A Novel Approach for faster diagnostic coverage closure aided by Software Test Libraries of CPU Cores

[1]Amresh Kumar Lenka, [2]Varun Kumar C, [3]Naveen Srivastava, [4]Subramanian, [5]Sekhar Dangudubiyyam
Design Verification, FDSHW Samsung Semiconductor India Research Bangalore, India
[1]amresh.lenka@samsung.com; [2]varun.k2@samsung.com; [3]naveen.sr@samsung.com; [4]ramanian.r@samsung.com;
[5]sekhar.d@samsung.com

***Abstract*- Ensuring functional safety in the processor cores has become increasingly critical in modern applications, particularly under the rigorous standards of ISO 26262 and IEC 61508. Diagnostic coverage (DC) [1] closure, a cornerstone of functional safety verification, depends heavily on robust stimulus generation and fault detection mechanisms. However, the complexity of contemporary processors makes it challenging to access adequate observation and detection points, potentially leaving diagnostic gaps and missed faults. This paper presents a novel approach to achieving faster and more comprehensive diagnostic coverage closure through a Software Test Library (STL) tailored for processor core's functional safety. STL is developed as a reusable Safety Element out of Context (SEooC) in accordance with ISO 26262 [1] and a compliant item in accordance with IEC 61508 [2] requirements. It is designed to be lightweight, with minimal memory footprint, enhancing reusability across different systems. By developing the STL as a standalone safety element, it can be seamlessly integrated into various project without the need for repeated compliance checks, supporting quicker time-to-market and significant resource saving in diverse safety-critical applications.**

For fault simulation, VC-Z01X tool has been employed to support the STL's fault coverage objective. VC-Z01X injects faults into a baseline "good machine (GM)" to create a "faulty machine (FM)" executing identical stimulus on both versions and comparing the outcome at designated observation and detection points. This comparison enables precise quantification of the STL's diagnostic performance, ensuring alignment with safety standards. In addition, the non-intrusive design of the STL allows external reporting, enabling rapid safety verification without requiring modification to the processor core's architecture.

Simulation and FMEDA (Failure Modes, Effects, and Diagnostic Analysis) results confirm the efficacy of our approach. By leveraging the VC-Z01X tool and STL, higher diagnostic coverage was achieved much faster in our testbed when compared to legacy system use-case test cases. For proof of concept we have used a subpart of the core, specifically the u_instruction_execute module, with just a single STL test case out of 5000+ feature STL test cases that verifies simple load and store instructions. This test case was used in fault simulation to produce FMEDA results for netlist or synthesized design built with proprietary 3nm library cells. The FMEDA results were published for 2 failure modes only.

For diagnostic coverage two detection statues or mechanisms have been considered namely, 1. DT (Detected by STL), this status will be reported if an incorrect operation is detected. 2. DW (Detected by watchdog), this status will be reported when the design under test (DUT) slips from the expected operational time. We have simulated 60916 faults out of which 55448 were testable faults and 6347 were Inconclusive faults due to significant divergence between GM and FM. With just 1 STL test case a cumulative diagnostic coverage of 7.03% for 2 failure modes has been recorded.

Keywords—STL, VC-Z01X, DC, ISO 26262, IEC 61508, FMEDA, FTTI, SEooC.

I. INTRODUCTION

The reason for pursuance of faster diagnostics coverage closure using a Software Test Library is, in high-stake applications like automotive, medical and aerospace, delays in fault detection can have serious consequences. The aim is to detect the fault safely before any hazard can occur. Several aspects on the need for fault simulation and faster detection is indispensable in each of these contexts have been explained.

A. Automotive Safety

As autonomous vehicles grow more complex, rapid fault detection is essential to prevent malfunctions that could lead to accidents. Faster diagnostic coverage ensures systems can identify and respond to issues within FTTI.

