2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Methodology to combine Formal and Fault simulator to measure safety metrics

Jain Gaurav, Infineon Technologies AP Pte LTD, Singapore

Kadambi Ranga, Infineon Technologies AP Pte LTD, Singapore

Bandlamudi Kirankumar, Infineon Technologies AP Pte LTD, Singapore

Busch Holger, Infineon Technologies AG, Munich, Germany

***Abstract*— ISO26262 recommends measuring single point fault metric (SPFM) for Hardware Safety Mechanisms, that are used for fault detection in safety critical modules. The SPFM is function of the safeness fraction and diagnostic coverage of the intended safety mechanisms. The Safeness fraction is given by 2 components of fault types: Faults which architecturally won't propagate the fault effects, also known as un-testable faults, e.g. blocked, tied or redundant faults and Faults which are safe from the given failure mode as Test stimuli is not sufficient to propagate the fault effects, these are known as unobserved faults and this might result in a higher SPF result. Creating test stimuli to cover every unobserved fault is tedious. To measure realistic safety metrics, we propose a methodology to use FORMAL techniques in conjunction with FAULT SIMULATIONS.**

Keywords*— *Fault Injection, Formal, Fault Simulations, Single Point Fault Metric, Safness Fraction

I. INTRODUCTION

This paper is organized as follows; this section explains the motivation behind the methodology discussed in this paper. Section II discusses different fault models and the fault model refers for proposed methodology. Section III discusses fault collapsing, sampling and the fault simulation campaigns using dynamic fault simulator. Section IV discusses modelling of faults using formal techniques. Section V discusses the proposed methodology. Section VI combines results from fault simulator and formal together to derive single point fault metrics. Section VII discusses the limitation of using formal tool and proposes mitigation steps. Section VIII draws conclusion.

A. Abbreviations and Acronyms

ASIL: Automotive Safety Integrity Level

SPFM: Single Point Fault Metrics

SPF: Single Point Fault

DC: Diagnostic Coverage

FMEDA: Failure modes, effects, and diagnostic analysis

CPU: Central Processing Unit

SPI: Serial Peripheral Interface

B. Motivation

ISO 26262 is the comprehensive automotive safety standard that addresses the safety of the growing number of electric/electronic and software intensive features in today's road vehicles. The standard defines fault injection testing as a relevant method to be applied for different parts of the standard. At the hardware level (ISO26262-5, Table 3 and Table 11) [1] fault injection testing is recommended for highest ASIL whenever a hardware safety mechanism is defined to analyze its response to faults. Table I shows the requirement of fault metric for various ASIL targets for each safety goal.

Table I. Metrics for various ASIL targets

Fault Metrics	Automotive Safety Integrity Level		
	ASIL B	***ASIL C***	***ASIL D***
Single-point fault metric	≥ 90%	≥ 97%	≥ 99%
Latent-fault metric	≥ 60%	≥ 80%	≥ 90%

The single point fault metric is function of the safeness fraction and diagnostic coverage of the intended safety mechanisms on the non-safe fraction of faults as explained by equation (1)

$$SPF = F_{SAFE,SR} + (1-F_{SAFE,SR}) * DC_{FM,\,SM} \quad (1)$$

Where $F_{SAFE,SR}$ is the safeness fraction (or amount of safe logic portion) in the safety related logic and

$DC_{FM,\,SM}$ is the diagnostic coverage of the safety mechanisms.

The Safeness fraction is given by 2 components of fault types:

1. Faults which architecturally won't propagate the fault effects, for instance blocked, tied or redundant faults - these are usually known as un-testable faults.

2. Faults which are safe from the given failure mode point of view. Poor test stimuli can't propagate the fault effects and might result in a higher SPF result.

Fault simulators use test cases to stimulate the logic under analysis. Many times, the test case has limited coverage and this eventually leads to a higher safeness fraction resulting in a very optimistic or sometimes incorrect single point fault metric computation. Table II exemplifies single point fault metrics with different safeness fractions. Diagnostic coverage of 50% is assumed in this example.

