jasper Gold**				**Fault List Reduction**	
	Nº of Faults	***Safe Faults***	***Nº of Faults***	***Safe Faults***	***Collapsed Faults***	***Run time (s)***	***by Safe Faults***	***by Collapsed Faults***
DMA	33428	106	33428	4921	8734	186	14,40 %	26,13 %
ac97	11192	134	11192	1401	2326	674	9,88 %	20,78 %
aes	4266	0	4266	49	1408	168	1,15 %	33,01 %
i2c	528	0	528	14	86	9	2,65 %	16,29 %
mem_ctrl	11044	8	11044	3933	2246	346	34,75 %	22,11 %

Design	XFS		Jasper Gold				Fault List Reduction	
	Nº of Faults	*Safe Faults*	*Nº of Faults*	*Safe Faults*	*Collapsed Faults*	*Run time (s)*	*by Safe Faults*	*by Collapsed Faults*
sasc	86	0	86	1	0	7	1,16 %	0,00 %
simple_spi	534	28	534	35	54	9	1,31 %	10,11 %
spi	1396	0	1396	12	324	13	0,86 %	23,21 %
ss_pcm	242	2	242	3	1	7	0,41 %	0,41 %
systemcaes	9302	0	9302	425	2664	40	3,37 %	47,38 %
systemdes	4104	64	4104	98	1806	41	0,77 %	47,84 %
tv80	1942	36	1942	51	206	49	0,73 %	15,48 %
usb_funct	20386	56	20386	8128	6483	665	39,38 %	32,17 %
usb_phy	364	0	364	3	58	8	0,80 %	18,62 %
vga_lcd	762	0	762	4	0	9	0,52 %	0,00 %
wb_conmax	106666	0	106666	2794	65216	186	2,61 %	61,31 %

VI. CONCLUSIONS

The combination between simulation and formal methods is becoming a stablished method for Fault Injection Analysis and appears as a promising practice for verification of Fault Lists. Looking at XFS and JG FSV as representatives of these technologies, we propose an alternative methodology for the evaluation of Fault Lists on the scope of ISO26262. Inclusion of redundancy as a method to detect malfunctions in a tool, is one of the standard suggested methods for achieving Tool Confidence Level. In addition, the formal methods applied by JG, provide improved information about the propagation effects of the faults, allowing the optimization of the Fault List, and therefore, reducing the number of required faults to be simulated. Preliminary results have shown that the Fault List of both tools are equivalent, allowing the use of JG to verify the outputs of XFS. Furthermore, the results from JG allowed an average reduction of 29.5% on the number of faults to be simulated.

ACKNOWLEDGMENT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 722325.

REFERENCES

[1] Y.C. Chang, L.R. Huang, H.C. Liu, C.J. Yang and C.T. Chiu, "Assessing automotive functional safety microprocessor with ISO26262 hardware requirements", 2014 International Symposium on VLSI Design, Automation and Test (VLSI-DAT).

[2] Q. Wang, A. Wallin, V. Izosimov, U. Ingelsson and Z. Peng, "Test tool qualification through fault injection", 2012 17th IEEE European Test Symposium (ETS).

[3] U. Krautz, M. Pflanz, C. Jacobi, H.W. Tast, K. Weber and H.T. Vierhaus, "Evaluating Coverage of Error Detection Logic for Soft Errors using Formal Methods", Proceedings of the Design Automation & Test in Europe Conference, 2006, vol 1.

[4] A. Bernardini, W. Ecker and U. Schlichtmann, "Where formal verification can help in functional safety analysis", 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD).

[5] K. Devarajegowda and J. Vliegen, "Deploying Formal and Simulation in Mutual-Exclusive Manner using JasperGold's Proofcore Technology", Cadence User Conference CDNLive EMEA 2017.

[6] S. Marchese and J. Grosse, "Formal Fault Propagation Analysis that Scales to Modern Automotive SoCs", 2017 Design and Verification Conference and Exhibition (DVCON) Europe.

[7] A. Traskov, T. Ehrenberg and S. Loitz, "Fault Proof: Using Formal Techniques for Safety Verification and Fault Analysis", 2016 Design and Verification Conference and Exhibition (DVCON) Europe

[8] Cadence Design Systems, "Xcelium Fault Simulator User Guide", Product Version 2018.03

[9] Cadence Design Systems, "JasperGold Functional Safety Verification App User Guide", Product Version 2018.03

[10] International Organization for Standardization, "ISO26262 - Road Veichles - Functional Safety - Part 8: Supporting processes".

[11] Cadence Research Berkeley, "International Workshop on Logic and Synthesis (IWLS) 2005 Benchmarks".