# Static Structural Analysis and Formal Verification of SoC with Software Safety Mechanisms for Functional Safety

Hyunsun Ahn, Samsung Electronics Co., Ltd., Korea (hyunsun.ahn@samsung.com)
Euisang Yoon, Siemens EDA, Korea (euisang.yoon@siemens.com)
Namyul Cho, Siemens EDA, Korea (namyul.cho@siemens.com)
Arun Gogineni, Siemens EDA, USA (arun.gogineni@siemens.com)
Ann Keffer, Siemens EDA, USA (ann.keffer@siemens.com)
Sungjin Park, Siemens EDA, Korea (sungjinpark@siemens.com)
Sungyun Yoo, Siemens EDA, Korea (sungyun.yoo@siemens.com)
Bumju Kim, Samsung Electronics Co., Ltd., Korea (bumju.kim@samsunmg.com)
Junhyuk Park, Samsung Electronics Co., Ltd., Korea (jh23.park@samsung.com)
Youngsik Kim, Samsung Electronics Co., Ltd., Korea (ys31.kim@samsung.com)
Seonil Brian Choi, Samsung Electronics Co., Ltd., Korea (seonilb.choi@samsung.com)

***Abstract*- Fault campaigns in large System on Chip (SoC) designs are complex and critical for ensuring the reliability of software safety mechanisms. This paper presents an integrated approach using static structural analysis and formal verification techniques to detect and resolve faults early in the design process. By integrating these methods, we aim to enhance diagnostic coverage and ensure compliance with Automotive Safety Integrity Level (ASIL) requirements. Our methodology involves a multi-step process that includes gap analysis, static structural analysis, and formal verification to identify and mitigate potential faults. A case study on a Samsung automotive project demonstrates the effectiveness of this methodology, showing improvements in fault detection and resolution. The results highlight the importance of combining static and dynamic approaches to achieve comprehensive diagnostic coverage and ensure the highest standards of safety and reliability in automotive systems.**

## I. Introduction

Conducting fault campaigns in large System-on-chip (SoC) designs is a tedious and challenging process. Early detection of faults in semiconductor circuits is crucial for ensuring the reliability and effectiveness of software safety mechanisms (SMs). This paper presents the benefits of using software safety mechanisms and software task libraries in an IC design to improve diagnostic coverage (DC). We will talk about how software test libraries (STLs) are used in functional safety to ensure the correctness of software safety mechanisms and how to apply an integrated approach to prove diagnostic coverage using both static and dynamic fault analysis through innovative structural analysis and formal verification techniques. This paper aims to show how this methodology improves the detection and resolution of faults and the overall safety and reliability of SoC designs. Our approach not only identifies faults in the early stage but also provides a framework for continuous improvement in fault detection and resolution processes.

## II. BACKGROUND: SOFTWARE SAFETY MECHANISMS

### *A. Definition: Software Safety Mechanisms*

Using software Safety Mechanisms (SM) to protect a design targeted for the automotive market is becoming a popular approach for ICs/SoCs because it facilitates a structured and effective approach to achieving coverage metrics. Software SMs have many benefits, such as:

1) Flexibility: They can be added to an existing safety-related IC/SoC after tape-out/production to increase safety coverage gaps without re-spinning the whole IC/SoC.

2) Consistency: They Ensure consistent code across different components in the design, projects or systems.

3) Efficiency: They Save time in test development, allowing teams to focus more on safety analysis and writing new code.

4) Real-time monitoring: Many include monitoring tools that check system performance, allowing for immediate detection of anomalies.

5) Improved documentation: They Help ensure thorough documentation of safety requirements, which aids in traceability and audits.

6) Contribution to ASIL metrics: They Contribute to ASIL metrics for ASIL level A through D.

7) Savings on hardware footprint: They Save chip real estate and cost.

Typical components of software safety mechanisms include software task libraries and self-test libraries. Software task libraries ensure safe access to memory regions, facilitate reliable data exchange between different parts of a system, and monitor system health.