Table II. Single Point Metric with Different Safeness Fraction

Failure Mode Distribution	*Safeness Fraction(F_{SAFE})*	*Diagnostic Coverage (DC)*	*Single Point Fault Metrics (equation 1)*
100%	20%	50%	60%
100%	40%	50%	70%
100%	60%	50%	80%
100%	80%	50%	90%

It is evident from Table II that the test case quality can give different single point fault metric numbers. It may help to SPFM target erroneously, whereas by improving the test suite quality diagnostic coverage can be measured accurately.

Alternatively, by using formal property checking with introducing faults in the design, has the potential to overcome the above limitation by performing exhaustive proofs/counterexamples for huge numbers of faults simultaneously. However, appropriate formal properties & fault models are required and formal proof times are longer in comparison to dynamic fault simulators which apply dedicates techniques like fault collapsing and sampling to reduce fault injection campaign times. In this paper, we propose a comprehensive framework to measure realistic single point fault metrics by combining the best of both fault simulator and formal worlds. In the same framework, an innovative approach is proposed to reduce the complexity in writing property rules for formal along with modelling faults in formal.

II. FAULT MODELS

A. *Fault Models*

- **Single stuck-at fault Model**: A single stuck-at fault model assumes to affect only the interconnection between gates. Each connecting line can have 2 types of faults: stuck-at-1 and stuck-at-0. A line with

stuck-at-1 fault will always have logic 1 irrespective of the correct logic output of the gate driving it. For a standard cell with "N" ports, there are "2N" single stuck-at faults.

- **Bridging faults**: These faults are the result of shorts between two neighbored nets. The knowledge of layout topology is mandatory to have realistic simulation of bridging faults.
- **Transient fault**: Transient faults are faults that occur once and subsequently disappear. These are of two types
    - **Single Event Upset**: occurs when a charged particle hits silicon transferring enough energy in order to provoke a bit flip in a storage element like a register or a memory cell
    - **Single Event Transient**: occurs when a single event may cause one or more voltages pulses (i.e. glitches) to propagate through the circuit.

Typically single event upset errors are possible at very high frequencies and lower geometries. Authors in [4] concluded that even at 28nm process, the frequency threshold to have SET is in the range of 0.9-2GHz range. ISO26262 recommends performing fault injection for D.C. faults (fault models like single-stuck at, bridging etc.) and transient faults (single event upsets) on ASIL D IPs.

III. FAULT ANALYSIS WITH FAULT SIMULATOR

Fault simulation is the capability to inject hypothetical physical defects called faults into a design and verifying that manufacturing test or verification suite or other mechanism can detect it. The underlying principle behind the simulation-based fault simulation is to compare the strobe signals between good machine (simulation without fault) and faulty simulation (simulation in the presence of fault). The fault effect is considered to be propagated (or detected) if strobe value mismatch between good and faulty simulations.

Single-stuck at fault model is most popular fault model in the industry. A circuit with N nodes consists of 2N possible single stuck-at fault sites. For modern designs, the number of single stuck-at is too high. To improve the run time for single stuck-at faults, fault simulators employ various techniques like fault collapsing. The reduction of single faults by removing equivalent faults is referred to as fault collapsing. Stuck-at fault collapsing typically reduces the total number of faults by 50-60% (2). Though this reduces the number of faults to be simulated, the number can be still high.

Fault sampling is another popular technique used to reduce the effort of fault simulation. In this technique, a subset of faults is randomly picked from the set of all faults. The faults in the sample are simulated and the sample coverage, i.e. the ratio of detected faults to all faults in the sample, is used as an estimate of the fault coverage in the complete fault set.

The accuracy (or error bound) of the estimated coverage depends only on the absolute number of faults in the sample. This number is known as sample size. Error in the estimate "Δ" is given by (2) . Theory and proof can be referred at [5]

$$\Delta = \frac{4.5}{N_S}\sqrt{1 + 0.44N_S x(1-x)} \quad (2)$$

Where N_S, is number of sampled faults

"x", is the coverage based on sampled faults. For example, with NS = 3000, x =96% the error = ±0.512%.

These techniques are used in the proposed flow in order to reduce fault injection time on stuck-at fault simulations. The bridging faults, unlike stuck-at faults are very much depends on the layout (proximity of metal lines, for example) and the number of bridging faults are less. However, the proposed methodology is applicable for all fault models.