B. Cybersecurity

At Black-Hat conference 2021, Ledger's Hardware Security Expert Olivier Heriveaux used lasers to induce faults into the chip, leading to retrieval of secure key used to decode the information stored on a Cypto Hardware Wallet using STM32L496 Microcontroller. Heriveaux found that using just two laser-induced faults were enough to compromise the security of the chip [3]. Fault simulation gives the quantitative measure of the functional safety of a device indicating how well design can recognize and detect the faults.

C. Medical Device Reliability and Patient Safety

Medical equipment, such as pacemakers, ventilators, and diagnostics machines, must operate reliably under every all conditions to avoid jeopardizing patient health. Faults in these devices can lead to life-threatening situations if they go undetected. Fault simulation allows engineers to test how medical devices will respond to faults, ensuring that critical functions are not compromised in case of hardware failures.

D. Space Applications

Spacecraft and satellites operate in extreme conditions with limited opportunities for repair, so any undetected fault can lead to mission failure. Fault simulation is essential in verifying that onboard systems can detect, isolate and recover from faults autonomously. This testing is critical given the high radiation exposure, temperature extremes, and mechanical stresses involved in space.

Several aspects of functional safety and the convergence of all the parameters considered can be explained in the below context.

A. Functional Safety

Ensures the system operates safely even in the presence of faults or failures, addressing risk reduction measures to avoid or mitigate hazards. Governed by standards like ISO 26262 and IEC 61508. Functional safety drives the need for detailed failure analysis (FMEA/FMEDA). It defines safety goals and diagnostic performance targets based on ASIL requirements. The outputs expected are Safety requirements, fault tolerance mechanisms, safety goals and target ASIL levels.

B. FMEA (Failure Modes and Effects Analysis)

Identifies potential failure modes, their causes, and their effects on system behavior. It's required to analyze failure scenarios and to evaluate severity (S), occurrence (O), and detection (D) to prioritize risks. For Functional Safety it helps to ensure all failure modes relevant to functional safety goals are identified. The expected outcome is a qualitative list of failure modes and their impacts.

C. FMEDA (Failure Modes, Effects, and Diagnostic Analysis)

Enhances FMEA by quantifying failure rates and analyzing diagnostic coverage for fault detection and isolation. It classifies failures into Safe, Dangerous Detected (DD), Dangerous Undetected (DU), etc. It considers diagnostic mechanisms to calculate DC. It provides quantitative data to justify compliance with functional safety requirements. FMEDA determines the diagnostic coverage needed for achieving safety goals. The results from the exercise are Failure rate data, diagnostic coverage, and safety metrics like SFF (Safe Failure Fraction) and λ values (failure rates).

D. Diagnostic Coverage (DC)

Measures the effectiveness of diagnostic mechanisms in detecting and managing faults. It is calculated as part of the FMEDA process. Higher DC is essential for meeting higher ASIL requirements to ensure fault detection and mitigation.

E. ASIL (Automotive Safety Integrity Level)

Represents the safety risk level of a system component, based on: Severity (S) of harm, Exposure (E) to the hazardous event and Controllability (C) of the hazard by the user. The categories range from ASIL A (lowest) to ASIL D (highest)

Due to all the above factors while on one hand, verification effort increases to deploy fault free functionality, on other hand the development cycle shortens demanding an alternate approach which will be discussed in this paper.

II. METHODOLOGY

In this methodology there is an integration of fault injection and simulation capabilities of the fault simulation tool with software test libraries (STL). STL enables us to simulate the full range of logical functionalities in the core. Since the STL exercises critical processor pathways, the injected faults are more meaningful by providing valuable faulty machines. We will discuss the methodology more elaborately in the *Execution Flow* section and other steps involved to realize our approach.

A. Failure Mode Effect Analysis (FMEA)

The design was carefully analyzed and divided into distinct parts, each responsible for executing specific functions within the overall systems. For each part, we identified potential failure modes and examined how these failures could impact the entire design. This analysis helped categorize the potential effects of individual part failures on system functionality. DC fault model was used for permanent faults. Other fault models were used to model dependent faults and transient faults, but that is beyond the scope of this paper. For our proof of concept, we focused specifically on failure modes related to incorrect operation, live lock and deadlock related only to permanent faults.