*B. Definition: Software Test libraries*

Software Test libraries (STLs) are used to test, validate, and ensure software safety mechanisms' correctness during systematic testing. STLs are beneficial for testing software safety mechanisms because they are:

1) Reusable: STLs provide a collection of reusable testing components, frameworks, and tools that can be reused across different designs and projects, which streamlines the testing process.

2) Efficient: Using pre-designed software libraries saves engineering time, and helps meet deadlines while ensuring ISO 26262 compliance.

3) Traceable: They help in tracking tests and coverage used throughout development used as work products for audit and certification.

4) Consistent: They provide a set of standardized tests for consistent, expected results.

5) Standardized: They Deliver standardized testing methods that align with the safety process defined by the ISO 26262 standard.

For this paper, we will refer to all of these as STLs, which are used as a software safety mechanism for functional safety.

*C. Execution of Software Safety Mechanisms*

Software SMs can be executed in several ways. Some popular methods are:

• Build software SM code as a binary and load it in internal RAM/ROM. It is preferable when the STL function size is small.

• Build software SM code as binary and load it to external DDR. This approach is best when the RTL function size is big. In Figure 1, the software SM can be loaded into both the internal RAM and DDR memory. This mechanism will protect four different modules: the DDR controller, peripherals, video system, and audio system.

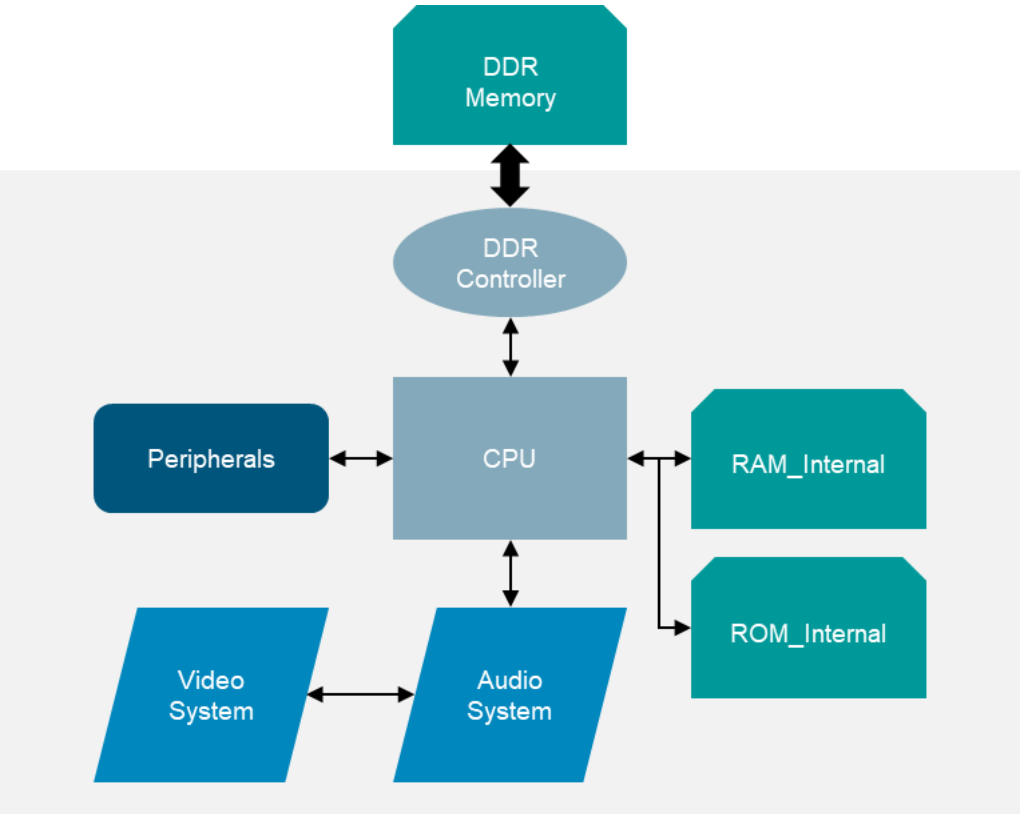


Figure 1. Simplified structure of complex SoC system

## III. STRUCTURAL ANALYSIS AND FORMAL VERIFICATION TO PROVE DIAGNOSIS COVERAGE

As mentioned above, implementing effective Software SMs can significantly contribute to meeting the target requirements of ASIL metrics. If the diagnostic coverage provided by these Software SMs can be evaluated in the early stages of implementation, the effectiveness of the mechanisms can be pre-validated. Structural analysis and formal verification techniques can be employed to achieve this.

