Challenges in Mitigating Errors in 1oo2D Safety Architecture with COTS Micro-controllers

Amer Kajmakovic*, Konrad Diwold*, Nermin Kajtazovic¶, Robert Zupanc¶

*Pro2Future GmbH & Institute of Technical Informatics, TU-Graz, Graz, AT
E-mail: (amer.kajmakovic, konrad.diwold)@pro2future.com
¶ Siemens AG, Graz, AT, E-mail: (nermin.kajtazovic, robert.zupanc)@siemens.com

***Abstract*—The number of Commercial-Off-The-Shelf (COTS) micro-controllers used in safety applications has increased significantly over the last decade. In contrast to safety-certified micro-controllers, they are produced without integrated protection against memory soft errors and limited in terms of available memory and computation power. However, due to constant optimizations of the memory's physical size and the voltage margins, the probability that external factors, such as magnetic fields or cosmic rays, temporally alter a memory state (and thus cause a soft error) rises. It is crucial to address such errors within safety-critical systems, and consequently a wide range of error mitigation strategies have been proposed. In the context of established brownfield automation systems, redesign and redeployment of new hardware is usually not feasible. Therefore, other approaches can be applied to existing fail-safe architectures to further improve their performance without the need for a partial rework or conceptual changes. This article identifies challenges associated with soft error detection and correction strategies in 1-out-of-2 with diagnostic (1oo2D) safety architecture. Moreover, it investigates mitigation strategies and their deployment challenges through different production phases of the systems (i.e., greenfield) as well as requirements and limitations when working with already existing systems (i.e., brownfield). Among other parameters, the memory usage profile and its effect on the mitigation strategies is explained. A brief overview and evaluation of already available hardware-based strategies along with the evaluation of the most prominent software-based strategies are presented. In addition, a discussion about potential mitigation strategies that rely on the underlying hardware features is outlined. The article demonstrates how to identify and assess trade-offs associated with different strategies to decide on suitable methods to enhance fault tolerance in existing and future automation systems.**

Keywords–soft errors; mixed-criticality; fail-safe; 1oo2D; COTS;

I. Introduction

This article extends the contribution "Challenges in Mitigating Soft Errors in Safety-critical Systems with COTS Microprocessors" of PESARO 2020 [1]. The contribution investigated challenges associated with software-based soft error detection and correction strategies, along with a short overview of currently applicable software-based mitigation strategies. Here, the evaluation is extended to include available hardware-based strategies and different phases in the development process of 1oo2D safety architectures. Furthermore, new ideas and approaches are presented utilizing existing features within 1oo2D architectures to avoid physical intervention on the system.

Given their ever-decreasing packaging size, semiconductors are increasingly susceptible to external influences such as alpha particles, cosmic rays, or magnetic fields [2]. Figure 1 shows the correlation of semiconductor technology/fabrication node size (nm) and their respective error rates (Soft Error Rate (SER) and Hard Error Rate (HER)). It is evident that the SER increases with decreasing node size, while the HER remains constant [3]. To counter the increasing number of soft errors, families of highly reliable safety-certified Micro-Controller Units (MCUs), with special integrated measures against soft errors, have been developed. The intended field of application of such micro-controllers is safety-critical applications where fault-tolerance is required.

Nevertheless, given their low cost and good performance, Commercial-Off-The-Shelf (COTS) micro-controllers are increasingly used in safety applications [4]. In contrast to safety-certified micro-controllers, they are not produced with integrated protection against soft errors. As a consequence, recent research proactively deals with environmentally induced soft errors by developing new methods for error detection, mitigation, and data recovery [5].

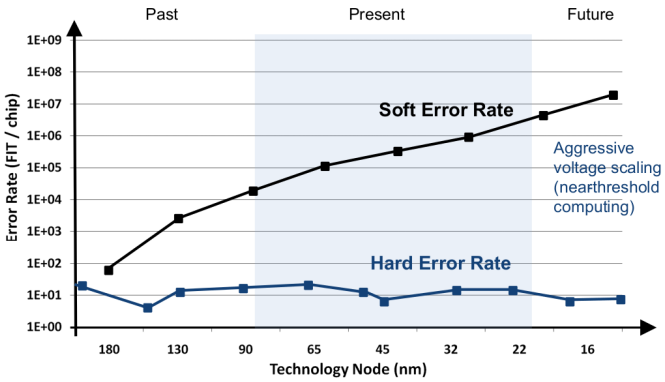

Figure 1. Correlation of error rate and technology/fabrication nodes

The importance of detecting and resolving soft errors is reflected by the numerous reports on soft error related problems within safety-critical applications. These reports originate from a wide range of industries, such as the automotive industry, space industry, and the medical industry. Duncan and Roche's analysis of semiconductor reliability in the context of autonomous driving [6] is devastating. They conclude a (soft error induced) failure rate of 1 part per million per year. Given that a single-car implements approximately 8,000 semiconductors, the likelihood of a car exhibiting semiconductor-induced errors within its lifespan (of 15 years) is around 12%. While the results of such failures are unclear during the operation of a car, semiconductor-based soft errors can be resolved (fairly easily) by restarting the affected component. However, not all safety-critical systems provide the luxury of resolving an error by "turning it off and on again". Consider, for example, safety-

critical nuclear power plant equipment: restarting a device in the event of a soft error is not an option and could lead to catastrophic fatalities.

When designing new safety-critical applications or enhancing existing systems with fixed underlying architectures and hardware, a wide range of methods is available. These methods target different stages within a system's development and are associated with trade-offs regarding required resources, costs, and complexity. To choose an appropriate strategy therefore requires a clear understanding of the available methods as well as their prerequisites. This article aims to demonstrate the variety of existing mechanisms for soft error detection and correction, by reviewing and outlining available methods. In addition, the article demonstrates how an appropriate methodology can be chosen, depending on the development stage of the application along with challenges that come with selecting the right strategy.

While architects are 'relatively' free when designing a new system from scratch, their options narrow down when they are enhancing an existing system, as the chosen methods must complement the existing system. Given the longevity of existing safety automation solutions, this article demonstrates an approach to improve/enhance existing fail-safe solutions. This is done utilizing an exemplary system with a 1oo2D safety-architecture and allows to demonstrate the impact and prerequisites of various safety strategies on the system's performance and design as well as their effects on non-functional requirements, such as reliability, safety, and availability. The discussed approaches range from enhancing the existing hardware solutions with additional software-based correction schemes to the utilization of additional hardware, resulting in novel hybrid approaches. These innovative approaches allow enhancement of non-functional properties such as availability, maintainability, and most importantly safety in existing safety architectures.

