Formal Fault Propagation Analysis that Scales to Modern Automotive SoCs

Sergio Marchese, Jörg Grosse

OneSpin Solutions GmbH, Munich, Germany

{sergio.marchese, joerg.grosse}@onespin.com

***Abstract*—Modern automotive Systems-On-Chip (SoCs) implement numerous safety-critical functions. Random hardware faults, e.g. single event latch-ups, may cause errors in circuit behavior and ultimately lead to dangerous system failures. Safety mechanisms can reduce the occurrence of failures, but compliance to ISO 26262 requires a quantitative analysis of their effectiveness. Formal verification tools can accurately inject faults in hardware models and rigorously analyze their propagation. For complex automotive hardware with huge number of potential faults, however, scalability and usability are challenging. In this paper, we present a two-mode formal fault propagation analysis (FPA) method that requires minimal user input. We show how this pragmatic approach can complement simulation-based fault analysis to deliver improved ISO 26262 metrics, while saving in engineering effort and simulation cycles. Moreover, we demonstrate that with this method, formal FPA can be applied successfully to modern automotive SoCs.**

***Keywords*—formal verification; functional safety; fault injection; fault simulation; safety mechanisms; ISO 26262.**

I. INTRODUCTION

Functional safety standards regulate the development of safety-critical systems for a number of industrial domains, and include requirements for hardware components. ISO 26262 is the functional safety standard for the automotive industry and applies to safety-related electrical and electronic (E/E) systems within road vehicles [1]. Automotive hardware must satisfy stringent expectations on the proportion of random hardware faults that could compromise safety-critical functions. Hardware and software safety mechanisms can detect anomalous circuit behavior and prevent, or control, dangerous failures. Safety mechanisms are key to satisfy ISO 26262 requirements and fault simulation is the dominant method to provide evidence that only a small proportion of faults might propagate to safety-critical functions without being detected.

A. Related Work

Formal methods [2] are widely recognized as a powerful technique to uncover design bugs and perform rigorous and efficient functional verification. The application of formal tools to the functional verification of automotive hardware is well established in the industry [8,9,10].

Another important characteristic of formal tools, particularly relevant to safety-critical applications, is the ability to finely control the injection of faults and exhaustively analyze their sequential effects. Crucially, formal tools have the potential to perform this task very efficiently, in terms of both user effort and computational demands, and non-invasively (no need for code instrumentation steps). Formal-based fault injection and analysis methods applied to automotive hardware are presented in [3,4,5,6,7].

These applications mainly target the verification of register-transfer level (RTL) hardware models, and require significant inputs from expert users with in-depth knowledge of the design under test (DUT).

B. Contributions

In this paper, we present an application of formal tools that is specifically targeted at the computation of fault metrics. This method requires minimal user input, is applicable to RTL and gate-level hardware models, and scales to large designs with millions of faults to analyze. We propose a two-mode approach where, in each mode, we apply dedicated formal algorithms to achieve different intermediate goals. The first mode, named Fast Fault

Propagation Analysis (FPA), targets large fault populations aiming at providing useable results for most of the examined faults. The second mode, named Deep FPA, uses formal algorithms targeted at hard-to-analyze faults, thus being applicable only to fault populations of limited size. In Section II we set the context of this work and present basic concepts and terminology in line with ISO 26262. After introducing the two FPA modes in Section III, we show how this approach can integrate with simulation-based fault analysis in Section IV. In Section V we report on results obtained running the two FPA modes on both RTL and gate-level designs. Section VI concludes the paper.

II. Basic Concepts

A. *Faults, Errors and Failures*

Failures happen when an element in a system is no longer capable to perform its required function. Failures are caused by errors, e.g. the output of a hardware component not having its specified value. Errors are caused by faults, e.g. a DRAM memory being corrupted by a cosmic ray. Note that failures at a certain level of hierarchy within a system can become faults at the next level: a failure of a hardware component might be seen as a fault at the vehicle level [1]. ISO 26262 defines two categories of failures: systematic and random (see Fig. 1). Systematic failures have a deterministic relation to certain causes or faults as, for example, specification mistakes, coding mistakes in software or in RTL, or even manufacturing defects. Random hardware failures occur unpredictably during the

