Proceedings of CONF-MLA 2026 Symposium: Intelligent Systems and Automation: AI Models, IoT, and Robotic Algorithms
DOI: 10.54254/2755-2721/2026.LD38773

A Review of Fault Tolerance and Fault Handling Technologies for High-Reliability Industrial Embedded Systems

Lanxin Zhu

Wuhan Guanshan Middle School, Wuhan, China
2907360887@qq.com

Abstract. Industrial embedded systems operate in safety-critical domains including aerospace, automotive, rail transit, and medical devices, where faults can cause catastrophic failure or loss of life. Under demanding environments and strict functional safety standards, achieving high reliability remains a fundamental challenge. This paper presents a systematic review of fault-tolerance and fault-handling techniques for such systems. We begin by clarifying reliability requirements and distinguishing systematic faults from random hardware faults. Next, we analyze three redundancy mechanisms: hardware, software, and time. We then survey fault detection and diagnosis methods, including watchdogs, consistency checks, EDAC, control-flow monitoring, model-based observers, signal feature extraction, and AI-driven diagnosis. A graded recovery strategy follows: fault masking, failover, system degradation, and safe shutdown—which constitutes a defense-in-depth architecture. Finally, two validated case studies are presented: a triplex flight control computer (avionics) and an ISO 26262-compliant automotive ECU. The review shows that modern fault tolerance relies not on isolated techniques, but on coordinated, multi-layered engineering. Future trends, including AI-driven predictive diagnosis, adaptive redundancy, formal verification, and RISC-V-native fault-resilient hardware, point toward more robust, verifiable, and efficient future systems.

Keywords: high-reliability industrial embedded systems, fault-tolerance mechanisms, fault handling, redundancy mechanisms

1. Introduction

Embedded systems—tight integrations of hardware and software tailored to specific applications—are foundational in safety-critical industries: aerospace, automotive electronics, rail transportation, and medical devices. Because they perform essential monitoring, control, and actuation functions, failures can cause equipment damage, operational disruption, or injury and loss of life. Ensuring their continuous, reliable operation—under harsh environments and despite faults—is therefore a core challenge for industrial control and functional safety. Unlike general-purpose computers, industrial embedded systems operate under extreme conditions (e.g., electromagnetic interference, thermal cycling, vibration, radiation) and must meet stringent, long-lifecycle safety standards—

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/2026.LD38773

including IEC 61508, ISO 26262, and DO-178C. Relying solely on higher-quality components cannot eliminate failure risk, instead, fault tolerance—built through deliberate redundancy and robust fault handling—is essential [1, 2].

Fault tolerance fundamentally uses redundancy to maintain functionality despite faults—a principle formalized by von Neumann's early work on fault-resilient computation [3]. Modern designs go beyond simple fault masking to encompass the full fault lifecycle: detection, diagnosis, isolation, recovery, and safe degradation. This paper reviews fault tolerance and fault handling in high-reliability industrial embedded systems. We systematically analyze hardware, software, and temporal redundancy, key techniques including watchdogs, consistency checks, EDAC, and model-based diagnosis, and hierarchical recovery strategies—fault masking, failover, graceful degradation, and safe shutdown. Case studies from avionics and automotive electronics validate these methods, and we conclude with emerging trends in intelligent, adaptive fault tolerance.

2. Reliability requirements and fault classification for industrial embedded systems

2.1. Functional safety standard requirements

EC 61508 serves as the "parent standard" for functional safety and defines four safety integrity levels (SIL 1 to SIL 4) [1]. The requirements for the hazardous failure probability for each level are shown in Table 1. Among these, SIL 3 and SIL 4 are commonly used in scenarios with the highest safety requirements, such as nuclear power plants and rail transit systems.