B. Categorizing Identified Failure Modes

We have categorized the failure modes into below types [4]

1. Mission: Failure modes of parts executing primary functions. The failure rate associated with this will be categorized as single point or residual faults.
2. Passive: Failure modes of parts responsible for fault detection. The failure rate associated with this will be categorized as multi-point-failures.
3. Active: Failure modes of parts responsible for safety measures. The failure rate can either be associated to single-point failure or multi-point failure.

C. Assessing Technology Specific Failure Rates

TABLE I
TECHNOLOGY SPECIFIC FAILURE RATES

Technology type	Fault type	Failure in FIT
Digital	Permanent	1FIT/gate
Memory	Permanent	1FIT/bit

D. Calculating the ISO26262 safety metrics

1. Raw failure rate [1] [4]: λ_p = (Failure in FIT for Digital technology type) × (Total area of the sub-part / (unit design element size)
2. S_p%: Safe fault fraction.
3. Dangerous fault fraction: $\lambda_{pd} = \lambda_p - S_p\%$
4. $DC\%$: Diagnostic coverage of STL for a failure mode.
5. Residual Faults: $\lambda_{RF} = \lambda_p \times (1 - S_p\%) \times (1 - DC\%)$

E. Requirement of STL

Standard testing procedures often rely on system use case scenarios or standard benchmarks like cache hit/miss and Dhrystone respectively. However, these scenario-driven tests may not comprehensively address all logical functions, leaving potential safety gaps. Using STL allows us to move beyond scenario-specific testing by individually targeting each design function. The STL targets each core function to maximize fault path activation, enhancing fault detection reliability. A common limitation in functional safety verification is incomplete stimulus coverage, which leaves potential gaps in fault detection. In contrast, the STL's comprehensive logic coverage increases the probability of activating and observing faults, significantly improving fault coverage when paired with fault simulation tools. The STL has also been optimized to ensure the detection of the fault within Fault Tolerance Time Interval (FTTI) [1], which is essential for safety-critical application where prompt fault detection is critical to prevent failure. By catching faults within the required FTTI, STL supports systems in meeting stringent safety integrity requirements. The STL's efficiency is further enhanced through its capability for both boot-time and run-time execution. During boot, it verifies core functionality, establishing a safe operational baseline. During normal operation, it periodically checks for latent faults without impacting system performance. This dual capability enables continuous diagnostic coverage, caching faults that might only emerge under specific operational conditions. With the STL, the status will be reported to an external pseudo-register, the strobing becomes simpler, making the setup reusable across projects.

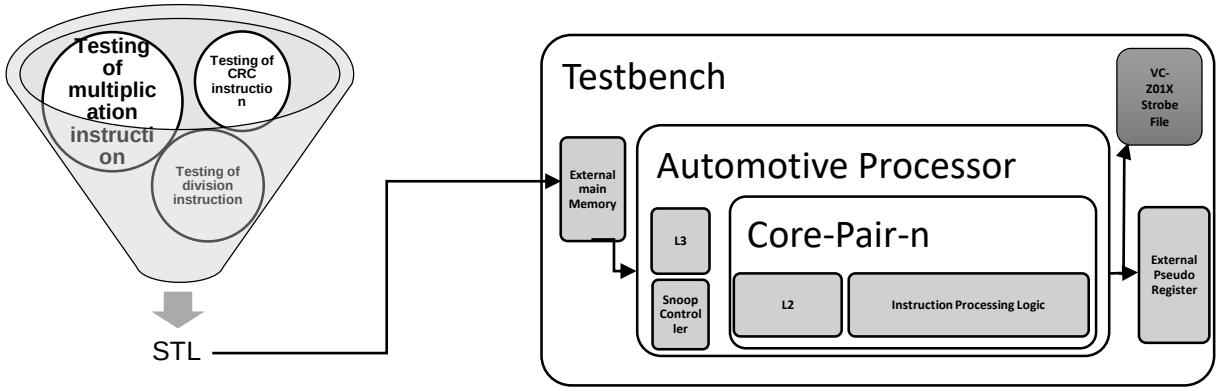

Figure 1. Representative image of STL synthesis and execution setup, Including the strobe file.

