Methodology to Combine Formal & Fault Simulator to Measure Safety Metrics

Jain Gaurav, Kadambi Ranga, Bandlamudi Kirankumar, Busch Holger
Infineon Technologies AP Pte Ltd, Singapore & Infineon Technologies AG, Munich, Germany

Motivation

ISO26262 recommends measuring single point fault metric (SPFM) for Hardware Safety Mechanisms that are used for fault detection in safety critical modules. Safety applications with ASIL D require SPFM of 99% as per the standard. SPFM is function of safeness & diagnostic coverage.

Safeness is influenced by two factors:

- Untestable Faults which architecturally won't propagate the fault effects
- Unobserved Faults which are safe as test stimuli does not propagate the fault effects. This can impact SPFM. Creating tests for covering such faults is tedious.

Proposed Methodology includes use of Formal techniques in conjunction with Fault Simulations to overcome this challenge.

Fault Analysis With Fault Simulator

Fault simulation has the capability to inject hypothetical physical defects called faults into a design and verifying that a verification suite or another mechanism can detect it.

Figure 1 explains the underlying principle of fault simulation, which is to compare the strobe signals between good machine (simulation without fault) and faulty machine (simulation in the presence of fault).

- The fault effect is considered to be propagated (or detected) if there is a strobe value mismatch between good and faulty simulations.

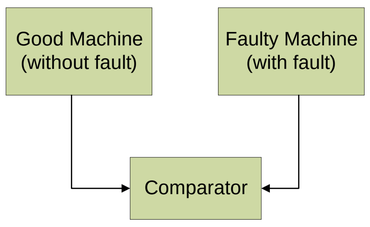

Figure 1

To improve the run time for single stuck-at faults, fault simulators employ techniques like

- **Fault collapsing**
- **Fault Sampling**

Formal Tool and Techniques

Formal Verification is the process of proving or disproving the correctness of the design using a formal specification or property on an abstract mathematical model of the system.

Formal tools can analyse digital designs given at register-transfer level in one of the common hardware description languages like VHDL or Verilog, or as a gate-level netlist. Figure 2 explains the analysis process

- Tools offer the advantage that they can generate exhaustive input scenarios which helps to explore all possible states of the design while proving a property. At the same time, constraints can be used to control the state space analysis, thus providing more controllability.
- Formal verification also allows for faults to be injected in the design and to activate them by property assumptions.

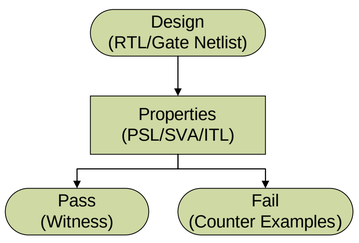

Figure 2

Phase I : Fault Extraction using Fault Simulator

Phase 1 (Figure 3) consists of Fault extraction for the safety critical IP

- Final tape-out netlist & cell libraries are input
- to a fault simulator.
- Fault simulator generates a list of untestable, prime & equivalent faults. Further, fault collapsing and sampling techniques are used to generate sampled list of prime faults.

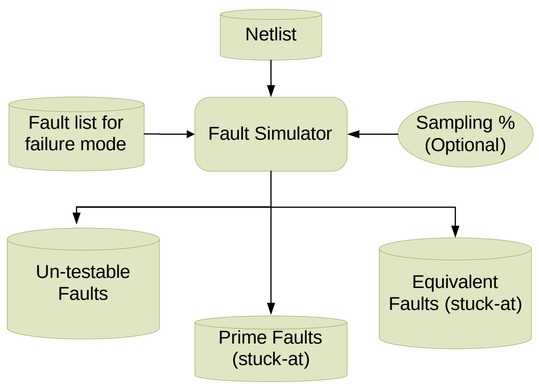

Figure 3

Phase II : Fault Simulations with Fault Simulator

In Phase 2 (Figure 4), a fault simulation campaign is performed using a fault simulator. A Prime fault list, generated in the Phase I, is simulated with the test suite. To measure fault metrics, the faults are classified into safe, dangerous and dangerous-detected.

Dangerous Fault: A fault effect propagated to the output ports of the module under analysis is considered to be dangerous.

Dangerous faults detected by a safety mechanism are counted as dangerous detected.

Safe Fault: A fault effect that does not propagate to any module output cannot violate a safety goal and is thus considered to be safe.

However, due to the non-exhaustive simulation test cases, some of the actual dangerous faults may be erroneously classified as safe faults. These faults are further analysed in Phase III with formal techniques.

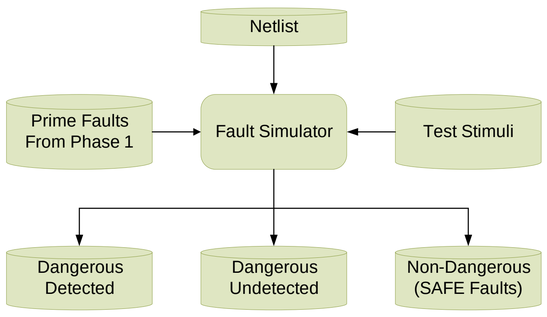

Figure 4

Phase III : Formal Analysis of Unobserved Fault

The non-dangerous faults from Phase II are subjected to fault injection using a formal tool, which stimulates the design exhaustively. The formal analysis helps to categorize the safe faults (as there is no dependency on test case), dangerous and dangerous detected based upon equivalence checking of the faulty and fault free design using System Verilog assertions (SVAs).