The remainder of the paper is organized as follows: Section II presents an overview of the mitigation strategies through the production phases. In Section II-E a screening of the market-available micro-controllers with mitigation strategies is presented. Section III describes 1oo2D safety architecture with a focus on its memory architecture. Section IV defines the challenges and requirements for soft error software-based mitigation strategies in safety-critical applications. Section V shows an evaluation of the mitigation strategies along with new mitigation ideas. In the last section, a summary and future work are presented.

II. Mitigating soft errors

While soft errors constitute the majority of memory errors, they can be prevented and/or corrected. To prevent soft errors, memories require resilience and/or fault-tolerance. Fault-tolerance denotes a system's ability to handle faults in individual hardware or software components, power failures, or other forms of unexpected problems, while still meeting the system specification [7]. There are different approaches/strategies to achieve fault tolerance. These approaches can be grouped into different levels regarding the stage in the development process they are utilized in as well as their underlying nature.

The most intuitive categorization can be made based on the different stages of a system's development. Protection and mitigation strategies can be designed and applied within the design and production processes of single components (i.e., memories) themselves. During system design, mitigation strategies can be actively integrated into the system by, for example, choosing appropriate components and system architectures (such as redundant architectures). If a system's architectural level has already been fixed (during or before the deployment stage), only software-based approaches can be used to enhance fault-tolerance on a system level (e.g., via additional features that will additionally secure a system). During system design, fault-tolerance mechanisms on a hardware-level (e.g., by hardening components and architecture) can also be utilized. Mitigation strategies thus either fall into the Hardware-based (HW) or Software-based (SW) classes. They are not mutually exclusive, meaning that a system might implement a set of different mitigation strategies to achieve required fault tolerance. Another categorization concerns whether or not an approach utilizes redundancy. Within a system, redundancy can occur on different levels: Hardware, Software, Information, and Timing, which are explained in more detail below. Figure 2 gives examples for mitigation strategies and their categorization.

Stages	Type	HW/SW	Examples
Component level	Early design stage	HW	Hardening of the component cases
Architecture level	Hardware redundancy	HW	2oo3 architecture
	Informational redundancy	HW	Memory with HW parity bit protection
System level	Software redundancy	SW	Parallel execution of redundant function
	Informational redundancy	SW	SW ECC code
	Timing redundancy	SW	Repetition of the same function execution

Figure 2. Categorization of the mitigation strategies

In the following subsections, state of the art mitigation strategies are outlined according to the system development levels they fall into, starting with the system level.

A. System level

Protection on the system level is applied when the hardware of a system is present, including internal design and system architecture. At this level, additional fault-tolerance can only be achieved via software-based approaches (as hardware and system architecture are fixed). Methods applicable to this phase can also be used to enhance existing (brownfield) automation systems that are already deployed and do not allow for hardware changes.

Software-driven fault-tolerant techniques are based on redundancy, which is applied to procedures, processes, data, or the whole execution code. The most common type of **software redundancy** in embedded systems is the multiplication of data. A simple way of achieving multiplication is to transform (e.g., with the Hamming distance of 4 or a simple inverse function) and store a copy of a variable in a different memory area. Comparison of the two versions of the variable enables the system to detect, mitigate, or recover corrupted data.

The main disadvantage of software redundancy is associated with memory consumption overhead, as the multiplication of data, code, and/or processes requires additional memory, which is usually very limited in embedded systems. Additionally, it can lead to a significant increase in code execution time [2], [5]. The other two types of redundancy which can be realized in the software itself are Informational and Timing redundancy.

Informational redundancy assumes the addition of supplementary information to the data to verify the soundness of the information. Usually, this additional information is in the form of codes, which are computed based on the data itself. Those codes (so-called Error Detection And Correction codes (EDAC)) were initially introduced in the context of data recovery in communication [7], but nowadays they are widely used in memories [8]. The family of EDAC codes is growing constantly. The most popular EDAC codes are: Parity Codes (error detection without recovery) [9], Hamming Codes (2-bit detection and 1-bit recovery) [9], Reed-Solomon and Bose-Chaudhuri-Hocquengham Codes (for multiple bits error masking) [8]. Some research has considered the implementation of other EDAC codes used in communication such as LDPC codes [10], RS codes, Turbo codes [11]. EDAC codes can be presented with the designation (n, k), denoting a block code that takes a k-bit data word and maps it to an n-bit codeword as shown in Figure 3.

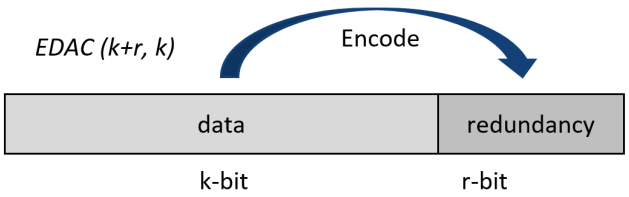

Figure 3. Representation of EDAC codes

EDAC codes have two main properties that need to be considered: speed and quality. Speed is defined as the time required to encode/decode EDAC codes and this time extends the overall memory access time. Quality denotes the number of faulty bits a specific code can detect and correct. Naturally, there is a trade-off between quality and speed. For higher quality, more complex EDAC codes are required, which allow for correction of multiple bit-flips. In this case, both code magnitude as well as computing demand increase due to these adaptations. Faster and less memory expensive correction schemes on the other hand are limited in terms of the number of bits they can correct.

Based on EDAC codes, a new method called **scrubbing** was developed. The idea behind scrubbing is to periodically re-write data in its original location to eliminate soft errors if they are correctable through EDAC [12] or copying of original data [13]. With this approach, an accumulation of soft errors inside one region of memory can be avoided.

Timing redundancy has been recently investigated and concerns a re-computation or retransmission of data at least twice. The results are then compared with previously stored copies [7]. This type of redundancy helps to distinguish between transient and permanent errors. If the fault is still present after repeating a test several times, then it is likely that the error is permanent.

B. Architecture level

HW-based Information redundancy: Software-based information redundancy raises the question of usability, as high-quality SW EDAC codes exhibit a trade-off and lead to a decrease of available memory as well as to an increase of required computation time, access time, and complexity of the overall system. To overcome these drawbacks, EDAC-related computations (encoding and decoding) can be outsourced on a special-purpose chip, which can be installed between memory and CPU in order to apply for on-the-fly informational redundancy. Most modern EDAC codes for memories are implemented via additional hardware [14]. EDAC addresses the perspective of system availability for safety, since the system will continue to run unabated in the presence of single-bit errors. However, EDAC adds significant cost to the memory portion of the device and slows down the CPU due to the added SRAM access time, which is required to make corrections on the fly. SRAM on a device constitutes about 1/3 of the hardware costs, and with additional HW-based EDAC this further increases by approximately 30%, resulting in a total price increase of around 40% [15]. To avoid an increase in chip size and hardware redesigns, software-based EDAC codes (explained in the previous chapter) have been proposed [16], [17]. In the past, HW EDAC codes were only available in the expensive safety-certified MCUs, but today conventional micro-controllers also possess HW-based EDAC protection. The flash memory, where operating code is stored, is usually protected with a Hamming code while parity bits protect selected parts of the SRAM [18], [19]).

