Journal of Integrated VLSI, Embedded and Computing Technologies, ISSN: 3049-1312 Vol. 2, No. 3, 2025 (pp. 63-72)

RESEARCH ARTICLE ECEJOURNALS.IN

Reliability Analysis and Hardware Fault Injection for Safety-Critical Embedded Applications

El Manaa Barhoumia[1], Hardley Caddwine[2]*

[1]College of Applied Science, University of Technology and Applied Sciences, Ibri, Sultanate of Oman.
[2]Faculty of Engineering, University of Cape Town (UCT),South Africa

KEYWORDS:
Safety-Critical Embedded Systems;
Hardware Fault Injection (HFI);
Reliability Analysis;
Fault Tolerance;
Redundant Design;
ECC;
Safety Standards;
Fault Recovery Rate (FRR);
FPGA-Based Injection;
Real-Time Monitoring;
ISO 26262;
MTTR.

ARTICLE HISTORY:
Submitted : 20.06.2025
Revised : 17.07.2025
Accepted : 04.09.2025

https://doi.org/10.31838/JIVCT/02.03.08

Abstract

A safety-critical embedded system is central to all applications that cause serious effect in case of failure, such as human injury, financial impact, or disruption of the entire system. Such systems are widely used in the realms of automotive control (e.g., automatic braking), airline navigation, automation of industrial processes and life-support medical devices. Therefore, their reliability and fault-tolerance of operation are a top priority. The proposed work presents a detailed framework of reliability analysis coupled with hardware fault injection (HFI) methodology that brings the serious fault tolerance capabilities of real-world fault scenarios of embedded systems under testability. The suggested approach will unite formal reliability modeling, where such metrics like the failure rate, mean time between failures (MTBF), fault recovery rate (FRR), and the mean time to recovery (MTTR) will be used, and the hardware fault injection based on the experimental hardware application, both FPGA-based platform and microcontroller-based platform. Classes of faults that the study systematically addresses are stuck-at faults, transient faults and memory corruption faults through the processor registers, memory arrays and within I/O peripherals. Two of their benchmark applications: an autonomous braking controller and a ventilator feedback system are safety-critical and made subject to controlled fault injection campaigns. The experimental data in terms of fault recovery prove that the fault recovery ratio is very high 94.7%, the system lacks significant fractional time in the fully down mode, and fast recovery is centralized on possessing redundant architectural elements and error-correcting codes (ECC). In addition, the trade-offs between area/power overheads and fault coverage are analyzed in terms of normal and abnormal detection latencies. The paper does not only prove the efficiency of the suggested fault-tolerant mechanisms, but also identifies certain potential weaknesses in the design that may go unnoticed in the traditional scheme of simulation-based testing. The framework complies with steps of functional safety, including ISO 26262, and IEC 61508, in terms of confirming pre-certification validation and soundness profiling of embedded systems. In general, the research introduces an ingenious scalable hardware-verified methodology of improving the resiliency of safety-critical embedded systems, and furthermore, leads the way to future applications in embedded system design involving the exploitation of smart fault prediction and fault mitigation policies.

Author e-mail: cadd.hardley@engfacuct.ac.za

How to cite this article: Barhoumia E, Caddwine H. Reliability Analysis and Hardware Fault Injection for Safety-Critical Embedded Applications. Journal of Integrated VLSI, Embedded and Computing Technologies, Vol. 2, No. 3, 2025 (pp. 63-72).

Introduction

Most mission-critical applications rely on safety-critical embedded systems where failure is not recognized as an option. They are developed and deployed to have the highest tolerances of reliability, availability, and fault tolerance as they are deployed in such environments as aerospace avionics, safety systems (e.g., airbags) in automotive systems, industrial automation controllers, nuclear instrumentation systems, and life-sustaining systems such as medical devices (e.g., ventilators, pacemakers, etc.). Any failure in such conditions may lead to disastrous results, i.e. personal harm or even death of people, destruction of environment, or colossal losses of money. As such, reliability verification of safety-critical embedded systems is both a design requirement and a regulatory requirement.

Reliability under embedded system can be defined as the capability of a system to do what it is supposed to do under defined conditions within a given time. Conventional software simulation or analytical modeling schemes, which are effective early in design can at times be insufficient in modeling the true-timing inter-relationships, timing deadlines, and fault-behavior propagation characteristics of the deployed system. These constraints are further exacerbated when the existence of transient faults like single-event upsets (SEUs) through radiations or permanent faults like aging induced wear-out are to be considered. Figure 1 these kinds of faults are able to go unnoticed in normal verification procedures resulting in security holes that have not been identified in the complete deployment.

