2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Fault Injection Strategy to Validate ASIL-D Requirements of BMS Components

Praneeth Uddagiri
Staff Engineer, DV Engg.
Analog Devices
praneeth.uddagiri@analog.com

Veera Satya Sai Gavirni
Senior Engineer, DV Engg.
Analog Devices
veerasatyasai.gavirni@analog.com

Prashantkumar Ravindra
Staff Engineer, DV Engg.
Analog Devices
prashantkumar.ravindra@analog.com

***Abstract*-Adoption of Battery EVs is growing at a significant pace due to the push for clean energy, favourable government policies and advances in technologies such as batteries, and automotive electronics. To ensure Battery EVs are safe and reliable, wired and wireless Battery Monitoring Systems (BMS) should identify and flag hazardous conditions within the battery pack. BMS components shall meet ASIL D safety requirements, and this demands significant safety analysis. In this paper, we will discuss the strategy adopted to validate the FMEDA safety analysis using Fault Injection. We will also present the inhouse fault campaign framework based on the EDA partner's FuSa solution, to ease its adoption. Further, we will present the flow and results to showcase how we are achieving the required target hardware metrics. We will end with detailing the flow used to reduce and achieve closure on the significant number of unclassified fault nodes.**

I. Introduction

ISO 26262 is the de facto standard for Functional Safety (FuSa) in automotive E/E Systems. It is designed to eliminate any unacceptable risk to human life. It provides a FuSa development process for the entire automotive supply chain including OEMs and their suppliers. It also provides an automotive-specific approach for determining risk classes known as Automotive Safety Integrity Levels (ASIL). Battery Management Systems (BMS) shall have components to mitigate concerns such as data authenticity, integrity, timeliness etc in the network to guarantee a tolerable level of risk. It is required to demonstrate that more than one of the components, as well as the entire system, can support safety applications rated for ASIL D.

II. Strategy, Challenges and Solution

ASIL D is the strictest level and demands SPFM of $\geq$ 99%, LFM of $\geq$ 90% and PMHF of < 10 FIT. In this paper, we will discuss the strategy adopted to achieve ASIL D requirements of the BMS component and the challenges faced in the process and the solutions for the same.

TABLE I
Hardware Safety Metric Requirements

	SPFM	LFM
ASIL B	>= 90 %	>= 60 %
ASIL C	>= 97 %	>= 80 %
ASIL D	>= 99 %	>= 90 %

A. FMEDA Validation Through FI

Failure Modes Effects and Diagnostics Analysis (FMEDA) is a systematic analysis technique to obtain failure rates, failure modes and diagnostic capabilities. A key metric that the DV team is responsible to validate is the Diagnostic Coverage (DC), which eventually ties up to the metrics such as SPFM and LFM.

ISO 26262 recommends fault injection as a method for

- Supporting the evaluation of the hardware architectural metrics
    - Evaluating the DC of a safety Mechanism (SM)

  - ○ Evaluating the diagnostic time interval and the fault reaction time interval
  - ○ Confirming the fault effect
- Supporting the pre-silicon verification of an SM concerning its requirements, including its capability to detect faults and control their effect (fault reaction)

B. Fault Injection Campaign using FCM

Fault injection typically includes injection of fault to simulate permanent and transient faults. The traditional approach would be reusing the DV environment and injecting the fault in a node by means of forcing via TCL and repeating the processing for tens of thousands of nodes. This requires the development of custom regression and data extraction scripts. This is a highly ineffective, time-consuming and human-resource intensive approach with a high chance of systematic errors.

The straightforward solution is to adopt any industry standard Fault Campaign Manager (FCM) based solution such as Cadence® FuSa solution (vManager™ Safety, Xcelium™ Safety, and Jasper™ Safety) for FuSa Verification. Figure 1 highlights the advantages of using the FCM flow over the traditional approach.