F. STL capabilities

The STL includes over 5000 feature test cases. In the table below, we highlight a selection of these test cases to demonstrate how each feature is specifically targeted within our DUT.

TABLE II
FEATURE SPECIFIC TEST LIST

Feature	Description
Feature 1	Testing of logical instructions AND, EOR and ORR
Feature 2	Testing of multiplication instructions
Feature 3	Testing of branch and addition instructions
Feature 4	Testing of GPR registers
Feature 5	Testing of CRC instructions
Feature 6	Testing of load/store instruction
Feature 7	Testing of bit operation instructions
Feature 8	Testing of subtraction instructions
Feature 9	Testing of shift instructions
Feature 10	Testing of conditional select instructions

G. VC-Z01X overview [5]

The below image presents a brief execution flow of the VC-Z01X tool, we will be using the tool for the functional safety use case (FuSa). The tool takes the design files, Test bench files, Strobe file and the Standard fault format file (SFF) to produce the fault DB. The flow will be explained more elaborately in *Execution Flow* section.

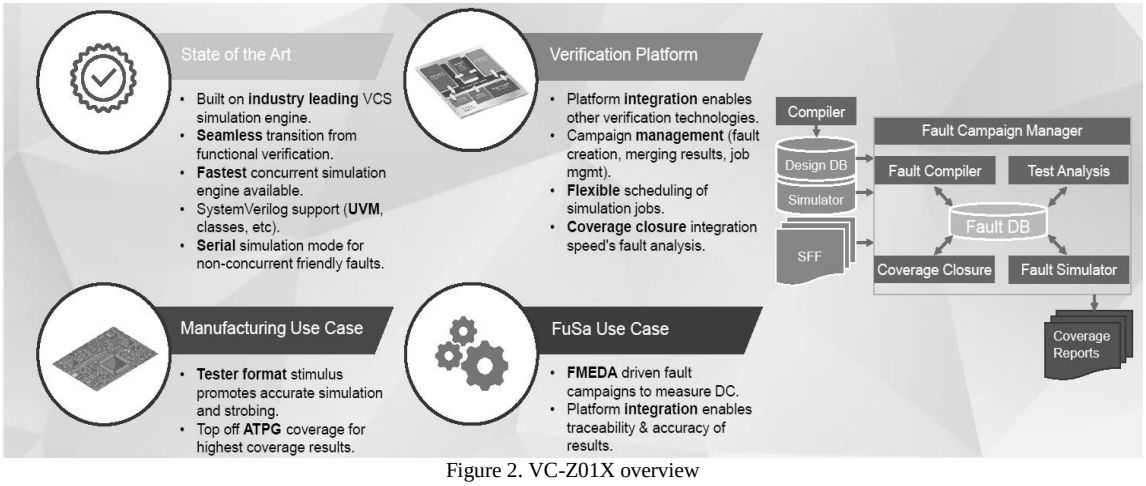

Figure 2. VC-Z01X overview

H. Execution Flow

We will start with generating the test case for required instruction, like for testing multiplication, division and addition instructions. After the build, compiled hex or elf files are loaded into external main memory. Logical simulation is run, to check the credibility of test case.

