HOW FORMAL REDUCES FAULT ANALYSIS FOR ISO 26262

DOUG SMITH, MENTOR GRAPHICS

WHITEPAPER

Mentor Graphics®

VERIFICATION

www.mentor.com

INTRODUCTION

The ISO 26262 standard defines straightforward metrics for evaluating the "safeness" of a design by defining safety goals, safety mechanisms, and fault metrics. However, determining those metrics is difficult. Systematic failure analysis can find simple faults, such as stuck-ats, but random hardware failure analysis, which poses a much harder challenge, is usually tackled through a process of *fault analysis*, including fault injection.

LIMITATIONS OF FAULT SIMULATION

Traditionally, safety analysis is handled by a *fault simulator*. Fault simulators use existing tests and regressions and randomly inject faults during simulation to determine if the injected fault impacts safety critical outputs. There are many possible methods. For example, using a simulation waveform dump from a golden run, a fault simulator injects faults and compares the results against the expected waveform to see if the fault eventually becomes masked or violates a safety requirement (e.g., a safety critical signal). Verification engineers use this fault information to calculate *failure rate metrics* (defined in the ISO 26262 standard) or *diagnostic coverage*, which is the proportion of the failure rate detectable or controlled by a safety mechanism. Or they use it to conduct function injection tests, which determine the effectiveness, correctness, and timing of a safety mechanism.

A complete safety analysis requires that engineers inject thousands of faults into hundreds of tests and simulate them to their outputs. This exponentially exploding workload swamps a simulation-based fault grading system. Additionally, simulators cannot determine which faults can be safely ignored, so they simulate a much larger list of faults than necessary. These issues force simulation-based fault testing systems to test a statistically insignificant number of safety-critical faults.

ADVANTAGES OF FORMAL VERIFICATION

A better approach and alternative to using fault simulation is using a formal verification technique known as *property checking* or *model checking*. Formal verification avoids the slowness of event-simulation by using mathematical algorithms and heuristics to prove the functional correctness of a design. Formal verification systems can also trace back a cone of influence from a safety critical signal or expression in order to prune the code of logic and reduce analysis time.

However, where formal really excels is in determining *transient faults*. Transient faults, such as a single event upset (SEU), occur while a design is operating. Formal engines have the ability to inject faults and formally ***prove*** whether the faults affect a safety requirement. Achieving a proof in simulation is usually impossible since it requires testing every possible input combination, but with formal technology, the safeness of a safety requirement can often be known absolutely and with the entire safety requirement's state space covered.

Consequently, formal tools explore both legal and illegal state spaces. This is easily remedied, however, using a technique called *Sequential Logic Equivalency Checking* (SLEC), which allows engineers to avoid testing invalid or undefined input paths. Using SLEC, the inputs do not need to be constrained (except as needed for clocks, resets, and any tied-mode bits), making formal verification an ideal method for performing safety analysis.

In this white paper, we will discuss how to use formal verification for static and transient fault analysis to generate the ISO 26262 safety metrics. First, we will look at some of the low-hanging fruit that formal analysis provides, and then we will discuss how to tackle the much harder task of fault injection.

FAULT PRUNING

One way in which formal can help with safety analysis is by reducing the set of fault injection points through a process referred to as *fault pruning*. While in theory all design elements could be faulty, only a subset of faults in a given design will affect a safety requirement. Therefore, any design elements not in the fan-in of a safety critical signal are automatically considered *safe*; i.e., contributing to the safe fault list (λ_S). A formal tool has the ability to trace back from the safety goal through the design elements to determine what is in the fan-in cone, also known as the *cone of influence* (COI). Any design elements not in the cone of influence are considered safe and do not need to be tested.