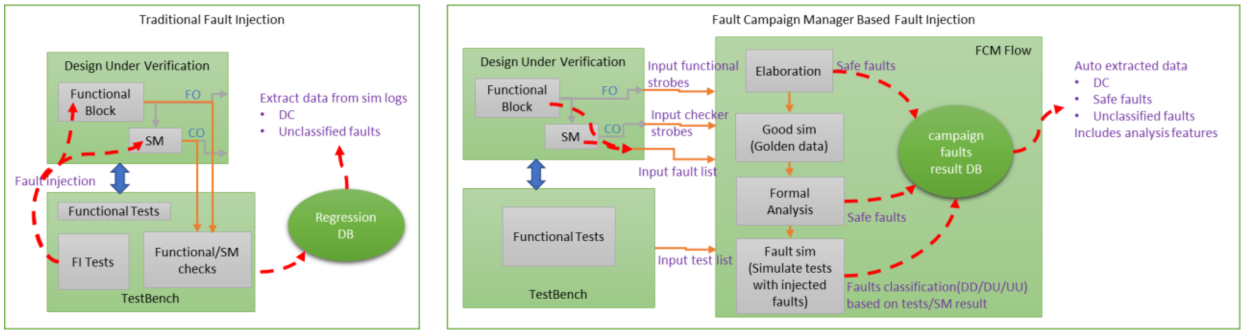


Figure 1. Comparing Traditional Fault Injection Flow and FCM based Fault Injection Flow

ADI's methodology team has developed the Fault Campaign Framework (FCF) which is a unified framework based on Cadence® FuSa solution and ADI's DV ecosystem, to enable quick turn-around, enable reuse of the FCM setup and ensure consistency across all FuSa projects. It includes all the required FCM setup files, config files and auxiliary scripts. It further includes a ready-to-run example for FCM flow exploration.

As shown in Figure 2, FCF is a result of close collaboration between the ADI methodology team, Product Lines and EDA partners and incorporates industry learnings. This has significantly reduced the fault campaign setup time, setup debug issues and ensured consistency across all FuSa projects in ADI and enabled smoother provision to deploy new methodologies.

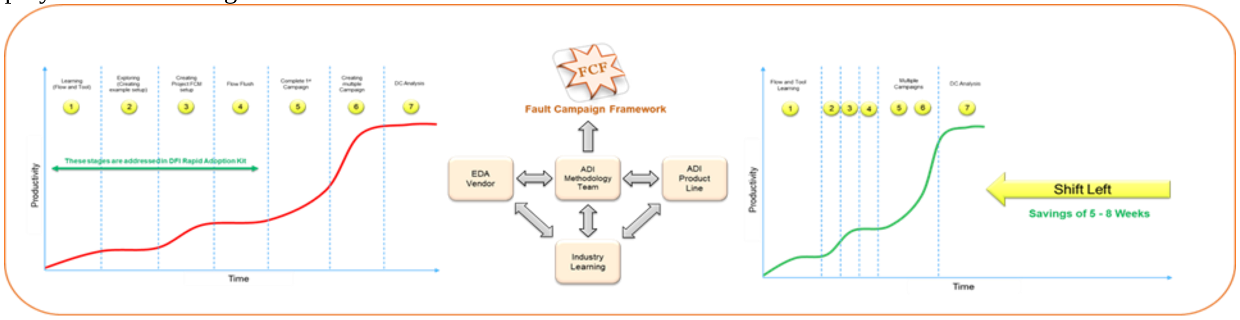


Figure 2. ADI's Fault Campaign Framework

The flow chart shown in Figure 3 summarizes the complete flow from identifying a fault campaign for a specific FM of a subpart from FMEDA (with estimated DC) to annotating back the achieved DC.