Structural analysis focuses on identifying design patterns and protocols that can reduce the scope of proof. By leveraging formal verification, we can evaluate failure states without requiring simulation execution or dependency on software stack readiness. This technique allows for an early proof of concept for software safety mechanisms, even before the complete software stack is developed.

The process involves three key steps:

### *A. Gap Analysis to Identify Areas for Coverage Improvement*

By reviewing the target fault space and design, we can identify areas where coverage improvement is possible through software SMs, rather than relying solely on pre-existing hardware SMs. Figure 2 shows a full matrix interconnection, where each slave port can access all the master ports depending on the address range. Pre-existing hardware SMs implemented in Master IP1, Slave IP1 do not cover the entire bus matrix. The uncovered areas are where software safety mechanisms need to be implemented to achieve coverage improvement.

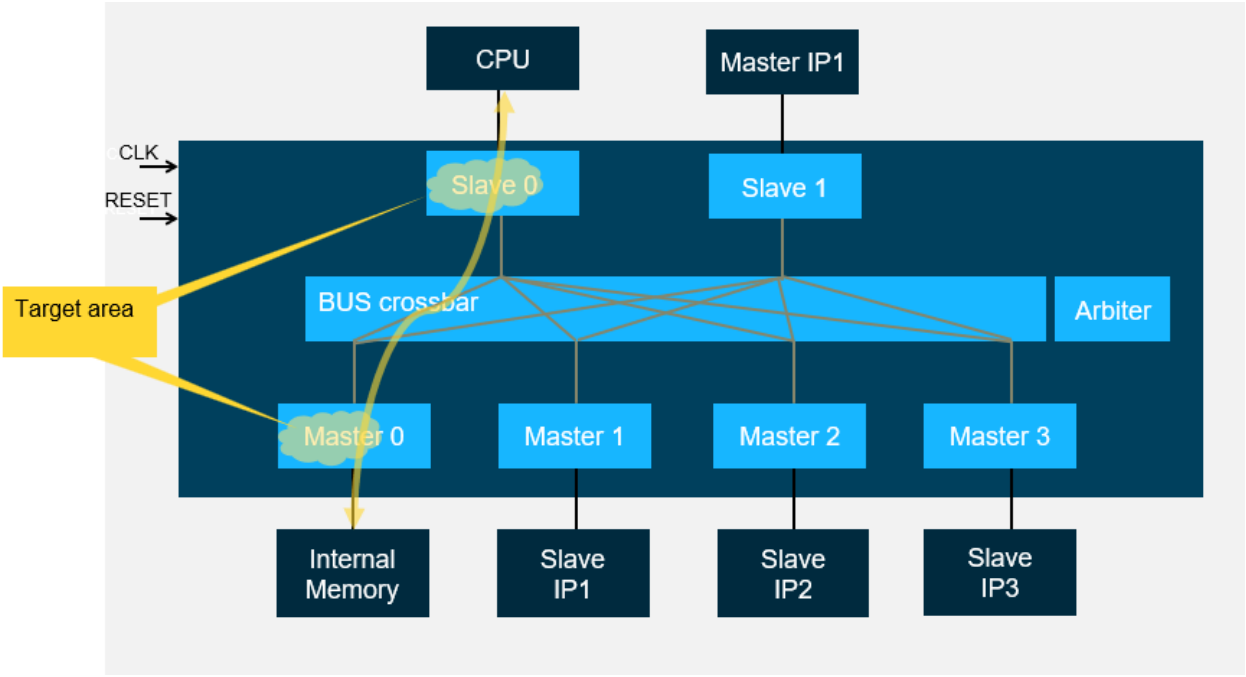


Figure 2. AXI interconnect bus matrix and IP connections

### *B. Static Structural Analysis to Prove Protocol Connectivity*

Once the target area for coverage improvement is defined through gap analysis, the connectivity of this target area can be verified using static structural analysis. Siemens EDA's SafetyScope™ can be used for this static structural analysis. SafetyScope supports multiple structural analysis modes, including connectivity analysis and data path analysis. We refer to this as the River Flow Mode (RFM) analysis of SafetyScope.