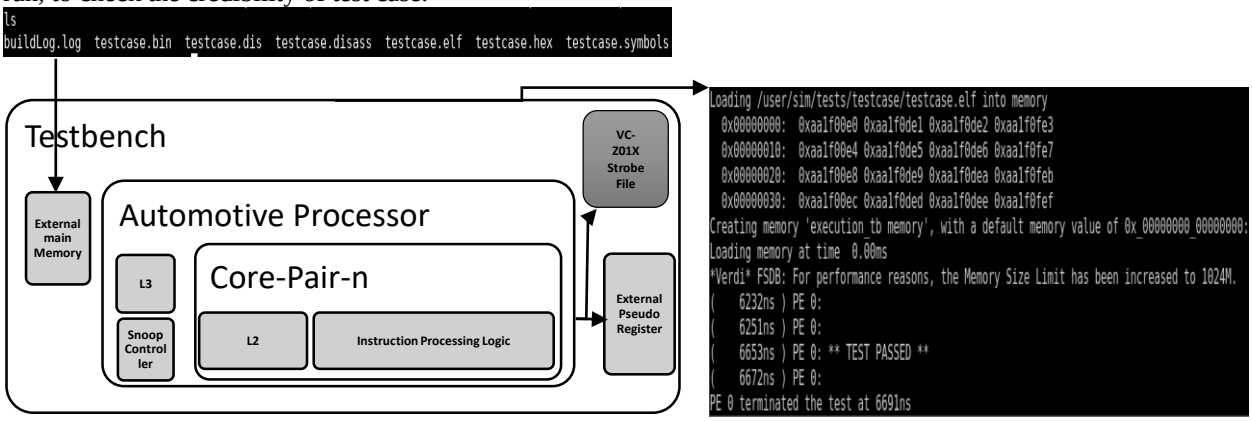

Figure 3. STL test case hex file loaded into the memory, followed by logical simulation of the elf file. The "TEST PASSED" status proves the sanity of the test case on a GM.

Once the test case's credibility is proved, we will proceed with the preparing the SFF for status definition and fault injection location.

```
FaultGenerate STL {
    NA [0,1] { PORT [INPUT,OUTPUT] "automotive_cpu_wrapper_execution_tb.u_automotive_cpu_wrapper.u_core_pair.u_core_0.u_cpu.u_instruction_execute.**"}
    NA [0,1] { PORT [INPUT,OUTPUT] "automotive_cpu_wrapper_execution_tb.u_automotive_cpu_wrapper.u_core_pair.u_core_0.u_cpu.u_instruction_fetch.**"}
    NA [0,1] { PORT [INPUT,OUTPUT] "automotive_cpu_wrapper_execution_tb.u_automotive_cpu_wrapper.u_core_pair.u_core_0.u_cpu.u_instruction_decode.**"}
    NA [0,1] { PORT [INPUT,OUTPUT] "automotive_cpu_wrapper_execution_tb.u_automotive_cpu_wrapper.u_core_pair.u_core_1.u_cpu.u_instruction_execute.**"}
    NA [0,1] { PORT [INPUT,OUTPUT] "automotive_cpu_wrapper_execution_tb.u_automotive_cpu_wrapper.u_core_pair.u_core_1.u_cpu.u_instruction_fetch.**"}
    NA [0,1] { PORT [INPUT,OUTPUT] "automotive_cpu_wrapper_execution_tb.u_automotive_cpu_wrapper.u_core_pair.u_core_1.u_cpu.u_instruction_decode.**"}
}
```

Figure 4. SFF code to target, various hierarchies with stuck-at-0 and stuck-at-1 faults, Fault injection at *u_instruction_execute* is only considered for FMEDA for this paper.

Once the SFF is prepared, we will proceed with the TCL script automation of the fault simulation, In the below image we will 1st set the load sharing facility (LSF) grid for the fault sim tasks at "*set_submit_cmd*", next we will