IV. FORMAL TOOL & TECHNIQUES

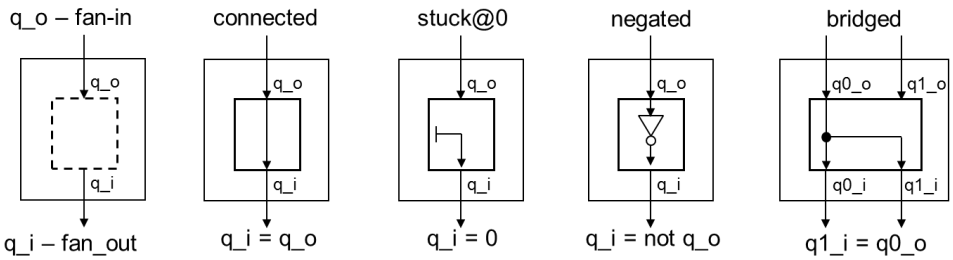
Formal Verification is the process of proving or disproving the correctness of a design using formal specification or property. The verification of the system is done by proving a formal proof/property on an abstract mathematical model of the system.

Formal tools handle 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. Properties can be developed using languages like SVA or PSL etc. Then analyze the design in presence of the properties using the formal tool. Properties can hold or fail in the analysis. Tools can generate witness (pass scenarios) or counter-examples (fail scenarios) which comes handy to analyze the design behavior. Based upon design complexities, formal analysis can take few minutes to hours' time to complete. Formal tools offer an 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 much needed controllability.

A. Formally Modelling Faults

The formal property checker allows any list of signals of an elaborated design to be instrumented for formal fault injection during compilation, including registers, internal nets, and variables. This instrumentation is achieved by specifying in a compilation option.

By way of extra formal assumptions which are added to formal properties [2], the new inputs can be specified with any required fault model at any time-point or interval within a property's examination window, or just with connected input- and output-parts in order to achieve the normal functional behavior of selected elementary parts.



B. Contribution of Formal Property-Checking to Diagnostic Coverage

When it comes to directed checking of single selected faults, running a simulation test-case with given stimuli is more efficient than a formal proof. On the other hand simulation test cases may be incomplete and thus fail to cover all faults. Here formal verification with fault injection comes again into play in order to decide the remaining undetermined faults. The different strengths of formal and fault simulation therefore suggest their combination.

V. REDUDANT BLACK BOXING(RBB) METHODOLOGY

From discussion in Section III, it's clear that there is need for comprehensive solution which can mitigate the impact of stimulus limitation during fault injection and can help to compute realistic SPFM. It is tedious to write test cases covering each and every uncovered fault.

From discussion in Section IV, it's clear that usage of formal tool gives advantage by generating random input stimulus for assertion analysis. Writing assertions based upon functionality is challenging and time consuming task. Developers need to be aware of design concept; otherwise it can lead to erroneous results. Also, formal tool has inherited problem of longer run time. Tool can take few second to hours' time to prove a single assertion.

With this paper, we are proposing a methodology to combine both fault injection using fault simulator and formal tool to measure realistic SPFM. This methodology is split into three phases. Figure 1 describes this methodology consisting of three phases. These phases are discussed in detail in this section.