A parity circuitry sets the parity bits when an SRAM word location is written and verifies that there are no single-bit errors in the word when it is read back. This is done within the read/write cycles, so no CPU overhead is involved. When the parity circuitry identifies an error, a high priority CPU interrupt is generated. In semiconductor devices, this detection mechanism is simple and relatively inexpensive to implement. Parity addresses the safe-state perspective for safety. As described earlier in Section I, virtually all SRAM failures in-system are likely to be single bit per word failures. This applies to both physical defect mechanisms as well as soft errors. Additional coverage can be provided by protecting the memory address bits with parity.

Hardware Redundancy: On a system level, fault tolerance can be achieved via hardware redundancy. Safety-critical systems often adopt an N-modular (where $N > 2$) architecture, where the components exist in certain redundancy N and perform the same computations in parallel. The correct result is established based on majority voting. If one of the modules fails, the majority voter masks the fault by identifying the result of the remaining fault-free modules [7]. N-modular systems can yield a higher Safety Integrity Level (SIL), as they provide inherent fault tolerance and consequently, a low failure rate. SIL is a quality indicator for systems that fulfill safety requirements in accordance with the IEC61508 standard. Many safety systems use simple architectures such as 1oo1D (1-out-of-1 with diagnostics) and 1oo2D (1-out-of-2 with diagnostics) [20]. In some cases, a diagnostic system is realized with an additional watchdog (i.e., challenge-response architecture) [21] or with an additional CPU like the lockstep architecture.

Lockstep systems are fault-tolerant computer systems that

run the same set of operations at the same time in parallel [22]. The redundancy (duplication) allows error detection in the system as well as in the memories. The stored values in memory are compared to determine if there has been a fault. The term ”lockstep” originates from army terminology, where it refers to synchronized walking, in which the marchers walk as closely to each other as physically possible.

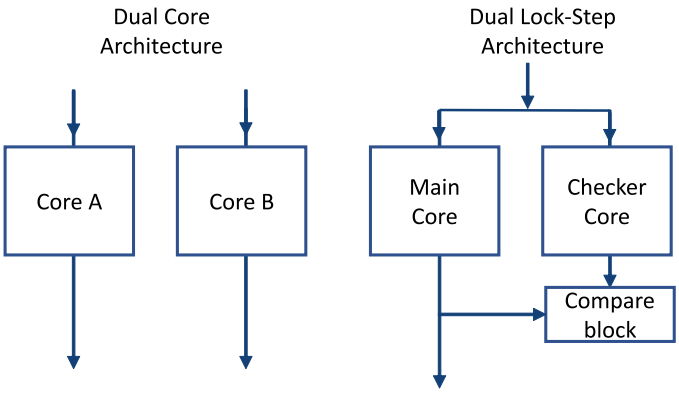

Figure 4. Dual Core and Lockstep architectures

These architectures are also known as a fail-safe, meaning that given a failure, the system inherently responds in a way that will cause no or only minimal harm to equipment, environment, and people. The main advantage of such architectures is the good balance between functional safety (i.e., achieving high safety integrity) and development process costs.

A shortcoming of hardware redundancy is its requirement for additional hardware. In the context of memory, it will increase cost, weight, size, power consumption, and thus impacts design and testing. Moreover, additional hardware needs to be budgeted for from the first stage of the chip design. It is therefore almost impossible to upgrade existing systems with additional hardware without degrading their performance, which limits the application of these methods in the context of brownfield applications.

C. Component level

Environments with high ionizing radiation (e.g., outer space, nuclear power plants, etc.) present special design challenges for integrated circuits, as the likelihood that particles cause an upset in the electronics (i.e., memory) is very high. Dealing with the consequences requires very reliable electronic components with sophisticated measures that can detect and correct errors. The first step in overcoming errors is to prevent them from happening, i.e., to stop particles on their way to the sensitive parts of the electronic circuits. This type of protection is achieved during the early stage designs, where different techniques and approaches are used to prevent errors. If these techniques can successfully protect electronics, in later phases they do not need additional detection and correction algorithms.

Shielding constitutes one of the first approaches that increase the resilience of components against radiation. Shielding is applied during the production phase, where a specific particle-resistant layer is deployed over the component’s package. The layer reduces exposure of the bare component/device and prevents environmental particles from influencing underlying layers of the package. Figure 5 depicts the penetration ability of various types of particles. As shown in the image, neutrons are capable of travelling further through different types of material than other particles, making it challenging for designers to find adequate materials for shielding.

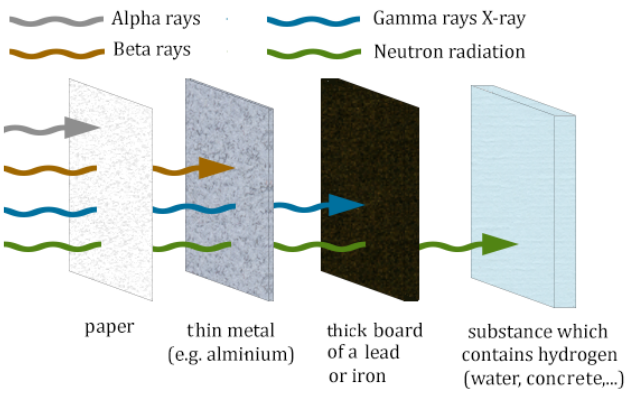

Figure 5. Penetration ability of radiation particles [23]

Radiation hardening is an approach where designers of electronic circuits use various physical means, such as insulating substrates, bipolar integrated circuits, or radiation-tolerant SRAM to harden the electronic system against the effects of radiation particles [24]. Hardened chips are often manufactured on insulating substrates instead of the usual semiconductor wafers (where energy from radiation can easily change the state of the material). Silicon on insulator (SOI) [25] and Silicon On Sapphire (SOS) [26] are commonly used. While hardening guarantees fewer errors to be caused by radiation, it requires special designs and techniques that increase the overall costs of the design and production process. Resistance to electrical charges can also be achieved by using specific structures and materials for critical points in the component (e.g., strengthening the gate of the transistors). One of these structures is the Dual Interlocked Storage Cell (DICE). In this technique, a transistor structure has redundant storage nodes and restores the original cell state when an error is introduced in a single node [27].