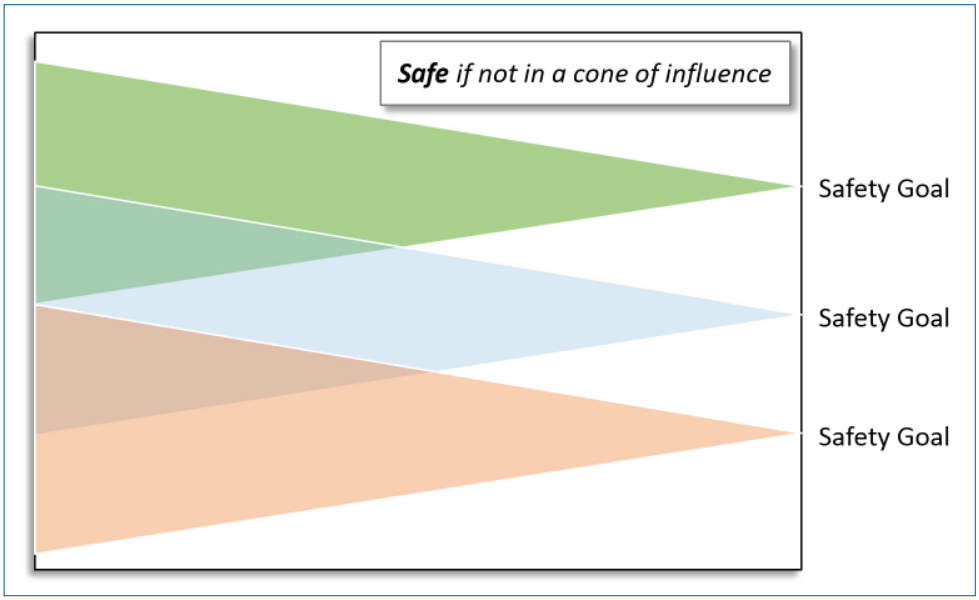

Figure 1: Formal tools trace back the cone of influence to perform their analysis.

As part of the ISO 26262 safety analysis, an engineer defines the safety requirements and safety mechanisms for a design. The first step in fault pruning is removing all design elements not directly in a cone of influence of a safety requirement, and thus consider only the design elements ***in*** a cone of influence. For example, here is a cone of influence report generated by a formal tool for a data bus:

```
--------------------------------------------------------------------------------
Questa Cone of Influence (COI) Report
--------------------------------------------------------------------------------
Questa Formal Version 10.5c linux_x86_64 23 Nov 2016
--------------------------------------------------------------------------------

Properties detected : 1
--------------------------------------------------------------------------------
cover_wb_dat_o
--------------------------------------------------------------------------------

 Registers:
  wb_dat_o    (32 Bits)
  divider     (16 Bits)
  ctrl[8:0]   (9 Bits)
  ss     (8 Bits)
  clgen.clk_out  (1 Bit)
```

```
  clgen.pos_edge (1 Bit)
  clgen.neg_edge (1 Bit)
  shift.tip   (1 Bit)
  shift.data    (128 Bits)

 Primary Ports:
  wb_clk_i    (1 Bit)
  wb_rst_i    (1 Bit)
  wb_adr_i[4:2]   (3 Bits)
  wb_dat_i    (32 Bits)
  wb_sel_i    (4 Bits)
  wb_we_i     (1 Bit)
  wb_stb_i    (1 Bit)
  wb_cyc_i    (1 Bit)
  miso_pad_i    (1 Bit)

 Counters:
  clgen.cnt   (16 Bits)
  shift.cnt   (8 Bits)

Active assumptions detected : 0
--------------------------------------------------------------------------------
```

As shown in Figure 2, the formal tool traces through the logic of the design finding all the major elements in the cone of influence back to the primary input ports. Formal tools can also give finer grain details, such as information about the signals or nets between the state elements.

