A Unified Functional Safety EDA Framework for Accurate Diagnostic Coverage Estimation

Bhowmik, Abhiroop; Babukutty, Subin; Taouil, Mottaqiallah; Fieback, Moritz

DOI
10.1109/VLSI-SoC62099.2024.10767815

Publication date
2024

Document Version
Final published version

Published in
2024 IFIP/IEEE 32nd International Conference on Very Large Scale Integration, VLSI-SoC 2024

A Unified Functional Safety EDA Framework for Accurate Diagnostic Coverage Estimation

Abhiroop Bhowmik*†, Subin Babukutty†, Mottaqiallah Taouil*, Moritz Fieback*
*Delft University of Technology, Delft, The Netherlands, {a.bhowmik, m.taouil, m.c.r.fieback}@tudelft.nl
†NXP Semiconductors, Eindhoven, The Netherlands, {subin.babukutty_1}@nxp.com

2024 IFIP/IEEE 32nd International Conference on Very Large Scale Integration (VLSI-SoC) | 979-8-3315-3967-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/VLSI-SOC62099.2024.10767815

***Abstract*—As electronics and software become more integrated into automobiles, Functional Safety (FuSa) per ISO 26262 becomes important. It assesses the risk level of automotive chips, reflected by the Automotive Safety Integrity Level (ASIL). Fault injection simulation verifies the FuSa of a design by injecting faults and classifying them based on whether safety mechanisms detect them. Discrepancies in classification results from FuSa EDA tools can lead to varying ASIL assignments and misrepresent associated risk. Thus, we evaluate two FuSa EDA tools, Cadence® XFS and Synopsys® VC Z01X, for RTL designs. We find that the fault space covered by the tools is not complete. Hence, we propose a novel verification methodology combining both tools to achieve maximum fault space coverage. We apply this approach to the AutoSoC benchmark suite and achieve a more accurate Diagnostic Coverage (DC) of 97.79%, over the baseline verification methodology of 98.36%, at the cost of injecting 1.31 times more faults. Our work ensures that the correct ASIL level is assigned through accurate DC estimation.**

***Keywords*—Functional Safety, ISO 26262, Fault Injection Simulation, EDA, Verification methodology**

I. Introduction

The automotive industry is experiencing a significant shift towards advanced electronic and software integration due to the increasing demand for self-driving and autonomous vehicles [1]. As automotive systems become more complex, the risk of malfunctions increases, necessitating robust safety measures. Functional Safety (FuSa), as defined by standards such as ISO 26262 [2], addresses these concerns by incorporating Safety Mechanisms (SM) to mitigate failures and hazards that could be life-threatening. ISO 26262 requires exercising faults on various design locations and analysis of safety mechanisms' detection capabilities. The fault space in modern designs is typically quite large and is associated with millions of design components, making FuSa verification a complex and time-consuming process [3]. Therefore, it becomes important to develop verification methodologies capable of covering the entire fault space while also achieving run-time efficiency.

Fault Injection (FI) simulation is the suggested ISO 26262 verification methodology for FuSa Verification and is widely seen in different solutions [4–6]. Formal methods and ATPG solutions are also seen in combination with FI simulation [5, 6] to identify safe faults that are undetectable. Testbench-based fault simulation [7, 8] involves utilizing existing functional verification testbenches with additional modifications for manual fault injection. However, this approach is not scalable to larger, complex designs and requires higher manual effort. Emulation-based techniques [9–11] can also be utilized for fault analysis using dedicated emulators. However, they also require additional hardware in the form of FPGAs to produce accelerated results in comparison to simulation platforms. Therefore, our work primarily focuses on simulation-based platforms for fault injection.

Leading vendors such as Cadence, Synopsys, and Siemens offer dedicated EDA tools for FI simulation. However, there is a lack of research on why one tool should be favored over another. It has been shown that there are inconsistencies when identifying safe faults while using different technologies like ATPG, formal and FI simulation [5, 6]. Hence, it is not clear whether different FI simulation tools will generate the same results. Further, as noted in [12], an important concern regarding the accuracy of results arises in FuSa verification at higher abstraction levels like RTL, contrasting with the predominant focus of most verification solutions on gate-level designs. Discovering bugs at the gate level would require subsequent changes to the RTL, prompting a repetitive and time-consuming process to achieve the desired coverage and stability of the design.