Table 1. SIL level and hazardous failure probability requirements

SIL degree	Probability of dangerous failure (unit: per hour)
SIL 1	$10^{-}5 \sim 10^{-}4$
SIL 2	$10^{-}6 \sim 10^{-}5$
SIL 3	$10^{-}7 \sim 10^{-}6$
SIL 4	$10^{-}8 \sim 10^{-}7$

The automotive industry adopts the ISO 26262 standard, which defines four safety integrity levels ranging from ASIL A to ASIL D [2]. The stringent requirements for the highest level, ASIL D, are as follows: Single-Point Failure Measure (SPFM) ≥ 99%, Latent Failure Measure (LFM) ≥ 90%, and Probability of Hardware Failure (PMHF) <10-8 per hour. These metrics serve as an important basis for designing system redundancy, implementing fault diagnosis, and configuring safety mechanisms. These standards not only define Safety Integrity Levels (SIL) and reliability metrics but also mandate that the entire development lifecycle must commence with Hazard Analysis and Risk Assessment (HARA), be interspersed with analytical techniques such as FMEA and FTA, and incorporate defensive programming, MISRA C, and testing requirements to achieve 100% MC/DC coverage. Consequently, fault-tolerance mechanisms should not be treated as remedial measures only triggered after a system failure occurs, rather, they should be comprehensively planned during the system requirements, architectural design, and development verification phases.

2.2. Fault classification and failure mechanisms

From various perspectives, faults in industrial embedded systems can be classified according to their fault source, formation mechanism, and duration.

DOI: 10.54254/2755-2721/2026.LD38773

From the perspective of fault origins, faults in industrial embedded systems primarily fall into two categories: hardware faults and software faults. Hardware faults include chip aging or damage, circuit short circuits or open circuits, bit flips induced by cosmic radiation or single-event effects, and transient anomalies caused by electromagnetic interference. Software faults encompass program logic errors, data overflow, illegal state machine transitions, and memory leaks.

IEC 61508 further distinguishes between systematic faults and random hardware faults: systematic faults stem from design defects and must be mitigated through rigorous development processes, random hardware faults, on the other hand, should be tolerated using redundancy and diagnostic coverage [1]. In practical engineering applications, these two types of faults often require different mitigation strategies: systematic faults are addressed through process-based controls (e.g., code review, static analysis, formal verification), whereas random hardware faults are handled via redundancy design (e.g., TMR, ECC, watchdog timers).

Additionally, faults can be classified by duration into transient faults (Temporary–e.g., power supply glitches or radiation-induced bit flips that can self-recover) and permanent faults (Permanent–e.g., chip burnout or circuit breakage that must be bypassed using redundant switching).

Therefore, the design of fault-tolerance mechanisms must take into account the characteristics of both types of faults simultaneously. However, the goal of fault tolerance is not to prevent the system from failing, but to ensure that the system still produces deterministic results even when faults occur. All subsequent discussions of relevant technologies will be centered around this principle.

3. Fault-tolerance core strategy: three major redundancy mechanisms.

Redundancy is an important approach for implementing fault tolerance in industrial embedded systems. Depending on the dimension of additional resources involved, redundancy can be classified into three main categories: hardware redundancy, software redundancy, and temporal redundancy [4, 5]. These three types of redundancy correspond to resource allocation in the spatial, design, and temporal dimensions, respectively, they can be employed independently or combined to form a hybrid redundancy architecture.

3.1. Hardware redundancy: using the space to obtain the reliability

Hardware redundancy achieves fault tolerance by physically replicating critical components, this is the most intuitive and most widely adopted fault-tolerance approach. Depending on the number of redundant modules and their operational status, typical architectures such as TMR, DMR, primary/backup redundancy, and N-mode redundancy can be implemented.

Three-Modular Redundancy (TMR): Three identical or functionally equivalent modules execute the same task in parallel, their outputs are fed into a majority voter. When at least two modules output the same value, the system adopts that majority result as its output. Under the assumption that the three modules are independent and the voter is reliable, if the reliability of a single module is R, then the reliability of the TMR system can be expressed as:

$$R_{TMR}=R^3+3R^2(1-R)$$

This model demonstrates that when R> 0.5, the reliability of an ideal TMR configuration is significantly superior to that of a single-module system [4]. The core advantage of TMR lies in its ability to transparently mitigate single-point failures, that is, even if any single module fails, the system still produces the correct output without requiring any switching time. Consequently, TMR is

DOI: 10.54254/2755-2721/2026.LD38773

often suitable for scenarios that demand continuous operation and high fault tolerance—such as avionics, industrial control systems, and safety-critical systems—where uninterrupted correct operation is essential.