Redundant black boxing (RBB) (Figure 5) approach is proposed to achieve this goal.

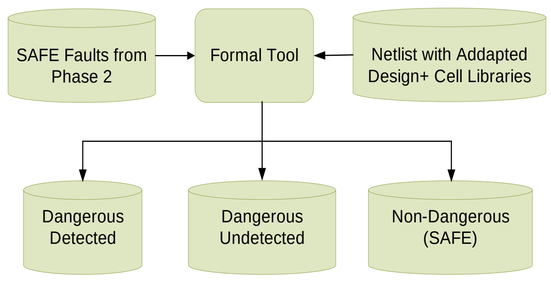

Figure 5

Redundant Black Box (RBB) Methodology

Figure 5 describes the architecture of a typical SoC consisting of safety-critical and non-safety-critical IPs.

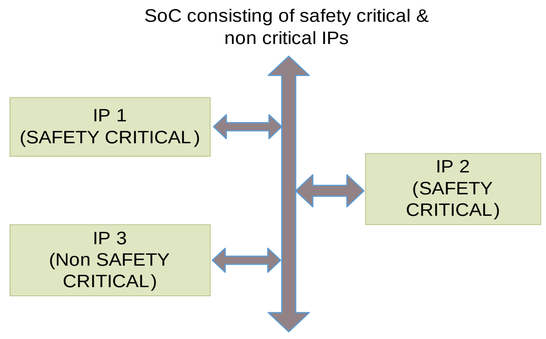

Figure 6

A Test architecture, according to the RBB methodology, is shown in Figure 7. Two instances of the safety critical IP are instantiated in the design wrapper replacing the existing instance.

SV assertions are developed to compare the output signals of both instances with one of these being fault-injected.

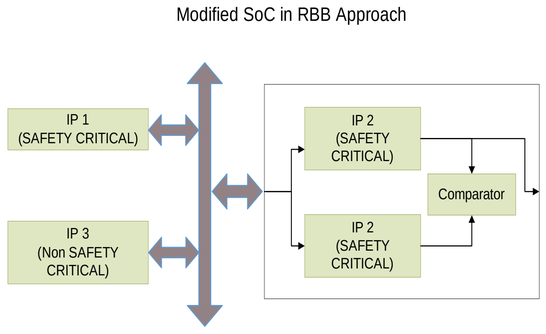

Figure 7

Advantages of **Redundant black boxing (RBB)** Methodology

- No impact on the existing design. Input signals drives both design instances. Output from faulty instance are mapped to the output signals.
- Automation to generate the modified design.
- Automation to generate System Verilog assertions.

Results

It is evident from chart in Figure 8 that,

- Fault Simulation results shows more SAFE Faults.
- SAFE Faults from Phase 2 are observed and detected with Formal techniques, thus formal overcomes the incompleteness of test stimuli.
- On combining the results of formal & fault simulation, the number of SAFE faults is reduced and the number of dangerous faults has increased.

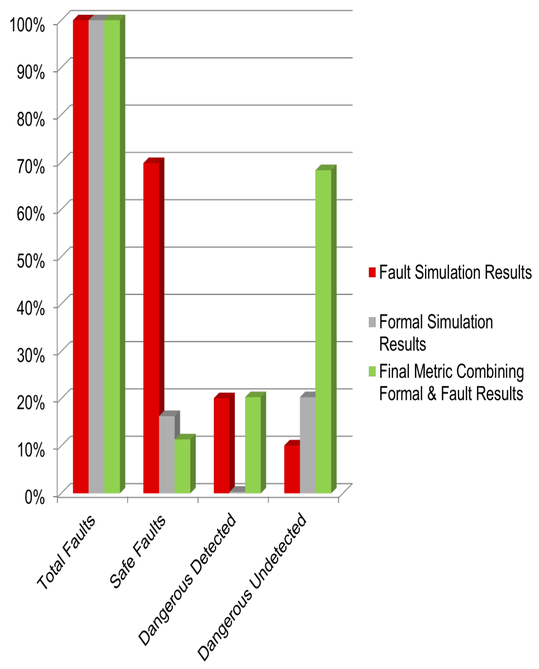

Figure 8

Conclusions

- Unobserved faults, which are safe as test stimuli does not propagate the fault effects, are detected with the formal techniques. Thus the efforts of creating new directed tests to check the fault effect are saved.
- Run time for formal analysis is minimized by using formal tool to generate counter-examples and by using functional constraints for the design.

Figure 9 highlights the final methodology. By combining the strengths of the Formal and Fault Simulations, it helps to achieve realistic Safety Metrics.

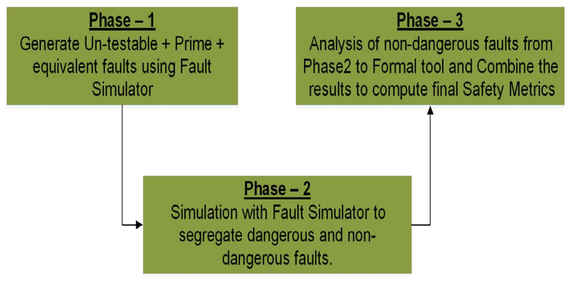

Figure 9