Considering the lack of research, we compare two FuSa EDA tools on RTL designs. We analyze the disparities in results, their limitations, and propose a verification solution to address the issues. The tools considered are: Xcelium™ Fault Simulator (XFS) by Cadence® and VC Z01X by Synopsys®. The main contributions of our work are:

- We compare the tools based on metrics such as correctness of results, fault space coverage, and simulation run-time to evaluate their performance, capabilities, and limitations.
- We propose a unified EDA framework combining the strengths of the tools to develop a verification methodology that maximizes the possible fault space while minimizing simulation time.
- We validate the proposed methodology on reference designs and an industrial-grade automotive SoC.

The rest of the paper is organized as follows: Section II discusses FI simulation along with FuSa concepts. Section III presents an analysis of the strengths and weaknesses of the FuSa EDA tools under consideration, highlighting the results produced by the tools. Section IV describes the proposed FuSa verification methodology. Section V discusses the results of the proposed methodology on reference designs. Sections VI and VII present discussion and conclusions.

TABLE I
TARGET VALUES FOR DC BASED ON ASIL LEVEL [2]

ASIL	DC
B	>90%
C	>97%
D	>99%

II. BACKGROUND

This section introduces key concepts related to FuSa verification as per ISO 26262 and outlines general verification flows used with FI simulation tools.

A. ISO 26262 FuSa concepts

One of the first steps in the FuSa lifecycle involves performing a *Hazard Analysis and Risk Assessment (HARA)* for different automotive components. HARA is used to identify and categorize hazards associated with components. Possible hazards are categorized based on three parameters: **severity** of a potential injury, **exposure** or how frequently an operational situation arises, and **controllability** of whether a situation can be managed to avoid injury. Based on these three parameters, the Automotive Safety Integrity Level (ASIL) is defined for each component. ASIL quantifies the risk associated with a component and determines the level of risk reduction necessary. ASIL is categorized into four levels: A, B, C, and D, with D representing the highest ASIL, requiring the most significant risk reduction measures.

Once ASILs are defined, safety goals are formulated followed by the implementation of SMs. An SM is an additional piece of logic designed to identify faulty behavior within the circuit and subsequently detect or correct these faults. The efficiency of these mechanisms is quantified by Diagnostic Coverage (DC). DC denotes the percentage of faults detected by SMs and is calculated based on fault simulation results. Table I presents the target values for DC for different ASILs.

B. Fault Injection (FI) Simulation

As mentioned earlier, FI simulation is the suggested ISO 26262 methodology for FuSa verification. A typical FI simulation flow consists of the following steps:

1) Fault targets (locations for fault injection) are defined along with available fault model types (Stuck-At (SA), transient faults)
2) A fault-free simulation is run to generate a reference database. Users must also define *functional* and *checker* strobes in the design. Functional strobes capture information related to functional outputs that directly impact the design output. Checker strobes are integral to SMs and can be regarded as signals for fault detection or alarms.
3) Fault simulations are run by injecting faults from the fault target list. Classification results are generated based on differences in strobe values of good and faulty simulations. Classifications are made based on whether the fault propagates to functional and/or checker strobes, as illustrated in Table II.

TABLE II
FAULT CLASSIFICATIONS

	Detected Functional	**Undetected Functional**
Detected Checker	Observed Diagnosed (OD)	Not observed Diagnosed (ND)
Undetected Checker	Observed Not Diagnosed (ON)	Not Observed Not Diagnosed (NN)

The DC can then be estimated by Equation 1. However, EDA tools offer additional classifications beyond these fundamental ones, allowing for modifications to the equation accordingly. For instance, there is also a class of **Safe** faults, which do not affect functional and checker strobes, for example, because they are unused or blocked by other signals, Not Controllable (**NC**) faults, which are signals that do not toggle during simulation) and Impossible x-state (**IX**) faults, which are transient faults injected on signals at an unknown state.

$$\text{Diagnostic Coverage (DC)} = \frac{OD{+}ND}{OD{+}ND{+}ON} \times 100\% \quad (1)$$

Our work determines differences in the classification results of FI simulation EDA tools when applied to a design. To the best of our knowledge, no previous studies have conducted a comparative analysis on this aspect. If discrepancies in classifications exist, they would result in varying DCs, consequently leading to varying ASILs. This will, in turn, misrepresent the associated risk of the component.

