A Comprehensive Safety Verification Solution for SEooC Automotive SoC

I. Introduction

The Automotive vehicles have evolved from mechanical machine to loaded electronic machine in last few decades. The demand for a safe automotive call for a need of multiple complex electronics to the mechanical machine. The complex electronics are making the automotive susceptible to malfunction during the operation. To ensure the safety of the human lives, ISO26262 defines Functional Safety in automotive as "Absence of unreasonable risk due to hazards caused by malfunctioning behavior of E/E systems". In brief, we can say Functional Safety ensures the safe operation of the vehicle even when there is a malfunction in the electronics due to unpredictable environmental condition or wear and tear. Considering the reusability aspects, the ISO26262 Functional Safety compliant automotive SoCs are developed with SEooC (Safety Element out of Context, ISO26262-10:2018 clause 9.1) [1]. Typically, these SoCs have a multi-master architecture where HOST CPU, Safety Master, Security Master, DMA etc. are integrated together to accomplish the safety and security aspects of the SoC. The ISO26262 guides to categorize the IPs in the SoC to different ASIL (Automotive Safety Integrity Levels), considering the complexity of the operating scenarios. Therefore, its quiet evident to have Safety and Non-safety blocks in a single SoC considering the co-existence of ASIL (B/C/D)/QM (Quality Managed) [3]. ISO26262 mandates to ensure FFI (Freedom from Interference) to ensure safety in this co-existence scenario where a lower ASIL compliant block interferes the operation of the higher ASIL. We need a Safety Controller, which can detect and report these interferences. In such scenarios, the SoCs are getting equipped with numerous Safety Mechanisms to ensure Functional safety. These safety mechanisms play a crucial role to detect and report the faults in the SoC. However, the challenge lies in handling all these faults at the SoC level and taking the appropriate action to ensure the safe operation. Fault Manager handles this task.

It is quite essential to have dedicated verification component to verify Safety Controller and the Fault Manager to ensure safety in the design. The proposed solution here is Safety Controller Verification Component (SCVC) and the Fault Manager Verification Component (FMVC).

II. Related Work

Modern automotive vehicles are reliant on the ISO26262 Functional Safety. The numerous complex functions (Failure Modes) are bringing in heightened risk of faults that can compromise the safety, reliability and performance of the vehicle. Even the dependent failures (i.e. the fault occurring in one component affecting the other component) and interference from QM/Lower ASIL element to higher ASIL elements contribute to the dependent failures. To avoid these type of failures the SCVC and the FMVC provide a holistic approach to verify the FFI and fault verification at the SoC level.

A. Safety Controller

Modern SoCs have a multi-master architecture where HOST CPU, Safety Master, Security Master, DMA etc. are integrated together to accomplish the safety and security aspects of the SoC. The ISO26262 guides to categorize the IPs in the SoC to different ASIL (Automotive Safety Integrity Levels) considering the complexity of the operating scenarios. Therefore, its quiet evident to have Safety and Non-safety blocks in a single SoC considering the co-existence of ASIL (B/C/D)/QM (Quality Managed). ISO26262 mandates to ensure FFI (Freedom from Interference) to ensure safety in this co-existence scenario where a lower ASIL compliant block interferes the operation of the higher ASIL. We need a Safety Controller, which can detect and report these interferences.

All the transactions across the SoC from different masters carry the safety attribute (configurable due to SEooC) of the respective master. SC will monitor these transactions and depending upon SC's configuration, these transactions will be allowed or blocked considering the transactions validity. In case of blocked transaction SC will generate error response, log the error and will report a fault to Fault Manager (FM) [2]. Being SEooC SoC, the safety property of the master can be configured as safe/non-safe, the configuration of safety property for the master will be done by the SAC (Safety Attribute Controller).