#### *B.1 RFM analysis basics*

RFM analysis is a data path coverage analysis that assigns safety mechanism coverage to the entire data path, from the generation (GEN) point, where the data is generated, to the check (CHK) point, where the data is consumed.

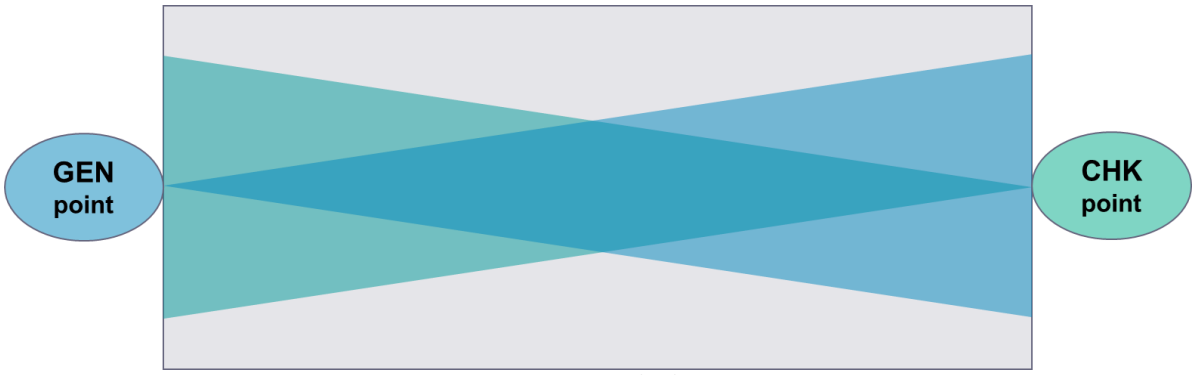

Figure 3. RFM analysis of SafetyScope

*B.2 Data Path analysis*

In data path analysis, we do:

1) Traverse backward from each CHK point and mark all visited logic.
2) Traverse forward from each GEN point and mark all visited logic.
3) Identify the logic that appears in both forward and backward traversals as the intersection.
4) To isolate the logic that is part of the data path only, remove all covered register/latch banks with bus width less than rfm_filter_width.
5) Repeat steps (1) and (2), but only traverse through the preserved registers and latches identified in step 4 for multi-level paths.

The logic cone that remains after this analysis will be considered part of the data path.

*B.3 Structural analysis for a targeted area*

The targeted area shown in Figure 2 is part of a data path. If we zoom into the internals of the crossbar, we see the following data packet flow as in Figure 4:

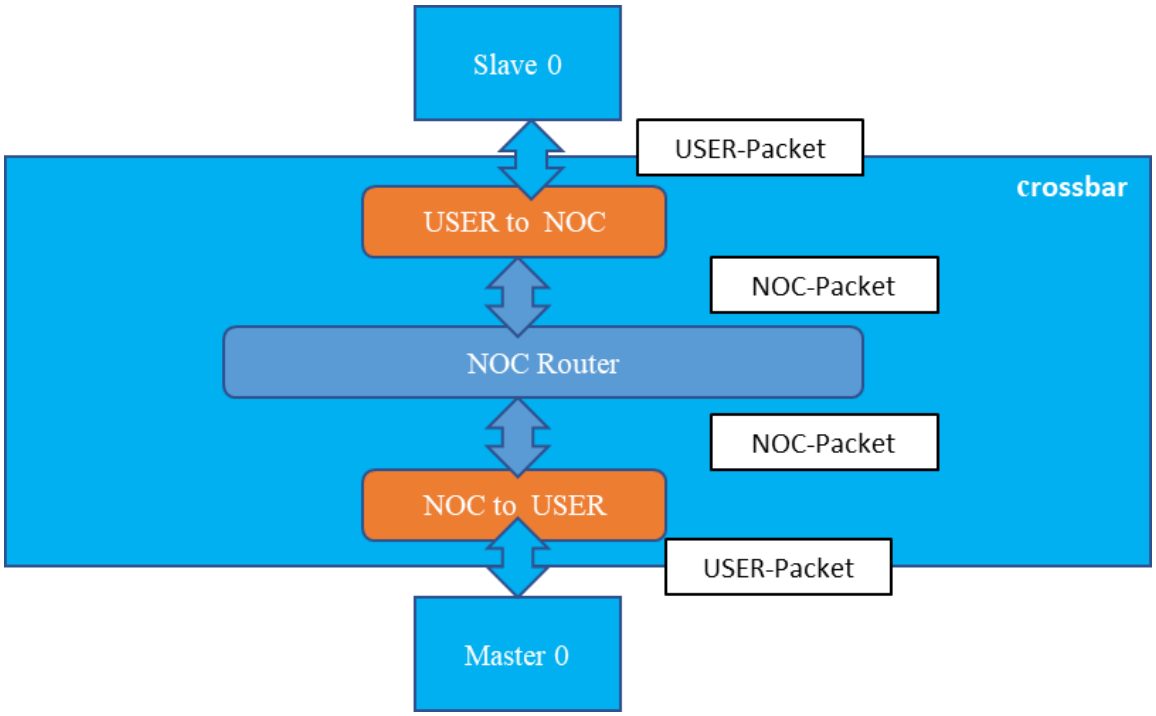

Figure 4. Data Packet flow in the bus matrix

1) Data starts as a USER packet (e.g., AXI transaction) at Slave0, which is connected to the crossbar.
2) In the crossbar, the USER packet is converted to a NOC packet by the protocol converter engine (USER to NOC).

3) The crossbar then routes the NOC packet to the appropriate Master (Master 0) NOC interface (NOC to USER).
4) The packet is converted back to a USER packet and sent to Master 0.
5) The data remains intact throughout this path, which is essential for the SafetyScope data path algorithm.

Using SafetyScope RFM, we extract all the data path nodes that can be covered with data path protection in software safety mechanisms. This list is then provided to the formal verification engine to prove the coverage.

### *C. Formal Verification on Reduced Space to Prove Fault Coverage*

Siemens EDA's Questa® Equivalent RTL is used as a formal verification solution to prove whether stuck-at-0 or stuck-at-1 permanent faults are detectable or not. The basic idea involves comparing normal RTL and faulty RTL through a formal equivalence check. This process closely resembles fault injection at a specific point within a conventional fault simulation procedure. As shown in Figure 5, the normal RTL is compiled as a design unit named SPEC, while the faulty RTL is compiled as a design unit named IMPL. To proceed, the prerequisite materials include RTLs of the module, the address map of the bus interconnect, the design architecture, the target fault list, and the formal verification tool Questa® Equivalent RTL.

While various other static and formal verification solutions available in the market could potentially be applied to this methodology, we chose Siemens EDA's SafetyScope and Questa® Equivalent RTL for this exercise to enable seamless integration with safety analysis, static analysis, and formal verification.

The main procedure is as follows:

1) Configure SPEC as a top module.
2) Configure IMPL as another top module.
3) Select a fault injection point from the fault list.
4) For the IMPL design, set the selected faulty point as a cut-point and drive to 0 or 1. It mimics a stuck-at-0/1 permanent fault.
5) Map the detection points and run the formal verification engine to compare and find the result.
6) Repeat this process until all faults in the list have been evaluated.

The formal verification result shows in three ways:

• Fired: In the original use case of Questa® Equivalent RTL, this means the two designs have differences. In this exercise, it indicates that the faulty point causes a deviation at the detection point, making the fault detectable. The counter-example waveform shows evidence that the fault injection point is causing the deviation.

• Proven: The faulty point does not cause any deviation at the detection point. This result indicates that the faulty point is not detectable.

• Inconclusive: The formal verification tool cannot resolve to proof or firing within the specified time.