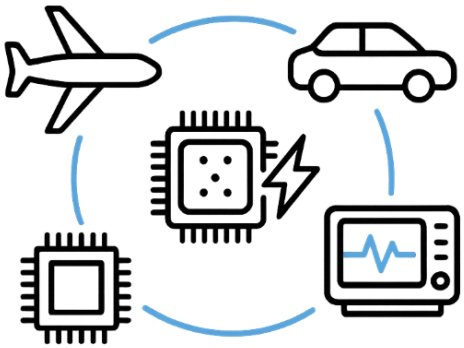

Fig. 1: Conceptual Overview of Safety-Critical Embedded Systems and the Role of Hardware Fault Injection in Reliability Evaluation

As a way of closing this verification loophole, Hardware Fault Injection (HFI) methods have taken strong grounds. The introduction of fault in HFI offers a more realistic and controlled way of introducing fault directly in the physical system or hardware abstraction layer. It allows realising the behaviour of the system under actual fault conditions to ensure engineers can perform analysis of fault detection, fault isolation and fault recovery mechanisms either within a hardware or software implementation of the system. Also, HFI is invaluable in pre-certification testing as it can be validated to industry safety standards like ISO 26262 (automotive), DO-254 (aerospace), IEC 61508 (industrial control).

The study proposes a comprehensive approach to reliability analysis that combines conventional reliability measurements with systematic approach of performing code injection fault on hardware. The proposed framework is applied on exemplary use cases that are related to safety-critical systems namely: an autonomous emergency braking controller and a ventilator feedback control system. The framework uses fault injection in the processor registers, memory subcomponents and the I/O interfaces to simulate fault types and test the effectiveness of redundancy, watchdog timers and error correction schemes as they apply to applying system functionality. Key performance indicators that are critical like the Fault Recovery Rate (FRR), Mean Time to Recovery (MTTR), and the downtime of the system are measured.

By close experimental analysis, this piece of work does not only confirm the strength of fault-tolerant designs, it also brings out the vulnerability that has been concealed, which can undermine reliability upon employment in the real world conditions. The results will assist in the development of stronger, certifiable, and predictable reliability-improved embedded platform compatible with the application in systems with high-assurance.

Literature Review

Over the last decades there have been a lot of research efforts in ensuring the reliability of safety-critical embedded systems. A number of fault tolerant design approaches, reliability modeling approaches and fault injection schemes have emerged and been enacted in the literature to deal with real time nature of failure modes of the embedded systems.

Embedded Fault Tolerance

Embedded systems in high-assurance systems need fault tolerance as a basic requirement. Error Correction Codes (ECC), watchdog timers, and Triple Modular Redundancy (TMR) are some of the widely used techniques to form watchdog states and detect possible hardware errors to recover.[1, 2] TMR has redundant copies of functional elements to produce correct outputs even in presence of fault and utilizes majority vote where it is guaranteed that no errors are left undetected, and ECC schemes (e.g., Hamming or BCH codes) diagnose and correct single/multi-bit errors[7] within the memory components. To reset the system in the event of an execution[8] hang-up, watchdog timers are typically used to do so, and fail-safe mechanisms are used to move the system to some known safe state in response to the detection of a fault.

Techniques of Modeling Reliability

There are a number of reliability modeling techniques that are used to examine the behavior of systems. Markov chains offer state-based models of probabilistic models of studying transitions between operational states and the failed ones.[9] The Failure Mode and Effects Analysis (FMEA) and Fault Tree Analysis (FTA) are qualitative methods, which are used to detect[10] the failure propagation chains, as well as to rank the risks

according to their impact and possibility. It is also strictly common to assess probabilistic reliability applicative in Weibull distributions, Poisson processes,[11] or Bayesian networks and compute such metrics as Mean Time To Failure (MTTF), Failure Rate (lambda 11, and Reliability Function R(t).[3, 4]

Hardware-based Methods of Fault Injection

Hardware Fault Injection (HFI) has also achieved popularity as a method of validating fault resilience in the embedded platforms.[12] Pulsed fault fault injection directly applies to system fault injection I/O pins to cause faults and measure behavior.[5] Bit-flip fault FPGA-based fault injection to cause[13] single-event upset (SEU) in configuration memory or internal logic registers to simulate transient errors that would be seen in radiation-sensitive conditions. Furthermore, the memory corruption emulation methods aim at the memory regions (RAM or Flash) in order to determine the strength of the ECC and[14] memory scrubbing algorithms. [6] The methods provide fidelity and observe fault in finding the fault propagation, and they are able to assess fault-tolerant mechanisms in realistic conditions.

Compliance with the safety regulations

Safety regulations have to be adhered to in industries like automobile, airline, and industrial automation. Automotive Functional Safety standards like ISO 26262, Design Assurance of Airborne Electronic Hardware like DO-254, and Functional Safety of Electrical/Electronic/Programmable Systems like IEC 61508 offer a stricter set of rules on faults classification, risk analysis,[15] and risk mitigation strategies. These standards pay increased attention to traceability, redundancy and diagnostic coverage diagnostic their limits, and validation into fault conditions and there is a strong suggestion that any HFI in the field can be utilized to assist with the verification of the design.