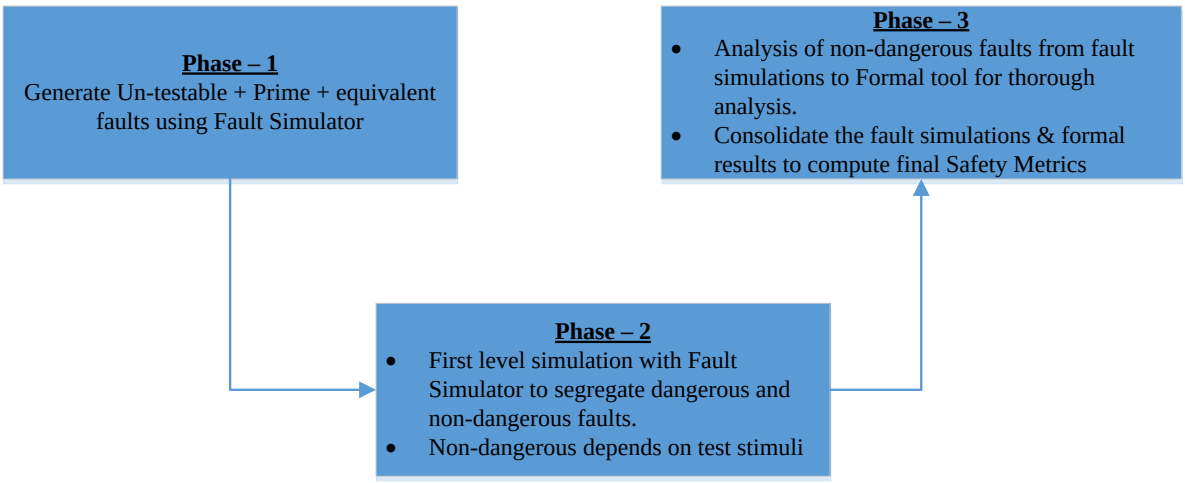


Figure 1: Methodology explaining combining of Fault Simulations & Formal Analysis to compute Safety Metrics

A. *Phase - 1*

Figure 2, describes the architecture of a typical SoC consisting of Safety critical (e.g. CPUs) & non-safety critical IPs (communication IPs like SPI). Safety critical IPs comes under the scope of fault injection.

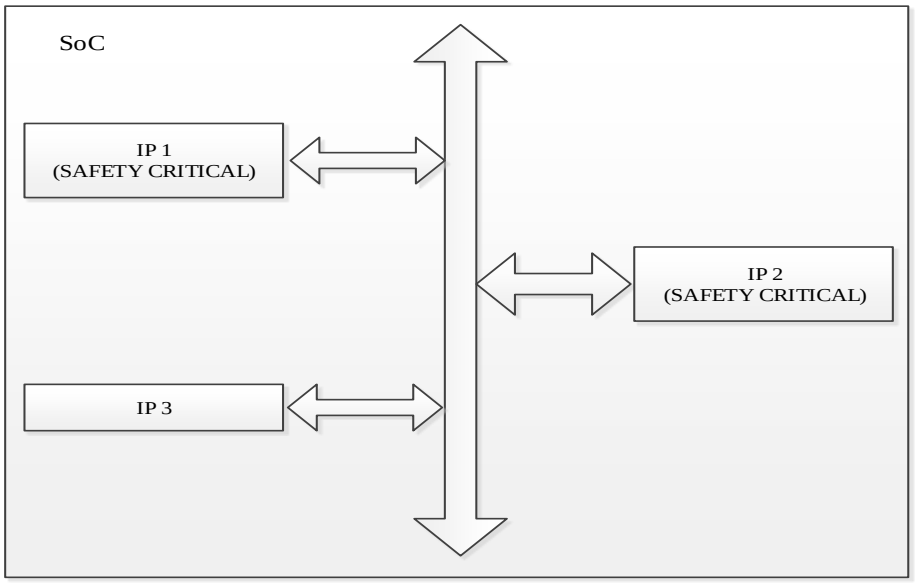


Figure 2: A SoC consisting of Safety Critical & Non-Safety critical IPs

Phase I consists of the 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. If the number fault is high, fault sampling is performed. Prime fault list is taken to Phase 2 for fault simulations. Figure 3 describes this phase.

