Optimizing Large-Scale Fault Injection Experiments through Martingale Hypothesis: A Systematic Approach for Reliability Assessment of Safety-Critical Systems

Saurabh Hukerikar, Atieh Lotfi, Yanxiang Huang, Jason Campbell, Nirmal Saxena
NVIDIA
Santa Clara, CA, USA
{shukerikar,alotfi,yanxiangh,jacampbell,nsaxena}@nvidia.com

***Abstract*—Functional safety of complex systems in safety-critical domains such as automotive, robotics and healthcare systems is paramount. Fault injection techniques play a pivotal role in the rigorous evaluation of functional safety. This paper introduces a novel approach to optimizing large-scale fault injection experiments by leveraging the Martingale hypothesis. By integrating such probabilistic models, our method enhances the efficiency of fault injection studies and enables deriving high confidence estimates of the diagnostic capabilities of safety critical systems using information gleaned from limited experiments. We demonstrate a pathway to achieving high functional safety by leveraging a combination hardware and software diagnostics. We validate our approach with extensive gate-level fault injection experiments performed over two years on NVIDIA's Graphics Processing Units (GPUs) spanning over 11 million simulation hours. The presented findings and methodologies have broad implications for functional safety analysis, highlighting the transformative potential of this approach in overcoming previous challenges and advancing the state-of-the-art in fault injection.**

***Index Terms*—Automotive Functional Safety, ISO 26262, Artificial Intelligence (AI), Graphics Processing Unit (GPU), Fault Injection, Random Hardware Faults, Transient Faults, Permanent Faults, Fault Detection, Software Diagnostics**

I. Introduction

Fault injection plays a pivotal role in evaluating the safety of a system. Experiments model the impact of introducing various faults into the system in order to assess the ability of the system's diagnostic features to detect the faults and mitigate potential failure events. For automotive systems, the ISO 26262 standard for functional safety [1] mandates rigorous safety assessments to ensure that stringent safety requirements are met. This demands a high level of precision and realism in fault modeling. Performing gate-level fault injection is the norm, but the process can be complex and time-consuming, especially in highly complex, multi-billion transistor circuits such as NVIDIA's Graphics Processing Units (GPUs). Furthermore, modeling and simulating faults at the gate level for safety evaluation presents numerous challenges, which include: (a) large number of simulated faults go undetected (masked faults) due to the limited scope of computation captured from a real application for a gate-level experiment, (b) limited input stimulus makes it difficult to draw statistically significant conclusions about the behavior of the circuit when the system would encounter much more diverse datasets in a range of scenarios during real-world operation, (c) evaluation of diagnostic mechanisms, notably reliable quantitative estimates of the interplay and potential synergies between hardware and software mechanisms, and their impact on the architectural safety metrics.

In this work we present an approach that leverages the Martingale hypothesis [2] to optimize gate-level fault injection experiments and enhance the estimation of the safety metrics of a system. The key idea behind the Martingale hypothesis is that the expected value of a random variable at a future time, given the information available up to the current time, is equal to its current value. Applying this hypothesis to gate-level fault injection of random hardware faults enables using information available from limited past experiments for making predictions about future fault injection outcomes.

The main contributions of this paper are:

- We demonstrate how the outcomes of fault injection simulations follow a Martingale process, which applied to gate-level fault simulations of a highly complex GPU design enables projecting fault outcomes for various computations performed on GPUs for a range of input stimuli arising from diverse operational scenarios
- We develop the mathematical framework that builds on the Martingale property of fault simulation outcomes to provide improved accuracy and reliability in estimating the key functional safety metrics
- We also demonstrate how the framework enables the integration and combination of metrics for hardware and software diagnostic mechanisms
- We present empirical evidence that validates our approach obtained through extensive gate-level fault injection experiments performed over 2 years totaling 11 million simulation hours