On the whole, the field of fault tolerance and modeling methods has already been widely researched via previous research papers, yet this area still lacks an appropriate combination of quantitative reliability research and real-time fault injection tests. A scalable, hardware-validated methodology able to currently satisfy the gaps is proposed in the following Table 1 this paper with the aim at early-stage certification, design optimization, and reliability enhancement.

System Architecture

Target Platform for Embedded

The injection target embodied platform is where the safety stable application is going to be executed and is the focal point of the fault injection framework. This framework is chosen by its applicability to the real-life implementation of a framework working in the automotive system, industrial control, and medical instrumentation.

Table 1: Summary of Key Literature Contributions in Fault Tolerance and Reliability

Area	Technique/Standard	Key Features	Strengths	Limitations
Fault Tolerance	Triple Modular Redundancy (TMR)	Replication with majority voting	High reliability, used in aerospace	High area/power overhead
	Error Correction Codes (ECC)	Hamming, BCH, SEC-DED	Effective memory error correction	Not suitable for complex logic faults
	Watchdog Timers	Time-out based reset logic	Simple, widely adopted	May not detect logical faults
	Fail-Safe States	System resets to safe mode	Ensures fail-operational behavior	Requires extensive design integration
Reliability Modeling	Markov Chains	Probabilistic state transitions	Captures failure behavior over time	Complex for large systems
	FMEA / FTA	Failure path identification	Qualitative insight, supports ISO 26262	Requires expert knowledge, subjective
	Weibull / Poisson Models	Quantitative MTBF and FIT estimation	Standard in reliability engineering	Assumes constant failure rate
Hardware Fault Injection	Pin-Level Injection	I/O manipulation (glitches, pulses)	Simple hardware setup	Limited fault types and observability
	Bit-Flip Injection on FPGA	Configuration bitstream modification	Emulates SEUs realistically	Requires FPGA design access
	Memory Corruption Emulation	SRAM/Flash bit manipulation	Validates ECC & scrubbing	Risk of unrecoverable corruption

To show the applicability and scalability of the proposed framework, two categories of embedded platforms are utilized in the given study:

STM32F407VG Microcontroller: An STM32 Microcontroller based on ARM Cortex-M4 core, yet still a resource-constrained platform in use in many applications, and typically used where price is also critical, such as industrial control loops, ventilators and automotive sensor modules. It is equipped with AMD integrating peripherals (ADC, PWM, UART, etc.), fault-tolerant clock domains as well as low-power modes that are essential when working on timing-sensitive applications. Fault injection through its debug interface or GPIOs and viewing its behavior in real time is done through serial logging.

Xilinx Zynq-7000 SoC: A compilation or hybrid platform that comprises an ARM Cortex-A9 processing system (PS) and programmable logic (PL), this platform fulfills the tasks of providing custom fault-tolerant logic interfaces, real-time fault insertion systems and application-specific accelerators to design. The PL area is sole exploited to conduct partial reconfiguration modules and specification fault injection circuits, whereas the PS bullets the primary application and recovery logic. Precise fault control and observation are possible with a low latency in the Zynq platform given its high flexibility.

The two platforms are chosen to illustrate the fault injection in a variety of hardware environments MCU-based sequential T2 execution environment and FPGA-based parallel frameworks, and this allowed the conduct of a full evaluation of the reliability framework.

FIC

The Fault Injection Controller is the main element that provides fault injection into the target embedded system in a systematic way. Its main purpose is to mimic realistic fault conditions, both temporary and permanent, and with no change to the functional logic of the application itself. The controller consists of hardware and software units that are tightly integrated in order to configure faults, initiate bug injection and monitor their execution.

Software Module: The fault injection software will operate on either an auxiliary microcontroller, or a host PC. It enables users to specify faults parameters which are: injection type: stuck-at, bit-flip, open/short circuit; location: register, memory, I/O; time: periodic, random, and event-triggered; and duration. The embedded platform is communicated through a command interface over UART or JTAG.

Hardware Module: In microcontroller systems, the cause of faults are injected as GPIO driven voltage spikes, commanded writes to memory at a specific address, or the hardware debugging interface. LUTs, flip-flops, block RAM contents can be changed in a hardware module in an FPGA based system through custom HDL based injectors, or through Xilinx Partial Reconfiguration (PR). Moreover, the fault can also be injected through high speed digital toggling or pulse generators to perform pin level fault injection.