Other types of memories that are not based on standard semiconductors but on different underlying concepts can be found. The most promising concept is Phase-Change memory (PCM), which constitutes a new type of memory that is achieving good results against particle radiation. PCM utilizes a Germanium Antimony Tellurium $Ge_2Sb_2Te_5$ (GST) alloy and takes advantage of rapid heat-controlled changes in the material’s physical property of amorphous and crystalline states [28]. These states, which correspond to logic 0 and 1, are electrically differentiated by high resistance in the amorphous state (logic 0) and low resistance in the crystalline state (logic 1). One cell of the PCM is shown in Figure 6. PCM, which reads and writes at low voltage, offers several substantial advantages over flash and other embedded memory technologies: PCM is faster than standard flash memories, and logical gates within PCM can be scaled down further than the NOR and NAND gates used in flash memories. PCM also showed good protection against bit-flips induced by highly energized particles hitting the memory. Even though phase change material is immune to high-energy particles, PCM memory still suffers soft errors. For example, in PCM chips, up to 40% of the entire area consists of CMOS circuits [29]. As

PCM is still in development, only a few types of this memory are available on the market [30]. Similar to radiation hardening, PCM memory is used in the early design stage of the MCU, and thus, it does not have additional effects on the MCUs' performances.

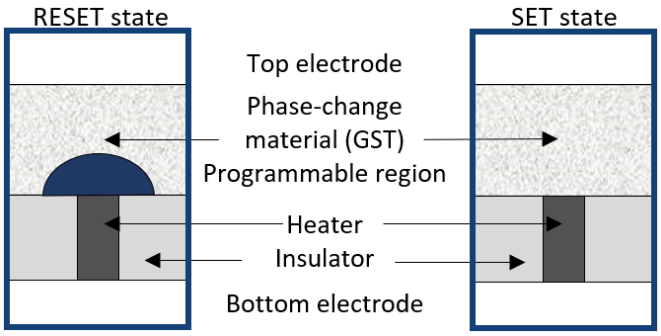

Figure 6. Phase changing memory - Reset and Set states [31].

Although techniques used on a component's level have shown very effective against soft errors, they always require additional or special materials, which significantly increases the cost of design and production.

D. Multi-phase

Some approaches to the production phase (i.e., component level) can also be used in later stages. For example, shielding can be applied after the entire system is developed, e.g., by installing a radiation-resistance shield over the system itself. Combining the best features of different protection schemes, to cover their weakness, constitutes a good way to create system tolerance for all kinds of failures. Mayuga et al. [12] combined different kinds of techniques to overcome failures in memory. Their approach uses EDAC codes to recover words with a single faulty bit, memory relocation for a word with more than one faulty bit, and a scrubbing method to avoid the accumulation of faulty bits. A hybrid approach seems very suitable in the context of mixed-criticality, as it allows to further customize the overall protection scheme, leading to a protection scheme with an even further reduced overhead in comparison to a scheme that is based on single redundancies.

E. Available solutions for safety-critical systems

When designing a safety-critical system from scratch, it is recommended to proactively consider soft errors through all production phases, in order to satisfy the safety requirements of the resulting system. As shown in the last section, designing a system from scratch allows us to utilize hardware solutions to mitigate or overcome errors, e.g., by choosing an appropriate architecture or components that already have integrated safety measures against soft errors in memories.

Manufacturers have developed special-purpose safety-certified micro-controllers that are highly reliable and contain additional features to overcome safety issues, including soft errors in memories. The design of these micro-controllers demands more time and effort, thus their development is far more costly than regular COTS micro-controllers. The main catalyst for these recent developments is the automotive industry. With high demands for functional safety in Autonomous (AV) and Semi-Autonomous Vehicles (SAV), the development of safety-critical micro-controllers has rapidly increased. Functional safety is required in almost every part of AV and SAVs, including all sensors, processing, and control units. Some MCU developers like STMicroelectronics are offering a wide portfolio of MCUs specialized for automotive applications. The latest achievement in safety from STM are the controllers from the Stellar series, a high-performance 32-bit automotive microcontroller family, which is based on the ARM R52 multi-core. It features an innovative embedded Phase Change Memory (ePCM) and built-in 28nm Fully Depleted Silicon On Insulator (FD-SOI) technology [32]. The combination of ECC and this memory can provide sufficient protection from soft errors in these processors.

In the context of automotive use-cases, the probability of a particle hitting the memory and flipping a bit is low, but the impact of a bit flip can be devastating. In space and the nuclear industry, besides devastating impact, the probability of a particle altering the memory is very high. Therefore, the need for radiation-resistant electronics is higher than in any other domain. This can be achieved e.g., via HARDSIL® a special technology that immunizes semiconductor devices against high temperatures or radiation-induced stress without the need for special design techniques (RHBD) or expensive specialized semiconductor processes (RHBP). HARDSIL® can enhance a broad range of semiconductor devices. It is a fully designed agnostic approach, where any standard manufacturing equipment and process geometry can be used with no resulting negative impact on performance, power consumption, or yield. Simulations have shown the ability to scale down to the most sophisticated leading-edge technologies like FinFET implementations [33]. One type of MCU that employs HARDSIL technology is the VA108X0 [34] micro-controller from Vorago technologies, based on the ARM®Cortex®-M0 processor with a radiation tolerant case.

Another example of highly reliable MCU is the TMS570 series from Texas Instruments' line Hercules. This safety micro-controller is targeted for safety applications, through hardware-based fault correction/detection features in the form of dual cores that can run in lockstep. Moreover, it has automated self-testing of memory and logic, peripheral redundancy, monitor/checker cores, and full path ECC. The full path ECC means that all memories in the MCU are protected with ECC (Flash, Data Flash for EEPROM, SRAM). This type of hardware-based ECC is performed by the CPUs and can correct single-bit errors and detect double-bit errors (SECDED). The ECC is evaluated in parallel to application processing, so there is no impact on latency or performance. The same integrated safety protection can be found in NXP's Kinetis Kx line. Their Flash and RAM memories are also protected with ECC codes (SECDEC).