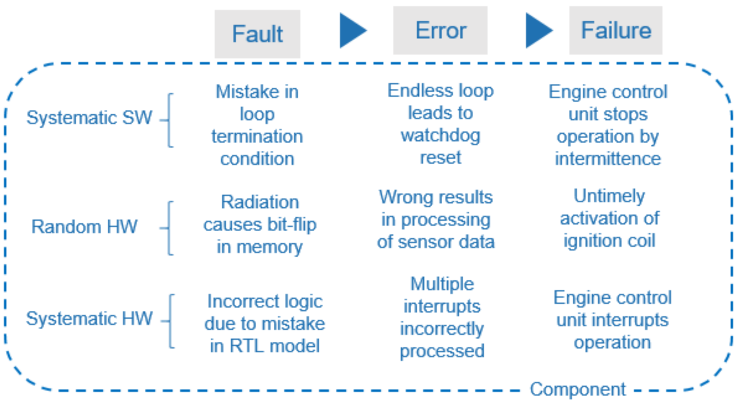

Fault
Error
Failure
Systematic SW
Mistake in loop termination condition
Endless loop leads to watchdog reset
Engine control unit stops operation by intermittence
Random HW
Radiation causes bit-flip in memory
Wrong results in processing of sensor data
Untimely activation of ignition coil
Systematic HW
Incorrect logic due to mistake in RTL model
Multiple interrupts incorrectly processed
Engine control unit interrupts operation
Component

Figure 1. Relations between faults, errors and failures

lifetime of hardware elements following a probability distribution: examples are failures caused by natural radiation or wear-out.

B. *Random Hardware Faults*

Random hardware faults can be categorized according to their duration [11]: permanent faults, e.g. a net becoming permanently stuck to the logical value 0 due to wear-out, intermittent faults, and transient faults. Single Even Upset (SEU) and Single Event Transient (SET) are transient faults. A typical example of a SEU is a bit-flip in a DRAM memory due to natural radiation. SET and SEU faults cause soft errors, as, unlike for hard errors, there is no permanent damage to the hardware. However, it is worth noting that a transient fault could be latent and long lasting.

C. *Safety Mechanisms and ISO 26262 Metrics*

ISO 26262 prescribes the use of hardware and software safety mechanisms that detect random faults, to prevent or control hardware failures [1]. Modern automotive SoCs may include a variety of software and hardware safety mechanisms (see Fig. 2), depending on their target application and associated safety integrity level. Software safety mechanisms may be implemented with self-checking routines, running at system power-up or even during operation. Hardware safety mechanisms, on the other hand, often involve some form of redundant logic responsible to detect and potentially correct errors, and report to other modules in the SoC, e.g. to a safety management unit (SMU).

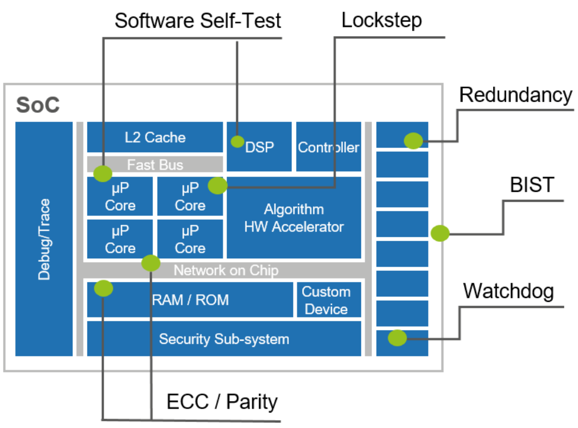
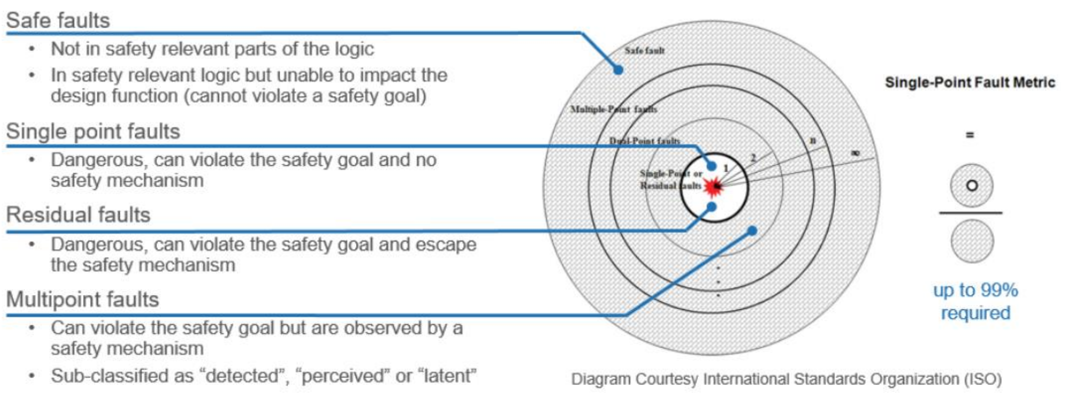