III. CAPABILITIES AND LIMITATIONS OF EXISTING FUSA EDA TOOLS

This section discusses the results of applying the tool flows from the two vendors on several reference designs. We describe the experimental setup and comparison metrics. We also analyze discrepancies in simulation results and their causes.

A. Tool setup and reference designs

The tool flows are tested first on two simple RTL designs: a full adder and 4-bit up counter. These examples serve as simple cases to analyze FI simulation results and enable manual inspection of classifications due to their small fault space. However, since they lack SMs, we also examine a FIFO design, provided by Synopsys® and illustrated in Figure 1. This design incorporates SMs like ECC for memories and module duplication. The FIFO is implemented using a dual port RAM, and encoded with ECC. A flags module is utilized to determine the status of the FIFO. The Read/Write Pointer modules are used to calculate the addresses for FIFO read/write. These three modules are duplicated to provide redundancy and generate errors in the event of faults.

We develop scripts to automate the tool flows, including placeholders for fault injection targets, fault models, and strobing options related to the different designs. Fault injection targets refer to design points where we want to inject faults. All locations are enabled for fault injection across the three designs. Further, the fault models employed include SA and

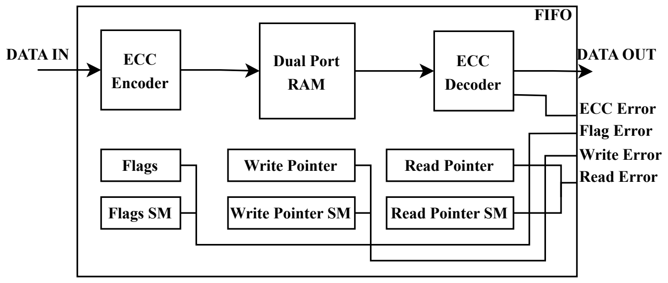

Fig. 1. FIFO with ECC and module duplication

TABLE III
CLASSIFICATION RESULTS FOR SA FAULTS ON ADDER, COUNTER AND FIFO

	Adder		Counter		FIFO	
	XFS	VC Z01X	XFS	VC Z01X	XFS	VC Z01X
ND	6	6	4	4	77	94
NN	0	0	0	0	98	1
OD	8	8	8	8	119	74
ON	2	2	8	8	84	112
Safe	-	-	-	-	44	88
NC	-	-	-	-	-	53
Total	16		20		422	

transient faults. Regarding strobing points, in the adder design, the sum and carry out signals are respectively designated as functional and checker strobes. Similarly, for the counter, the upper and lower 2 bits of the output work as functional and checker strobes respectively. The selection of these strobing points is arbitrary to verify fault classification results. However, in the case of the FIFO design, a more informed decision is made by assigning the data out signal as a functional strobe, and the error signals as checker strobes.

VC Z01X offers various options for instrumenting faults across different location types, including PORT, PRIMITIVE, FLOP, ARRAY, WIRE, and VARIABLE. XFS, on the other hand, does not support all of these distinct options for fault instrumentation. Consequently, the fault space of VC Z01X is larger than that of XFS due to the possibility of a signal instrumented as different location types. Therefore, for initial comparisons, we make the same fault lists for both tools to facilitate a fair comparison. Later, we extend the fault space to include all possible options supported by the tools.

B. Comparison metrics

We consider the following metrics to compare the tools:

1) **Correctness of results**: The tool results must be consistent and in line with expected fault classifications. There are rules defined by tools regarding fault propagation on different locations such as PORTS, FLOPS, WIRES etc. Based on how faults propagate in a design, the tool classification should match the expected result.
2) **Fault space coverage**: The tools should be able to cover the entire SA and transient fault space to provide an accurate estimation of DC.
3) **Run-time**: For large, complex designs, the fault space increases drastically, requiring more FI simulation time. We measure the fault simulation run-time for equal fault lists.

The first priority when comparing results is to thoroughly check their correctness. If the results are not correct, the tool cannot be trusted. Further, the tools should be capable of covering the entire fault space without any coverage limitations. This is important for obtaining an accurate DC metric. Finally, run-time is important when there are thousands of faults to be tested, as verification tests can take long simulation times. In the next section, we compare the tools based on these metrics and present the results.

C. Analysis of classification results

1) Correctness of results: Table III illustrates the classification results of the two tools applied on the adder and counter designs for SA faults. As seen from the table, the fault classifications obtained from the tools are the same and in line with the manual analysis done on the injected faults. However, for SA faults instrumented on the ports of the FIFO design, we observe multiple differences in fault classifications.