create the campaign by providing the SFF, compiled DB simv.daidir, campaign name, fault sampling percentage, And the scope of the DUT at "*create_campaign*". After creating the fault campaign, we will set the campaign as the default for the next tasks at "*set campaign*". After setting the default campaign, we will proceed to set the max number of parallel fault simulation that can be simulated at "set_config". Next at "*create_testcases*" we will provide the name of the STL test case along with the required elf file for the toggle simulation. Once the toggle simulation is over we will proceed to fault simulation at "*fsim*" and finally we will get the report containing DC at "*report – campaign automotive_cpu_ss*".

```
set_submit_cmd -grid_type LSF -cmd {bsub -R "select[(osversion==RHEL7.9||osversion==RHEL6.10)] rusage[mem=300000]"} -task_type default
create_campaign -args "-full64 -daidir simv.daidir -sff ./stl.sff -campaign automotive_cpu_ss -sample percent:20 -dut automotive_cpu_wrapper_execution_tb.u_automotive_cpu_wrapper -overwrite"
set_campaign -campaign automotive_cpu_ss
set_config -global_max_jobs 1000
report -campaign automotive_cpu_ss  -report before_fault_sim.rpt -showfaultid -overwrite
create_testcases -name {[redacted]} -exec "./simv" -args "+elfname=./tests/testcase/testcase.elf" -fsim_args "-fsim=fault+dictionary"
fsim
report -campaign automotive_cpu_ss -report diagnostic_coverage.rpt -showfaultid -overwrite
```

Figure 5. TCL script code for fault simulation automation.

III. FMEDA RESULT*

After the successful execution of faultsim, below diagnostic coverage report is generated. This particular report is with testing just load and store instruction and only u_instruction_execute module is targeted for both the cores present in our DUT.

```
#------------------------------------------------------------------------
# Number of Faults:                                 60916 100.00%
#
# Untestable Faults:                                 5468   8.98%  100.00%
#   Untestable Unused                          UU     5264   8.64%   96.27%
#   Untestable Tied                            UT      204   0.33%    3.73%
#
# Testable Faults:                                  55448  91.02%  100.00%
#   Hyperactive                                HA     6347  10.42%   11.45%
#   Not Detected by STL                        ND    45650  74.94%   82.33%
#   Detected by STL                            DT      360   0.59%    0.65%
#   Detected by Watchdog                       DW     3091   5.07%    5.57%
#
# Status Groups ---------------------------------------------------------
#   Hyper                                      HG     6347  10.42%
#   Untestable                                 UG     5468   8.98%
#
# Coverage --------------------------------------------------------------
#   Diagnostic Coverage                                 7.03%
#------------------------------------------------------------------------
```

Figure 6. The Diagnostic coverage report.

VC-Z01X also facilitates the fault simulation status per hierarchy, hence the complete picture of DC status for all hierarchies from top to bottom is obtained.

```
# Statuses: ND, HA, UU, UT, DT, DW
Total             ND             HA            UU            UT            DT             DW Scope
----- -------------- -------------- ------------- ------------- ------------- -------------- -----
60916 45650  (74.94%) 6347  (10.42%) 5264   (8.64%) 204   (0.33%) 360   (0.59%) 3091   (5.07%) automative_cpu_wrapper_execution_tb
60916 45650  (74.94%) 6347  (10.42%) 5264   (8.64%) 204   (0.33%) 360   (0.59%) 3091   (5.07%) -u_automative_cpu_wrapper
30458 22917  (75.24%) 3620  (11.89%) 2632   (8.64%) 102   (0.33%)  56   (0.18%) 1131   (3.71%) --u_core_pair
30458 22917  (75.24%) 3620  (11.89%) 2632   (8.64%) 102   (0.33%)  56   (0.18%) 1131   (3.71%) ---u_core_0
30458 22917  (75.24%) 3620  (11.89%) 2632   (8.64%) 102   (0.33%)  56   (0.18%) 1131   (3.71%) -----u_cpu
30458 22917  (75.24%) 3620  (11.89%) 2632   (8.64%) 102   (0.33%)  56   (0.18%) 1131   (3.71%) ------u_instruction_execute
30458 22733  (74.64%) 2727   (8.95%) 2632   (8.64%) 102   (0.33%) 304   (1.00%) 1960   (6.44%) ---u_core_1
30458 22733  (74.64%) 2727   (8.95%) 2632   (8.64%) 102   (0.33%) 304   (1.00%) 1960   (6.44%) -----u_cpu
30458 22733  (74.64%) 2727   (8.95%) 2632   (8.64%) 102   (0.33%) 304   (1.00%) 1960   (6.44%) ------u_instruction_execute
```

Figure 7. The Diagnostic coverage report per hierarchy.