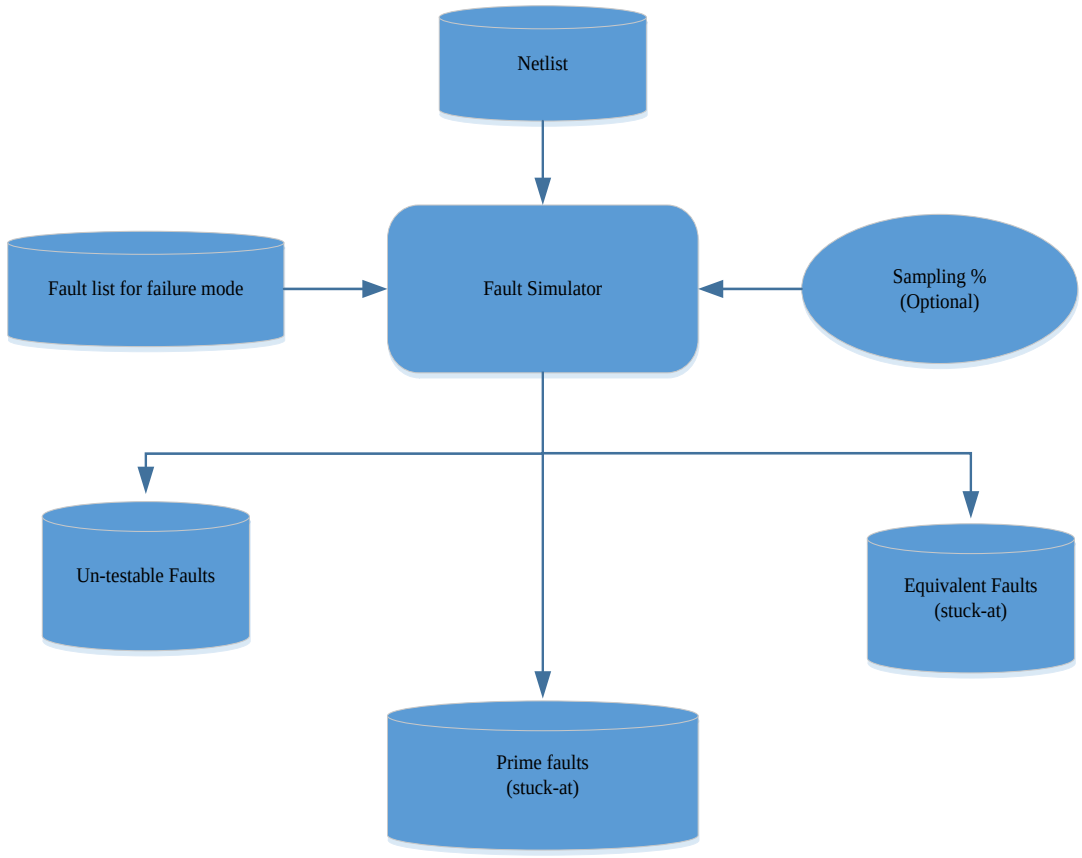


Figure 3: Methodology Phase 1 extracting fault using fault simulator

B. Phase – 2

Fault simulation campaign is performed using fault simulator. Prime fault list, generated in the Phase I, is simulated with the test suite. To measure fault metric, the faults are classified into safe, dangerous and dangerous-detected. Figure 4 describes this phase.

- 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.

Fault simulator run times depend upon the individual test case run time. However, due to the non-exhaustive simulation test cases, some of the actual dangerous fault may be erroneously classified as safe faults.