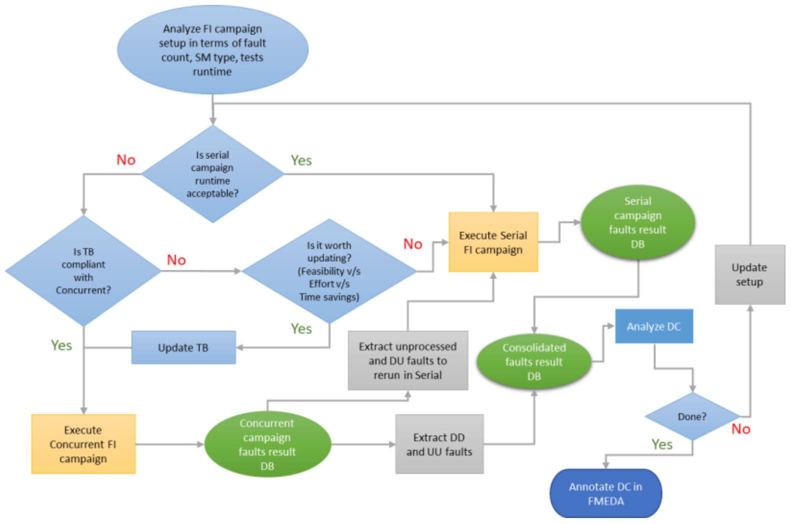

Analyze FI campaign setup in terms of fault count, SM type, tests runtime
Is serial campaign runtime acceptable?
No
Yes
Is TB compliant with Concurrent?
No
Is it worth updating? (Feasibility v/s Effort v/s Time savings)
No
Execute Serial FI campaign
Serial campaign faults result DB
Update setup
Yes
Update TB
Yes
Extract unprocessed and DU faults to rerun in Serial
Consolidated faults result DB
Analyze DC
Done?
No
Yes
Execute Concurrent FI campaign
Concurrent campaign faults result DB
Extract DD and UU faults
Annotate DC in FMEDA

Figure 3. Digital Fault Injection Flow using FCM

C. *Achieving Target Diagnostic Coverage*

The flow chart in Figure 4 summarizes the flow followed to fine-tune the fault campaign to meet the target DC, this is followed by results from a critical sub-part/block. This process had to be repeated for all fault campaigns. Meeting high DC for each of the FM with its associated SM to meet the stringent ASIL D SPFM, and LFM requirements was extremely challenging.

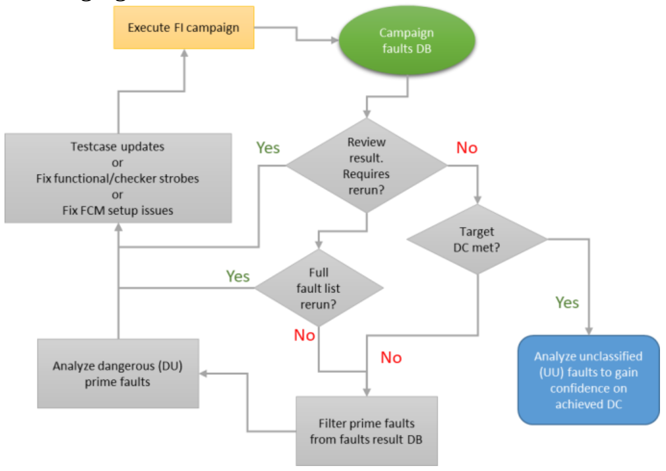

Execute FI campaign
Campaign faults DB
Testcase updates or Fix functional/checker strobes or Fix FCM setup issues
Yes
Review result. Requires rerun?
No
Target DC met?
Yes
Full fault list rerun?
Yes
No
No
Analyze dangerous (DU) prime faults
Filter prime faults from faults result DB
Analyze unclassified (UU) faults to gain confidence on achieved DC

Figure 4. Flow to improve Diagnostic Coverage

Analyzing faults in GL netlist is a challenging task due to the tool-generated random naming of nodes. In this project following approach was used for the analysis which improves the quality and efficiency.