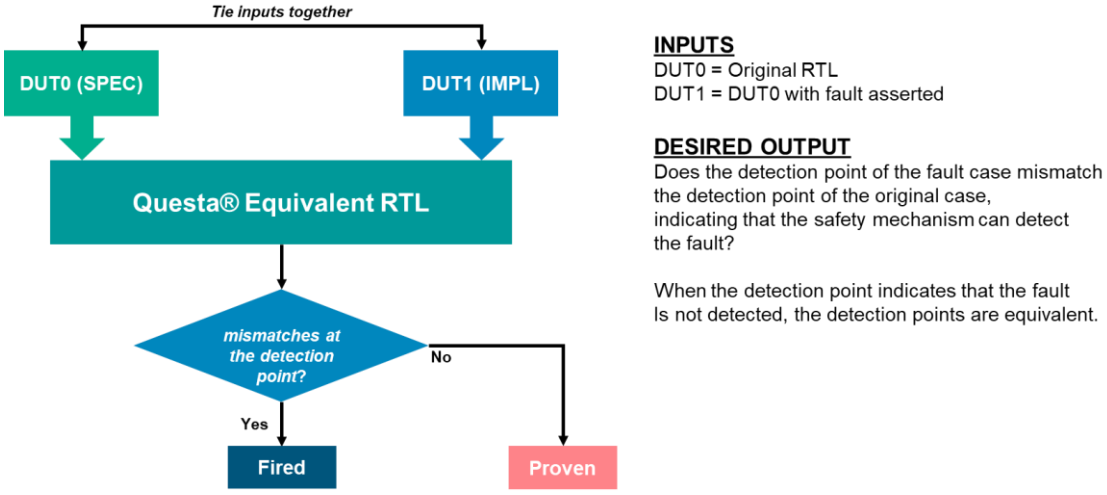


Figure 5. Configuration and the operation flow of formal verification, with Questa® Equivalent RTL

This approach is applied to the AXI interconnect bus matrix in Figure 2, which is the reduced design scope from the structural analysis.

If the locations of multiple faulty points differ, the detection points (mapping points of Questa® Equivalent RTL) also vary accordingly. In this experiment, faults are propagated to the endpoints through the AXI channel, causing differences at the outputs of the AXI ports, which are considered detection points. If Questa® Equivalent RTL catches the deviation, the result is reported as 'Fired'.

TABLE 1
Target Area and corresponding Detection Points.

| Target Area | Channel | Detection Point (IO) | Condition |
|---|---|---|---|
| Slave 0 | Write Data | Master 0 | Master0.WREADY&&Master0.WVALID |
| | Read Data | Slave 0 | Slave0.RREADY&&Slave0.RVALID |
| | Write Address | Master 0 | Master0.AWREADY&&Master0.AWVALID |
| | Read Address | Master 0 | Master0.ARREADY&&Master0.ARVALID |
| Master 0 | Write Data | Master 0 | Master0.WREADY&&Master0.WVALID |
| | Read Data | Slave 0 | Slave0.RREADY&&Slave0.RVALID |
| | Write Address | Master 0 | Master0.AWREADY&&Master0.AWVALID |
| | Read Address | Master 0 | Master0.ARREADY&&Master0.ARVALID |

As shown in TABLE 1, If the faulty point is WDATA in the Slave 0 interface area, the detection point should be set to the Master 0, as specified by the address map. The AXI bus communicates using VALID/READY signals in a handshake method, so the VALID/READY signal should be given as a detect condition at the detection point. Besides the data and address signals listed in the table, many types of AXI bus sideband signals can be analyzed in the same manner as the data signals.

In practical applications, multiple bus interconnectors are used. In typical SoC designs, there are numerous bus interconnections, and the master side of an IP can access many slave IPs. It allows the CPU's software safety mechanism to prove diagnostic coverage, and the formal verification tool can detect the fault propagation path to the endpoint connected to some slave IPs. However, appropriate constraints need to be applied for the formal verification tool to report accurate results and improve the run time.

Below are how we used constraints for this experiment:

• Formal VIP for AMBA AXI: We attached Questa® Formal Library (QFL) to constrain control points to be AXI protocol compliant. The QFL includes formal assumptions. For example, it ensures that the DATA/ADDRESS does not change during the VALID asserted time and that the READY signal is de-asserted.