2833-292X/24/$31.00 ©2024 IEEE
DOI 10.1109/DSN-S60304.2024.00036

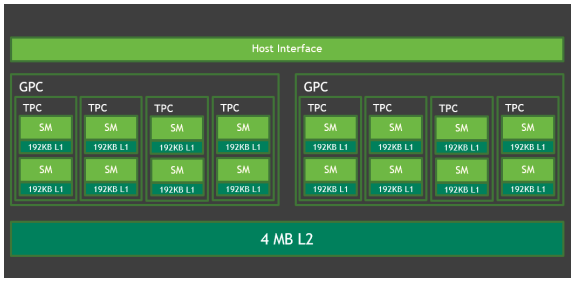

Fig. 1: Ampere GPU in the Orin SoC

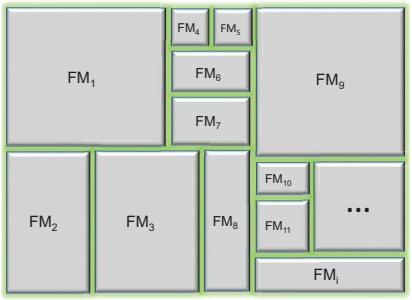

Fig. 2: Failure Modes in GPU Design

II. GPU Diagnostic Mechanisms

The NVIDIA DRIVE autonomous vehicle platform is based on the Orin SoC (Figure 1), which contains an integrated GPU based on the Ampere architecture. It is composed of 2 Graphic Processing Clusters (GPCs), 8 Texture Processing Clusters (TPCs), 16 Streaming Multiprocessors (SM), 192 KB of L1-cache per SM, and 4 MB of L2 cache. There are 128 CUDA cores per SM in Ampere and 4 tensor cores per SM. The GPU provides a total of 2048 CUDA cores and 64 Tensor cores delivering up to 4096 FP32 TFLOPs of CUDA compute and up to 131 Sparse TOPs of INT8 Tensor compute.

A. Hardware Diagnostic Mechanisms

The GPU contains various safety mechanisms built into its design, which are capable of detecting abnormal operations such as invalid instructions, out-of-bounds addresses, or memory translation errors. Fault events in the GPU logic covered by these diagnostics commonly raise machine check exceptions (MCE). The GPU design also contains various watchdog timer mechanisms that signal the stall of the processor due to a fault event. Additionally, various RAM structures in the GPU are protected using parity or error correcting codes (ECC). These diagnostic mechanisms can detect transient or permanent random hardware fault events.

For evaluation of ISO 26262 safety metrics, the design of the GPU is partitioned into failure modes (FM) (illustrated in Figure 2). The circuits that make up each FM are associated with a specific set of conditions under which a failure could occur. Therefore, the scope of each failure mode is defined to contain a part of the logic in the processor that is associated with a single hardware diagnostic mechanism.

B. Software Diagnostic Library

For certain parts of the GPU logic, the implementation of effective hardware-based diagnostic mechanisms is too difficult to design, or known solutions are too expensive to implement. We have developed a range of software diagnostics that are designed to run as online periodic tests to detect the presence of permanent random hardware faults. This software diagnostic library (SDL) consists of tests written in a high-level programming language by abstracting the various functional units in the GPU into data and control paths, and implementing code fragments that exercise these paths. Each test computes a hash value that is compared against a reference value. The SDL tests can be scheduled opportunistically to run on the GPU concurrently with the mission application and are particularly effective at detecting permanent random hardware faults in the GPU datapath that have potential to cause silent data corruption (SDC).

III. Gate-Level Fault Injection Methodology

While the use of analytical methods is permitted by the ISO 26262 [3] standard, the ever increasing complexity of hardware components in safety-critical systems makes it challenging to accurately analyze the effectiveness of diagnostic mechanisms in handling random hardware faults. In the ISO 26262 standard the Single-Point Fault Metric (SPFM) assesses the resilience of a system to single-point faults (SPF) and residual faults (RF), which can either be mitigated by safety mechanisms or those that are inherently safe by design. The Probabilistic Metric for Hardware Failure (PMHF) is a long-term reliability metric that is an estimate of the time-averaged probability of the potential violation of a safety goal over the operational lifetime of a component. Diagnostic Coverage (DC) is the ratio of the failures detected and/or controlled by a safety mechanism to the total failures. A comprehensive description of the safety metrics is contained in [3], [4].

For DC estimation and evaluating the hardware architectural metrics (SPFM, PMHF) gate-level fault injection simulations are preferred over analytic methods since they provide highly accurate modeling of the low-level behavior of circuits in the presence of faults. However, fault injection simulations at the gate-level are computationally intensive and time-consuming especially for complex systems such as GPUs, which contain millions of gates. The focus of this work is on random hardware faults, specifically permanent faults in the GPU, which may occur in the field due to circuit degradation on account of thermal stress, electromigration, and material fatigue, harsh environmental operating conditions, manufacturing variability, etc.

