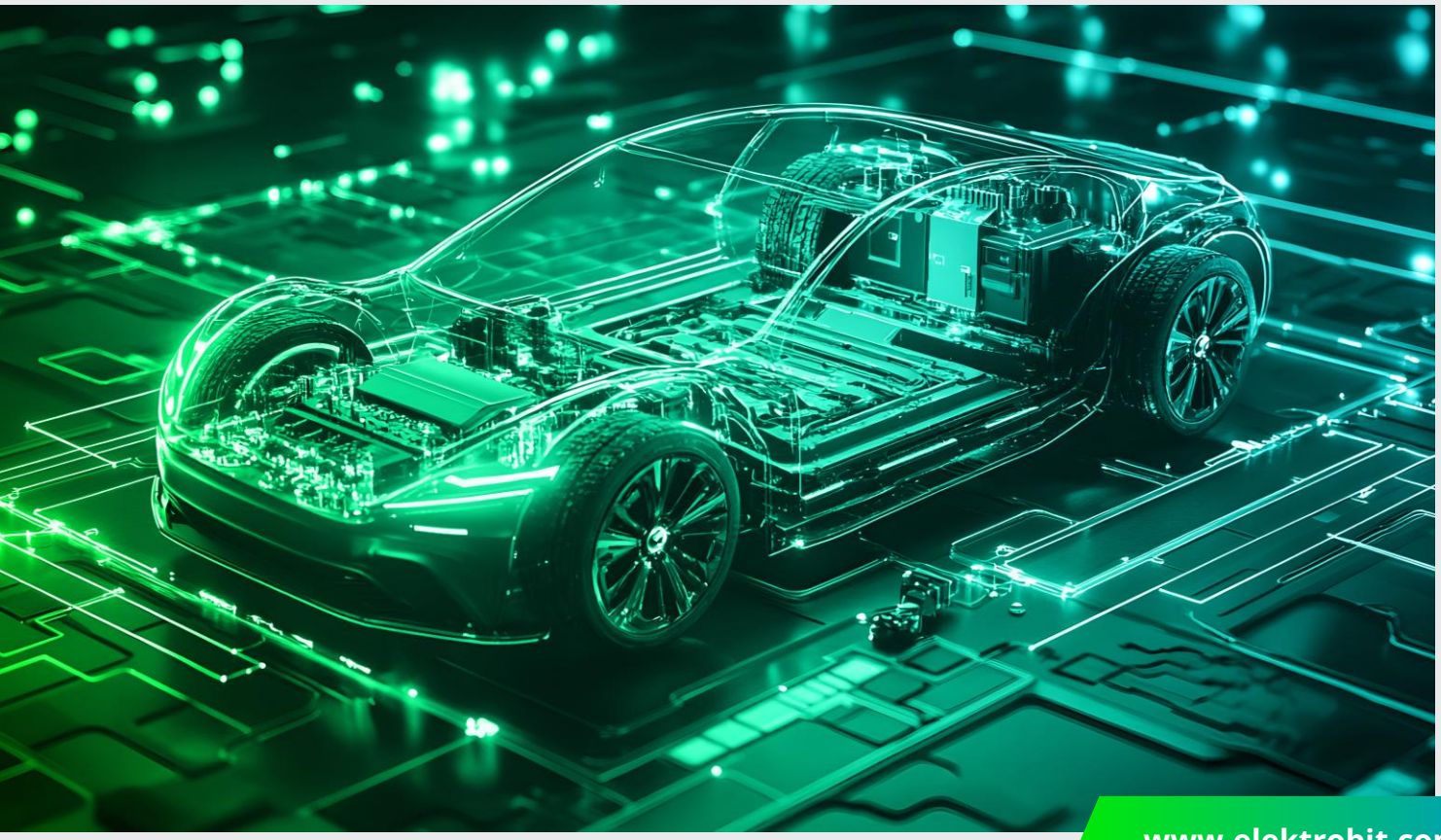
Elektrobit

Tech paper

Highly reliable x-by-wire systems



www.elektrobit.com

Table of contents

1. Software for fail-operational systems – highest availability and maximum performance? No compromise is needed!

According to safety standards like ISO 26262[1] and according to public expectations, the availability of steer-by-wire or brake-by-wire systems needs to be at least as high as the availability of traditional mechanical or hydraulic systems. To achieve this, we need to use fail-operational systems which can tolerate random hardware faults by using redundancy as well as to avoid systematic faults by using appropriate development processes. This paper presents a systematic software-based approach to achieve highest availability, while providing the maximum system performance, thus reducing system costs.

Fault tolerance and avoidance

By their definition, the functionality of x-by-wire systems depends on the correct functionality of electronics and software implementing the system. If the required functionality is safety-critical (for example, in the case of a steer-by-wire system), the controlling system usually shall be fail-operational, that is, it will continue operation even in case of electronic or software faults, as there is no safe state of the system.