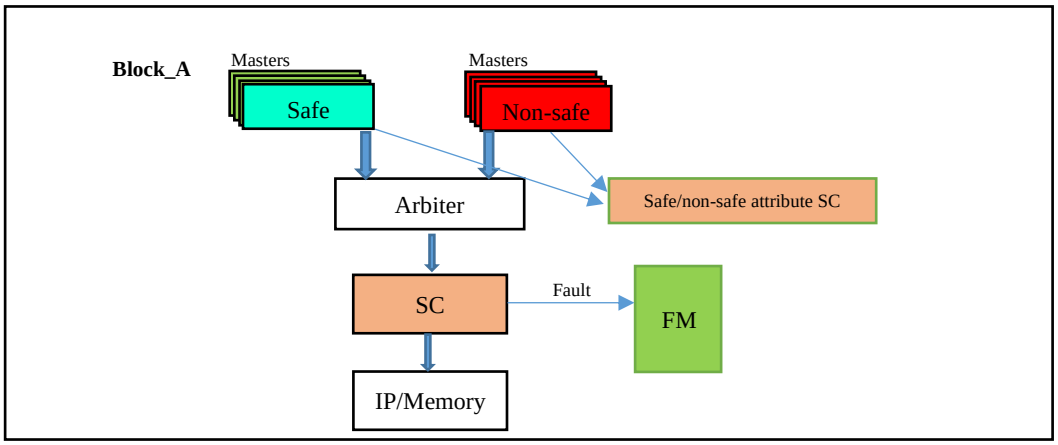

Figure 1. Diagram with SC and FM communication in single block

Safety Controller across the blocks in the SoC will provide protection to predefined address region from non-safety master access. It assigns memory access permission for each address region and checks that incoming traffic is allowed or not. Safety Controller operations are applied to only Non-Safe Masters. i.e. Safe Masters can bypass Safety Controller and access the regions though they may be blocked by Safety Controller.

In the SoC there will be several slave regions, to support those slave regions, Safety Controller has 32 windows modules, where each window module will be programmed to monitor certain slave region as per user request, and the size of the window are configurable, with this windows implemented inside the Safety Controller user can program to monitor different address regions.

Once the Safety Controller is enabled, it will watch the traffic and determine whether the traffic can proceed or not by comparing permission and the address of the requested transaction. The 32 windows modules are responsible to check address match and permission. If a transaction carries the required permission, no window will block the transaction. Otherwise, every window will compare the address of the transaction. If one of the windows flags the traffic not to proceed, Safety Controller will block the traffic and return error response.

Once Safety Controller blocks the traffic, Safety Controller will generate a fault, and that fault are connected to Fault Manager. Safety Controller will also log the traffic details in the error logging unit, the traffic details captured in error logging unit is read by the core to see which traffic has tried accessing the safe region.

B. Fault Manager

In the context of an automotive System on Chip (SoC), it anticipated that each chip might harbor more than 5000 potential faults. The total number of faults increases with increase in the number of dies. An automotive SoC typically consists of a number of blocks and each block is equipped with numerous safety mechanisms.

These safety mechanisms include Clock Monitor Unit(CMU) to monitor the frequency of the clock, Voltage Monitor Unit (VMU) to monitor the voltage, a Temperature Monitor Unit(TMU) to monitor the temperature of the chip, ECC, etc., As we know, ECC is capable of detecting 3 faults: SERR (Single bit error), DERR (Double bit error) and DEC_ERR (Decode error). Given that an automotive SoC typically consist of 1000+ memories i.e. 1000+ ECC instances, the number of faults contributed by ECC alone crosses 3000. Considering the faults more different safety mechanisms, the fault number shoots past 5000.