The examples and Figure 7 outline a selection of MCUs specialized for safety-critical applications. These MCUs are certified according to automotive (ISO 26262[35]) and industrial (EC 61508[36]) safety standards. As a result, they are significantly more expensive than standard COTS MCUs. This cost difference may lead engineers and designers of safety systems to look at cheaper solutions based on COTS MCUs. As outlined above, some COTS MCUs already integrate simple hardware memory protection such as parity bits. The computation of these simple protection schemes requires less resources than more complex EDAC (SECDED) codes. Adding them to a controller does not significantly increase the overall costs. The

Safety-processor	Architecture	Memory protection	Additional safety features
ARM Cortex R52	Lockstep / Dual core	ECC (SECDED)	• High-coverage built-in test (BIST) • Safety package
Infineon Aurix	Up to 3 CPUs / Dual-lockstep	ECC (SECDED)	• Architectural diversity
Intel Xeon D-1529	Lockstep	ECC (SECDED)	• Windowed watchdog timers. • Mixed safety-critical task execution
MIPS i6500-F core	Configurable clusters of 64bit CPUs	ECC (SECDED)	• Rigorous QMS processes addressing • Heterogeneous inside and outside
NXP S32S24	Lock step with hardware hyper vision	ECC (SECDED)	• Large integrated flash memory for multiple sets of application code
STM SPC5	Lockstep / Dual core	ECC (SECDED)	• High-coverage built-in test (BIST)
Texas Instruments Hercules	Lockstep	ECC (SECDED)	• BIST • ADC self-tests, • CRC on communication
Xilinx Zynq 7000	FPGA -> Two independent channels	yes	• FPGA

Figure 7. Safety-certified (IEC61508, ISO26262) micro-controllers

parity bit is a prime example of a simple protection scheme, however, it comes with drawbacks as it can only detect odd-numbered bit errors (single, triple, etc.) in the protected word. In addition, using parity bits only allows the detection of the error, and and without a proper safety architecture, memory recovery is practically impossible.

For example, the STM32L4 series and some MCUs of the STM32Fx series from ST Microelectronics have parity protection for 25% of their memory in addition to EDAC codes for flash memory. When an error occurs in protected memory, an interrupt with high priority is activated on the CPU side. In this way the error is detected, but there is no way to recover it. Advanced MCUs have additional ECC protection for SRAM memories. For example, in the case of Cortex R4 based CPUs, the EDAC encoding/decoding is done by the CPU (in-built), whereas in the case of Cortex-M3 and ARM7TDMI-based CPUs, the ECC encoding/decoding is done by the RAM wrapper. The main advantage of having the encoding/decoding within the CPU is to speed up memory access by removing the time-consuming SECDED block. SECDED is based on the Flash/RAM technology design of the controller and is adapted accordingly. Some designs have two SECDED modules that operate in parallel. The results are then compared and accepted only if both are the same [37].

III. Prevailing Safety Architectures

Safety-critical systems often adopt an N-modular (where $N > 2$) architecture. The components exist in certain redundancy and perform the same computations in parallel. The correct result is established based on majority voting. If one of the modules fails, the majority voter masks the fault by identifying the result of the remaining fault-free modules [7]. Although N-modular systems can achieve a higher SIL level, as they provide inherent fault tolerance and consequently a low failure rate, many safety systems use simple architectures such as 1oo1D and 1oo2D [20]. The main advantage is that they have a good balance between functional safety (i.e., achieving a high safety level) and development process costs.

In 1oo2D architectures, all hardware including sensor inputs is independently implemented twice. This leads to a multi-core architecture similar to the one described in [38]. The output of these parallel lines is checked and selected by a voter [39]. For a safety system, it is quite often not important if the final result (chosen by the voter) is correct, as long as it is safe. In case the two outputs differ, the result leading to a safe and non-critical state is preferred and opted for by the voter.

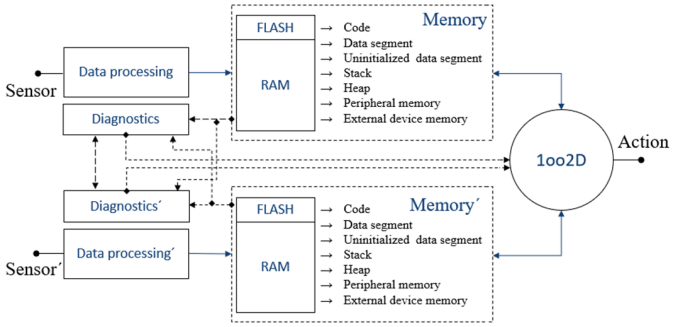

Figure 8. Memory model in 1oo2D architecture

For memory, a 1oo2D architecture provides independent memories for each parallel line of the computing system. Two independent parallel memories ensure system hardware and software redundancy. This means that besides memory-specific data which is required for synchronization, identical data can be found on both memories (Figure 8 depicts the memory model in a 1oo2D architecture). Data in mixed-critical memories can be categorized into safety-relevant and safety non-relevant data. The different criticality levels of data in combination with duplicated memory in the 1oo2D architecture are shown in Figure 9.

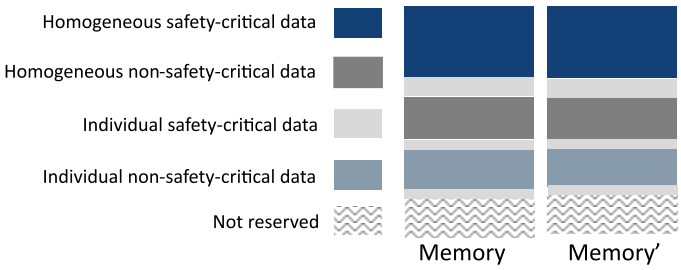

Figure 9. Example of the mixed critical memory in the 1oo2D safety architecture

All regions are equally exposed to faults, however, different forms of protection can be applied to different regions. Experts advise that protection should be implemented in the form of periodical test runs over data. As a guide, we refer the reader to the Safety manual [40] provided by STMicroelectronics for their micro-controllers. To enhance the coverage of hard errors on SRAM, detection tests like Galloping [41] or March classes [42] have been proposed.

For soft errors, STMicroelectronics advises redundancy to be implemented for all safety-relevant variables. Typical solutions provide a copy of original data on the same memory chip or on an additional (redundant) chip. The copied data is periodically compared to the original, in order to detect the presence of errors [43]. If an error is detected, it is not clear which memory (or part of the memory) is affected. Hence,

such a solution leads to detection but not a correction of the soft error and will result in the system transitioning into a safe-state.

As we have seen in Figure 1, the number of soft errors shows a negative correlation with the size of the underlying transistors, leading to a rapid increase of soft errors. In the context of automation, this means that systems are more likely to go into safe states that can disrupt or stop the automation process. These unwanted halts affect the availability of the system [44]. A solution for overcoming this problem is to add mechanisms on top of the existing architecture, which allow for the recovery of faulty data and to extend the on-line time of the system. Recovery mechanisms in this context are usually ECC based. As outlined before, ECC is mostly hardware-based and requires additional time and additional hardware for computing. It is not feasible to extend existing brownfield automation systems with additional hardware, as this would lead to the need for a complete redesign of the system. Another option is to apply software-based ECC approaches, which are complex and expensive in terms of computation.