Injection Modes:

To model real life fault scenarios, the fault injection controller has three modes of operation. Periodic faults are injected by the time-triggered mode to test the system to assess its susceptibility to the periodic interruptions with predefined frequency of interruptions. The event-based mode allows assessments under realistic runtime conditions and provides the possibility to inject faults on the basis of particular events occurring within or external to the system (e.g., on the basis of an interrupt

Table 2: Comparative Overview of Target Embedded Platforms

Feature	STM32F407VG (MCU)	Xilinx Zynq-7000 SoC (FPGA + PS)
Core Type	ARM Cortex-M4 (single core)	Dual-core ARM Cortex-A9 with programmable logic
Architecture Type	Sequential (MCU-based)	Parallel and reconfigurable (SoC)
Clock Speed	Up to 168 MHz	Up to 1 GHz (PS) + custom PL clock
Memory	1 MB Flash, 192 KB SRAM	On-chip BRAM + DDR3 via PL & PS
Fault Injection Support	GPIO manipulation, register overwrite	Partial reconfiguration, bitstream fault injection
Real-Time Monitoring	UART/USB logs	On-chip ILA, UART, JTAG
Typical Applications	Ventilators, automotive sensors, control loops	Autonomous braking systems, real-time fault emulation
Use Case Focus	Cost-sensitive, lightweight deployments	High-performance fault-tolerant applications
Flexibility for Fault Modeling	Moderate (fixed logic)	High (dynamic PL reconfigurability)

or a sensor trigger). Finally, the randomized mode is an approximation of unknown faults due to statistical models (e.g., Poisson or exponential) and are useful in modeling flaws that are temporary such as soft faults produced by radiation in the aerospace or high-altitude world. These multimodal injection abilities allow a thorough fault coverage and strong reliability assessment (Fig. 2).

The controller achieves accurate spatial and temporal control of fault injections that can be used to conduct repeatable experiments and analysis of their system response in finer detail with fault injection.

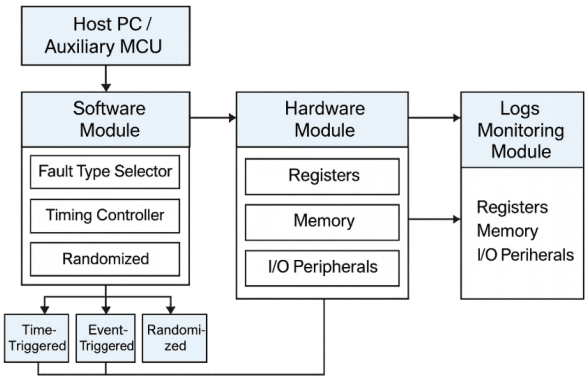

Fig. 2: Block Diagram of the Fault Injection Controller Architecture for Safety-Critical Embedded Systems

METHODOLOGY

Test-retest analysis

Reliability analysis forms a basic part of evaluating the dependability of safety-critical embedded systems so far as there is fault-prone operating condition. It entails both the evaluation of failure probabilities in quantitative terms and modeling of system behaviour in fault cases in qualitative terms. Here, the essential reliability measures and modeling strategies that the suggested frame embraces to describe and estimate system soundness are summarized.

Measurements of Quantitative Indices Reliability

There are three major quantitative indicators to deal with:

Failure Rate (lambda): The expected number of failures that occur in an hour in a component or a system is represented by this metric which is denoted in failures per hour. In embedded systems it is common to get 0 (0) from empirical testing or reliability data supplied by the vendor. It is important in the determination of operation safety during a certain mission time.

Mean Time between Failures (MTBF): This is the opposite of the first part factor, that is; the failure rate (MTBF = 1/lambda); this figure is the average operational time between two consecutive failures. When addressing embedded systems, a high MTBF will imply that the design will work well with minimal disruption that is crucial in systems such as autonomous braking units or even life-support units.

Failures in Time (FIT): If n is the number of expected failures per billion (10 9) hours of operation FIT is a typical measure used in the analysis of the reliability of semiconductors. Take an example of table 3 where a component having 50 FITs, is projected to fail 50 times in a billion hours of use. FIT is also very effective during processors, memory module, and ASIC reliability benchmarking where such are used on an embedded platform.

These measurements are used as the foundation to compare quantitatively among the several configuration of the hardware and implementations of a fault-tolerant design in the present study.

Quall Mel Modelling

In order to supplement numerical measurements, qualitative reliability modeling is done, with the following:

Reliability Block Diagrams (RBDs): In these diagrams sections of the system are modeled in terms of blocks which are connected to each other and where each block represents some sub system or a component of the system with a reliability value. RBDs are employed to determine the system reliability and the series, parallel as well as the hybrid arrangements. As an

Table 3: Quantitative Reliability Metrics

Metric	Definition	Units	Application in Study
Failure Rate (λ)	Expected number of failures per time unit	Failures/hour	Derived from empirical/standard data
MTBF (Mean Time Between Failures)	Inverse of failure rate (MTBF = 1/λ)	Hours	Indicates system longevity and robustness
FIT (Failures in Time)	Failures per 10^9 hours of operation	FITs	Used for semiconductor reliability benchmarks

illustration, a triple modular redundancy (TMR) logic block may be modeled as a parallel device with a voting scheme, which enhances the reliability of the whole system, as contrasted with its single-path analog.