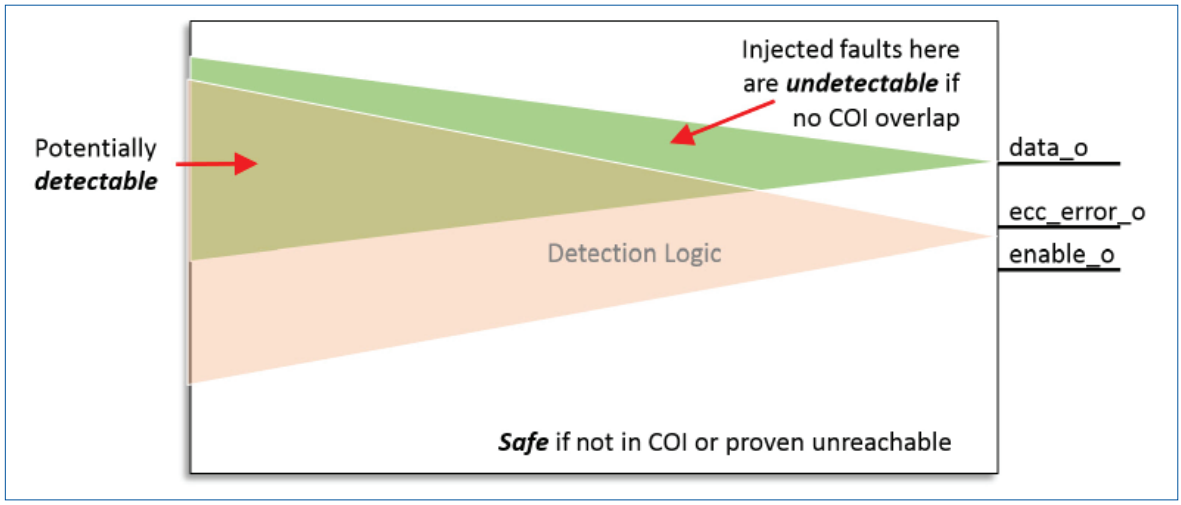

Figure 2: Anything outside the COI of the detection logic is considered unsafe.

The next step is to look at what design elements in the cone of a safety requirement overlap with the cone of the safety detection mechanism. For example, suppose a data bus is considered safety critical. Along with the data bus are extra error-correcting code (ECC) bits for detecting any errors. As long as the ECC bits flag any transient faults, the data bus is considered *safe* because it can be detected and dealt with downstream. Any design elements in the cone of logic of the data bus that are not in the cone of the ECC logic cannot be detected by the safety mechanism and so must be considered *unsafe*.

As shown in Figure 2, not all elements in the data path are part of the cone of influence of the detection logic, so these elements can be treated as undetectable or unsafe. Faults in this area would also contribute to a single point fault (λ_{SPF}) or residual fault (λ_{RF}) list. By pruning out logic whose faults will be detected by ECC, we reduce the number of design elements that need fault injection.

Likewise, the cone of influence can be reduced by applying any top-level constraints—such as disabling DFT, debug, and test—or other non-operational modes. As with synthesis, applying these constraints causes the formal tool to reduce the cone of logic, further pruning the faults required to be injected.

FAULT INJECTION WITH SLEC

Once the number of design elements have been pruned down to a more manageable and meaningful subset, formal can be used to inject faults. However, design inputs need to be constrained the same as with simulation. For example, proving a fault does not propagate in a non-operational mode is of very little value. This is where sequential logic equivalency checking (SLEC) plays a crucial role.

SLEC should not to be confused with *logical equivalency checking*. Logical equivalency checking (LEC) checks two designs to make sure that the clouds of logic ***between*** their state elements are equivalent. If the logic between registers is equivalent in both designs, then we are guaranteed that the designs will behave the same way, regardless of design modifications like DFT insertion. Thus, logical equivalency checking provides an exhaustive comparison between the internals of the two designs.

On the other hand, SLEC compares the ***outputs*** of two designs. The implementation of the two designs can be totally different as long as the outputs are always the same. For example, SLEC can be used to compare a VHDL design that has been ported to Verilog (or vice versa) to see if the two RTL designs are functionally equivalent. Conceptually, a SLEC tool is nothing more than a formal tool with two designs instantiated, constraining the inputs to be the same, and having assertions specifying the outputs are equivalent for all possible internal states.