Fault tolerance mechanisms are usually applied to mask random hardware faults, as the rate of random hardware faults of single hardware instances is usually higher than the required minimum needed to achieve the availability targets of x-by-wire systems. The resulting system then consists of two or more identical or similar redundant subsystems handling the control function. There is a need for some voting-mechanism to identify the correct control signals, or the redundant sub-systems shall be fail-silent (figure 1).[2]

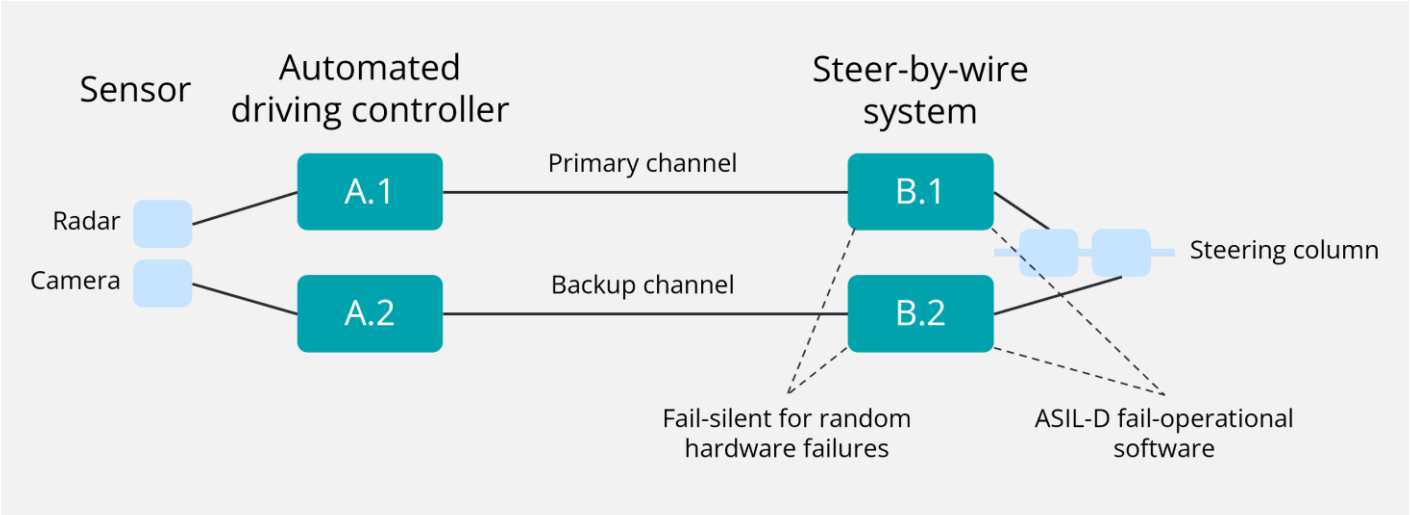


Figure 1: Example of a fail-operational system

Tolerance of systematic design faults, both in hardware and software, by using redundant subsystems only work to a certain extent, as it is practically impossible to avoid all common mode failures in the design of the two subsystems, even when using hardware from different vendors and doing both software and hardware design by strictly separated teams. Even when assuming that the subsystems are sufficiently independent, the ISO 26262-10:2018[3] requires the subsystems to be developed according to ASIL-B (D), if the system requirements are ASIL-D. As such, we recommend using fault avoidance mechanisms as the primary means to deal with systematic design faults, thus doing software and hardware design according to a sufficient ASIL level, usually ASIL-D.

A fail-operational system is more than a redundant fail-safe system

Advanced driver assistance systems (ADAS) are usually based on fail-safe peripheral systems, for example in case of EPS (electronic power steering systems) The basic safety requirement of a fail-safe system is defined as providing the correct functionality or no functionality at all, as these systems provide a “usually reduced” also in case of a failure of the electronic control system.

The safety requirement of a fail-operational system is to ensure correct and continuous execution. One might assume that a fail-operational system may be built up from redundant fail-safe systems, but this does not work, since there is no such safety requirement – correct and continuous execution – for these redundant fail-safe systems.

Especially when using almost identical software on redundant subsystems, fail-safe systems may fail concurrently at any point in time, by no longer providing their defined service. Even under assumption of when assuming fail-safe systems with completely independent failure modes, we still cannot give any guarantees for correct and continuous operation of the system as there are no guarantees at all for the availability of the redundant subsystems which may serve as base to calculate an overall availability (figure 2).