Figure 3 illustrates the possible outcomes of a gate-level fault injection experiment. When an injected fault triggers one of the hardware safety mechanisms, the simulation terminates and the MCE or timeout is logged. Such fault outcomes are classified as Detected Uncorrected Errors (DUE). Faults that

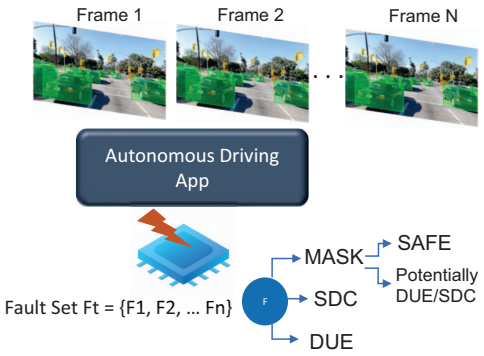

Fig. 3: Fault Outcomes from Simulation of Application Traces on a GPU Netlist

do not trigger hardware-based safety mechanisms, but alter the outputs of the simulations are classified as Silent Data Corruption (SDC). Certain faults trigger neither hardware-based safety mechanism, nor cause a SDC; such outcomes are treated as MASK. The MASK fault outcomes are challenging to account for in the analysis of DC. It is possible that some faults with a MASK outcome will never cause a SDC or a DUE due to it being SAFE. It also likely some faults with a MASK outcome may not have provoked a diagnostic mechanism, or caused a SDC, due to limited computation contained by the trace or due to inadequate stimulus in the trace. The inability to conclusively classify a MASK fault adds uncertainty to the estimation of the hardware architectural metrics. The primary objective of this paper is to rigorously test and demonstrate that the fault outcomes adhere to the Martingale property.

A key characteristic of a Martingale process is that the expected value of the next observation in the sequence, given all past observations, is equal to the current value. Mathematically, if X_t represents the value of the process at time t a process X_t is a Martingale if:

$$E[X_{t+1}|X_1, X_2, X_2, ..., X_t] = X_t \tag{1}$$

In applying the Martingale hypothesis, we treat all *MASK* faults as unsafe and estimate their potential to become *SDC* or *DUE* based on known fault behaviors. This hypothesis posits that the expected value of the next observation in a sequence for a FM is influenced by the fault outcomes previously observed. Applying the Martingale hypothesis to diagnostic coverage of residual faults (RF), the ($K_{FMi,RF}$) for FM$_i$ is calculated as:

$$K_{FMi,RF} = \frac{DUE + MASK \times K_{FMi,RF}}{DUE + SDC + MASK} \tag{2}$$

For the combination of hardware diagnostics and the SDL, let SDC_{sdl} and $MASK_{sdl}$) be the faults detected by SDL that caused SDC and MASK outcomes based on built-in hardware diagnostic alone. On the basis of these fault outcome relationships, the combined coverage of SDL and built-in HW diagnostic detection is given by:

$$\begin{aligned} K_{FMi,RF,SDL} = & \frac{DUE}{DUE + SDC} + \frac{SDC_{sdl}}{ALLFAULTS} \\ & + \frac{MASK_{sdl}}{ALLFAULTS} \\ & - \frac{MASK_{sdl}}{ALLFAULTS} \times \frac{DUE}{DUE + SDC} \end{aligned} \tag{3}$$

This method of calculating $K_{FMi,RF}$ for the combination of hardware and software diagnostics assesses the coverage provided by the built-in diagnostic mechanisms and the additional coverage offered by the SDL for SDC and MASK faults, but eliminates any overlap in coverage offered by both the SDL and the built-in diagnostic mechanisms.

IV. Experimental Setup

A. Fault Injection Framework

The experiments were conducted using a gate-level simulation infrastructure equipped with the capability to inject faults in the netlist of the GPU. The netlist was extracted from industry-standard Electronic Design Automation (EDA) tools ensuring the fidelity of the representation of the design. Since the focus of this work is on permanent random hardware faults, we simulate faults using the stuck-at fault model, which has been shown to be adequately representative of various types of permanent defects [5] [**?**] and is widely used in industry EDA tools. The target fault site (logic gate, latch or flip flop) in the design is set to a stuck-at fault (either a stuck-at 0 or 1) prior to initialization of the trace simulation. The stuck-at value persists for the total duration of the trace simulation. Each simulation run is injected with a single permanent fault at the selected fault site in the GPU netlist. The simulation framework is integrated with the error handler routines and low level software components in order for any interrupt events caused by the built-in hardware detection mechanisms to be recorded and logged.