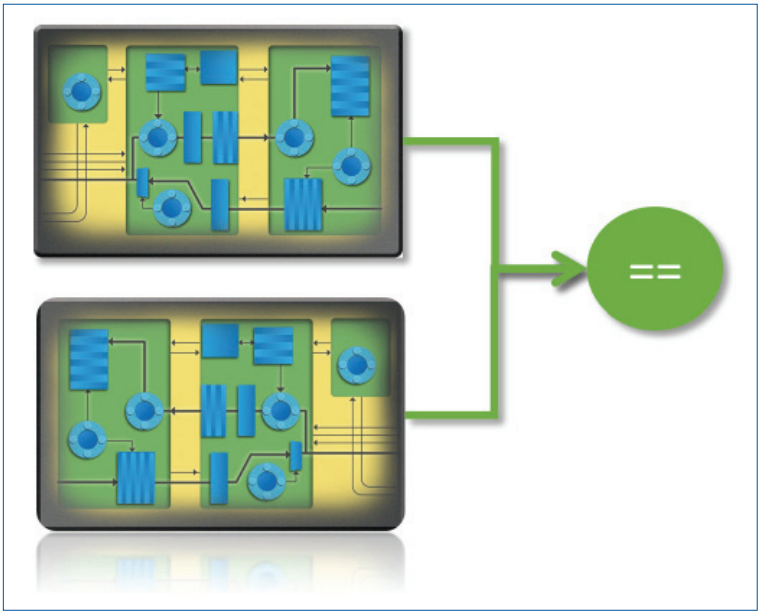

Figure 3: SLEC is used to determine if two designs have the same outputs, regardless of their implementation.

Under the hood, however, there are special formal techniques used by SLEC engines to pick equivalency points and abstract logic for handling designs with large state spaces. SLEC tools also automatically instantiate the designs and generate the necessary input constraints and output assertions.

While there are many uses for SLEC, fault injection is a particular sweet spot. Formal tools have the ability to inject both stuck-at and transient faults into a design, clock the fault through the design's state space, and see if the fault is propagated, masked, or detected by a safety mechanism. Furthermore, both *single-point faults* and *multiple-point faults* can be observed. Once the transient fault is injected, the formal tool tests all possible input combinations to propagate the fault through the design to the safety mechanism—this is where SLEC becomes necessary.

By instantiating a design with a copy of itself, all legal input values are automatically specified for SLEC, just as a golden reference model in simulation predicts all expected outputs for any input stimulus. The only possible inputs are those values that can legally propagate through the reference design to the corresponding output. By comparing a fault injected design with a copy of itself without faults, the formal tool checks if there is any possible way for the fault to either escape to the outputs or go undetected by the safety mechanism.

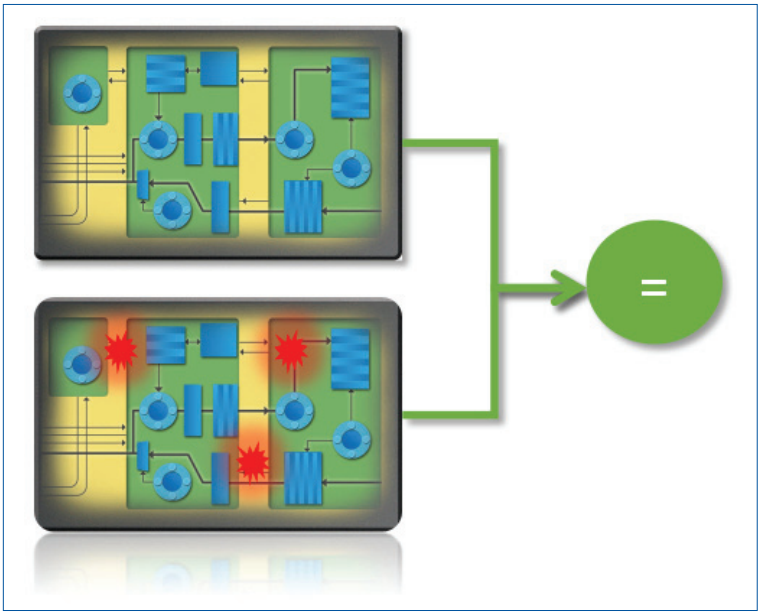

Figure 4: SLEC compares a fault injected and non-fault injected copy of the design.