The first major difference lies in NN fault classification. NN faults often require manual analysis to determine whether they are actually safe or if they remain undetected due to test limitations. The difference of 97 faults in this regard is due to the classification of an additional 44 Safe and 53 NC faults by VC Z01X. While both tools identify 44 common Safe faults, VC further categorizes 97 additional faults, saving users' debugging time for NN fault classification.

The disparities in other fault classification categories (ON, OD, and ND) arise primarily due to variations in how faults in ports are modeled by the two tools. To better understand this, let us consider the example of the *FLAGS* module, which is duplicated as an SM to detect faults, as illustrated in Figure 2. Any difference in the status signals arising from the two modules is triggered as a *FlagErr*. According to fault propagation rules of the two tools, faults injected at input ports should propagate inwards towards lower hierarchies, whereas output port faults should propagate outwards towards higher hierarchies. However, when we inject a fault at an input port, for example, *FLAGS_SM.Write*, XFS propagates this fault on the wire connecting the port. Thus, it affects the value of *FLAGS.Write* as well. Technically, such behavior is not incorrect, but the tool should be able to consider the effect of both fault types. Otherwise, we would lose out on a particular section of the fault space. The disparities in fault classifications are essentially due to this difference in fault modelling.

The SA fault space is then further extended to include all types of signals - intermediate nets, registers, variables, etc. The total number of faults instrumented for XFS and VC Z01X increased to 706 and 1408 respectively. This difference is seen as a result of VC Z01X instrumenting signals as multiple location types. For example, faults are injected at *FLAGS_SM.Write* as two types: PORT and WIRE. For the latter, the classification is the same as the one obtained from XFS, and is correct for the considered location type. XFS,

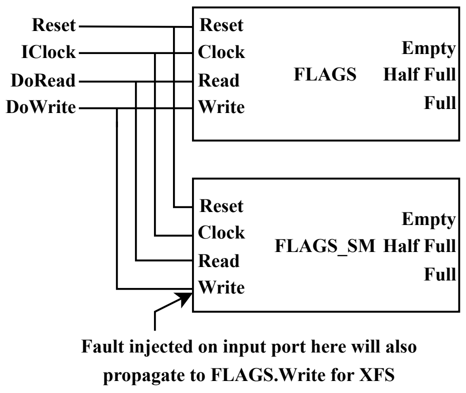

Fig. 2. Difference in fault modeling of input/output ports of tools

on the other hand, instruments a signal as a single type only. The effect of a fault and its corresponding classification can vary depending upon whether it is injected at the port itself, or the wire connecting the port. This is not taken into consideration by XFS. There are no further discrepancies in fault classifications for the extended list.

The analysis is also extended to the transient space by injecting faults on all locations within a range of timestamps. There are no additional discrepancies seen in terms of fault classifications. However, there are differences in how transient faults are modeled by the two tools. Modifications are made to fault injection commands to make similar fault lists. The strobing mechanism also makes a difference in the final result depending on whether the signals are strobed at every timestamp or at clock edges. Such differences can also result in different classifications for a given transient fault. Further, the IX classification is observed for certain signals that remain in an unknown state because the testbench fails to assign any value after a certain timestamp. Upon further analysis, it is seen that they can be reclassified as Detected. However, neither tool offers a feature to manually update fault classifications, presenting a limitation common to both.

2) Fault space coverage: The SA fault space for VC Z01X is quite extensive and takes into account the effect of different fault types. XFS does not instrument a signal as different types, resulting in the omission of certain fault effects and thereby not contributing to the final diagnostic coverage. However, VC Z01X cannot inject transient faults on inputs and intermediate wires, leaving a portion of the fault space uncovered. On the other hand, XFS does not restrict transient fault placement on inputs or nets.

3) Run-time: As shown in Figure 3, VC Z01X fares better than XFS in terms of fault simulation run-time owing to its concurrent engine support. Although XFS does provide a concurrent flow, it exhibits limitations, including limited compatibility with RTL designs, unsupported constructs, and the incapability to handle VHDL and SystemVerilog designs. Also, VC Z01X exhibits better scalability in terms of run-time, particularly for larger and more complex designs, as it can instrument thousands of faults in one shot.