Given that 1oo2D already provides the possibility to detect memory errors, the question arises how existing architectures (i.e., 1oo2D) can be combined with other approaches that allow correction of detected soft errors and exhibit very little overhead. In addition, these methods should be flexible in terms of their configuration, to enable their application in the aforementioned mixed-critical scenarios where safety-critical data requires more detailed monitoring.

From previous discussions it seems obvious that a solution enables better memory error detection and correction strategies for existing automation products must take the best of two worlds, i.e., utilizing the properties of the underlying architecture to the fullest extent and combining them with flexible software-based soft error correction methods which show little overhead and can be adjusted in terms of mixed-criticality of the prevailing system.

IV. Challenges in mitigating soft errors

To overcome soft errors and consequently lower their impact on the non-functional properties of a system, various methods for error detection, correction, and mitigation were introduced. As already stated in the previous section, the available methods can be divided into hardware- and software-based correction mechanisms. Hardware-based mechanisms provide error detection and correction on an architectural level and use specific hardware. Hardware approaches are not applicable in the brownfield, i.e., existing devices or systems, and usually involve redesign and redeployment. For brownfield systems or devices, software solutions fit better because they can be implemented with a simple update or software patch and consequently minimize costs. Software-based correction mechanisms operate on the memory itself without altering the underlying hardware or architecture. Depending on the application, adequate correction quality is required. Quality denotes the fault magnitude that the strategy is capable of detecting, mitigating, and/or recovering. Given that there is no such thing as a free lunch, soft error strategies require additional execution time and/or memory space, and therefore affect processor run-time and can cause increased memory overhead. On the other hand, hardware-based strategies are more reliable, more powerful, and faster when it comes to computing EDAC codes.

These observations lead to a general trade-off problem for the design and deployment of error detection and correction, as it is always required to balance the quality of detection (required by the underlying application) and the resources required to implement appropriate correction and detection strategies. Higher quality error correction requires more computation time, more memory capacity, and sometimes additional hardware. Depending on the target system, this might lead to a violation of the system's requirements in terms of cost, available memory space, or computation time of the system's applications. In the following, the system requirements are outlined in more detail.

1) Run-time performance: The development of methods, which provide sufficient error coverage, while keeping the impact on a system's run-time or memory overhead minimal, is particularly important in the context of safety-critical systems. This is due to the fact that such systems have very strict timing requirements (i.e., norms in the field define specific timing limits, such as Fault Tolerant Time Interval (FTTI) (see Figure 10) in ISO26262 or Process Safety Time (PST) in the IEC61508 standard). The FTTI constitutes the time-span between a fault and the hazard which results from it [36], [35].

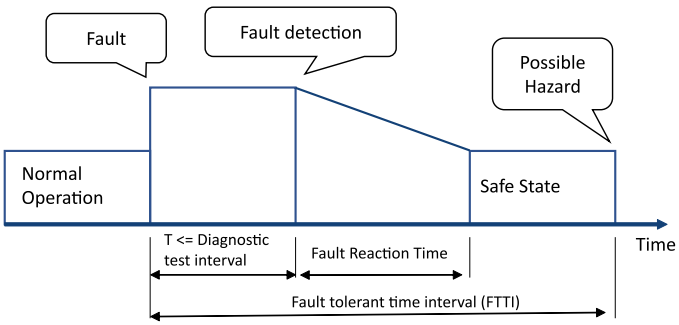

Figure 10. Fault reaction time and Fault Tolerant Time Interval (FTTI) [35]

Faults must be detected and corrected within this interval. If a correction is not possible, the system must be guaranteed to reach a safe state within the FTTI. Therefore, the run-time performance of correction strategies plays a crucial role in the context of safety-critical systems, as its application must not lead to a violation of these FTTI requirements. For example, when using software calculated EDAC codes, the computation time required to calculate redundant bits needs to be evaluated and taken into consideration. If additional hardware is calculating redundant bits, it will increase memory access. This time will probably not significantly affect overall run time, but engineers need to be aware of it [15].

2) Memory consumption: Many software-based strategies require additional memory space for their implementation, which is used to store copies of data or code, or additional information required by the method, such as Parity bits or EDAC. Compared to a similar software solution, EDAC codes exhibit the smallest overhead. The ratio between additional bits required for protection and protected bits is always less than one in EDAC, whereas this is not the case for full redundancy. While in most cases EDAC codes can have a large memory footprint, parity bits constitute their most lightweight form. They allow monitoring of the consistency of a memory region with a defined length based on a single bit, which

denotes whether the number of one-bits in the region is odd or even. Decreasing the size of the protected region can lead to increased memory overhead. To give an example: the protection of a 32-bit word via Hamming code will result in a 3.15% memory overhead. One-bit recovery of a 32-bit word, using Hamming code, would require an additional 7 bits and result in a memory overhead of 22%. The EDAC calculation always requires additional hardware components that will do the calculations and store the calculated bits. Different redundant architectures also require additional components, or entire multiplied systems as is the case with 1oo2 or 2oo3 safety architectures.

3) Mitigation quality: The quality of a strategy is defined by its capability to detect and correct (i.e., recover from) faulty bits. A system's detection and correction capabilities are reflected in the number of faulty bits that can be detected and corrected. The simplest EDAC code (Parity) can detect all odd-numbered bit flips but does not provide recovering capabilities. A 2oo3 system can detect and correct all bit flips, but its complexity and consequently costs are much higher. A short overview of the quality for some mitigation strategies is given in Figure 11

Type	Detection	Correction	Safety	Availability	Complexity/Cost
Parity Bit	Yes[1]	No	Yes	No	Low
SECDED	Yes	Yes (1bit)[2]	Yes	Yes (1bit)[2]	Medium
DECTED	Yes	Yes (2bit)[3]	Yes	Yes (2bit)[3]	High
1oo2	Yes	No	Yes	No	Medium
NooM (M>=N>1)	Yes	Yes (All)	Yes	Yes	High
1oo2 + parity	Yes	Yes(All)	Yes	Yes	Medium
Shielding	-	-	Yes	Yes	High

[1] Yes in the case of odd number of bit-flips
[2] Yes in the case of single- bit flips
[3] Yes in the case of single- and double-bit flips

Figure 11. Mitigation strategies and quality parameters

In fail-safe systems, detection of an error is usually reflected with the safety feature because detection is enough to trigger activation of the safe state, which prevents further safety issues. Between error detection and safe-state activation, the system has a defined allowed time for recovery. If recovery is not possible for any reason, the system will transition into the safe state and its availability will be affected.