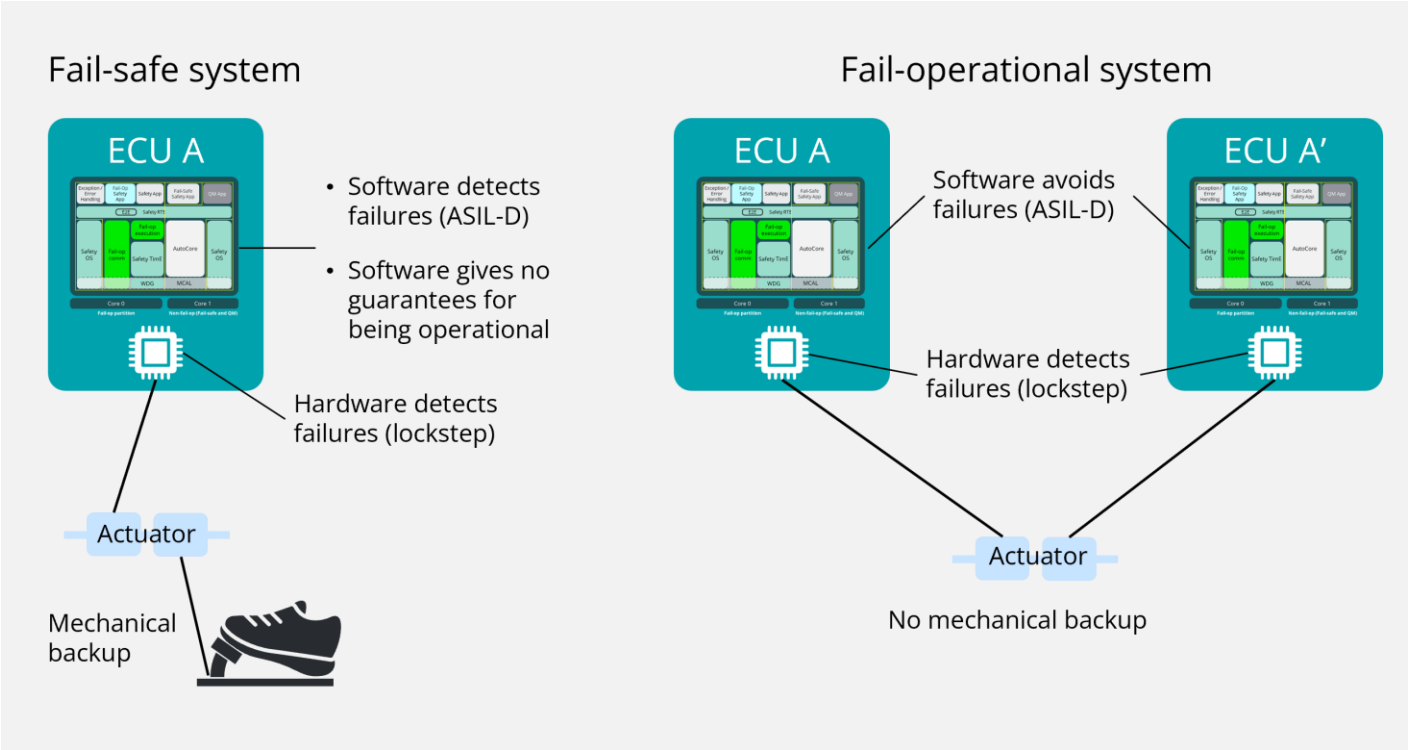


Figure 2: Fail-safe vs. fail-operational

Reduction of complexity

One of the measures that ISO 26262-6:2018[4] demands from ASIL-rated software is the enforcement of low complexity. In case of a fail-operational system, this basically tracks down to reducing the size and functionality, and overall complexity of the software to the required minimum, ultimately streamlining the interactions among different parts of the software.

ISO 26262 and ASIL levels

ISO 26262 for functional safety in road vehicles provides an automotive-specific risk-based approach to determine Automotive Safety Integrity Levels (ASILs)[5] and uses them to specify which of the requirements of ISO 26262 are applicable to avoid unreasonable risk. The applicable ASIL is determined according to the severity (from no injuries to fatal injuries), probability (from incredible to high probability), and controllability (from controllable to uncontrollable) of possible hazardous events. A system shall be developed according to the highest ASIL-D level if there are hazardous events having the highest level for each of these defined classes, and respective lower ASIL levels if all hazardous events have lower levels for one or several of these classes.

Reducing the functionality of the fail-operational software does not necessarily mean reducing the functionality of the whole system, but to carefully partition the system's software into fail-operational and non-fail-operational parts, aiming to minimize the amount of fail-operational software. This partitioning can be done according to timing and functional aspects (figure 3):

- **Operating mode of system:** Software which is executed during safety-critical operating modes of the system may be fail-operational, all software which is executed during other operating modes of the system (while the parking brake is applied) doesn't need to be fail-operational. This non-fail-operational software provides things like initialization routines, startup, shutdown, and sleep functionality, if the software may not tamper the system in a way which prevents the fail-operational software from operating correctly later.
- **Functionality of the software:** Even software executed during safety-critical operating modes of the system doesn't need to be fail-operational if it performs non-fail-operational relevant tasks only. This includes supporting tasks for diagnosis or firmware updates, or tasks which perform convenience functionality only.