• Black Box: In most designs, a module has many submodules not within the Cone of Influence (COI). We can remove these modules to reduce run time; however, only validated modules can be black-boxed since some can cause false results.

• Apply constants for the unused Slave/Master Ports: Unused ports should be tied to 0 or 1. Untied ports can cause the formal verification tool to generate unnecessary packets, leading to incorrect results.

• Clock/reset for Clock Domain Crossing Point in Asynchronous Bridge: Sometimes, the clock domain crossing point makes the formal verification tool difficult to pass packets through them. The exact clock and reset settings are needed for this point.

There are expected limitations to this formal verification methodology. First, considering a fault in the ADDRESS path, the fault propagating through the ADDRESS path cannot reach the appropriate port; hence, the result for the ADDRESS cannot be 'fired'; it can be 'proven' or 'inconclusive'. Second, if a fault is injected into handshake signals like VALID/READY, the entire transaction delivery can differ from the original; therefore, the faulty path may be undetectable, even if the formal verification result is reported as 'fired'.

Despite these limitations, there is potential to further extend the application of this method beyond the practical experiment being discussed. Although this experiment only addressed permanent faults, it can be easily extended to analyze transient fault injection. Transient faults can be mimicked by applying appropriate sequential assumptions as constraints, instead of driving the faulty point to 0 or 1.

## IV. TEST DATA ANALYSIS

The above approaches were applied to improve coverage gaps in dynamic fault simulation at Samsung, with the goal of achieving Automotive Safety Integrity Level (ASIL) requirements for an automotive project.

### *A. Diagnostic Coverage Achieved Using Traditional Fault Simulation Techniques*

At Samsung Design, traditional fault simulation techniques were employed to address a total of 6000 statistically random-sampled faults. The paper "Are My Fault Campaigns Providing Accurate Results for ISO 26262 Certification?" presented at DVCon US 2024[4], described how we planned and verified fault campaign activity on the design. The results of this fault campaign, detailed in Table 2, are from the latest fault campaign with fully configured safety mechanisms, which is up-to-date from the results published in [4].

TABLE 2
Result of fault campaign with fully configured safety mechanisms

| | Categories | % Of faults |
|---|---|---|
| 1 | Alarm Detected | 66.50 % |
| 2 | Residual | 11.33 % |
| 3 | No Deviation | 13.57 % |
| 4 | Not Injected | 8.60 % |

The practical experiment in this paper addresses the process of meeting the DC target for ASIL-B through the analysis and additional experiments on Not Detected (Residual, No Deviation, Not Injected) faults. Traditionally, faults in the No Deviation and Not Injected categories are considered safe. However, taking a conservative approach, we conducted engineering analysis and judgment on 33.50% of the Not Detected categorized faults.

Through the judgment work of designers and verification engineers, we obtained safe judgments for 6.76% in the Residual category and 15.95% in the No Deviation category. Most cases were judged safe by designer confirmation, as the faults propagated through unused paths due to a lack of verification scenarios or unintended scenarios, which do not occur in actual use cases.

As shown in TABLE 3, despite significant efforts, after judging 22.71% as safe, approximately 11% of the total faults remained conservatively unsafe. To achieve the ASIL-B target, considering a Margin of Error (MOE) of 1.06%, an additional effort of around 3% was required.

TABLE 3
Result of engineering judgement

| | Detected | Residual | No Deviation |
|---|---|---|---|
| Fault campaign result | 66.5% | 11.33% | 22.17% |
| Judgement result | | Safe: 6.77% | Safe: 15.95% |
| | | Not Safe: 4.57% | Not Safe: 6.22% |

### *B. Final Coverage Goal Achievements*

Through a gap analysis of Not Safe faults, we identified a gray area that hardware SM could not cover, accounting for 4.88% of the total. To improve this 4.88% and achieve the remaining 3% coverage required for ASIL-B, we collaborated with the software engineering team to apply software SM. The software SM was developed and implemented by the software engineers following the guidance of hardware designers. The Read after Write (RAW) safety mechanism was specifically considered with STL in this experiment. During discussions with the software team, we excluded some fault paths from the target application due to their location in the flash memory area, where the RAW SM could not be applied. Consequently, we proposed coverage for 4.47% through software SM from the total uncovered faults of 4.88%.