Phase-1: Debug with good simulation waveform

1. Dump waveform of a good simulation by re-running the good simulation step of FCM flow.

2. Open the generated waveform in graphical waveform debugging tool, search the fault node of interest and send it to the schematic tracer.
3. Trace its loads to the closest flop or signal whose behavior is identifiable based on the signal name.
    a. Tracing the fault node to the nearest flop or signal whose behavior is identifiable simplifies the analysis of the functional effect of a fault on the node under consideration
    b. Now analyze the path between the fault node and the traced signal to understand what configuration(s) along with this fault can affect the nearest flop or signal that eventually affects functionality.
4. If tracing loads leads to large fan-outs, then trace its drivers to its closest flop or signal whose behavior is identifiable based on the signal name.
5. In a netlist, a traced nearest flop by the above methods may be affected by multiple functional configurations of the design. Figure-out which function(s) uses this fault path and further analyze configuration(s) that effect these functions when this fault is set.
6. Finally, check if all the derived fault conditions have SMs associated with them and if these SMs have been implemented in the test suite. If all SMs are implemented then we have a genuine safety violation (DU). Otherwise missed SM(s) have to be implemented in the test and the campaign has to be re-run with the modified test.

Phase-2: Debug with fault simulation waveform

1. If the Phase-1 turns out to be difficult then dump waveform for the node of interest by re-running the test in fault simualtion step of the FCM flow.
2. Open the generated waveform in graphical waveform debugging tool, search the functional strobe on which the fault is detected and send it to the schematic tracer.
3. Trace the signals from functional strobe and compare it with good simulation waves to understand how fault propagated to the functional strobe.
    a. This helps in understanding effect of this fault on the functional strobe.
    b. Then analyze why this fault effect is not seen on checker strobe (SM) and take corrective action.

Using the approach listed above the **DC boosted from 90% to 99%.** Figure 5 summarizes the iterations, with the details, that were necessary to improve the DC. This DC was with ~47% unclassified faults, which are reduced with the methods discussed in the next section.

iteration	Description	Total faults	Faults executed	DD faults	DU faults	Safe faults	UU faults	DC (%)
Initial campaign	Initial campaign after fixing all setup issues	51214	51214	23982	2461	681	24090	90.69
Rerun: Iteration 1	Fixed missing checker strobe and invalid functional strobe	51214	2461	24768	1675	681	24090	93.66
Rerun: Iteration 2	Rerun with added malfunctions in DFMEA after consulting FSE	51214	1675	25984	459	681	24090	98.26
Rerun: Iteration 3	Analyzed ~30 faults and created a new test configuration	51214	459	26133	310	681	24090	98.82
Rerun: Iteration 4	Analyzed ~20 faults and created a new test configuration	51214	310	26307	136	681	24090	99.48
Rerun: Iteration 5	Analyzed ~20 faults and created a new test configuration	51214	136	26400	43	681	24090	99.83

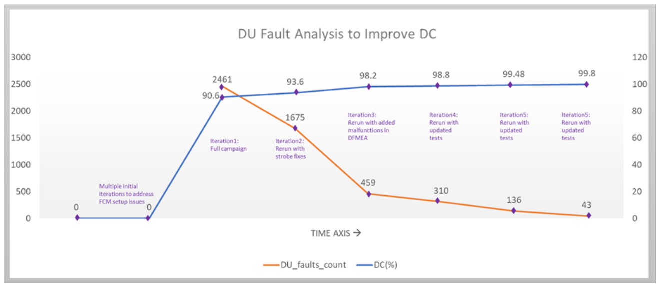


Figure 5. Results showing improvement in Diagnostic Coverage

D. Gaining Confidence in Achieved DC

Meeting the target DC for a specific FM is not enough if there are a considerable number of unclassified (UU) faults from the fault campaign. Without a thorough analysis of these UU fault nodes, it is extremely risky to sign off on DC. It is possible with appropriate stimulus these nodes could become dangerous and if the SM fails to detect them, it could lead to safety vulnerability. Hence, we have come up with flow and methods to get the UU nodes classified with fault simulation and/or formal analysis. Further, the remaining smaller subset is subjected to expert judgement taking FSE into confidence.