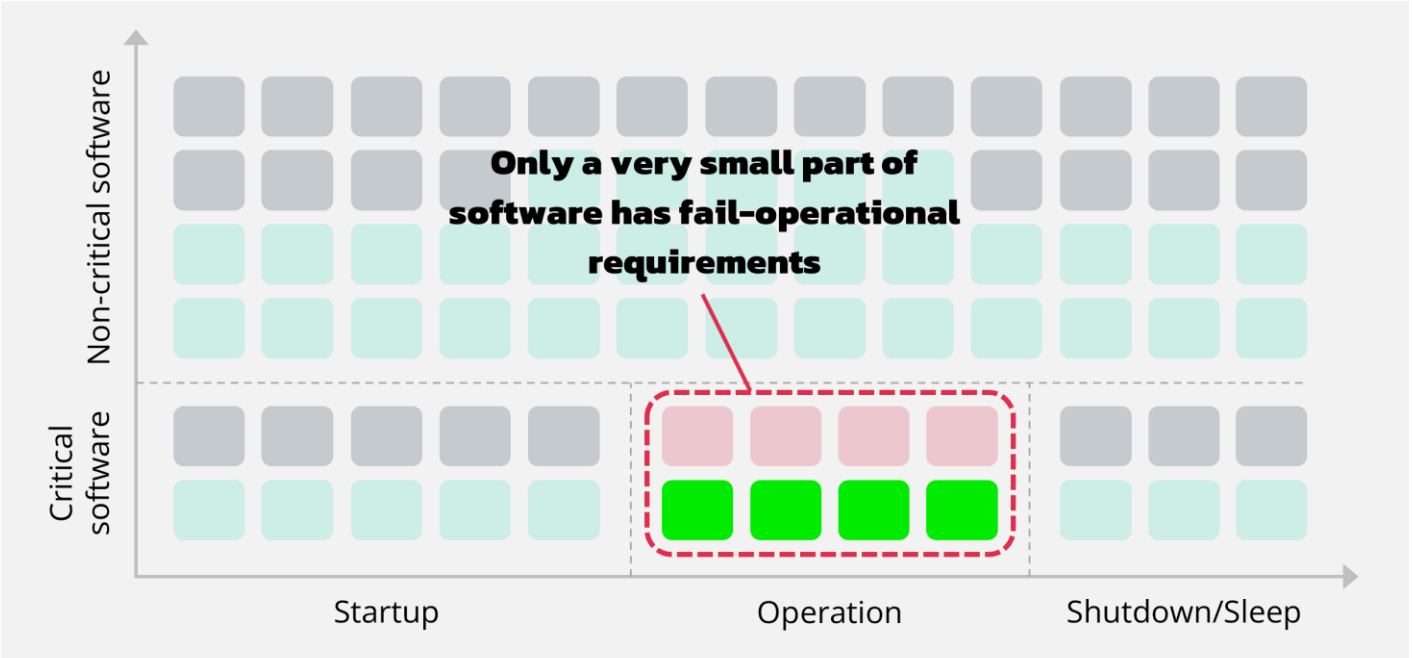


Figure 3: Software partitioning

Reducing the possible interaction scenarios between tasks may be done by executing a mostly static task scheduling, avoiding interruptions (use polling mechanisms instead) and higher priority, dynamically created tasks, while reducing task preemptions in situations where they are really needed, for example within tasks of different periods. The use of lock-free communication primitives further simplifies interaction scenarios by eliminating the need for critical section handling.

Advantages of complexity reduction

The development of ASIL-D software is very expensive because of the high reliability requirements, resulting in a complex development process. By reducing the size and complexity of the ASIL-D rated software, we can also reduce development cost significantly.

On the other hand, the availability of the system is basically defined by the availability of the fail-operational part of the system. Both the size reduction and the complexity reduction of the fail-operational software will reduce the probability of failures within the fail-operational software, and thus increase the availability of the system.

A crucial factor for the availability of fail-operational systems is the ability to execute each task within its defined processing deadline. By missing a deadline, a task may not only defer necessary communication with the environment, such as actuators, but also cause internal inconsistencies in task interactions or trashing situations. To avoid such scenarios, the worst-case execution time (WCET) of all fail-operational relevant tasks will be determined and the necessary amount of time will be reserved for them. To be able to calculate the WCET of a task, the task needs to follow a strict set of rules allowing it to limit the execution time of all code blocks, especially for loops. If all fail-operational tasks have reduced complexity, the efforts to calculate their WCETs significantly decreases.

Spatial firewalls

To support the partitioning of the software into a less critical and a fail-operational part, we ensure that potential problems with the less critical software do not jeopardize proper operation of the fail-operational software. Spatial and temporal firewalls are the right software mechanisms to enforce the separation of such partitions.

Spatial firewalls are basically implemented by memory protection units (MPUs). There are two kinds of MPUs: Core MPUs and Bus MPUs (figure 4). A Core MPU is part of a microcontroller unit (MCU) core and allows/prevents access to certain memory regions initiated on this core. To use a Core MPU for reliable protection of the memory which is used by fail-operational tasks, both the MCU core shall be trustworthy, that is, it will detect random hardware errors with a reasonable probability, such as by using a lockstep-architecture, and the software controlling the MPU will adhere to the same ASIL level than the one required for the fail-operational software. A Core MPU cannot protect memory from access by direct memory access (DMA) units outside of MCU cores.