4) Mixed criticality: Safety-critical applications usually exhibit different levels of criticality in terms of their underlying data. While a fraction of data is system critical (i.e., if affected by an error the consequences can be catastrophic), errors affecting non-critical data will not impact the safety of operation. This phenomenon is known as mixed-criticality [45]. Incorporating mixed-criticality into the design of mitigation strategies, by devising and applying different detection and correction strategies on memory areas holding data of different levels of criticality, allows further improvement of a system's availability while guaranteeing a correct treatment of system-critical events [45]. While adequate protection needs to be provided for the whole system, safety-critical data requires stronger protection. Several recent studies have investigated mixed-critically in memories, with a focus on data delivery and prioritization according to data criticality [46].

Taking mixed-criticality into account when designing memory detection and correction strategies allows the reliability and safety of the underlying system to be enhanced, as such strategies aim to increase the protection of safety-critical memory parts. By defining different parts of memory to have different criticality, the overhead of correction strategies can be reduced, in contrast to applying rigid correction/detection strategies to the entire memory. In addition, incorporating mixed-criticality can increase a system's availability, as faults in non-system critical memory areas will not necessarily lead to a halt of the system. Figure 9 shows the example of mixed critical memory in the 1oo2D safety architecture.

5) Frequency of access: One interesting phenomenon that can be discussed in the context of protecting memories is access frequency. Two classes of memory access can be distinguished here: low-frequency and high-frequency memory access [47]. Memories with high frequency are more general purpose and can be updated several times per execution cycle.

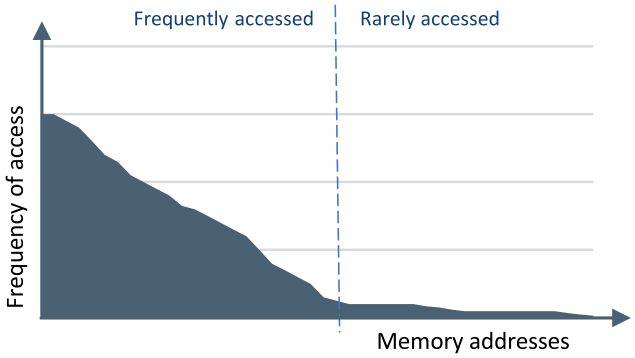

Figure 12. Sketch of potential memory usage profile

The parts of the memories that have lower access frequency usually include on-demand or periodically accessed data, with large time intervals between consecutive accesses. Memories used on-demand could, for instance, store the address of a function that takes the system to the safe-state. As safe-state activation does not happen often, the function will remain unused for long periods of time and thus will not be tested often. Nevertheless, it must always be available. The accumulation of soft errors on these resources can be of high relevance, for example, when the system needs to comply with specific normative requirements (e.g., SIL3 according to the IEC61508 standard [36]). Figure 12 depicts an exemplary memory usage profile. To obtain a realistic memory usage profile, a safety-critical device must be analyzed, as memory usage depends on the applications.

To give an example let us imagine a system with hardware-integrated parity bit protection. Parity bit protection can only detect odd bit flips (single, triple, etc.) without correction, and detection is only triggered when the protected part of memory is accessed. In the context of rare access, the possibility exists that this part of memory experiences more than two separate bit flips between two accesses (protection activation). This can lead to errors going undetected at the next access and can lead to an unsafe-state of the system. The explained scenarios and the effect of accumulation are shown in Figure 13. In sequence (a), an error will be detected, because the parity bit will not respond to the data, while in sequence (b), the test will not detect an error in data because the calculated parity bit responds to the data. This second phenomenon is called the accumulation of error.

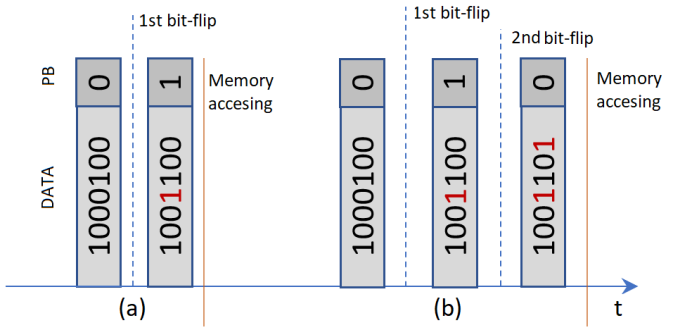

Figure 13. Sequence of the events: (a) when error will be detected , (b) when error will go undetected

6) Memory organization: Due to environmental changes, occurrences of soft memory errors are not continuous, and the chance of a cell being hit by an error is randomly distributed. Errors can appear at any time and in any type of memory or memory part, which can aggravate protection and detection mechanisms as they are type-dependent. One can distinguish between two types of memory in embedded systems: non-volatile and volatile memory. Non-volatile memory sustains stored information during a loss of power (e.g., flash memory), while volatile memory requires constant power to retain stored information (e.g., SRAM) [48].

Embedded memories exhibit various regions: program memory, data memory, registers, and I/O ports [49]. From a software point of view, the memory layout of C/C++ programs consists of the different sections that are saved in different memory regions. Typical memory representations of C/C++ programs consist of a code segment, data segment, uninitialized data segment (bss), stack, and heap. All of this can impact the design of correction/mitigation mechanisms.

7) Availability vs Safety: Safe-state activation often leads to a functional degradation of many system components, and as it often results in a system halt it is associated with high costs. It decreases the availability of the system to ensure the safety of the system and its environment.

Especially in production lines, where every minute without service is associated with high costs, a system's availability is of utmost importance. However, a highly available system is costly, because it demands complex redundant architectures. Therefore, a trade-off between safety and availability exists, that needs to be optimized. One way to increase availability while keeping functional safety on the demanded level is to postpone or avoid the unnecessary activation of safe-states. In [44] the concept of Predictive Fail-safe was proposed, which aims to increase a system's availability by applying data analytics on safety-relevant data to predict and prevent future failures.

8) Usability: Soft errors have been a focus of research for the past 60 years. Although many approaches have been introduced and tested with good results, only a few have found their way into real-world applications. This is due to the associated required resources (e.g., computation power and time or memory consumption), which limits their applicability. The strategies outlined in Section II-E are the only ones that are currently applicable given the performance capabilities of available micro-controllers and embedded memories. Usability in this context is defined as a quality attribute that assesses how easily a mitigation strategy can be implemented. Usability addresses questions related to integration, including the following: What do developers need to do to successfully configure and deploy a chosen mechanism? What is the limitation of the strategy, and is it possible to define the part(s) of the system that needs to be protected? Hence, the usability parameter of the strategy depends on three factors:

- The base system/device's properties.
- The requirements/limitations of the strategy.
- The non-functional requirements of the user.

For example, DECTED (Double Error Correction, Triple Error Detection) EDAC codes [7] show very good performance against soft errors, but integrating such approaches (hardware or software) into devices requires additional computational power as well as additional computing time, which are usually both limited in COTS devices.