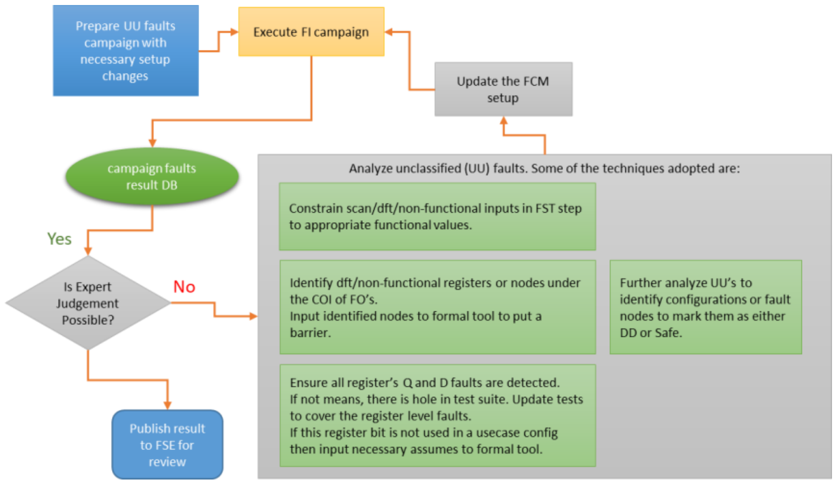

Prepare UU faults campaign with necessary setup changes
Execute FI campaign
Update the FCM setup
campaign faults result DB
Yes
Is Expert Judgement Possible?
No
Publish result to FSE for review
Analyze unclassified (UU) faults. Some of the techniques adopted are:
Constrain scan/dft/non-functional inputs in FST step to appropriate functional values.
Identify dft/non-functional registers or nodes under the COI of FO's.
Input identified nodes to formal tool to put a barrier.
Further analyze UU's to identify configurations or fault nodes to mark them as either DD or Safe.
Ensure all register's Q and D faults are detected.
If not means, there is hole in test suite. Update tests to cover the register level faults.
If this register bit is not used in a usecase config then input necessary assumes to formal tool.

Figure 6. Flow to improve Diagnostic Coverage

Reclassifying the Unclassified fault list using formal based structural analysis

Formal based Structural analysis app is used with appropriate inputs to re-classify the unclassified faults as "Safe".

1. If an input of a block or sub-block is non-functional (DFT/Scan), then
   a. Supply the "Functional Mode" value as a constraint (assume) to the formal based structural analysis tool and run the campaign till the structural analysis step in FCM.
   b. If a fault is found to be structural safe with the constraints, then FCM categorizes these as "Safe".
      i. Example: scan_enable input is always 0 in functional mode. If its value is constrained to 0, the tool will categorize all the faults in paths from this input to different flops SE port that results in value 0 at SE port as "Safe".
      ii. The tool can also categorize all faults (SA0 and SA1) that exclusively drive SI (scan input) of different flops as "Safe".
      iii. Figure 7 shows a dummy design to demonstrate the impact of constraints on the structural analysis

Constraining (assume on) SE_in to 0 will classify following faults as safe in FST structural analysis

- top.SE_in.SA0
- top.I2.SA0
- top.I2.SA1
- top.FF1.SE.SA0
- top.FF1.SI.SA0
- top.FF1.SI.SA1
- top.buf1.A.SA0
- top.buf1.Q.Sao
- top.FF2.SE.SA0
- top.FF2.SI.SA0
- top.FF2.SI.SA1

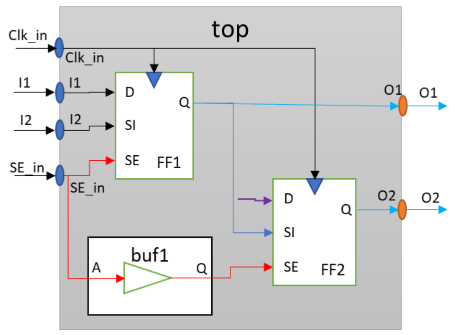


Figure 7. Impact of input constraints on the structural analysis

2. If a signal is not used in functional mode and is in the COI of any functional strobe, then
    a. Place a stopat (breakpoint) on this signal which will block the fault propagating through this signal to the functional strobe.
    b. FCM categorizes all faults (SA0 and SA1) that are exclusively driving this signal as “Safe”.
        i. Example: DFT status register is not read in functional mode, but this register and faults which only drive this register are under COI of the functional strobe. As these nodes are not active in the functional mode they are classified as “Safe” by the tool.
        ii. Figure 8 shows a dummy design to demonstrate the impact of “stopat” on the structural analysis