A Bus MPU allows/prevents memory access on the base of peripheral units and/or MCU cores. This enables the protection of memory regions from access by MCU cores where either hardware or software is not fully trusted, for example non-lockstep cores or cores running a non-trusted OS, and from access by DMA units being part of peripheral units.

While a Bus MPU allows us to allocate fail-operational and non-fail-operational software partitions onto different cores of an MCU, the Core MCU supports allocation of these partitions onto the same core under the control of an operating system adhering to the required ASIL level.

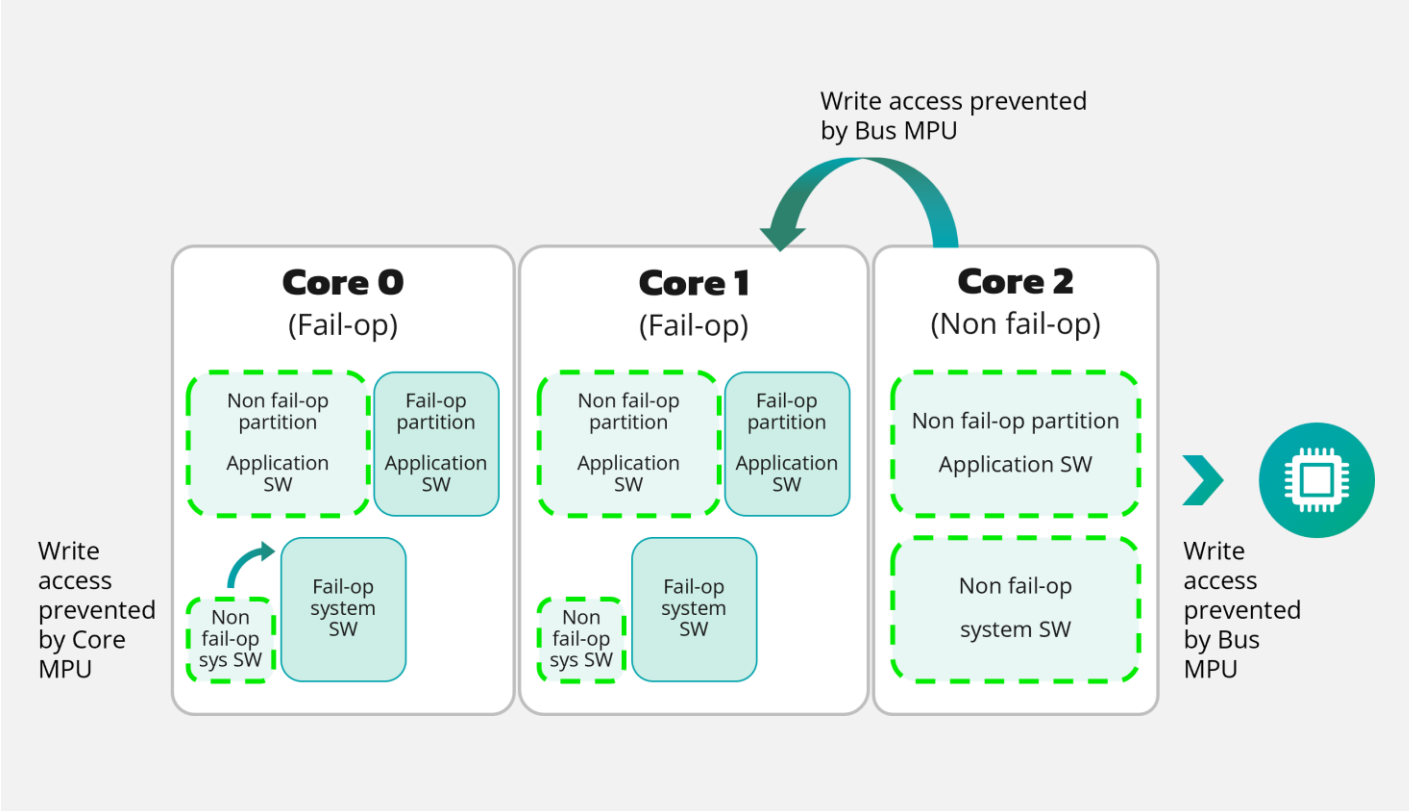


Figure 4: Example for spatial firewalls

Temporal firewalls

To protect fail-operational software from being tampered with arbitrary faults within less critical software, we need not only to protect the memory from being overwritten, but we also shall guarantee that the execution of fail-operational tasks is not blocked by less critical tasks. There are several ways to achieve this:

Execute less critical tasks on other cores than fail-operational tasks

In this scenario the tasks of different criticality do not compete against processing resources. A fail-operational task is blocked by a less critical task only in case it actively waits for some event controlled by the less critical task, like some kind of semaphore. This situation may be avoided by using lock-free communication mechanisms instead of critical regions.