Fault Tree Analysis (FTA): FTA is a deductive approach to failure analysis which begins with a pre-selected place of system failure (top event) and derives all the bombination of component failures (basic events) that can result in the top event. The fault propagation paths are modelled with logic gates, i.e. AND, OR, or k-out-of-n. In this paper, the FTA is used to model the scenario of failure as an incident like the loss of braking signal or the timing error in the ventilator, by which critical fault points can be discovered to be mitigated.

The combination of these approaches allows us to have an overall reply regarding the failure behavior of the system. As the presented reliability framework was designed by mixing both numeric estimation of reliability based on MTBF and FIT and structural analysis based on RBD and FTA, such an approach provides both predictions and analysis. Figure 3 This two-fold method makes it easy to identify components which are at danger early in the design process, assisting the optimization of the design and also makes sure that the system is functional as stated in international safety standards like ISO 26262 and IEC 61508.

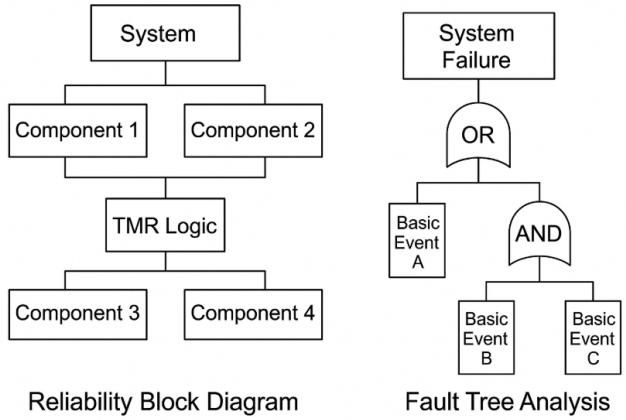

Fig. 3: Qualitative Reliability Modeling Using Reliability Block Diagram (RBD) and Fault Tree Analysis (FTA)

Hardware Fault Injection Strategy

A Hardware Fault Injection (HFI) Hardware Fault Injection (HFI) technique is an effective method of validation that is adopted to evaluate the fault tolerance of embedded systems in a real but controlled fault environment. Embedding faults in the hardware elements of the system commonly works as the fundamental strategy of HFI that is executed artificially to simulate erratic behavior of operations that could arise in multiple system operational anomalies with reference to the environmental influences, manufacturing flaws, or richness effects. The suggested fault injection framework uses a methodical method that is typified as injection types, injection points and trigger mechanisms so as to make sure complete fault coverage.

Injection Types

Stuck-at Faults (SA0, SA1)

These are permanent logic level faults in which a digital line is permanently set at logic 0 (SA0) or logic 1 (SA1) irrespective of whether or not they are expected or are intended to be used in a computation. They normally mimic things such as broken wires or short circuits. They can be introduced at different nodes on the datapath, within control logic, or in connections between memory to evaluate the efficiency in signal flow and system dynamics.

Transient Faults

The faults are temporary disturbances, due to environmental factors (electromagnetic interference (EMI), electrostatic discharge (ESD), or radiation (e.g., cosmic rays in aerospace). They usually appear in the form of single-event upsets (SEUs) in flip-flops or memory. Transient faults play an important role in exercising the soundness of real-time control systems since they may unpredictably corrupt control flow, or data values.

Fault/Usable Faults

Bridging faults are the electrical connection of two signal lines that are supposed to be electrically isolated causing logic contention or functionality errors. Open faults are faults in which the signal path has been destroyed or incomplete because of poor interconnects. Bus lines or peripheral interface to injection these are the kind of faults that are injected to test their error detection capability and fallback routine.

Injection Points

Processor Registers

Manipulating internal registers like Program Counter (PC), Status Register (SR), or General Purpose Registers (GPRs) allows an assessment of the impact of logic corruption on the control flow and the data processing. Faults at the register level assist in demonstrating execution pipeline and exception vulnerability.

SRAM, Flash (Memory Bits)

Injection of faults into SRAM or flash emulates bit-flips and corruption of data, both within the instruction and the data volumes. Such faults are fatal to systems where the systems depend on either a static system configuration or on the sensor calibration data in non-volatile memory. The usefulness of error detecting codes (EDC/ECC) and memory scrubbing routines can be confirmed by this type of injection.

I/O Peripherals

The response of the system to external disturbance or error condition in sensors is tested using faults inserted into I/O interfaces (e.g. SPI, UART, and ADC). This is more applicable in the case of applications such as the ventilators or autonomous vehicles where sensor-actuator feedback is crucial.

Mechanisms of Perturbation of Faults

Time-Based (Intermittent injection)

The introduction of faults is controlled with step time intervals and this can be repeated to obtain repeatable experiments that can measure the system behavior over similar fault patterns. This mode is applicable in debugging watchdog timers, the periodic diagnostics, and the scheduled failsafe processes.

Event-Based Injection

The faults are induced as a reaction to a system event like interrupt servicing, mode changing or crossing a sensor-threshold. This closely follows the working trend of the real world and it tests the fault tolerance of the system in serious transitions.