B. Fault Sampling

For comprehensively evaluating the diagnostic safety mechanisms that are part of the GPU design, faults are sampled for every FM in the design. The number of fault locations for simulation within each FM is weighted by the size of the logic associated with the FM based on sampling theory and the requirements for error margins in the safety metric estimation. For any FM_i, the fault sites are randomly selected. The total size of the sample for our experiments selected from the gate netlist across all the FMs in the GPU is 5000 faults. We use a commercial EDA tool that performs testability analysis on the fault set classifying faults into testable and untestable faults; we prune the fault set and only simulate the testable faults. The pruned fault set contains 2789 fault locations.

C. Automotive Application Traces

For selecting workload traces for the fault injection studies for validating our Martingale hypothesis, we prioritized the representation of real-world automotive applications to ensure the relevance and applicability of our findings. The traces were

Fig. 4: Driving Scenarios Simulated with Fault Injection

TABLE I: Application Traces used for Gate-level Fault Injection Studies

Trace	Description
Localization	Enables the vehicle to accurately determine its own position with respect to its surrounding using GPS, IMU, radar and lidar sensors
Path Perception	Provides the vehicle with the ability to perceive and plan the trajectory including lane keeping, following traffic rules
Object Detection	Understands the presence of obstacles, pedestrians and other vehicles along a planned path
Autonet	Consists of various detection tasks including wait conditions, free space, light source detection

sourced NVIDIA's DRIVE software stack from an autonomous vehicle (AV) driving application. The chosen traces are representative of the critical part of the perception, planning and control stages of the AV pipeline.

The four traces representing the stages in the AV application framework are derived from the stages described in Table I. Each trace consists of numerous deep learning neural network-based computations intrinsic to the perception, decision-making, and control tasks and operates on a video input dataset of a realistic driving scenario.

D. Driving Scenarios

AV systems operate in dynamic and complex environments. Varied driving scenarios encompass a wide range of challenges, which include varying road conditions, traffic patterns, weather conditions, and user behaviors. By subjecting the GPU to fault injection at the gate-level in a variety of driving scenarios helps assess the system's ability to detect and respond to fault events in unexpected situations. We selected 20 varied yet typical driving conditions encountered by an AV system; frames from each video input data that we simulated are shown in Figure 4. Each application trace (localization, path perception, object detection and autonet), which has individual fault-free runtimes ranging from 2 to 7 days, was simulated for each of these 20 scenarios for all the 2789 gate-level faults. These experiments required 2 years of continuous simulation on a cluster consisting of about 2000 nodes taking a total of 11.2 million simulation hours.

E. SDL Traces

Traces for the software diagnostics are captured from the SDL test programs to execute on the gate-level simulation infrastructure. SDL traces contain a hashing function that computes the output signature, which is dumped upon completion of the simulation. Mismatches between the computed signature value from the simulation and the reference value is used to determine whether the diagnostic detects an injected stuck-at fault. The SDL fault injection experiments are performed with the same fault set containing 2789 stuck-at 0/1 faults used for the AV application traces.

V. Results and Discussion

The fault injection experiment outcomes of the AV application are used to calculate the aggregate diagnostic coverage of the safety mechanisms across all the FMs in the GPU. The built-in hardware diagnostic mechanisms are activated by the application workload exercising the logic associated with the FMs. The DC is weighted by the relative size of the FMs in terms of their failure rates. Figure 5 summarizes the aggregate DC for each of the four AV traces for all 20 driving scenarios. We observe that the DC variation for different workloads is small and is always within the margin of error