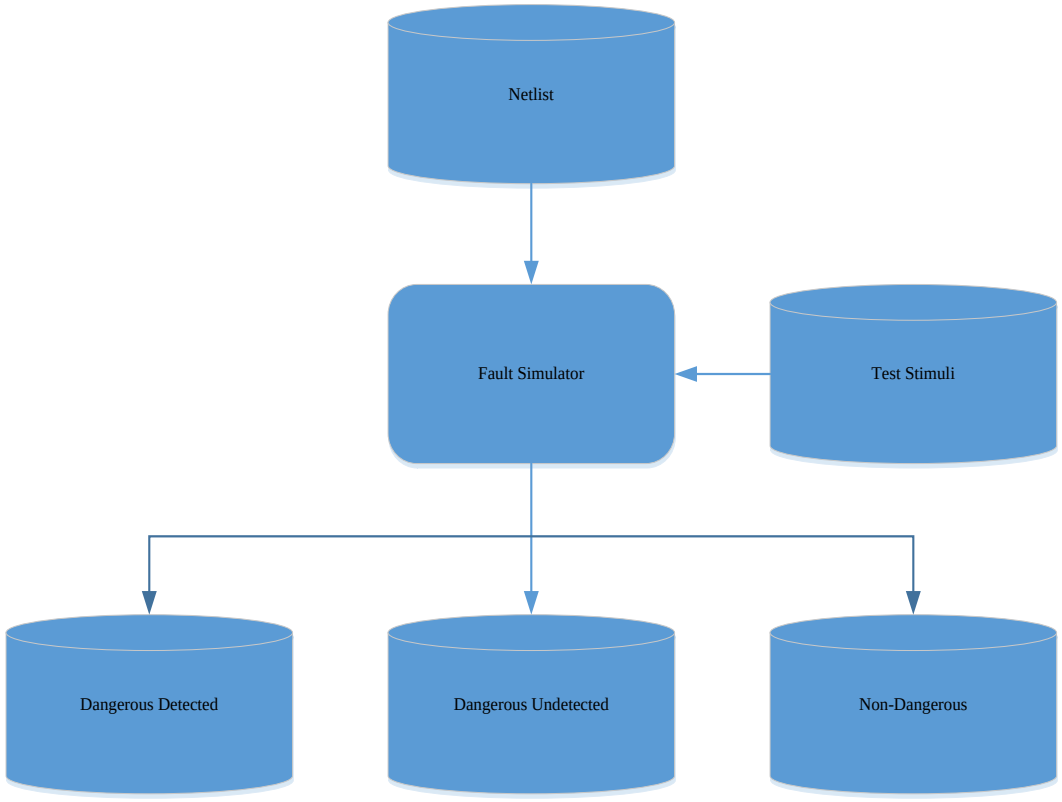


Figure 4: Methodology Phase II performing fault simulations

C. Phase – 3

The non-dangerous faults from Phase II are subjected to fault injection using a formal tool, which stimulates the design exhaustively. The formal tool categorizes truly safe faults (as there is no dependency on test case), dangerous and dangerous detected. A redundant black boxing (RBB) approach is proposed to allow for automated property generation and proofs in order to save the manually property development effort and reduce proof complexity. Figure 5 describes this Phase.

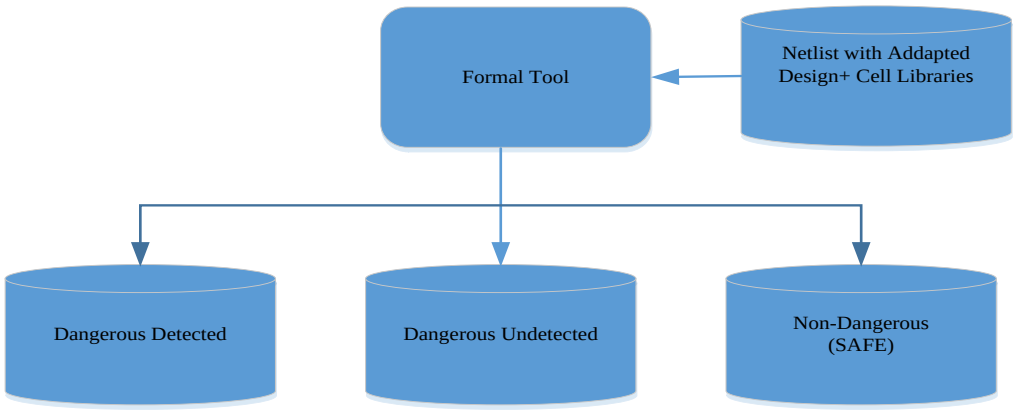


Figure 5: Methodology Phase III Fault Analysis with Formal Tool

Figure 6 describes the design adaptions done in RBB methodology. Property based equivalence checking methodology [3] is applied and characteristics of the adaptive design are

1. A design wrapper with same input and output connections replaces the existing IP; thus it does not impact the existing interfaces.
2. Two instances of the reference design are instantiated inside the wrapper. Output of one of the design instance drives the output interface signals.
3. One of the reference design instance acts as GOLDEN i.e. no fault is injected on this instance. Other acts as FAULTY design.
4. A comparator is instantiated to compare the output of the two design instances.
5. System Verilog assertions are developed to check the validity of the comparator outputs. The output nodes can be Hardware Safety Mechanism also, which are part of the design to detect the fault in the design.