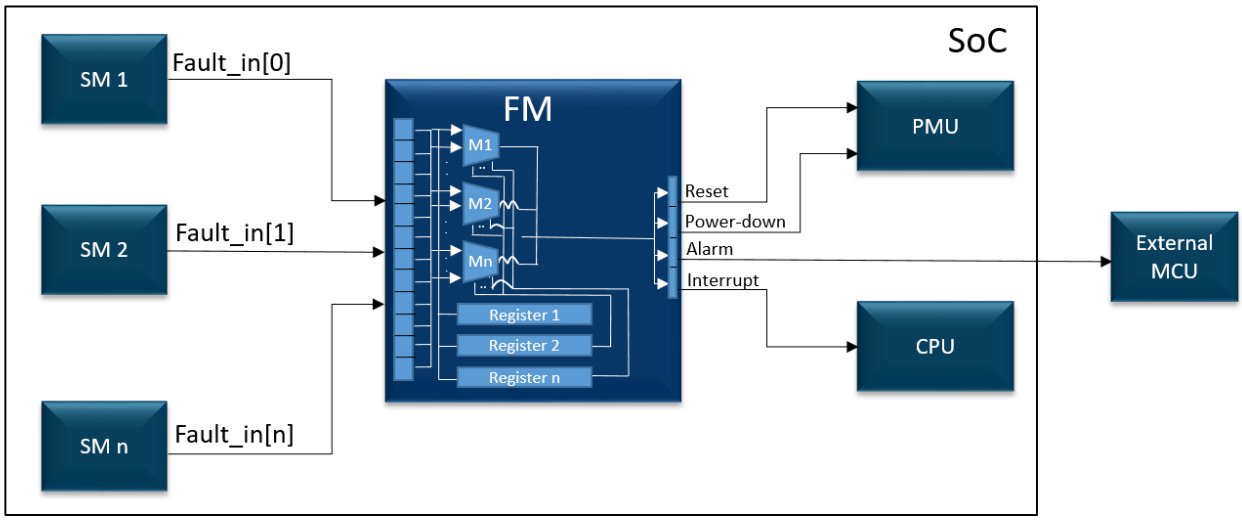

Figure 2. Fault Manager Block Diagram

Considering the complexity of the design and the multitude of faults, it is crucial to note that not all the faults can be handled uniformly. For instance, fault from Temperature Monitor Unit necessitates a power-down request due to its critical nature, while a voltage monitor fault may be addressed through a reset request to power management unit. Similarly, a clock monitor fault can trigger an alarm request to the external micro-controller unit and an ECC SERR fault be communicated to the CPU via an interrupt request to Generic Interrupt Controller (GIC).

Thus, the handling of fault solely depends on the priority and the use case. To effectively address these diverse fault types, an IP is required, which is capable of detecting and reporting the faults to the respective master. That is the function of Fault Manager (FM). Based on the priority and the use case of the fault, the Fault Manager (FM) can be configured to route the fault to the following destinations:

i) Interrupts to the respective CPU (IRQ)
ii) Power mode switching request to Power Management Unit (PMU)
iii) Reset request to Reset Management Unit (RMU)
iv) Alarm indication outside the SoC

C. Safety Controller Verification Challenges

The verification of Safety Controller at every block becomes a mammoth task, for each Design Verification engineer to understand and configure the entire relevant configuration. A single miss can become hazardous for the SoC.
To perform configuration and driving of transactions to Safety Controller, user has to program certain set of registers which includes

1. Programming of the register to enable Safety Controller
2. Programming of the window registers which can be enabled or disabled for write or read access to monitor the incoming traffic.
3. Programming the size of the window
4. Programming the slots of the window, (each window is further divided into 32 slots). Each Slot represents a portion of the address region of the Window.
5. Programming of error logging unit
6. Programming of FM for fault

All the above-mentioned programming of the registers has to be executed by the individual block design verification engineer to verify one particular instance of safety controller in a block.

This is for one Safety Controller instance, in a SoC there will be more than 70 instances. So verifying all the Safety Controller instances with master and slave being safe and non-safe will be a tedious task to DV engineer. Where each

DV engineer has to understand the Safety Controller IP, Fault Manager IP, and DV engineer needs to program several registers, and if any scenario is missed to verify that could lead to silicon bug, and 100% functional coverage of all Safety Controller instances also can't be guaranteed.

III. Safety Controller and Fault Manager Verification COMPONENT (VC)

To address the above mentioned challenges, the Safety Controller Verification Component (SCVC) is developed to perform configuration and driving of transactions to Safety Controller, SCVC expects a simple and minimal configuration from the block owners and verifies Safety Controller and do robust End-to-end verification using the FMVC that takes care of the generated faults.

Safety Controller Verification Component takes Safety Controller instance name, targeted master name, address range to be monitored, and fault number as a inputs from user and checks the functionality of Safety Controller with master and slave being safe and non-safe combinations, here programming of the master and slave as safe or non-safe will be done by SCVC itself. Safety Controller Verification Component drive the traffic to the targeted Safety Controller with 4 combinations of master and slave safety property, which includes.

Table 1
SCVC Traffic

Master	Slave
Safe	Safe
Non-Safe	Non-Safe
Safe	Non-Safe
Non-Safe	Safe

When Non-safe master accessing safe slave region, Safety Controller will generate error response and fault, in remaining three combinations Safety Controller has to generate non-error response and no fault. Safety Controller Verification Component will check for the response in each combination and report error if there is any discrepancy in any combination, Safety Controller Verification Component will also program and check for error logging unit. Safety Controller Verification Component also program Fault Manager IP for fault reachability check, Safety Controller Verification Component will also clear the fault through register write and check the fault clear status.

Safety Controller Verification Component will also make sure all the combinations of master and slave pair are verified, and ensure 100% functional coverage for any given Safety Controller instance. Safety Controller Verification Component also provides different verification modes to the DV engineer. where each mode checks certain combination of master and slave pair, for example if a given master is non-safe by default and the DV engineer wants to check only non-safe combination checks of master, MAS_NON_SAFE_MODE can be used. In addition, ALL_CHECK mode involves all possible checks involving the Safety Controller region access and altering the SAFETY property of Master.