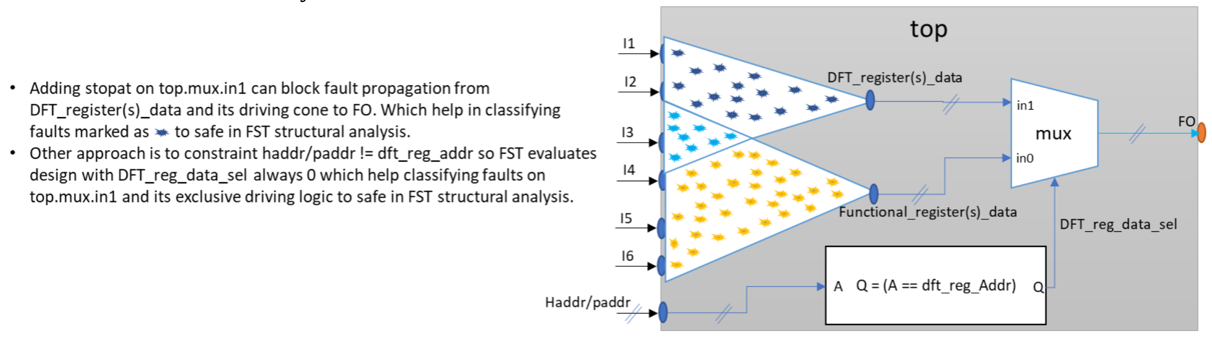

- Adding stopat on top.mux.in1 can block fault propagation from DFT_register(s)_data and its driving cone to FO. Which help in classifying faults marked as ✱ to safe in FST structural analysis.
- Other approach is to constraint haddr/paddr != dft_reg_addr so FST evaluates design with DFT_reg_data_sel always 0 which help classifying faults on top.mux.in1 and its exclusive driving logic to safe in FST structural analysis.

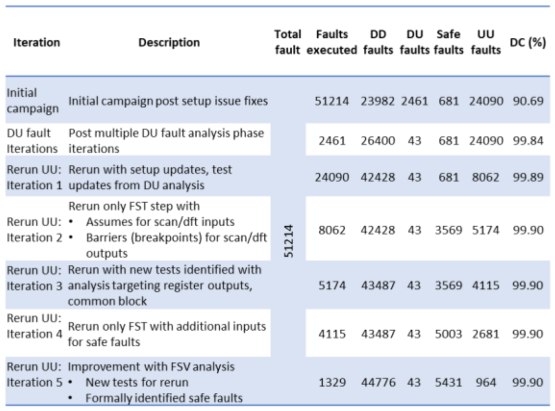
Figure 8. Impact of “stopat” input on the structural analysis

Reclassifying the Unclassified fault list using formal based safety verification

Formal Safety Verification app is used to re-classify the remaining faults as “Safe” or “Dangerous”. Tools also help to identify stimuli necessary to further classify dangerous faults as DD or DU.

1. Formal Safety Verification app auto generates the possible cover properties to cover all possible stimuli for all faults in a design w.r.t. functional strobes.
2. While proving the auto the generated properties tool can either classify fault as
    a. Safe - No propagation to the functional strobes with all possible input stimuli.
    b. Dangerous - Fault propagates to one or more functional strobes.
3. If a fault is “Dangerous” then there is a provision on the tools to visually show the input combinations (stimuli) leading the fault to be classified as “Dangerous”
    a. The campaign is rerun for the unclassified nodes with the new test (with the stimuli identified above) to classify fault as DD/DU.
    b. It is necessary to constrain the tool as appropriate to avoid the generation of cover properties for invalid input combinations.