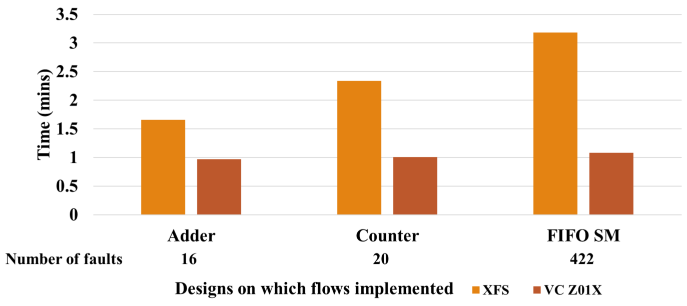

Fig. 3. Run time comparisons of tools on Adder, Counter and FIFO

D. Comparison conclusions

Both tools do not cover the entirety of the fault space required for FuSa verification. So, the first step in a verification flow should be to address all faults required for FI simulation. Second, if there is an overlap of fault space coverage between the tools, the runtime of the tools needs to be taken into consideration. Therefore, in the next section, we introduce a novel methodology aimed at providing an extensive and robust FuSa verification framework to address these issues.

IV. Unified FuSa Verification Framework

This section discusses the concept of the proposed verification methodology along with its design and implementation.

A. Concept

One of the key objectives of our work is to offer a precise evaluation of DC by covering all faults within the design space. VC Z01X is extensive for SA faults, but this is not the case for transient faults. On the other hand, XFS does not restrict fault placement for transient space, but lacks options for location types as compared to VC Z01X. Thus, the main aim of the proposed verification methodology is to combine the strengths of the tools to cover the entire fault space. If there is any fault space overlap, VC Z01X is chosen owing to its faster simulation capabilities. Further, both tools share a limitation in their inability to update fault classifications of signals when necessary. We take care of this aspect by introducing a feature in the flow to update classifications with an external file.

B. Design and Implementation

Figure 4 provides an overview of the proposed verification methodology. The different steps of the flow are outlined below:

1) We develop automated VC Z01X scripts to concurrently run stuck-at and transient fault campaigns in the first step. All possible location types and fault targets are enabled for SA faults. Further, all supported configurations for transient faults are also enabled. Start and end cycles for fault injection are specified based on design toggle activity and strobe detection possibilities.
2) Signals not subjected to transient fault injection with VC Z01X are identified from reports generated in the first stage.

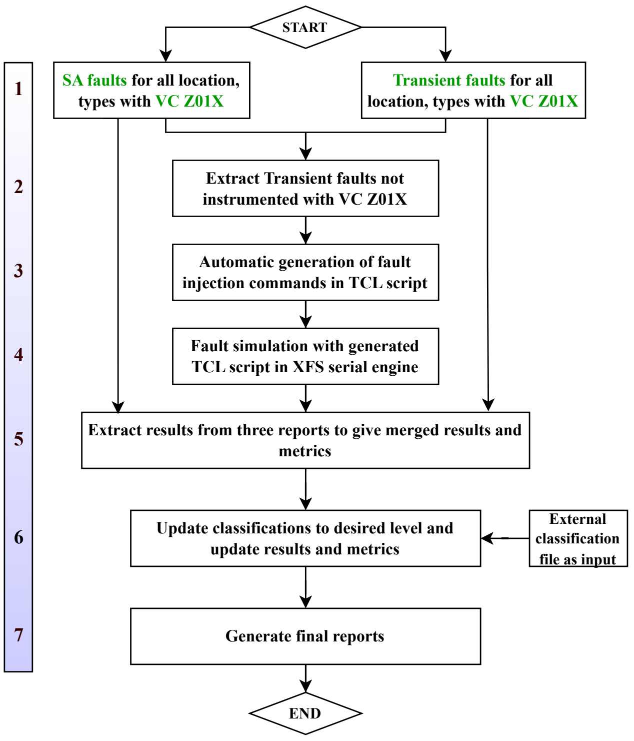
START
1
2
3
4
5
6
7
SA faults for all location, types with VC Z01X
Transient faults for all location, types with VC Z01X
Extract Transient faults not instrumented with VC Z01X
Automatic generation of fault injection commands in TCL script
Fault simulation with generated TCL script in XFS serial engine
Extract results from three reports to give merged results and metrics
Update classifications to desired level and update results and metrics
External classification file as input
Generate final reports
END

Fig. 4. Proposed verification methodology

3) Fault injection commands are automatically generated using developed scripts in Tool Command Language (TCL) format, considering parameters like clock period, reset time, start cycle, end cycle, and hold time range.
4) Fault simulation is run with generated TCL script using XFS serial engine.
5) The three reports generated are consolidated together to provide individual results along with the merged results, containing the updated DC in a final report.
6) A feature is introduced to manually update fault statuses. An external classification file is used to specify signals for status conversion along with source and destination classifications. Fault classes are updated for specified signals along with classification numbers for individual reports and the final report.
7) Both original and modified merged reports are kept as separate files to track differences before and after using the update classification feature.