Execute less critical tasks on the same cores than fail-operational tasks with lower priority

In the previous scenario, interactions between fail-operational and less critical tasks need to cross core-boundaries, making them time-consuming, especially when considering task synchronization. Allowing to execute less critical tasks on the same cores together with the fail-operational tasks leads to higher resource utilization on the one hand. On the other hand it enables more efficient communication between these two types of tasks.

To prevent less critical tasks from deferring the execution of fail-operational tasks, we intend to restrict the execution of the less critical tasks on these cores in two ways:

- Each less critical task will be preemptive and has a lower priority than any fail-operational task. This way the scheduling of fail-operational tasks is not tampered by less critical tasks (figure 5).
- The less critical tasks may not disable task switches or use exclusive resources shared with fail-operational tasks, except when there is a defined maximum lock-time for these resources which is monitored and enforced by the operating system.

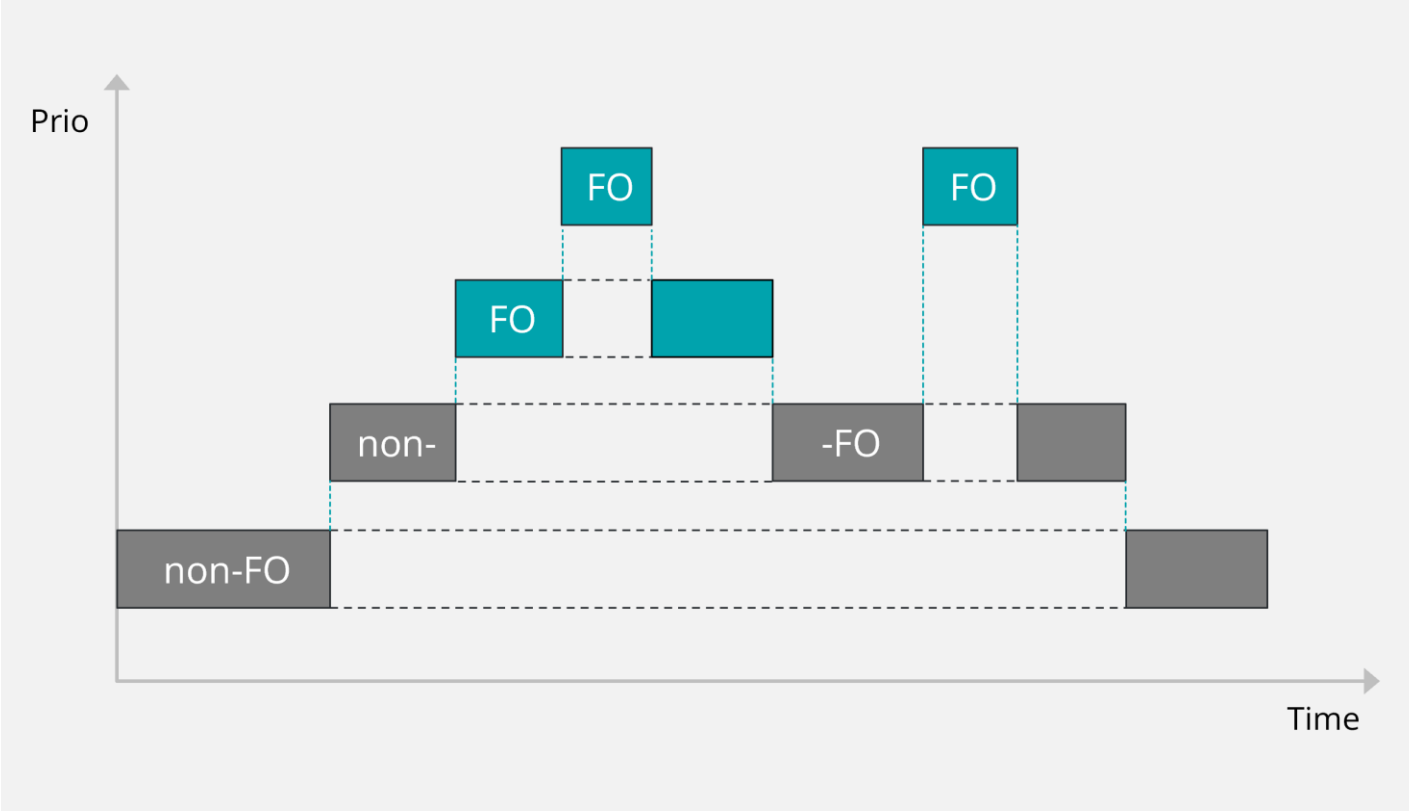


Figure 5: Non fail-operational tasks have lower priority

Use temporal firewalls between fail-operational and less critical tasks

The drawback of allowing only low priorities for less critical tasks is, that it prevents execution of chains of tasks with different criticalities, for example a task chain A – B – C, where tasks A and C are fail-operational, while task B does some non-safety-critical optimizations which could be left out if necessary. Additionally, there may be less critical tasks which need to be executed with a higher frequency than some fail-operational tasks, what is not possible with a priority-based scheduler, if the less critical tasks have lower priority.