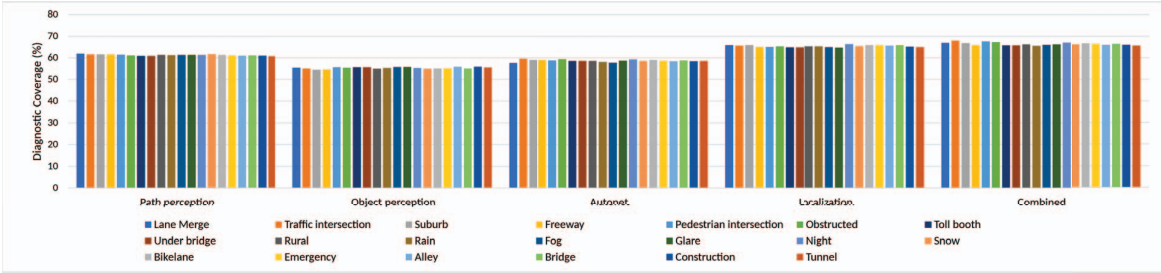

Fig. 5: Diagnostic coverage summary with built-in hardware detection mechanisms

for the fault sample size. The four traces are representative of distinct functions in the NVIDIA Drive AV pipeline that are executed per frame during system operation. Therefore, it is useful to estimate the aggregate DC from the combination of the four traces. The outcomes are combined such that final outcome is a MASK only when the fault is MASK for all traces' simulations; the final outcome is a DUE when the simulation of any one trace experiences a DUE; the outcome is a SDC, when any one trace causes SDC and none of the other traces trigger a DUE event. The aggregate DC of the combination of traces case has a small gain in DC compared to individual traces, since some faults trigger hardware diagnostic mechanisms for one of the traces but not for others. However, the variation is within the margin of error (2.43%) indicating a weak dependence of the aggregate DC on the workload.

Figure 6 summarizes the DC for the traces based on the combined diagnostic capabilities of the built-in hardware mechanisms and the SDL tests. Since the SDL diagnostics are designed to detect random permanent hardware faults in the GPU datapath where hardware mechanisms are insufficient, the aggregate DC of the combination of hardware and software diagnostics is increased to 87.5% (averaged across all across traces). Additionally, the DC is within a tighter range (between 87.53% and 88.42% for all traces/scenarios) since the SDL detects many of the faults that remain undetected by the HW mechanisms further weakening the impact of application workload on DC.

Figure 7 shows the percentage of fault simulation outcomes for each input scenario that result in *MASK* fault, i.e. the fault remains unobserved at the end of the simulation. It is observed that a significant fraction of faults are unobserved (on average 43.53% MASK outcomes) per frame. Some of these faults may manifest as SDC or DUE in an end-to-end application run, but don't provoke a MCE or timeout mechanism or cause a SDC due to limited computation and/or input stimulus in the trace. Some unobserved faults are SAFE i.e. they will never cause a SDC or a DUE. This large proportion of unobserved faults in the gate-level fault simulations adds uncertainty to the analysis of the DC for the various FMs, and consequently for estimating the hardware architectural metrics PMHF and SPFM of the GPU.

To overcome the uncertainty inherent in the MASK outcomes, we postulate that the outcomes for these faults in an end-to-end application follow a Martingale process. For validating our hypothesis, we analyze the fault outcomes by treating the simulation of the scenarios as consecutive frames to understand the temporal resolution of MASK faults across frames. Figure 8 (a) shows the trend in the resolution of MASK faults across 20 successive frames with only built-in hardware safety mechanisms. Figure 8 (b) similarly shows the trend in MASK faults across the scenarios with a combination of built-in hardware safety mechanisms and the SDL. A MASK fault is resolved in the subsequent frame if it manifests as a DUE or SDC in a subsequent frame. With only built-in hardware safety mechanisms, MASK faults are reduced from 43% to 37% across the 20 frames. With the combination of hardware and software diagnostic mechanisms, the MASK faults decrease from 27% to 23% across the 20 frames. On average 13% of the faults that are MASK after the simulation of the first frame resolve into SDC or DUE over the subsequent 19 frames.

In each scenario the workload characteristics cause different faults to be exposed as SDC, DUE or MASK. However, the ratio of DUE and DUE+SDC remains stable and within the margin of sampling error. The reduction in MASK faults between successive frames is also influenced by mutual entropy in input scenarios and the intrinsic fraction of SAFE faults. If one assumes that none of the MASK faults are SAFE, a Poisson density function based estimate predicts 468 additional frames to reduce the *MASK* faults to zero. However, the downward trend across the simulated frames and the resolution of MASK faults into SDC or DUE across frames validates our hypothesis on the Martingale property of the fault simulation outcomes.

The confirmation of the Martingale property within the experimental data serves as compelling evidence that the prevailing pessimism surrounding the treatment of MASK faults in the analysis of DC, and estimation of SPFM and PMHF metrics is unwarranted. Furthermore, it permits the SDL tests to be targeted towards detection of faults not covered by the built-in hardware diagnostics that can potentially cause SDC. This synergy of hardware and software diagnostics enables achieving high diagnostic coverage without relying on techniques such as lock step redundancy.