Figure 2. Examples of safety mechanisms

ISO 26262 requires a quantitative analysis of the effectiveness of safety mechanisms, which must take into account faults occurring in the safety logic itself. For example, a fault compromising the error detection and correction capabilities of the safety logic could remain undetected and stay latent. When another fault occurs, the detrimental effect of the latent fault could finally become apparent. Single point faults, residual faults and latent faults as defined in Figure 3 are a threat to safety and must be minimized. For ASIL D applications, for example, engineers might have to demonstrate that the proportion of single point faults is below 1%.



Safe faults
- Not in safety relevant parts of the logic
- In safety relevant logic but unable to impact the design function (cannot violate a safety goal)

Single point faults
- Dangerous, can violate the safety goal and no safety mechanism

Residual faults
- Dangerous, can violate the safety goal and escape the safety mechanism

Multipoint faults
- Can violate the safety goal but are observed by a safety mechanism
- Sub-classified as "detected", "perceived" or "latent"

Safe fault
Multiple-Point faults
Dual-Point faults
Single-Point or Residual faults
Single-Point Fault Metric
=
up to 99% required
Diagram Courtesy International Standards Organization (ISO)

Figure 3. Classification of faults in ISO 26262

D. Fault Injection and Fault Analysis

Fault injection is an established technique to understand the effects of faults on fault-tolerant systems [11,12]. ISO 26262 highly recommends fault injection as a method to verify the completeness and correctness of safety mechanisms against safety requirements, during hardware, software and system development [1]. When analyzing RTL and gate-level designs, the verification environment shall be able to inject faults and keep track of how they propagate through the design. The main challenges are to implement an easy-to-use and scalable environment capable to either quickly provide evidence that the design achieves its intended fault coverage targets, or clearly point out to shortcomings.

Fault simulation has gained widespread adoption as a method for fault injection and analysis. Simulation does scale to large designs, however, even for small designs, it cannot be exhaustive, and thus cannot prove that a fault will never propagate to critical design's outputs or registers. Even if a fault appears to be safe when exercising the design with a large and smartly constructed set of input stimuli, there is always the chance that a new test could show the very same fault causing a dangerous failure. Moreover, engineers might have to examine millions of faults, which is not feasible with simulation. The typical workaround is to examine a sample of faults and provide statistical evidence for fault coverage metrics.

III. Formal Fault Propagation Analysis

Most formal tools for assertion-based verification are optimized to work on RTL models. Modern tools can perform structural analysis on the design, including cone-of-influence analysis on signals and assertions, have apps to automatically generate assertions for specific tasks, and rely on numerous proof engines and strategies that users can tune to reduce proof runtimes or turn inconclusive results into pass or fail ones. Formal equivalence checking tools for synthesis verification, on the other hand, are routinely applied on large gate-level netlists. In recent years, sequential equivalence checking algorithms complementing traditional combinatorial equivalence checking have enabled the verification of advanced synthesis optimizations, including in FPGA flows. All these methods can play a role in formal FPA.