The concept of temporal firewalls[6], which protect the correct timing behavior of fail-operational tasks from being tampered by less critical tasks, allows us to overcome these restrictions. A temporal firewall is basically a deadline for a less critical task which terminates the respective task if it has not finished at its deadline. Fail-operational tasks which are scheduled after this less critical task, or which are preempted by it are guaranteed to be activated not later than at the occurrence of the deadline (figure 6).

The task scheduler provides the trusted services to start tasks and occasionally terminate them if they violate their deadlines, thus it is a crucial service to guarantee that all fail-operational tasks complete in time. It should be realized as ASIL-D software service, ideally as part of the operating system. At design phase, an application developer will create a schedule for the fail-operational tasks considering the WCETs, periods and dependencies for these tasks. This schedule may then be enriched with less critical tasks of arbitrary priorities and assumed execution times. To guard fail-operational tasks against less critical tasks which do not finish on time, we provide these less critical tasks with deadlines at the point in time, when they have to finish latest. This way we may mix the two types of tasks without jeopardizing the safety properties of the fail-operational tasks.

Usually the WCET calculations for fail-operational tasks overestimate the tasks' WCETs by some factor, depending on the MCU architecture and the structure of the code. The presented approach allows us to use the difference between the calculated WCET and the actual execution time of a fail-operational task to increase the amount of CPU time available for less critical tasks. As less critical tasks are allowed to fail from the safety perspective, their correct timing needs to be validated by conventional methods like testing and not by calculating theoretical limits. That is, all the execution time which was allocated to fail-operational tasks but not yet consumed, contributes to the available execution time of succeeding less critical tasks and increases the performance of the overall system – without making any compromises on safety.