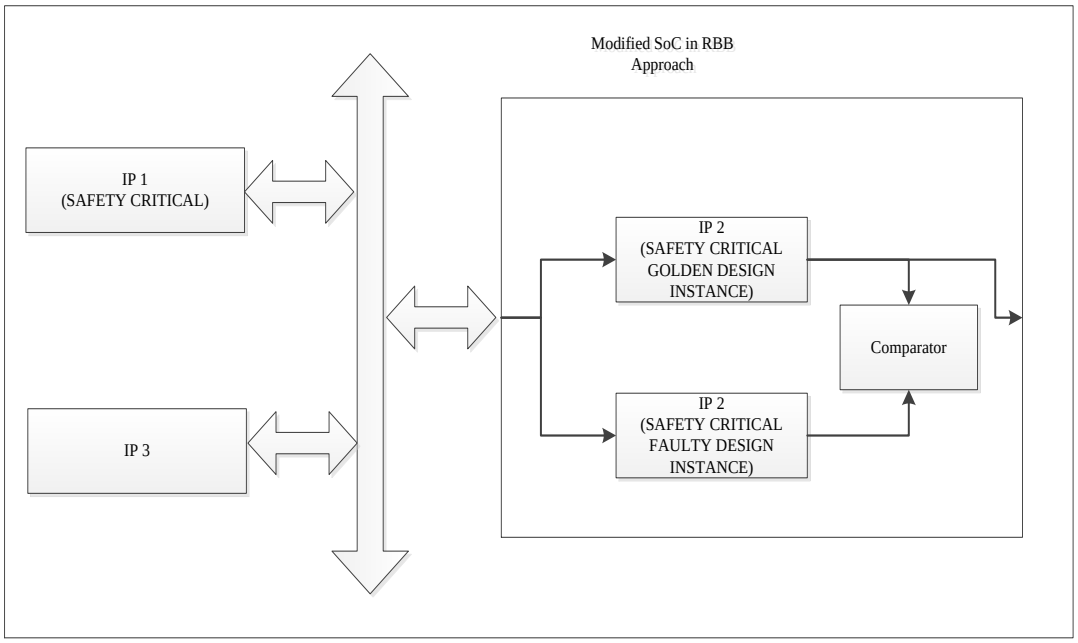


Figure 6: SoC in RBB Methodology

Following steps are performed for fault analysis using formal tool.

- Design wrapper and cell libraries are read in the formal tool. Then design is elaborated and compiled to generate the formal model.
- Firstly, assertions are proved on the design without fault. If assertions are failing then input constraint are refined till assertion holds. Counter-examples can be generated and analyzed further for the failing scenario. Elaboration & compilation steps are performed with updated constraints.
- Then Fault Injection campaign is carried out. Fault is injected on one of the design instance and outputs are compared between GLODEN design & FAULTY design. This behavior is validated by checking if assertion holds or fails.
    - If any of the assertion fails, this implies that output of the design with fault is different from the design with no fault. So, fault effect is propagated to the output node and causing comparison signal to fail. Hence fault is classifies as DANGEROUS fault.
    - If assertion holds, this implies that output of the both design instances are same and fault effect can't propagate to the output of the design with fault. Hence the fault is classified as SAFE fault.

VI. RESULTS ANALYSIS COMBINING FAULT SIMULATIONS & FORMAL ANALYSIS

This methodology was executed and proven on one of the safety critical IP present in microcontroller SOC. Results from different phase are presented below

A. Phase-1 Execution & Results

Table III mentions the total number of faults present in the design with metric consisting of testable & untestable faults. It also mentions the prime fault present in the design with testable and untestable fault numbers.

Further, sampling is done in order to perform statistical analysis on the prime fault list. After sampling, 13977 prime faults are considered for fault injection to be performed in Phase II.

Table III: Fault Metrics (Output of Phase I)

Design	***All FAULTS***	***PRIME FAULTS***
Total No of Faults	185188	133014
Testable Faults	185111	132968
Untestable Faults	77	46

B. Phase-II Execution & Results

As it is clear from the Table IV, fault simulation results contains very high percentage of safe faults. This can be due to limitation in the test stimuli which did not exercise the fault exhaustively. Thus fault did not propagate to the output node. This fault will be further analyzed in Phase III with RBB approach.