Poisson-Distributed Randomized Injection

To accommodate the probabilistic behavior of fault occurrences in the real world, the faults are activated in random temporal frequency following a statistical mathematical distribution such as Poisson or exponential. Figure 4 this approach is necessary to check long-term reliability and efficiency of anomaly detecting algorithms in uncertain operating conditions.

Experimental Setup

Experimental prototyping of the postulated hardware fault injection platform was carried out on two offline representative embedded systems: Xilinx Artix-7 FPGA and STM32F407 microcontroller (ARM Cortex-M4). These platforms were chosen because of their relevance to safety-critical deployments in the field and also in that

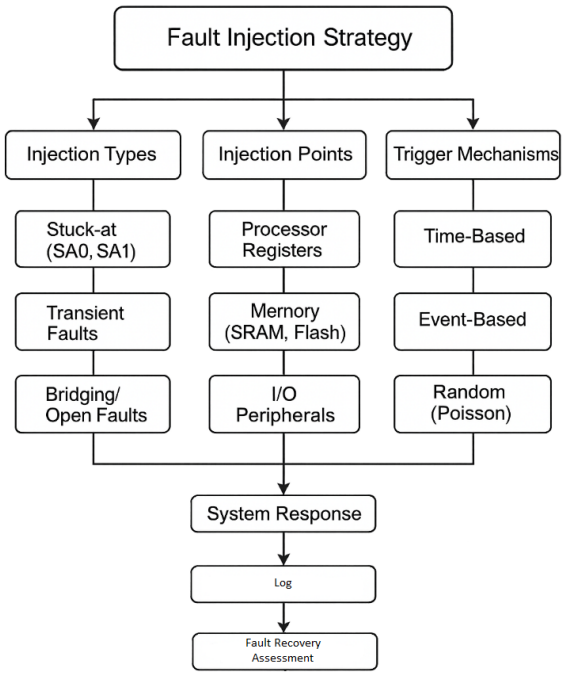

Fig. 4: Categorized Hardware Fault Injection Strategy - Types, Targets, and Trigger Mechanisms for Embedded Reliability Evaluation

such a platform was used to exhibit such scalability of the framework in both programmable logic and fixed-function hardware architectures. The following two benchmark applications were synthesized to assess the behavior of a system under the conditions of faults: (1) a logic that implements an autonomous emergency braking system, which models the processing of signal inputs in real-time in a vehicle, decision-making, and control of actuators and (2) a control system that ventilates and allows closed-loop airflow control via ECG and respiratory signals. The custom-made toolchain was implemented to inject faults. In the case of the FPGA, a bitstream-level fault injector was developed to induce bit-flips in logic blocks, registers, and memory elements through partial reconfiguration and model hardware transient and permanent faults like SEUs and stuck-at faults. On the microcontroller side, the automated GPIO-based peripheral disturbance was applied, including the toggling of control lines and register overwrites through JTAG/SWD interfaces, simulating circumstances like open connections or gaps in the peripheral or bridge connections. Figure 5. Both of these platforms had onboard monitoring modules that recorded system status, recovery actions and fault metadata in real time over UART/USB-based communication channels. These logs also assisted in conducting the post-injections analysis in order to determine the rate of fault recovery (FRR), the mean time to recovery (MTTR), and the total downtime of the system. The approach lays out

a high fidelity and realistic environment simulating the operating conditions to assess the soundness of fault tolerant designing attributes and authenticity of embedded systems in mission-critical applications.

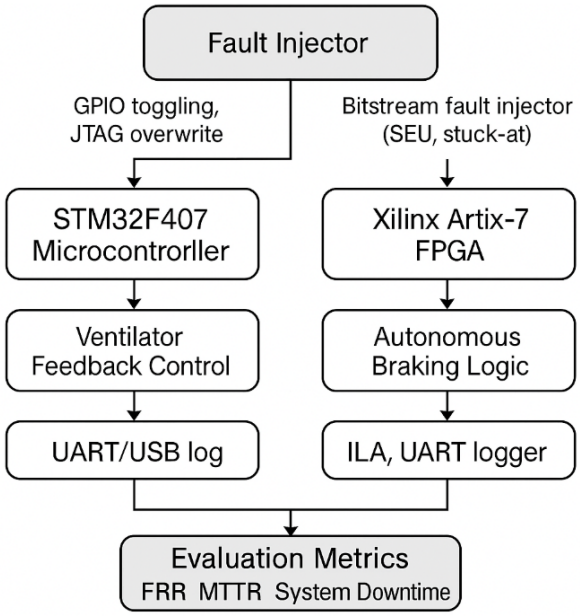

Fig. 5: Experimental Setup for Fault Injection and Monitoring on FPGA and Microcontroller Platforms