A. FMEDA:

DT status will be flagged when an incorrect operation is detected, and DW status will be flagged when a livelock is detected.

As we got 6347 hyper active (HA) or inconclusive faults, For the DC calculation HA numbers will be removed from total testable faults, Hence the actual testable faults become 55448 – 6347 = 49101 faults.
Now the DT when adjusted for the new denominator of 49101, it becomes 360/49101 = 0.00733 => 0.733%.
Similarly, DW when adjusted for the new denominator of 49101, it becomes 3091/49101 = 0.06295 => 6.295%.

TABLE III
FMEDA for failure mode: Livelock

Part	Failure mode	Tech node	Safety related	Failure mode type	No of unit design elements	λ_p	S_p%	λ_{pd}	λ_p %	DC%	λ_{RF}
u_instruction_execute	Livelock	3nm	YES	MISSION	350	350	8.89%	318.885	0.02%	6.295%	298.81

TABLE IV
FMEDA for failure mode: Incorrect operation

Part	Failure mode	Tech node	Safety related	Failure mode type	No of unit design elements	λ_p	S_p%	λ_{pd}	λ_p %	DC%	λ_{RF}
u_instruction_execute	Incorrect operation	3nm	YES	MISSION	350	350	8.89%	318.885	0.00%	0.733%	316.54

λ_p : Raw Failure rate in FIT.
S_p%: Safe fault fraction.
λ_{pd}: Dangerous fault fraction.
λ_p % : Failure rate distribution for the Failure mode as a percentage of whole design's failure rate.
DC%: Diagnostic coverage.
λ_{RF} : Residual Faults.

With just one STL test case we achieved a cumulative DC of 6.295 + 0.733 = 7.03% (rounded off from 7.028%), Now when 5000+ test cases will be simulated for all the sub-parts of the design the cumulative DC will be much greater than current number. The 7.03% is achieved fairly quicker due to simulation of STL. The same quick result is expected from the complete suite.

CONCLUSION

A cumulative DC of 7.03%, was achieved with the execution of 1 STL test case, for just a single test case targeting the load-store function, The DC number is considered good for a single test case targeting failure modes, Incorrect operation and live lock. Once the complete suite of the STL is simulated the cumulative DC is expected to surpass the current DC achieved by a single STL test case. Since the STLs were coded to test single functions rather than specific sequence of functions (which can be inferred as a scenario). The DC numbers achieved through STL cannot be directly compared to our previous fault analysis methodology employing standard processor test cases like Dhrystone, cache hit/miss scenarios. Therefore, author is unable to produce a comparison figure.
Fault simulation's traditional role in developing chip manufacturing tests has been expanded into other areas equally critical. During chip development, it helps to ensure that verification methodologies are robust enough to catch all design bugs. The strict requirements of functional safety standards such as ISO 26262 require fault simulation to demonstrate that safety mechanisms can detect faults in the field so that corrective action can be taken. Methodology discussed integrates the fault simulation capabilities of the tool with the benefits offered by Software Test Libraries. High reusability provides tremendous future scope and would help in fast closure of fault simulation with FMEDA report generation.

REFERENCES

[1] ISO 26262-1:2018 Road vehicles — Functional safety 2nd edition, Dec 2018.

[2] IEC 61508-1:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 1: General requirements, 2nd edition, ISBN number 9782889105243, April 2010.

[3] Olivier Hériveaux, “Defeating a Secure Element with Multiple Laser Fault Injections” Black Hat USA 2021.

[4] Accellera, “Functional Safety Working Group White Paper”, December 2023.

[5] Synopsys, “VC Z01X Fault Simulation for Functional Safety Verification”