VI. Related Work

The practice of using gate-level fault injection is recommended by safety standards such as the ISO 26262 [6] and is

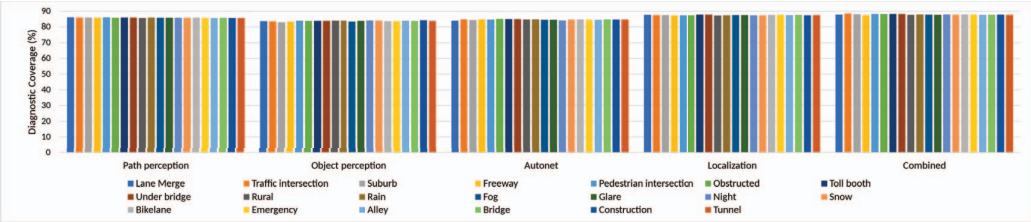

Fig. 6: Diagnostic coverage summary with SDL and built-in hardware detection mechanisms

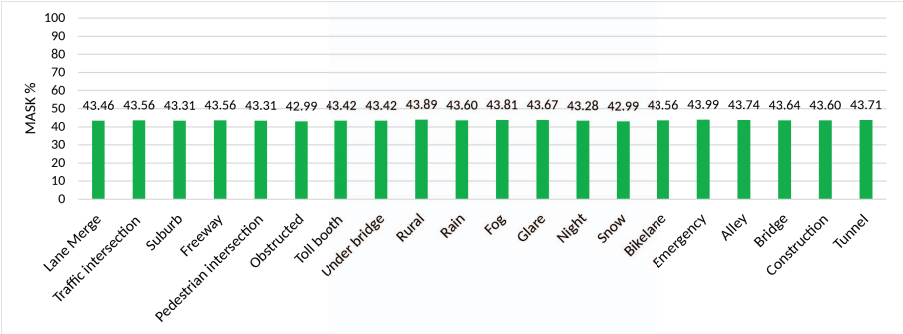

Fig. 7: Masked faults for each scenario

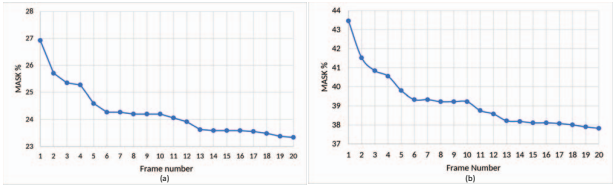

Fig. 8: Mask fault resolution in successive frames with: (a) built-in hardware diagnostics (b) combination of hardware diagnostics and SDL

standard practice in evaluation of safety-critical systems due to the realism in studying physical effects of faults on individual gates. To mitigate the massive computational resource demands of modeling and simulating gate-level faults, especially for large and complex circuits several studies have focused on developing comprehensive fault models at higher levels of abstraction such as RTL-based injection [7] [8], emulation [9] [10], using error models [11], or combining approaches such as Automatic Test Pattern Generators (ATPG), formal methods and simulation [12] to improve confidence in the safety metrics. Software-based fault injection techniques [13] [14] afford higher simulation speed, but the lower fidelity of the fault propagation behavior result in the metrics derived from these studies to be less precise. Radiation-based testing [15] [16][17] has been done, but such testing only evaluates the robustness of circuits to random transient faults and tend to have a large fraction of unobserved faults. In view of these challenges of loss of precision and realism in fault behavior prevalent in alternate approaches, the methods presented in this paper enable using gate-level simulation and leveraging the Martingale property in fault outcomes to derive high confidence estimates in the metrics despite availability of limited data. While it is demonstrated through our experiments that the approach is valid for stuck-at faults, the Martingale hypothesis is broadly applicable for any type of combinational fault.

VII. Conclusion

For rigorous functional safety evaluation of circuits using gate-level fault injection enables understanding the physical effects of random hardware faults on individual logic gates and their propagation, which provides a realistic representation of how faults manifest in safety-critical systems. This paper advances the state-of-the-art in fault injection for evaluation of safety critical systems through an adaptation of the Martingale hypothesis in projecting fault outcomes in gate-level fault injection experiments. The mathematical framework also enables evaluating the combination of hardware diagnostic mechanisms and software-based techniques such as SDLs, which fosters a holistic approach to safety evaluation and optimization. The Martingale property for the outcomes of the gate-level fault injection experiments was validated through extensive simulation data. This approach to analyzing fault injection experiments enables estimating key ISO 26262 safety metrics with high confidence despite limited scope of computation and input stimulus, and despite a large number of injected faults remaining undetected during the simulation scope.