Table IV: Fault Simulation Results (Output of Phase II)

Fault Metrics	***No. of FAULTS***	***% Representation (wrt Total Faults)***
Total No of Faults	13977	100
SAFE UNDETECTED	9759	69.8
DANGEROUS DETECTED	2808	20.1
DANGEROUS UNDETECTED	1410	10.1

C. Phase-III Execution & Results

It is evident from Table V that undetected faults computed in Phase II can be observed and detected using the RBB methodology. With this approach, number of safe faults is reduced to 16.3% of the original SAFE faults. Simultaneously, number of faults which can be detected at the output node has increased to 83.7%. Thus, limitation of simulation can be resolved using formal tool and hence helps to compute the safety metrics realistically.

Table V: Formal Analysis Results (Output of Phase III)

Fault Metrics	***PRIME FAULTS***	***% Representation (wrt Total Faults)***
Total No of Faults	9759	100
SAFE UNDETECTED	1587	16.3
DANGEROUS DETECTED	29	0.3
DANGEROUS UNDETECTED	8143	83.4

Finally combining the results from Phase II & Phase III, it is evident the more faults are detected with this approach. Table VI highlights this observation. Number of SAFE faults is reduced to 11.4% from the original 69.8%. Also, number of dangerous fault has increased to 88.6%.

Table VI: Combined Results of Formal Analysis & Fault Simulation Results

Fault Metrics	***PRIME FAULTS***	***% Representation (wrt Total Faults)***
Total No of Faults	13977	100
SAFE UNDETECTED	9759-8143-29=1587	11.4
DANGEROUS DETECTED	29+2808=2837	20.3
DANGEROUS UNDETECTED	8143+1410=9553	68.3

VII. LIMITATION AND MITIGATION

Major limitation of using formal proof for analyzing fault analysis is the run time. Run time is more as compared to fault simulations. This limitation is mitigated using following techniques

1. Using formal tool to generate counter examples only. As the emphasis of this methodology is to find fault which can propagate to output node, feature of generating counter examples with formal tools can be referred.
2. Constraints: User can specify functional constraints based upon the design while compiling the design. For example, scan enable signal or test related signals can be disabled using constrain. This will help formal tool to reduce input scenarios for assertion analysis and hence speeds up the analysis.

VIII. CONCLUSION

ISO26262 recommends to measure SPFM for hardware safety mechanisms used in safety critical IPs. Fault simulations are most common method used to measure SPFM. This helps to check the correctness & completeness of Hardware Safety Mechanism. Fault analysis with this approach depends upon the quality of the test suite referred. If test stimulus cannot exercise the data path where fault is present, fault can't be checked. Hence completeness of the hardware safety mechanism will not be verified. Fault will be classified as SAFE fault and lead to wrong computation of SPFM. This needs to be addressed.

With this paper, we have discussed in details about the fault injection using fault simulators and capability of formal tool. We have listed down pros and cons of both techniques. Based upon we are proposing a methodology where capabilities of both fault simulator and formal tool are combined to achieve realistic safety metrics. This concept is proven with the data measured from a real life example of a microcontroller.

REFERENCES

[1] ISO 26262 Standard for Road Vehicles – Functional Safety, Parts. 1-10, 15 Nov. 2011.

[2] H. Busch, "Formal Safety Verification of Automotive Microcontroller Parts," ITG/GMM-Workshop ZuE 2012, Bremen, July 2012

[3] H. Busch, "Automated Safety Verification for Automotive Microcontrollers," DVCon US-2016.

[4] Mahatme, N.N.; Gaspard, N.J.; Jagannathan, S.; Loveless, T.D.; Abdel-Aziz, H.; Bhuva, B.L.; Massengill, L.W.; Wen, S.; Wong, R., "Estimating the frequency threshold for logic soft errors," Reliability Physics Symposium (IRPS), 2013 IEEE International , vol., no., pp.3D.3.1,3D.3.6, 14-18 April 2013 doi: 10.1109/IRPS.2013.6531991

[5] Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits M. L. Bushnell, V. D. Agarwal