Applying formal technology to fault propagation analysis entails the injection of faults into hardware models and the setup of proof problems to assess if one or more signals in the corrupted design, typically referred to as observation points or safety-critical signals, have a different functional behavior compared to the original fault-free design. Moreover, should a fault affect an observation point, it may be important to know if a fault detection signal is activated. A crucial advantage of formal verification over simulation is that it does not require input vectors. The formal analysis is equivalent to examining all input sequences, enabling engineers to provide rigorous evidence that a certain fault is safe.

Ideally, engineers with no knowledge on formal methods should be able to apply the same push-button formal flow on any RTL or netlist design of arbitrary complexity, and quickly find all safe faults. In practice, this is not possible due to complexity issues, and engineers may have to explicitly select methods and proof strategies, depending on the design and the set of faults under analysis. The pragmatic approach we propose in this paper consists in defining two formal FPA modes, fast and deep, targeting two generic application scenarios. The main goal is to achieve optimal computational performances with two highly automated solutions covering all the application scenarios of advanced automotive SoCs developments.

A. Fast Mode

The Fast FPA mode requires as inputs a list of observations points, and an RTL or gate-level design model, and targets large fault populations. Runtimes to examine the entire fault population could range from minutes to several hours, with overnight runs acceptable for complex designs. In the fast mode the formal tool should automatically select appropriate proof methods and strategies to find the majority of safe faults, while aborting the analysis of hard-to-prove faults as soon as possible, thus minimizing effort spent on inconclusive results. To easy the task, the generation of debug information for propagatable faults can be dropped.

B. Deep Mode

The Deep FPA mode requires as inputs a list of observation points and a small list of faults. The methods and algorithms used in this mode shall be targeting hard-to-prove faults and produce detailed debug information for faults that are proven to propagate to observation points. Acceptable runtimes could range from seconds to a few hours per fault. In deep mode, it shall also be possible to leverage design knowledge in order to achieve performance improvements. For designs with long initialization sequences, for example, the user might benefit from starting the formal analysis after design initialization, a feature available on most modern formal tools.

IV. Integration with Simulation

A key benefit of the two-mode formal FPA approach outlined in the previous section is that it integrates with fault simulation flows, boosting quality of results (QoR) and productivity. Engineers can run formal FPA in fast mode on the whole fault population (see Fig. 4), before simulation. Faults that are proven safe immediately contribute to the fault coverage metrics and do not need to be further analyzed in simulation. Fault simulation is then applied to the reduced fault population, providing evidence to classify the majority of faults into detected and propagatable. Before spending effort analyzing the remaining potentially propagatable faults and attempting to improve the simulation testbench, engineers can run formal FPA in deep mode to identify additional safe faults that were missed by fast FPA, and to automatically generate input stimuli demonstrating how certain faults propagate to safety-critical outputs or registers. This approach reduces simulation effort and speeds up achieving the fault

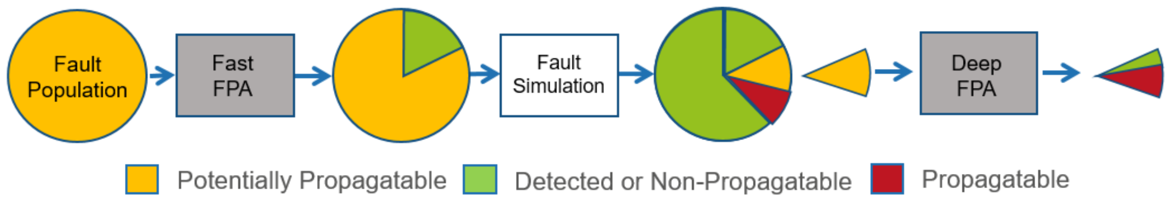


Figure 4. Integration of formal FPA with fault simulation

coverage results required to attain ISO 26262 certification.