To prove the coverage of these faults, we opted for static analysis and formal verification instead of traditional fault simulation, which has difficulties preparing the necessary input stimuli. First, we verified through structural analysis that the RAW SM had no structural restrictions in covering specific target faults. Then, we conducted a formal verification to confirm that the actual valid RAW operations could detect these faults. Following the previously described flow, we performed structural analysis and formal verification using SafetyScope and Questa® Equivalent RTL, and confirmed that 4.41% of the faults were covered by the software SM.

As a result, as shown in TABLE 4, we could claim 93.62% coverage for the NOC in Safety Island for Samsung design.

TABLE 4
Result of static and formal analysis

| BUS Logic | Result 1(from [4]): Fault simulation with partially configured hardware SMs Detected (%) | | Result 2: Fault simulation with fully configured hardware SMs Detected (%) | | Result 3: Fault simulation + Judgement results Detected (%) | | Result 4: Fault simulation + Judgement result + static and formal analysis for software SMs Detected (%) | |
|---|---|---|---|---|---|---|---|---|
| NOC in Safety Island | Total 65.87% | | Total 66.50% | | Total 89.21% | | Total 93.62% | |
| | Fault Simulation | 65.87% | Fault Simulation | 66.50% | Fault Simulation | 66.50% | Fault Simulation | 66.50% |
| | | | | | Judgment | 22.71% | Judgement | 22.71% |
| | | | | | | | Software SM | 4.41% |

## V. LIMITATION AND FUTURE WORKS FOR FURTHER EXTENSION

The methodology outlined in this paper facilitates the early analysis of coverage for software safety mechanisms. It focuses explicitly on data path protection. Although this approach can be extended to other use cases, such as control path protection or software safety mechanisms with appropriate observation points, we will restrict the formal verification to a specific sequential depth rather than encompassing design hierarchies.

Future works on this approach include:

- Creating design cones that must be addressed using the SafetyScope RFM method to limit the design size that needs formal attention.
- Create seamless communication between the design partitioner (SafetyScope) and the formal verification engine so that the setup needed for formal verification is auto-generated.
- Exploring the possibility of dynamic formal analysis based on input stimulus to reduce the target design space.

The proposed methodology is suited for any SM that needs a proper vector to prove it. For ASIL level D, if duplication is used to achieve the ASIL level, other approaches will enable faster proof. However, if software SM is used to achieve some of the coverage goals, this approach will help in faster closer for software write-up with early metrics and identifying the areas that are not covered using software SM models. For the other ASIL levels, this approach can be used to prove the effectiveness of both hardware and software safety mechanisms.

## VI. CONCLUSION

Structural analysis and formal verification allow for enhancing fault detection resolution without the need for a traditional fault simulation process. In this paper, we presented a methodology involving multi-step processes, including gap analysis, static structural analysis, and formal verification, to identify and prove the fault resolutions early in the design process. A case study on a Samsung automotive design highlighted the effectiveness of this approach, demonstrating that we achieved comprehensive diagnostic coverage for the targeted ASIL goals by combining static and dynamic analysis methods. Through our partnership with Siemens EDA, we aim to automate and integrate this methodology into their Functional Safety solutions, thereby significantly enhancing the overall safety and reliability of automotive systems. All static and dynamic analyses, including engineering judgment, should be managed within a single database and fully automated to report the final metrics. By continuously improving our approach, we strive to stay ahead of emerging challenges and ensure the highest standards of safety and reliability in SoC designs.

## REFERENCES


[1] ISO 26262 Part 5: Product development at the hardware level, Second edition 2018-12.
[2] ISO 26262 Part 11: Guidelines on application of ISO26262 to semiconductors, Second edition 2018-12.
[3] Validating the complex safety mechanisms, Siemens Verification Academy.
[4] Are My Fault Campaigns Providing Accurate Results for ISO 26262 Certification?, DVCon US 2024.
[5] Austemper SafetyScope User Guide – Safety Analysis
[6] Questa SLEC (Equivalent RTL) User Guide – Safety Mechanisms