In engineering practice, special attention should be paid to the fact that the voting device itself can represent a potential single-point failure, therefore, it is often necessary to implement redundancy measures for the voting device—for example, by interlocking three voting devices. Additionally, the three modules should be physically isolated as much as possible to mitigate the risk of simultaneous failure of multiple modules due to common-cause failures resulting from power supply anomalies, electromagnetic interference, or radiation.

Dual-Mode Redundancy (DMR) and versus Dual-Unit Hot Standby: Two identical modules operate in parallel, and their outputs are fed into a comparator. DMR can only detect inconsistent data outputs and trigger a switch or generate an alarm, however, it cannot directly correct errors.

This approach is typically suitable for scenarios such as server dual-unit hot standby or the failover between primary and backup CPUs in industrial PLCs, where the typical switch time requirement is <300 ms. Compared to TMR, DMR eliminates the need for an additional hardware set, but it sacrifices transparent fault tolerance, once the comparator detects a discrepancy between the results from the two units, the system must rely on upper-layer software or manual intervention to determine the next course of action.

Cold standby, Warm standby, and Hot standby: In Cold standby, the standby module is either powered off or idle, a full startup is required during the failover—this mode takes the longest time but consumes the least power, in Warm standby, the standby module has the program loaded but does not participate in actual computations—the failover is relatively fast, in Hot standby, the standby module operates in full synchronization with the primary module the failover is virtually imperceptible, but this mode incurs the highest cost. The choice of one of these three modes depends on the system's trade-off between failover time and resource budget.

N-module redundancy (NMR) is an extended form of TMR that extends to multi-module redundancy, it involves selecting N modules based on fault-tolerance requirements—typically, using an odd value for N provides fault-tolerance advantages (enabling majority voting). In the aerospace sector, five-module redundancy (5MR) is employed in extreme radiation environments (e.g., during missions to Jupiter) because it can tolerate the simultaneous failure of two modules.

3.2. Software redundancy: combating common-cause failures through diversity

Software redundancy is implemented by employing different algorithms or programming approaches to achieve the same functionality, its core principle is to leverage diversity and independence to mitigate systemic risks arising from common design flaws [1].

Typical techniques include: N-Version Programming (NVP), in which multiple versions of the same functionality are independently developed by different teams, executed concurrently, and then a voting mechanism is applied to make a decision— thereby mitigating the risk of common-cause failures from a diversity perspective, the Recovery Block approach, in which the main program is initially executed, if acceptance testing fails, the system rolls back to the previous checkpoint and then resumes execution of a backup program—this approach is well-suited for computationally intensive tasks, and diversified data representations and algorithms that employ different methods to implement the same functionality, such as hybrid floating-point and fixed-point arithmetic or cross-validation.

The challenge of software redundancy lies in the difficulty of ensuring "independence"—since different teams may be working with the same requirement documentation, which can lead to

DOI: 10.54254/2755-2721/2026.LD38773

ambiguity arising from shared misunderstandings. Therefore, in engineering practice, it is common to require each versioning team to use different programming languages, distinct algorithmic approaches, or even different compilers in order to maximize the benefits of diversity.

3.3. Time redundancy: trading time for reliability

Time redundancy reduces the impact of transient faults on system operation by repeatedly executing the same operation, recalculating data, or retransmitting data at different times, it is suitable for scenarios involving intermittent errors. Typical techniques include instruction retry (Retry), transaction rollback and redo (Rollback and Redo), and cyclic redundancy check (CRC) combined with retransmission, among others [5].

The core assumption is that transient faults are not persistent in time, therefore, repeatedly executing a task is highly likely to yield the correct result. The trade-off, however, is an increase in execution time: instruction retries cause the average execution time to become (1+p) times the original value (where p is the retry probability). Time redundancy is often combined with hardware/software redundancy to form a hybrid redundancy architecture, for example, the approach of "TMR + instruction retries" can simultaneously handle both permanent and transient faults.

4. Fault detection and diagnosis technology

A fault detection and diagnosis technology-based fault-tolerant system must first "detect" the presence of a fault, as undetected faults cannot be masked or repaired. The following are commonly used fault detection and diagnosis techniques in industrial embedded systems [5,6].