Results and Analysis

In a bid to assess the effectiveness and robustness of the suggested fault injection framework, a set of experiments were carried out with three phenomenal fault scenarios on two of the safety-critical embedded systems: an autonomous emergency braking system and ventilator control architecture. The key metrics measured were the detection of faults, downtime of the system, feasibility of recovery mechanism and safety effect. Table 1 provides the results of the system, and an impressive Fault Recovery Rate (FRR) is observed as 94.7%, which means an efficient injection and tolerance of the introduced faults. The max brakes-stuck-at-1 fault detection time was 5.4 ms and a redundant channel switch was able to recover which had a low safety impact and the system had a modesty system downtime of 7.8 ms. The watchdog timer issued a bit-flip in the control register that tripped after 2.1 ms rebooting the system, restoring functionality after 11.2 ms; it was quite effective but has a medium degree of safety because some component has lost control. Memory corruption in SRAM was the most gracefully addressed defect and an internal ECC-style mechanism identified and fixed the bug with minimal system overhead deeply- literally- no effect is seen in the performance; the error is recovered in a command and half, 0.5 ms.

The overall Mean Time to Recovery (MTTR) was calculated to be 8.3 ms, proving that the system is presented with a limitation regarding real-time functionality even in case of fault conditions. In addition to this, inclusion of fault detection logic provided area overhead of 11.2% and power overhead of 6.5% which are satisfactory considering the limitations of a safety-sensitive embedded system design. These findings confirm the compromise of enhanced system observability and low cost of resources. Notably, the experimental results support the efficacy of hybrid recovery mechanisms-redundancy, watchdog timers and ECC, when applied towards providing graceful degradation, or recovery in the event of hardware faults. The data also shows in figure 6 that different types of fault apply different effects to the system and the need to have fault tolerant strategies that are application specific. In Table 4 overall, the data proved the feasibility of the proposed HFI framework to serve as a pre-certification method to test the reliability of an embedded system in real and faulty operating conditions.

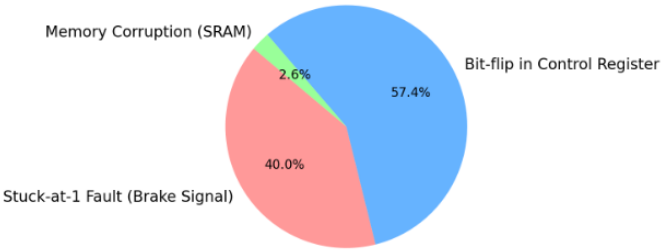

Fig. 6: System Downtime Distribution across Fault Scenarios in Safety-Critical Embedded Applications

Table 4: Summary of Experimental Results for Injected Fault Scenarios in Safety-Critical Embedded Systems

Fault Type	Detection Time (ms)	Recovery Mechanism	System Downtime (ms)	Safety Impact
Stuck-at-1 on Brake Signal	5.4	Redundant Channel Switch	7.8	Low
Bit-flip in Control Register	2.1	Watchdog Reset	11.2	Medium
Memory Corruption (SRAM)	8.6	ECC Correction	0.5	None

Discussion

The fault injection experiment outcomes can be of great help to determine how safety-critical embedded systems behave and how resilient they are, under different fault conditions. It is worth noting that the suggested Hardware Fault Injection (HFI) architecture is successful in revealing the vulnerabilities in the redundancy feature of the systems, especially when the system is used in timing-sensitive workloads. The scenarios demonstrated that failover logic and redundant paths might not be fast enough when it was being used in tight control loops, and this could cause critical delays, albeit only of a short duration. Of the tested types of faults, bit-flip faults in control registers turned out to have the most severe fault propagation severity, directly interrupting the execution of instructions, and control logic itself in many cases necessitating a full system reset to get it back into a stable state. By contrast, faults involving memory, particularly SRAM, were also addressed with great efficiency using incorporated Error Correction Codes (ECC), and these automatically detected and corrected single-bit errors, leading to recovery times of 1 ms or less and undisrupted functionality. Nevertheless the fault coverage provided by such a flawless coverage imposes a quantifiable architectural penalty upon the design: the use of this detection and recovery circuits creates a moderate area (~11.2%) and power (~6.5%) cost. Figure 7 Even though such overhead can be tolerated by high-assurance applications, the concern that fault resilience should be balanced with the constraint on the resources available to a system is being highlighted. On the whole, the discussion makes stronger the idea that fault injection proves to be not a mere diagnostic tool but also a key that enables to optimize the design of embedded safety systems.