REFERENCES

[1] International Organization for Standardization, "ISO 26262 Standard, Road vehicles - Functional safety Part 1: Vocabulary," 2018. [Online]. Available: https://www.iso.org/standard/68383.html

[2] J. L. Doob, "Regularity properties of certain families of chance variables," *Transactions of the American Mathematical Society*, vol. 47, pp. 455–486, 1940. [Online]. Available: https://api.semanticscholar.org/CorpusID:54001642

[3] International Organization for Standardization, "ISO 26262 Standard, Road vehicles - Functional safety Part 11: Guidelines on application of ISO 26262 to semiconductors," 2018. [Online]. Available: https://www.iso.org/standard/68392.html

[4] ——, "ISO 26262 Standard, Road vehicles - Functional safety Part 10: Guidelines on ISO 26262," 2018. [Online]. Available: https://www.iso.org/standard/68392.html

[5] E. McCluskey and C.-W. Tseng, "Stuck-fault tests vs. actual defects," in *Proceedings International Test Conference*, 2000, pp. 336–342.

[6] International Organization for Standardization, "ISO 26262 Standard, Road vehicles - Functional safety Part 5: Product development at the hardware level," 2018. [Online]. Available: https://www.iso.org/standard/68383.html

[7] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, "Fault injection into vhdl models: the mefisto tool," in *Proceedings of IEEE 24th International Symposium on Fault- Tolerant Computing*, 1994, pp. 66–75.

[8] J.-C. Baraza, J. Gracia, S. Blanc, D. Gil, and P.-J. Gil, "Enhancement of fault injection techniques based on the modification of vhdl code," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 6, pp. 693–706, 2008.

[9] F. Ferlini, L. O. Seman, and E. A. Bezerra, "Enabling iso 26262 compliance with accelerated diagnostic coverage assessment," *Electronics*, vol. 9, no. 5, 2020.

[10] A. Blin, Y. Laarouchi, and P. Quéré, "Fault-Injection using Virtualization for Critical Software Validation in Automotive," in *Embedded Real Time Software and Systems (ERTS2014)*, Toulouse, France, Feb. 2014. [Online]. Available: https://hal.science/hal-02272451

[11] N. Saxena and A. Lotfi, "Error model (em)—a new way of doing fault simulation," in *2022 IEEE International Test Conference (ITC)*, 2022, pp. 324–333.

[12] F. Augusto da Silva, A. C. Bagbaba, S. Hamdioui, and C. Sauer, "Combining fault analysis technologies for iso26262 functional safety verification," in *2019 IEEE 28th Asian Test Symposium (ATS)*, 2019, pp. 129–1295.

[13] J. Sini, M. Violante, and F. Tronci, "A novel iso 26262-compliant test bench to assess the diagnostic coverage of software hardening techniques against digital components random hardware failures," *Electronics*, vol. 11, p. 901, 03 2022.

[14] L. Osinski, T. Langer, M. Schmid, and J. Mottok, "Pyfi - fault injection platform for real hardware," in *ARCS Workshop 2018; 31th International Conference on Architecture of Computing Systems*, 2018, pp. 1–7.

[15] J. Karlsson, P. Liden, P. Dahlgren, R. Johansson, and U. Gunneflo, "Using heavy-ion radiation to validate fault-handling mechanisms," *IEEE Micro*, vol. 14, no. 1, pp. 8–23, 1994.

[16] A. Lotfi, S. Hukerikar, K. Balasubramanian, P. Racunas, N. Saxena, R. Bramley, and Y. Huang, "Resiliency of automotive object detection networks on gpu architectures," in *2019 IEEE International Test Conference (ITC)*, 2019, pp. 1–9.

[17] D. Oliveira, S. Blanchard, N. Debardeleben, F. F. Dos Santos, G. P. Dávila, P. Navaux, C. Cazzaniga, C. Frost, R. C. Baumann, and P. Rech, "Thermal neutrons: a possible threat for supercomputers and safety critical applications," in *2020 IEEE European Test Symposium (ETS)*, 2020, pp. 1–6.