Safety Controller Verification Component also provides automated test case generation script to DV engineer, where DV engineer will generate their block Safety Controller instance test case by providing minimal inputs to script, this will ease the verification challenges of DV engineer. The integrated checkers and the functional coverage of the instances reduces the risk of reviewing the test cases again and again. The gaps can be easily caught using top level review.

In nutshell the Safety Controller Verification Component can verify all the below aspects of the Safety Controller and ensures the safety of the SoC.

a. Configurable safety attribute of the master as safe or non-safe
b. Allow or block transaction based on safety attribute
c. Check Safety Controller fault output to Fault Manager (FM) and GIC
d. Configuration of Safety Controller with minimalistic user inputs
e. Error logging of the blocked transaction
f. Functional Coverage for all the Safety Controllers in SoC
g. Automated test case generation

h. Different verification mode supported for user configuration
i. Scalable to other verification solutions to catch FFI issues in bus component

D. FMVC

Since there are more than 5000 faults in a typical automotive SoC, verifying all the faults with every RTL development upgradation, is a humungous task that is levied upon DV engineers. With this, multiple risk factor is associated in verifying the faults.

- Firstly, it is a very time consuming approach, as the FM registers have to be configured before the generation of the fault in respective block. Therefore, DV engineers should know the functionality and the working of FM.
- Secondly, if there are any changes in the faults that are mapped in RTL, the engineer would have to re-visit their test to make the necessary FM register configuration. Post FM register re-configuration, rerun of the test is required to close on the verification of the fault scenario.
- Mainly, even if a single fault is not verified, that might lead to a silicon bug causing potential safety hazard.

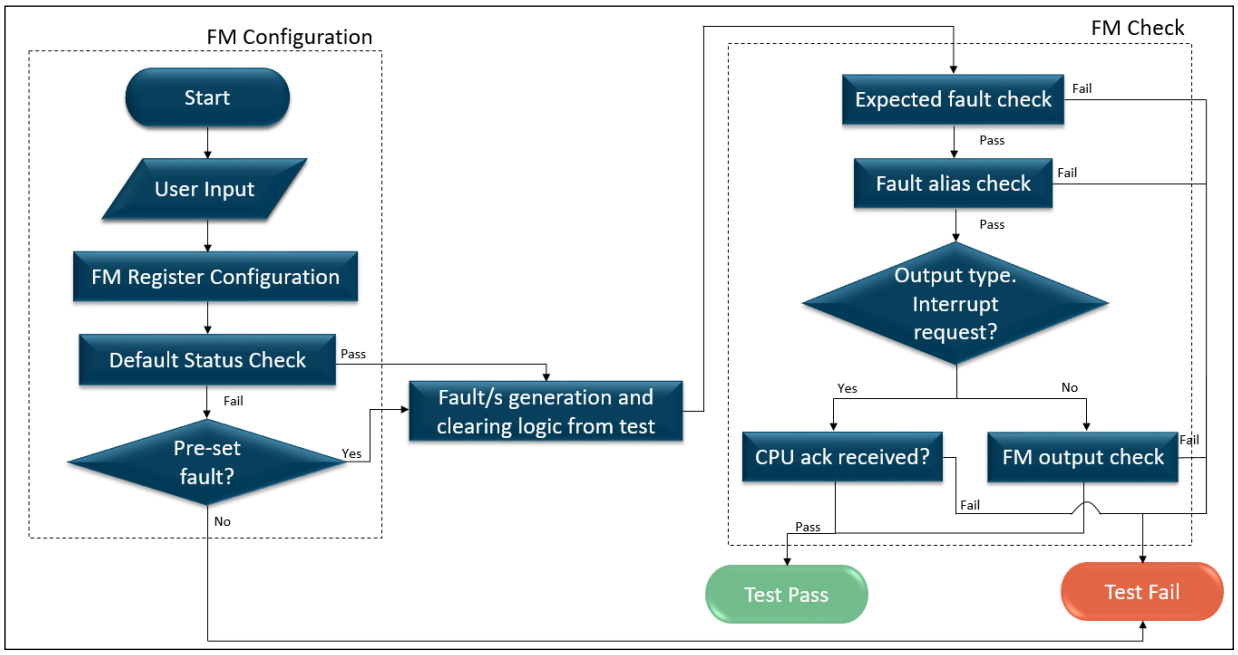

Figure 3. Fault Manager Verification sequence

Hence to ease the verification of the faults by DV engineers across various teams, FMVC was developed, in which hooks are provided to the user that needs to be called in their respective tests to make the respective FM register configuration and also to check if the intended fault is asserted in that particular scenario. The FMVC also does a fault aliasing check where the sequence checks if there are any unintended faults that are asserted in the design.