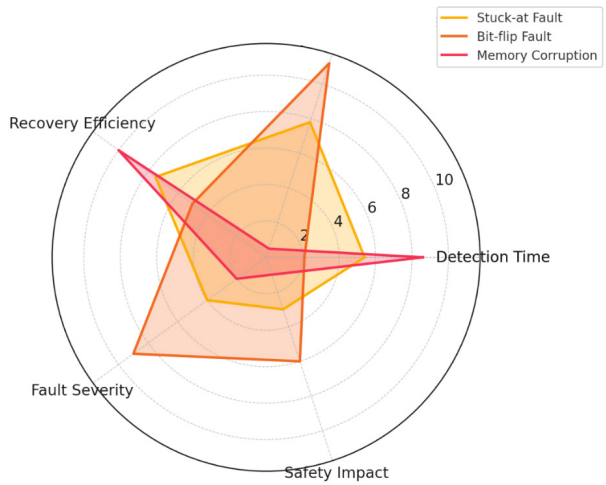

Fig. 7: Comparative Radar Analysis of Fault Types Based on Recovery and Impact Metrics

Conclusion

This paper describes an experimentally verified fault injection and reliability analysis process that is applicable to safety-critical systems embedded in the system. Quantitative reliability measurements the use of real-time fault injection paradigms of hardware faults together with structured monitoring and quantitative reliability measures, therefore, provides an effective way of studying system robustness given realistic fault conditions. The fact that it supports a dual-platform (a combination of STM32 microcontrollers and Xilinx FPGAs) version demonstrates that the framework is universal and may be applied to various embedded systems as well. As illustrated in experimental results, there are also high fault recovery rates, quick detection time, and effective mitigation measures by use of features like watchdog resets, redundancy, and ECC, hence a demonstration of the effectiveness of the design through different types of faults like stuck-at, bit-flip and memory corruption faults. Notably, the analysis points out not only the advantages of the common fault-tolerant features but also their drawbacks, especially when timing constraints make both control register corruptions and fault tolerance responses particularly vulnerabilities in timing workloads. The trade-offs are also recognized in the study, most prominent of them being the additional cost of area and power overhead needed to actually install the extensive fault detection and recovery mechanisms. The proposed framework should align with other relevant industry standards like the standards ISO 26262 and IEC 61508, thus it could be deployed in various fields including automotive safety, medical devices, industrial automation, aerospace, and space systems, and pre-certification. Altogether, this paper has built a strong base towards the incorporation of hardware-verified fault tolerance in the design of embedded systems that will lead to development of more resilient and certifiable next generation embedded applications.

References

1. Pradhan, D. K. (1986). *Fault-tolerant computing: Theory and techniques* (Vol. 1). Prentice-Hall.
2. von Neumann, J. (1956). Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C. E. Shannon & J. McCarthy (Eds.), *Automata Studies* (pp. 43-98). Princeton University Press.
3. Rausand, M., &Høyland, A. (2004). *System reliability theory: Models, statistical methods, and applications* (2nd ed.). Wiley-Interscience.
4. Andrews, J. D., & Moss, T. R. (2002). *Reliability and risk assessment*. Longman Scientific & Technical.

5. Reorda, M. S., Sterpone, L., &Violante, M. (2008). Fault injection techniques for SRAM-based FPGAs. *ACM Transactions on Design Automation of Electronic Systems*, 13(1), 1–23. https://doi.org/10.1145/1297666.1297668
6. Madeira, H., Costa, D., & Ferreira, R. R. (1999). Injection of faults using software tools: Practical experience in real-time systems. *IEEE Transactions on Reliability*, 48(4), 337–346. https://doi.org/10.1109/24.809567
7. Laprie, J. C., Arlat, J., Beounes, C., &Kanoun, K. (1990). Definition and analysis of hardware-and software-fault-tolerant architectures. *Computer*, 23(7), 39–51. https://doi.org/10.1109/2.56830
8. Avizienis, A., Laprie, J. C., Randell, B., &Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1), 11–33. https://doi.org/10.1109/TDSC.2004.2
9. Salfner, F., Lenk, M., &Malek, M. (2010). A survey of online failure prediction methods. *ACM Computing Surveys (CSUR)*, 42(3), 1–42. https://doi.org/10.1145/1670679.1670680
10. Dhir, A., &Sarje, A. K. (2003). Modeling and evaluation of reliability and availability in fault-tolerant systems using Petri nets. *Microelectronics Reliability*, 43(9–11), 1483–1491. https://doi.org/10.1016/S0026-2714(03)00163-3
11. Kavitha, M. (2025). Hybrid AI-mathematical modeling approach for predictive maintenance in rotating machinery systems. Journal of Applied Mathematical Models in Engineering, 1(1), 1–8.
12. Surendar, A. (2025). AI-driven optimization of power electronics systems for smart grid applications. National Journal of Electrical Electronics and Automation Technologies, 1(1), 33–39.
13. Madhanraj. (2025). Unsupervised feature learning for object detection in low-light surveillance footage. National Journal of Signal and Image Processing, 1(1), 34–43.
14. Arvinth, N. (2024). Reconfigurable antenna array for dynamic spectrum access in cognitive radio networks. National Journal of RF Circuits and Wireless Systems, 1(2), 1–6.
15. Mäkinen, R. (2024). The Role of Digital Twins in Improving Business Processes and Quality Management. National Journal of Quality, Innovation, and Business Excellence, 1(2), 23-29.