4.1. Watchdog timer

The watchdog timer is a commonly used hardware monitoring mechanism, it is essentially a hardware counter that operates independently or relatively independently of the CPU. During normal system operation, the counter must be reset periodically, if the watchdog is not fed before the timeout occurs—due to, for example, a program crash or a system freeze—the watchdog overflow will trigger the system's reset function or an Non-Maskable Interrupt (NMI).

Modern MCUs typically support various types of watchdog mechanisms. For example, the Independent Watchdog (IWDG) usually operates using an independent clock source and continues to function even when the main clock fails, it is primarily used to detect system freezes caused by hardware faults or electromagnetic interference. In contrast, the Window Watchdog (WWDG) requires the system to execute within a specified time window and is designed for precisely detecting software runtime compliance and logical anomalies.

In multitasking systems, it is common practice to design a high-priority monitoring task to oversee the "software watchdog" behavior of multiple monitored tasks, the failure of any single task will trigger a hardware watchdog timeout reset. This two-tiered "watchdog + task monitoring" architecture both detects hardware anomalies and covers software logic errors.

4.2. Consistency check

The consistency check rapidly identifies abnormal deviations during computation or execution by comparing the output results of redundant modules, enabling immediate fault detection. If the results of two modules are inconsistent, it indicates that at least one of the modules has encountered an

DOI: 10.54254/2755-2721/2026.LD38773

issue. This is the fundamental mechanism upon which redundant architectures such as TMR and DMR rely.

The implementation of the consistency check can be hardware-based (e.g., a comparator integrated within an FPGA) or software-based (e.g., result comparison in a dual-core lockstep architecture).

4.3. Error detection and correction code (EDAC/ECC)

EDAC technology is specifically designed to address bit-flip errors in memory systems and serves as the core component of fault-tolerant storage systems. During the data writing phase, parity codes (e.g., Hamming code, BCH code, Reed-Solomon code) are automatically generated and applied for synchronous storage, during the data reading phase, real-time parity checking is performed, enabling fully automated correction of all single-bit flip errors as well as precise detection of double-bit or higher-order multiple-bit errors, with immediate error reporting.

Different coding schemes offer varying error correction capabilities: Hamming codes can correct single-bit errors and detect double-bit errors, making them suitable for SRAM, BCH codes can correct multiple-bit burst errors, making them suitable for NAND Flash, Reed-Solomon codes provide the highest error correction capability and are commonly used in spaceborne solid-state memory systems. In aerospace, rail transit, and high-performance server applications, EDAC has virtually become a standard configuration.

4.4. Control flow detection

This monitoring program checks whether the program execution path deviates from the expected path and detects any software logic anomalies.

Common methods include:

Instrumentation during the compilation phase—inserting instrumentation points to verify at runtime whether the program counter has traversed the expected sequence of basic blocks,

(2) Utilizing specialized hardware (e.g., ARM's ETM tracing unit) to record the instruction stream in real time and compare it against an expected pattern. This approach can effectively detect program divergence scenarios caused by stack overflow, wild pointers, or other issues.

4.5. Model-based diagnosis

When a system possesses an accurate mathematical model, precise fault diagnosis and localization can be achieved by comparing the actual output with the expected output through state estimation and residual analysis [6]. Specific methods include: (1) State observer method: Design a Luenberger observer or a Kalman filter to estimate the system state, a fault is detected when the residual exceeds a predefined threshold, (2) Parameter estimation method: Perform online identification of system parameters, parameter drift indicates component aging or failure, (3) Odd-even equation method: Utilize the system's analytical redundancy relation to construct an odd-even vector for fault detection and isolation (FDI).

These methods offer high accuracy but require more precise models and are less suitable for complex nonlinear systems.

DOI: 10.54254/2755-2721/2026.LD38773

4.6. Signal-based diagnosis

When an accurate system model is unavailable, signal-based diagnostic analysis can be employed to examine characteristic variations in sensor signals (e.g., vibration, temperature, current, noise, etc.) and extract features from the time domain (mean, variance, skewness), the frequency domain (spectral density, envelope spectrum), and the time-frequency domain (wavelet transform, EMD) to identify faults [6]. This method does not require an exact system model and is suitable for complex nonlinear systems, however, it depends on a large amount of historical data to train the feature library.