FMVC requires minimal inputs from the users to make the required configuration or to do the respective fault check. Inputs that are required from the users are

a. Intended fault list - Intended fault list is the list of fault that the user expects in their scenario. FMVC checks if these faults are generated in the design.
b. Pre-set fault list - Pre-set fault list is the list of fault that are already asserted in the design (maybe because of the developing RTL, scenario where that respective fault is always asserted, etc.). FMVC ignores the check of the provided pre-set faults during the exhaustive fault aliasing check.
c. Output type - Output type is the intended output on where the fault will be routed. As mentioned, every fault can be routed to various masters like the PMU, Alarm unit or the GIC.

The features of the FMVC can be split into 2 tasks i.e. FM_CONFIGURATION and FM_CHECK.

- FM_CONFIGURATION - With the input that the user provides, the first task that FMVC does is the FM register configuration, where the input and the output registers are configured. Post this; FMVC does a default check on the status registers to make sure that there are no faults already set in the design. Considering that there are no faults set in the design, the user would proceed to the fault generation and clearing logic from their test. If the default check fails, the pre-set fault list is compared with the faults asserted in the design. If that is a match, user would proceed to the fault generation and clearing logic from their test.

- FM_CHECK - Post the clearing of the fault from respective test, FMVC checks if the intended fault is asserted in the design. For a given fault, there may be multiple sources. With the help with assertion, the sequence makes sure that the respective fault only is asserted and no other faults are asserted in the design. Post that, FMVC checks the output type that the user has provided. If the input provided is IRQ, FMVC waits for the CPU to acknowledge the fault. If the input is of any other type, FMVC checks if the respective output is asserted at the FM output boundary.

If any of the check fails, the test will be marked as fail. In short, there is only one way in which the test would pass and n ways in which the test might fail.

To sum it up, the FMVC supports below features to ensure the incoming faults are handled correctly.

a. FM auto-configuration with minimalistic user inputs
b. Checkers for intended fault type, polarity, effect, etc. including spurious fault(s) detection
c. Fault reporting and SoC reaction check
d. Fault aliasing and connection check
e. Functional coverage for all the faults
f. Sanity checks for FM connections
g. Scalability to multichip solutions

IV. Preliminary Results

The SCVC in conjunction with the FMVC helped in finding critical bugs in the SoC. The plug and play nature of the verification components enabled the engineers to deep dive the safety aspects of the SoC and able to find bugs that attributed to the architectural changes. SCVC helped to find major bug where interference issues would have created major safety violation in the SoC. FMVC helped to verify the fault polarity, fault aliasing and connectivity issues at the early stage of the project.

Below table summarizes the major outcomes of using both these components.

Table 2
SCVC and FMVC Results

SCVC Results	FMVC Results
Overriding of safety attribute by SoC components, **~30 RTL bugs**	Timing issue causing FAU failing to capture the faults
Missing of fault connectivity from SC to Fault Manager, **~10 RTL bugs**	Incorrect Fault Polarity and Fault type (Level vs Pulse)
Interference issues found during page table walk resulted in IP bug fix **(>30 blocks)**	Fault aliasing issues and missing connection issues, **~100** connection mismatches
100% functional coverage for all SC combinations	Incorrect Fault merging issues (ORing Level and Pulse faults together)
Cross functional scenarios covered at SoC level	Spurious fault triggering issues, **~30 bugs**

V. Conclusion

The UVM based SC and FM Verification component helped in reducing the verification time of the Automotive SoC. It helped the block DV teams to plug and play the verification component and the VC did thorough verification of the safety aspects. The ease of use aspect enabled the SoC engineers explore corner case scenarios as the SC and FM verification challenges were overcome by the verification component. With these the safety is ensured in the SoC within the stringent SoC timelines. Below are the outcomes of the verification components.

a. **95%** DV efforts reduced due to the plug and play features
b. **200+** tests added to verify Safety Controller and the time reduced from 2 man months to 1day
c. Direct Architecture impact as corner case scenarios were done at ease at SoC level
d. The FMVC reduced the repetitive fault verification time on every level from **3days to ~10mins**

VI. References

[1] Safety Element Out of Context - ISO26262-10:2018 standard
[2] Product Development at Hardware Level - ISO26262-5:2018 standard
[3] ASIL Decomposition ASIL Related Safety Analysis - ISO26262-9:2018 standard
[4] Functional Safety Requirements with Production/Operation/Maintenance/Decommissioning - ISO26262-7:2018 standard
[5] Road Vehicles - Functional Safety ISO26262:2018 standard