Formal SLEC makes it easy to eliminate the input set up for safety fault analysis so that fault testing becomes a push-button, automated process. No more testbench required and no more test cases. Simply provide a list of safety critical requirements along with the safety detection logic and the formal tool automatically prunes the fault set, injects stuck-at and transient faults, and determines the diagnostic coverage of the design.

SUMMARY

Fault simulators generate metrics, but formal tools provide more meaningful safety failure rates. A formal tool can determine the number of safe faults (λ_S) by finding the unreachable design elements, those outside of a cone of logic, or those that do not affect the outputs (or are gated by a safety mechanism). Engineers can use the fault set from fault pruning to determine accurate numbers for single-point failures (λ_{SPF}), residual faults

(λ_{RF}), and multi-point failures (λ_{MPF}), and they can throw faults that prove too hard for formal (i.e., inconclusive proofs) into the undetectable or unsafe category.

Unlike simulation where it is never known if the design has been simulated enough or given enough input, formal verification conclusively determines if faults are safe or not, making the failure rates from formal analysis more than an arbitrary number determined by fault simulation. Formal analysis tools that apply SLEC techniques, like Questa® Formal from Mentor Graphics®, are an ideal solution for fault pruning, fault analysis, and determining diagnostic coverage. For more information, please check out our web pages at www.mentor.com and Mentor Consulting.

FURTHER READING

[1] ISO 26262 Road Vehicles Functional Safety Standard. www.iso.org.

[2] ISO 26262. Wikipedia. https://en.wikipedia.org/wiki/ISO_26262.

[3] Avidan Efody. Picking Your Faults: Advanced Techniques for Optimizing ISO 26262 Fault Analysis. White Paper. Mentor Graphics. https://www.mentor.com/products/fv/resources/overview/picking-your-faults-advanced-techniques-for-optimizing-iso-26262-fault-analysis-c5878d8e-20b4-4c56-88bf-e2f7ee16f6f5

[4] Avidan Efody. Getting ISO 26262 Faults Straight. Mentor Graphics. Verification Academy. https://verificationacademy.com/topics/planning-measurement-and-analysis/articles/Getting-ISO-26262-faults-straight.

[5] Avidan Efody. ISO 26262 Fault Analysis – Worst Case is Really the Worst. Mentor Graphics. Verification Academy. https://verificationacademy.com/topics/planning-measurement-and-analysis/articles/ISO-26262-fault-analysis-worst-case-is-really-the-worst.

[6] Questa Formal Verification Apps. https://www.mentor.com/products/fv/questa-formal-verification-apps.

Corporate Headquarters
Mentor Graphics Corporation
8005 SW Boeckman Road
Wilsonville, OR 97070-7777
Phone: 503.685.7000
Fax: 503.685.1204

Sales and Product Information
Phone: 800.547.3000
sales_info@mentor.com

Silicon Valley
Mentor Graphics Corporation
46871 Bayside Parkway
Fremont, CA 94538 USA
Phone: 510.354.7400
Fax: 510.354.7467

North American Support Center
Phone: 800.547.4303

Europe
Mentor Graphics
Deutschland GmbH
Arnulfstrasse 201
80634 Munich
Germany
Phone: +49.89.57096.0
Fax: +49.89.57096.400

Pacific Rim
Mentor Graphics (Taiwan)
11F, No. 120, Section 2,
Gongdao 5th Road
HsinChu City 300,
Taiwan, ROC
Phone: 886.3.513.1000
Fax: 886.3.573.4734

Japan
Mentor Graphics Japan Co., Ltd.
Gotenyama Garden
7-35, Kita-Shinagawa 4-chome
Shinagawa-Ku, Tokyo 140-0001
Japan
Phone: +81.3.5488.3033
Fax: +81.3.5488.3004

MGC 3-17 TECH15310-w