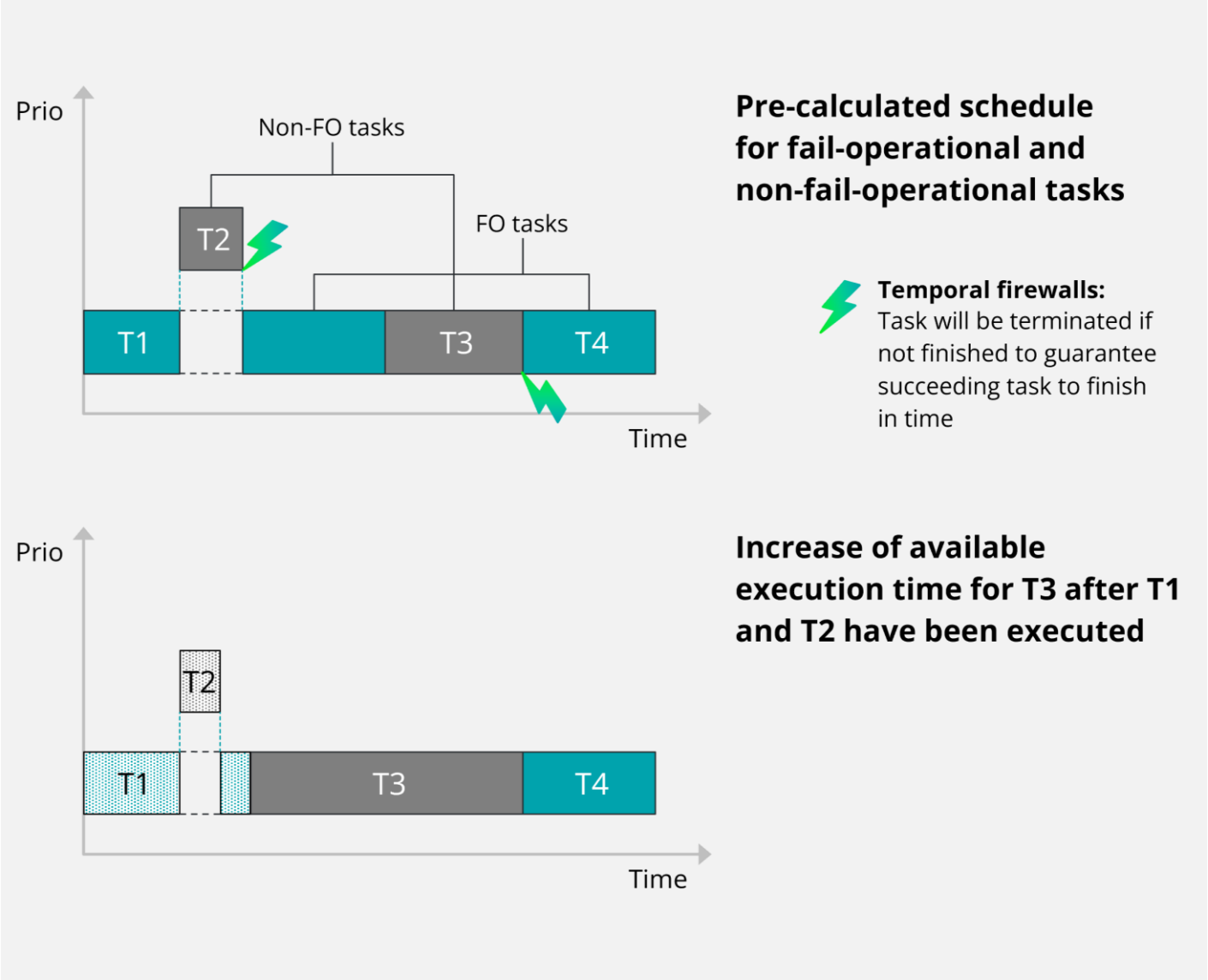


Figure 6: Temporal firewalls for fail-operational tasks

A failure, like a timing failure within a less critical task may be handled in various ways:

- In specific situations it is sufficient to terminate only the failing task and take no further actions, or
- In other cases, it may be necessary to terminate the whole subsystem of less critical tasks and activate a fail-operational only mode with reduced functionality. This may be used to perform a last minimal risk maneuver or to continue the trip with reduced functionality of the vehicle.

Reduce qualification costs by using spatial and temporal firewalls

Safety qualification efforts for safety critical software have a considerable contribution to the overall costs of safety critical software. The presented approach of using spatial and temporal firewalls reduces these costs in two ways:

- Only the dedicated fail-operational part of the software needs to be developed according to the respective ASIL level, the remainder may be developed according to a lower level, usually QM level. This highly reduces overall costs.
- After any changes to the software, usually the whole software needs to be re-qualified. By having spatial and temporal firewalls enforced by the operating system, changes in the less critical software do not require a re-qualification of the software, since the fail-operational software is protected against bad effects of changed or even erroneous behavior of the less critical software.

2. Conclusion

The presented approach for software for fail-operational automotive systems reduces both the cost for development, qualification, and maintenance of the software and increases the amount of software which can run on a certain MCU by allowing to mix fail-operational and less critical software on one core, and by utilizing the difference between actual and worst case execution time of fail-operational tasks for the execution of less critical tasks.

Building such design allows us to achieve high availability of the system by reducing the amount of safety critical software. Fail-safe or convenience functions, which require less development and qualification efforts, may be easily and quickly added to the system in a shorter time.

3. Author

Johannes Reisinger
Chief Expert, Automotive Middleware and Basic Software
Elektrobit Automotive GmbH

After his Master and PhD thesis about 35 years ago, he has been working as a researcher and product developer for several companies, both in the role of a Technical Manager and an Architect, focusing on three major areas: embedded systems, fault tolerant real time systems, and hardware-software co-design.

Creating innovative solutions by using his wealth of experience and combining it with new concepts fitting to the actual problem domain is one of his major strengths. Being a part of the automotive industry for about 25 years, gives him the knowhow and confidence to create modern automotive architectures.

4. References

[1] ISO 26262-5. (Dec 2018). *Road vehicles – Functional safety – Part 5: Product development at the hardware level.* ISO.

[2] Architecture, T. A. (September 2025). Von The Autonomous: https://www.the-autonomous.com/news/the-autonomous-working-group-safety-architecture-second-report/abgerufen

[3] ISO 26262-10. (Dec 2018). *Road vehicles – Functional safety – Part 10: Guidelines on ISO 26262.* ISO.

[4] ISO 26262-6. (Dec 2018). *Road vehicles – Functional safety – Part 6: Product development at the software level.* ISO.

[5] ISO 26262-3. (Dec 2018). *Road vehicles – Functional safety – Part 3: Concept phase.* ISO.

[6] Kopetz, H., & Nossal, R. (1997). Temporal firewalls in large distributed real-time systems. *Proceedings of the Sixth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems.* Tunis, Tunisia: IEEE.

About Elektrobit

Elektrobit is the trusted partner in the transition to the software-defined vehicle (SDV). With over 35 years of award-winning automotive software expertise, Elektrobit's innovative portfolio and comprehensive SDV ecosystem empower OEMs, Tier 1s, along with ODMs and Big Tech to build future-ready solutions with speed and confidence. Its SDV building blocks include operating systems, middleware, embedded software, digital cockpit solutions, engineering services, and development workflows – driving faster innovation and seamless integration across the vehicle lifecycle. Elektrobit software powers over five billion devices in more than 630 million vehicles worldwide. It is a wholly owned, independently operated subsidiary of AUMOVIO.

For more information, visit us at **elektrobit.com**

Elektrobit Automotive GmbH
Am Wolfsmantel 46
91058 Erlangen, Germany

Phone: +49 9131 7701 0
Fax: +49 9131 7701 6333

sales@elektrobit.com

www.elektrobit.com