4.7. Knowledge-based diagnosis

This approach relies more heavily on empirical methods, such as employing artificial intelligence techniques—including expert systems, neural networks, and support vector machines—for fault detection [6]. Expert systems are well-suited for scenarios where the fault patterns are known, in contrast, neural networks leverage a data-driven approach to automatically learn fault characteristics, making them ideal for handling complex nonlinear mapping relationships. In recent years, deep learning has achieved significant success in applications such as bearing fault diagnosis and battery State of Health (SOH) estimation.

5. Fault recovery and hierarchical handling strategy

Upon detecting a fault, the system should employ a hierarchical response strategy based on the fault severity and its recoverability. The team of Amel provided a comprehensive review that outlines the entire fault-tolerant lifecycle—from fault detection to recovery handling [7]. The core principle of this hierarchical strategy is to shield the system whenever possible, if shielding is not feasible, switch to an alternative mode, if switching is not possible, degrade the system, and if degradation is also not feasible, initiate a safe shutdown—following this step-by-step, progressive approach.

Fault Masking directly masks faults through redundant voting, ensuring that the system output remains unaffected by faults, it is suitable for transient faults. TMR is a typical example of a fault masking mechanism—a single-module failure is transparently masked via majority voting, without any perceptible impact on the system or the need for any software intervention. The advantage of Fault Masking is a zero response time (pure hardware logic), the disadvantage is that it incurs the highest hardware cost.

Fault Switching involves switching the faulty module to the standby module (primary-standby switching). For example, in a PLC primary—standby CPU switching scenario, the engineering requirement is that the switching time be less than 300 ms. And it must address three key issues: (1) State synchronization.The standby module must possess the same latest state as the primary module, otherwise, the system behavior will experience a discontinuity after the switchover, (2) Switching arbitration.When both the primary and standby modules determine that they are operating normally, a third-party arbitration mechanism or priority-based monitoring rules must be employed, (3) Switching consistency.Any transactions in progress during the switchover must be fully rolled back or committed to prevent data inconsistency.

System degradation involves disabling non-critical functions while maintaining the operation of core functions. The in-vehicle "Limp Home" mode employs this response strategy: when the Engine Control Unit detects failures in certain sensors, the system continues to operate at reduced performance levels, ensuring that the vehicle can still safely pull over—even under power-restricted

DOI: 10.54254/2755-2721/2026.LD38773

conditions—without experiencing sudden loss of control [5]. The key to designing this degradation strategy is that the core functions must be identified through HARA analysis to determine which functions are safety-critical and must be preserved under all circumstances.

Safe Shutdown: When a fault cannot be repaired, the system enters a safe mode—the final line of defense for fail-safe design, ensuring that the safety function is not compromised even in the event of a single-point failure. Examples include cutting off power supply, releasing pressure, stopping mechanical motion, or closing valves. The design of a safe shutdown mechanism must adhere to the "passive safety" principle: even if the control system fails completely, the safety mechanism should still activate through mechanical, hydraulic, or gravitational means—such as a brake that engages under gravity or an air valve that resets via a spring.

This four-tiered strategy forms a progressive defense system: ranging from "user-perceptible" fault isolation to "functionally restricted yet safe operation" (downgrading), and finally to a "fail-safe" fallback shutdown. Each tier has clearly defined triggering conditions and corresponding response actions, collectively constituting a layered defense approach.

Software recovery and checkpointing technology In addition to hardware-level recovery, software recovery techniques involve periodically pausing and restarting the software or clearing its internal state to prevent severe failures and mitigate software aging phenomena (such as the accumulation of memory fragmentation). This technology should be combined with checkpointing technology, which periodically saves critical system states to non-volatile storage, upon a failure, the system can then roll back to the most recent valid checkpoint and resume execution. The frequency of checkpointing requires careful trade-off: too frequent checkpoints incur significant system overhead, while too infrequent checkpoints result in excessive rework after a failure. Therefore, a compromise is typically made based on the real-time requirements of the application, resulting in a checkpointing interval ranging from dozens to hundreds of milliseconds.

6. Typical application case study

6.1. Avionics: a three-redundancy flight control computer system based on "Dual-Factor Monitoring"

Avionics systems for flight control computers impose extremely stringent reliability requirements, typically requiring a failure rate below 10-9 per hour. Traditional N-modular redundancy relies on a central voter, however, if this voter fails, the entire system will become inoperative, representing a single point of failure. Pan et al. proposed a design scheme for a three-redundancy airborne computer system, Cheng et al. further investigated fault-tolerance logic techniques for channels in redundant flight control computers [8, 9].