The proposed flow covers the maximum possible fault space supported by the tools, improving the accuracy of the calculated DC. Additionally, using VC Z01X's concurrent engine ensures efficient simulations, minimizing runtime. There is also the option of triggering multiple runs with the XFS serial engine, which can help speed up the remainder of the fault simulation, albeit at the cost of using multiple licenses. This methodology is further validated in the next section.

V. Validation

This section discusses the validation of the proposed methodology on the discussed designs and an automotive SoC.

A. *Experimental setup*

The proposed methodology is validated on the three designs previously discussed. Further, to evaluate the flow's effectiveness at a larger scale, we select the Automotive SoC (AutoSoC) [13, 14], an open-source benchmark suite designed for automotive SoC applications. AutoSoC consists of configurable hardware IPs integrated into an SoC, offering diverse SMs (Dual core lock step processor, bus parity, checkpoint control, ECC on memories, Software Test Libraries) and automotive software applications.

SMs are enabled in the design with the help of additional defines (for example, `+define+MEMECC` for ECC on memories). The user can create a new configuration based on a combination of SMs. Functional strobes, consistent across all configurations, include signals for instruction and data buses, as well as Special Purpose Register accesses to external units like cache and Memory Management Unit. The checker strobes vary depending on the SMs enabled. For instance, if ECC is enabled, the ECC error detection from individual modules is configured as checker strobes. Fault targets and exclusions are user-dependent and adjustable to meet specific module-level injection needs.

We identify potential areas of improvement based on an initial analysis of results obtained from the baseline verification flow (XFS) used in AutoSoC. Thus, we implement ECC on all internal memories, along with the duplication of fetch, control, and load-store unit module duplication with temporal redundancy. This is the configuration on which the proposed verification flow is validated. SA and transient faults are enabled for fault simulation.

We introduce a new metric called "Total relative effort", which highlights the increase in the number of faults to be injected as compared to the individual tools. This provides an insight into the trade-offs required to get an accurate estimation of DC.

B. *Results*

Table IV presents a comparison of the results of fault simulation campaigns run with the individual tools and the proposed verification flow. As seen from the results, the proposed verification flow provides a more accurate DC estimation between the bounds of individual tool coverages. Our work provides a better representation of the fault space to be actually considered for FuSa verification purposes using the combination of two tools. Of course, we have to inject more faults compared to individual tool flows, thereby signifying more fault simulation effort. Nonetheless, for the largest design, AutoSoC, the overhead is 31% and 22% for XFS and VC Z01X, respectively. This trade-off results in a more accurate DC calculation which in turn leads to the correct assignment of ASIL level. For example, the initial results obtained from the proposed verification flow on the AutoSoC design results

TABLE IV
COMPARISON OF DC, NUMBER OF FAULTS AND TOTAL RELATIVE EFFORT

		XFS	VC Z01X	Proposed methodology	Total relative effort (XFS)	Total relative effort (VC Z01X)
Adder	**DC**	87.50%	82.14%	83.42%		
	Fault count	32	44	62	1.94	1.4
Counter	**DC**	60.00%	58.62%	59.70%		
	Fault count	40	38	48	1.20	1.26
FIFO	**DC**	68.96%	80.47%	70.36%		
	Fault count	5797	5412	6982	1.20	1.29
AutoSoC	**DC**	98.36%	96.12%	97.79%		
	Fault count	676213	726097	885839	1.31	1.22

in a DC of 95.88% (ASIL B) as compared to the DC from the baseline verification flow (98.36% specifying an ASIL C). With the update classification feature of the flow, we are able to elevate certain fault classifications, resulting in the final DC of 97.79%. However, such varying classifications could have dire implications on the ASIL, if the fault space is not correctly evaluated. Our work addresses this concern by considering the maximum possible fault space covered by the tools to provide an accurate estimation of DC.