V. RESULTS

We applied the two formal FPA modes using the Fault Propagation Analysis App (FPA™) from OneSpin's Safety Critical Verification Solutions portfolio. This app enables the two-mode approach through a simple interface, which also allows the user to adjust high-level proof settings. However, the complexity of proof methods and strategies employed by the tool is kept under the hood. The FPA app accepts a user-defined fault population, but can also create one that includes all potential stuck-at-0 and stuck-at-1 faults within the design. For each design, we provided a list of observation points that were the targets of the fault propagation analysis. The chosen observation points consisted in safety-critical outputs or registers. We also provided a small list of control inputs that were tied to their inactive logic values during the analysis. This list typically included test mode and debug mode inputs. The goal was to identify most of the safe faults during fast FPA. For deep FPA the goal was to reach conclusive results on a small set of faults. A subset of faults that were proven propagatable to observation points were further investigated to analyze their propagation path. Table I and Table II show results for the fast and deep FPA modes respectively. It is worth noting that the overall runtime in fast mode is usually proportional to the size of the fault population. In the deep mode, on the other hand, the runtime per fault can vary significantly.

Table I. Formal Fast FPA Results

	Identify Safe Stuck 0/1 Faults	
	RTL	***Gate-level***
Description	Open Core Processor – 10K LoC – 794 FFs	Summary of various designs
Fault Population	12.260 faults	Up to 2 million faults
CPU Time	2 minutes	Up to several hours
Safe Faults	3257 (26%)	Up to 14%

Table II. Formal Deep FPA Results

	Identify Safe and Propagatable Stuck 0/1 Faults	
	RTL	***Gate-level***
Description	Open Core Processor – 10K LoC – 794 FFs	Summary of various designs
Fault Population	12.260 faults	Up to 2 million faults
CPU Time	Averaging at 20 seconds per fault	Averaging at several minutes per fault
Safe Faults	13%	Up to 2.1% of the remaining faults

VI. CONCLUSION

Modern automotive SoCs include complex safety-critical functions and safety-mechanisms. To comply with ISO 26262 demands and compute the required fault metrics, engineers might have to inject and analyze millions of fault scenarios on both RTL and gate-level design models. Formal verification tools can provide a powerful means for the injection and rigorous analysis of faults. The method presented in this paper requires minimal user input, exploits a pragmatic divide-and-conquer approach, and integrates with simulation-based methods. Large fault populations can be addressed before any simulation effort. Dedicated formal algorithms can quickly derive results that reduce the subsequent simulation workload. Small fault populations, typically representing faults that could not be classified after simulation, can be addressed with a different set of algorithms and proof strategies that require longer runtimes but are more likely to provide conclusive results. The results presented demonstrate the applicability of this method to modern safety-critical designs.

REFERENCES

[1] "ISO 26262. Road vehicles - Functiona safety- Parts 1-10". First edition, Nov 2011.

[2] https://shemesh.larc.nasa.gov/fm/fm-what.html

[3] H. Busch, "An automated formal verification flow for safety registers", in proceedings of DVCon Europe 2015, Munich, Germany.

[4] H. Busch, "Formal safety verification of automotive microcontroller parts", ITG/GMM-Workshop ZuE 2012, Bremen.

[5] H. Busch, "Automated safety verification for automotive microcontrollers", in proceedings of DVCon 2016, San Jose, CA, USA.

[6] H. Busch, "Quantification of formal properties for productive automotive microcontrollers". in proceedings of DVCon 2013, San Jose, CA, USA.

[7] A. Traskov, T. Ehrenberg, S. Loitz, A. Ayari, A. Efody, J. Hupcey III, "Fault proof: using formal technques for safety verification and fault analysis", in proceedings of DVCon Europe, 2016, Munich, Germany.

[8] R. Baranowski, M. Trunzer, "Complete formal verification of a family of automotive DSPs", in proceedings of DVCon Europe, 2016, Munich, Germany.

[9] T. Blackmore, S. Marchese, F. Bruno, "Formal verification of a key block of the TriCore2 microprocessor", Euro DesignCon, 2004.

[10] T. Blackmore, F. Bruno, J. Bormann, S. Beyer, A. Maggiore, M.Siegel, S. Skalberg, "Complete formal verification of TriCore2 and other processors", in proceedings of DVCon 2007.

[11] H. Ziade, R. Ayoubi, and R. Velazco, "A survey on fault injection techniques", in The International Arab Journal of Information Technology, Vol. 1, No. 2, July 2004.

[12] Hsueh M. C., Tsai T. K., and Iyer R. K, "Fault injection techniques and tools," IEEE Computer, vol. 30, no. 4, pp. 75-82, April 1997.