Using the approach listed above the **DC boosted from 99.83% to 99.9% Safe faults increased from 1.3% to 10.6% while the Unclassified faults dropped from 47% to 2%.** Figure 5 summarizes the iterations, with the details, that were necessary to improve the confidence in DC by significantly reducing the unclassified faults. Overall **96% reduction in unclassified faults** is achieved.



Iteration	Description	Total fault	Faults executed	DD faults	DU faults	Safe faults	UU faults	DC (%)
Initial campaign	Initial campaign post setup issue fixes	51214	51214	23982	2461	681	24090	90.69
DU fault Iterations	Post multiple DU fault analysis phase iterations		2461	26400	43	681	24090	99.84
Rerun UU: Iteration 1	Rerun with setup updates, test updates from DU analysis		24090	42428	43	681	8062	99.89
Rerun UU: Iteration 2	Rerun only FST step with • Assumes for scan/dft inputs • Barriers (breakpoints) for scan/dft outputs		8062	42428	43	3569	5174	99.90
Rerun UU: Iteration 3	Rerun with new tests identified with analysis targeting register outputs, common block		5174	43487	43	3569	4115	99.90
Rerun UU: Iteration 4	Rerun only FST with additional inputs for safe faults		4115	43487	43	5003	2681	99.90
Rerun UU: Iteration 5	Improvement with FSV analysis • New tests for rerun • Formally identified safe faults		1329	44776	43	5431	964	99.90

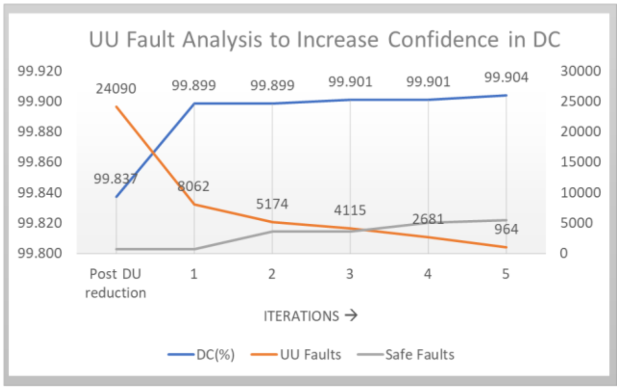


Figure 9. Results show a reduction in UU and increasing confidence in Diagnostic Coverage

IV. SUMMARY

FuSa verification and validation is a key requirement for all safety-critical automotive electronics. Early TTM being a critical factor, it is important to adopt methodologies that enable this. FuSa cannot be an afterthought and must be planned in the project planning phase, this includes identifying the FuSa requirements, strategy, methodologies, resources, skillset etc. Close engagement between Product teams, Methodology teams and EDA vendors is equally critical as tools, methods and technique are evolving at a rapid pace. Advanced methodologies such as Concurrent fault simulation, Fault simulation run time optimization, and Formal based structural analysis and fault pruning are deployed in this project to validate the FMEDA and support the ASIL D safety case. This strategy is generic enough to be adopted in any FuSa project. Overall **99.9% DC was verified with less than 2% unclassified faults** which were made possible by a 96% reduction in unclassified faults.

ACKNOWLEDGMENT

The authors would like to thank Siri Rajanedi, Ujjwal Chembuli and Vinod E. for their contribution to the work. The authors would like to further extend thanks to Alan Whooley, Ponnambalam Lakshmanan, Shelly Jose, Shreya Dasgupta for their guidance. The authors would like to express their sincere gratitude to the Cadence FuSa solution team for the training, support, and valuable input.

REFERENCES

[1] "*ISO 26262-1:2018*", www.iso.org, 2018
[2] Prashantkumar Ravindra, Mangesh Pande, and Vinay Rawat, "*Fault Campaign Framework for FuSa verification of ISO26262 compliant automotive products*" DVCon India, Dec 2021.