VI. Discussion

We summarize the main takeaways from the results in the following points:

1) **Fault space coverage**: In order to correctly determine the ASIL of an automotive component, an accurate estimation of DC is important. We make sure that the FI simulation space is completely covered by a combination of the tools.
2) **Simulation run-time**: By basing our solution on VC Z01X concurrent engine, we are able to minimize the simulation overhead. Nonetheless, it is also possible to base the framework on XFS.
3) **Transient fault space complexity**: The transient fault space is quite extensive due to the possibility of multiple injection and hold times during a simulation. Fault space pruning needs to be considered to effectively choose transient faults that will propagate to the strobing points, thus preventing the simulation of safe faults.
4) **Future work**: Future methodologies could involve implementing automated test generation for undetected faults. This will reduce manual verification efforts further and expedite fault simulation campaigns, particularly at higher abstraction levels like RTL. Additionally, it remains to be seen if other FuSa EDA tools, such as Siemens'® Kaleidoscope™, can provide better results based on the defined metrics.

VII. Conclusion

The increasing demand for safety-critical electronic components in automobiles necessitates a thorough research of FuSa EDA tools used for fault simulation purposes. In this paper, we compare two FuSa EDA tools from Cadence® and Synopsys® for FI simulation to identify discrepancies in results and evaluate their effectiveness in covering the fault space. Neither tool covers the fault space entirely, and therefore, they do not contribute to an accurate DC estimation. Therefore, we introduce a novel verification methodology that combines the tools along with additional features and utilities. This proposed flow is tested on an automotive SoC, achieving a DC of 97.79%. Compared to the baseline verification flow, which yields a DC of 98.36%, the methodology provides a more comprehensive estimation by considering the maximum possible fault space, with an increased fault simulation effort of 1.31x. Our automated approach provides an end-to-end verification framework to conduct FuSa verification and obtain a highly accurate DC.

References

[1] J. P. Trovao, "Trends in automotive electronics [automotive electronics]," *IEEE Vehicular Technology Magazine*, vol. 14, no. 4, pp. 100–109, 2019.

[2] "ISO 26262 Road Vehicles - Functional Safety," International Organization for Standardization, Dec. 2018.

[3] A. Cagri Bagbaba *et al.*, "An automated formal-based approach for reducing undetected faults in ISO 26262 hardware compliant designs," in *2021 IEEE International Test Conference (ITC)*, 2021, pp. 329–333.

[4] A. Nardi *et al.*, "Functional safety methodologies for automotive applications," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 970–975.

[5] A. C. Bagbaba *et al.*, "Combining fault analysis technologies for iso26262 functional safety verification," in *2019 IEEE 28th Asian Test Symposium (ATS)*, 2019, pp. 129–1295.

[6] F. A. d. Silva *et al.*, "Efficient methodology for ISO26262 functional safety verification," in *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2019, pp. 255–256.

[7] K.-L. Lu *et al.*, "FMEDA-based fault injection and data analysis in compliance with ISO-26262," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2018, pp. 275–278.

[8] D. Alexandrescu *et al.*, "EDA support for functional safety — how static and dynamic failure analysis can improve productivity in the assessment of functional safety," in *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2017, pp. 145–150.

[9] F. Ferlini *et al.*, "Enabling ISO 26262 compliance with accelerated diagnostic coverage assessment," *Electronics*, vol. 9, no. 5, 2020.

[10] C. Lopez-Ongil *et al.*, "Autonomous fault emulation: A new FPGA-Based acceleration system for hardness evaluation," *IEEE Transactions on Nuclear Science*, vol. 54, no. 1, pp. 252–261, 2007.

[11] O. Ballan *et al.*, "Verification of soft error detection mechanism through fault injection on hardware emulation platform," in *2010 International Conference on Dependable System and Networks Workshops (DSN-W), Jun*, 2010.

[12] A. Sherer *et al.*, "Ensuring functional safety compliance for ISO 26262," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–3.

[13] F. A. da Silva *et al.*, "Special session: AutoSoC - a suite of open-source automotive SoC benchmarks," in *2020 IEEE 38th VLSI Test Symposium (VTS)*, 2020, pp. 1–9.

[14] "Autosoc benchmark suite." (2020), [Online]. Available: https://www.autosoc.org/home. (accessed: Jan 13, 2023).