This case employs a triple-redundancy flight control computer fault-tolerant architecture based on "Dual-Factor Monitoring", its core innovation lies in eliminating the central voter in favor of a triple-redundancy architecture utilizing "Dual-Factor Monitoring." Each channel manages itself while simultaneously monitoring its neighbors. Specifically, this architecture comprises three monitoring factors:

Self-monitoring: Each channel utilizes built-in BIST (Built-In Self-Test) and voltage/temperature monitoring to independently verify whether its CPU, memory, and power supply are functioning correctly,

Cross-comparison: Channels compare their respective data through cross-linking mechanisms to detect output or interface faults, this cross-comparison employs bidirectional CRC verification to ensure data integrity,

DOI: 10.54254/2755-2721/2026.LD38773

(3) State sharing: The bottom data bus aggregates the health status of all channels, enabling global awareness, upon which each channel can independently make isolation decisions.

The operational logic of this architecture is as follows: when a channel experiences internal hardware failure (self-monitoring alarm) or detects inconsistency in its own data compared to the other two channels (cross-comparison alarm), that channel is automatically isolated. The remaining two healthy channels can still perform majority-voting to issue correct commands, ensuring that the aircraft continues to operate normally during a single fault and switches to emergency flight mode during a secondary fault. Compared to the traditional "four-redundancy" approach, this three-redundancy architecture achieves extremely high reliability while reducing hardware cost, power consumption, and weight by approximately 25%, offering a significant cost-performance advantage [8, 9]

6.2. Automotive electronics: ECU fault-tolerant design

Automotive electronic systems (e.g., Engine Control Unit (ECU), Anti-lock Braking System (ABS), Electronic Power Steering (EPS)) must comply with the ISO 26262 Functional Safety standard and achieve safety levels ranging from ASIL B to ASIL D [2]. Key aspects of the fault-tolerant design include: Sensor redundancy – employing multiple sensors (e.g., wheel speed sensors, acceleration sensors) for cross-validation, when a single sensor fails, the system can continue operating based on the data from the remaining sensors, Actuator monitoring – monitoring actuator feedback signals to detect faults such as jamming, short circuits, or open circuits, Communication fault tolerance – utilizing bus protocols such as CAN FD or FlexRay, which incorporate CRC checking, retransmission mechanisms, and time-slot scheduling, Synchronized CPU architecture – the main CPU and the Checker CPU execute identical code, the failure protection unit compares the results from the two cores and can almost immediately detect and diagnose any discrepancy that may arise from random faults, the two cores execute code for 1.5-2 cycles each, and rotational and inversion techniques are employed to provide temporal and physical diversity, Comprehensive on-chip diagnostics – permanent monitoring of clock and voltage signals, ECC protection for all memory blocks, and instant fault detection via BIST logic.

7. Future development trends

Artificial Intelligence and Machine Learning – These technologies are further driving the transformation of industrial embedded systems from passive fault tolerance to active fault prediction. By analyzing system operational data through historical analysis systems, AI models can predict potential failures before they occur and trigger fault tolerance mechanisms in advance. Interdynamic system intermittent fault diagnosis techniques are evolving toward a data-driven, model-integrated approach. Typical applications include predicting the Remaining Useful Life (RUL) of bearings using LSTM/Transformer-based models, employing autoencoders for unsupervised anomaly detection to identify deviations without requiring labeled fault samples, and implementing adaptive fault tolerance strategy selection based on reinforcement learning to dynamically adjust redundancy levels according to system status.

Adaptive fault tolerance will become a key development direction for high-reliability industrial embedded systems. These systems can dynamically adjust their fault tolerance strategies based on the environment and the tasks at hand: they can enable full redundancy when resources are abundant and switch to a degraded mode when resources are scarce, thereby achieving a dynamic balance between system reliability and resource consumption. Additionally, such systems must possess

DOI: 10.54254/2755-2721/2026.LD38773

environmental sensing capabilities to assess their own health status, the severity of the external environment, and the current task's criticality in real time, and based on this assessment, make informed decisions regarding their fault tolerance strategy.

Furthermore, formal methods can be employed to rigorously verify the correctness of fault-tolerance mechanisms, providing a mathematical proof that the system can still maintain a safe state even under specific fault scenarios. Formal verification techniques include Model Checking, Theorem Proving, and Abstract Interpretation, among others. These methods are particularly important for systems operating at the highest safety levels, such as ASIL D or SIL 4 [1, 2].The DO-178C standard explicitly lists formal methods as a recommended verification approach for software with high assurance levels.

With advancements in semiconductor manufacturing processes, security mechanisms such as lock-step CPUs, on-chip ECC, and hardware BIST are being integrated into MCUs (for example, the TI TMS570 series has received ASIL D and SIL 3 certifications), significantly lowering the barrier to implementing highly reliable fault-tolerant systems. Additionally, the customizable nature of the RISC-V architecture opens up new possibilities for fault-tolerant design—enabling the addition of redundant instructions or verification instructions at the instruction set level, thereby providing fault tolerance support at the underlying hardware level.

8. Conclusion

Fault tolerance in high-reliability industrial embedded systems is a multi-layered, integrated engineering discipline—built on three redundancy pillars: hardware (e.g., TMR, dual hot standby), software (e.g., N-version programming, recovery blocks), and temporal (e.g., instruction retry, transaction redo). Complementing these are robust fault detection and diagnosis techniques—including watchdogs, consistency checks, EDAC, control-flow monitoring, and model-based observers—that cover the full system lifecycle. Recovery follows a tiered strategy: fault masking, failover, graceful degradation, and safe shutdown, enabling defense-in-depth.

In aerospace, triplex flight control computers implement "Dual-Factor Monitoring" to replace centralized voting units—eliminating single-point failures while cutting power use and hardware complexity, all without compromising ASIL D reliability. In automotive electronics, ASIL D–compliant ECUs integrate sensor redundancy, lock-step CPUs, on-chip BIST, and ISO 26262–aligned diagnostics—proving fault tolerance has matured from theory into reusable, production-ready engineering.

Looking ahead, AI shifts fault tolerance from reactive to predictive: time-series anomaly detection models can spot early degradation signs in operational data, enabling preventive reconfiguration. Adaptive mechanisms dynamically allocate redundancy based on safety-criticality, environmental stress, and equipment health—optimizing reliability against computational and energy cost. Meanwhile, RISC-V's open ecosystem accelerates development of custom fault-resilient IP cores, lightweight formal verification tools, and hardware-assisted diagnostics. Ultimately, fault tolerance is not just a compliance requirement—it is the foundational enabler of secure, intrinsically safe critical infrastructure. Mastery of its principles, constraints, and frontiers is essential for moving beyond certification to genuine resilience.

References

[1] IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems [S].
[2] ISO 26262: Road vehicles—Functional safety [S].

DOI: 10.54254/2755-2721/2026.LD38773

[3] von Neumann, J. (1956). *Probabilistic logics and the synthesis of reliable organisms from unreliable components*.
[4] Lyons, R. E., & Vanderkulk, W. (1962). The use of triple-modular redundancy to improve computer reliability. *IBM Journal of Research and Development, 6*(2), 200–209.
[5] Zhou, D. H., Shi, J. T., & He, X. (2014). A survey of intermittent fault diagnosis techniques for dynamic systems. *Acta Automatica Sinica, 40*(2), 161–171.
[6] Wen, C. L., Lü, F. Y., Bao, Z. J., et al. (2016). A survey of data-driven methods for small fault diagnosis. *Acta Automatica Sinica, 42*(9), 1285-1299.
[7] Solouki, M. A., Angizi, S., & Violante, M. (2024). Dependability in embedded systems: A survey of fault tolerance methods and software-based mitigation techniques. *IEEE Access, 12*, 180939-180967.
[8] Pan, J. H., Zhang, S. B., Zhang, X. L., et al. (2013). Design and implementation of a triple-redundant airborne computer. *Journal of Northwestern Polytechnical University, 31*(5), 798-802.
[9] Cheng, J. Q., & Yang, J. P. (2014). Research on fault logic technology for multi-redundant flight control computers. *Modern Electronics Technique, 37*(10), 43-46.