V. Evaluation of Strategies

As shown in Section IV, it is crucial to estimate the performance and overheads of soft error mitigation strategies in order to identify appropriate strategies for one's problem domain given the underlying system requirements. This section demonstrates how to evaluate potential techniques in the context of an existing 1oo2D safety architecture. The 1oo2D safety architecture is considered fixed, and the goal of the evaluation is to provide the means to enhance this existing system regarding soft error mitigation.

The least complex solution (demanding only effort and time and no additional hardware) is to apply a software-based mitigation technique. However, the problem with software-based approaches is their memory consumption and computation time requirements as well as complexity of implementation.

Given an existing architecture which already provides certain features (e.g., 1oo2D safety architectures inherently provides redundancy, that can detect but not correct errors), a hybrid approach can be taken. In such an approach a system's existing features (e.g., error detection) are complemented with additional software-based mitigation techniques to achieve increased fault-tolerance (e.g., providing a correction mechanism to complement 1oo2D detection mechanism).

In the following, an analysis of the mitigation strategies explained in Section II will be presented, along with new ideas that utilize existing peripherals of the micro-controller. Techniques will be explained top-down to outline system safety-enhancement prospects for designers involved in different development stages of the system. In addition, the top-down order reflects the amount of effort required to implement a strategy in the system, as alterations in earlier phases might require a system redesign.

A. Software-based techniques

These techniques belong to the deployment phase according to Section II. While they do not require additional hardware per se, their overhead can affect the system's performance. To choose an appropriate strategy requires the comparative assessment of potential strategies. This section demonstrates how such an assessment could be performed via an exemplary calculation and comparison of memory consumption and runtime performances using the example of Parity Bit (PB)

and Extended Hamming Code (EHC). A similar approach can be used when assessing other software-based mitigation strategies.

The evaluation is performed for varying lengths of protected data, as strategies scale differently with different lengths. For the representation of the codes, a common annotation (n, k) is used, where n denotes the number of total bits and k the number of protected data bits. The number of required check bits can be easily calculated as $n - k$. Utilizing these parameters, memory consumption (mc) is calculated in (1) and exhibited in Figure 14.

$$mc[\%] = (n - k)/k \cdot 100\% \quad (1)$$

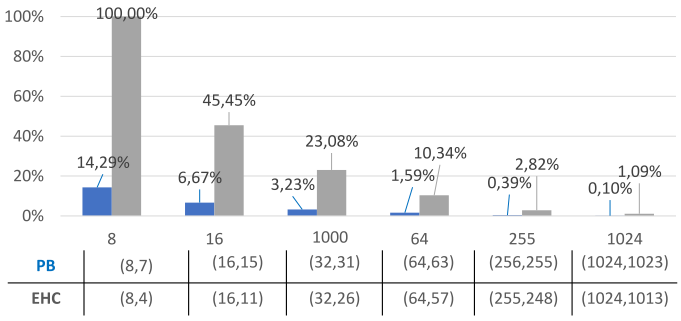

Figure 14. Memory overhead for different types of Parity Bit (PB) and Extended Hamming Code (EHC), where the x-axis denotes the total length of the word and y denotes the percentage of the memory overhead.

The run-time performance of a given strategy is closely connected to the complexity of the underlying algorithm. A good indicator of an algorithm's complexity is the number of logical XOR operators it requires for implementation.

In the context of PB, a calculation stemming from [50] was used. The algorithm is based on the consecutive application of shift and XOR operators. Alternatively, a lookup table could be used to calculate the parity bits of 8-bit words. While using a look-up table will slightly increase the memory consumption of the algorithm, it will decrease its complexity by 3 XORs.

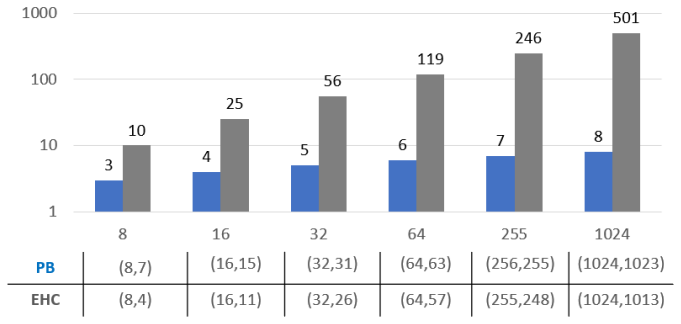

Figure 15. Number of XORs for encoding process for different types of PB(n, k) and EHC(n, k), where y-axis denotes the number of total XORs gates and the x-axis the number of the protected data bits.

Equation (2) was used to calculate the number of the XORs gates for EHC.

$$XORs(k) = 2^{k+1} - k - 3 \quad (2)$$

where parameter k can be derived from the following form of Hamming code annotation $H(2^k, 2^k - k - 1)$. Equation (2) stems from [50], where it was calculated for the EHC recursive encoding computation. Figure 15 shows the number of XOR operators for varying lengths of protected bits.

PB and EHC differ significantly in terms of mitigation quality. While PB is only capable of detecting odd numbers of bit-flips errors (including single-bit errors), EHC can detect double-bit flips errors and correct only single-bit errors. In the context of safety-critical systems, this low mitigation quality will have a big impact on availability and safety.

In [51], a detailed report is presented on the number of soft errors in SRAM memory (512K x 8-bit) that were observed in space. Errors were recorded in a nanosatellite circulating the Earth's orbit. During the 2510 days of recording a total of 247593 soft errors occurred, which could be categorized into four types. The majority of the errors (i.e., a total of 244150 errors constituting 98.6% of the recorded errors) fell in the single-bit error class. Only 2996 errors (i.e., 1.21% of the recorded errors) constituted double-bit errors. Multiple bit (> 2) errors occurred at an even lower rate (corresponding to a total of 217 errors (0.08%)), while the remaining errors (230 (0.09%)) were classified as severe errors.

Let us consider the capability of the algorithms under test (PB and EHC) for this recorded error distribution. PB would detect all single-bit errors and some of the multiple bit errors, leading to a detection rate of 98.75%. PB detection alone is not enough and would not increase the availability of the system, because without recovery the sole identification of an error would lead to the system being put into a safe-state. Using EHC, 99.8% of errors would be detected and 98.6% would be corrected. This means that the system's availability could be increased significantly as it would only be stopped (put in a safe-state) for 1.4% of the errors. This leads to the conclusion that (on its own) EHC is significantly better when it comes to safety and availability, however, this can be associated with the higher memory overhead and complexity (as shown before). Furthermore, one should keep in mind that the SRAM used was relatively old (approximately 20 years), and thus, exhibits a lower probability for multiple bit errors because of the higher technology node in use. With newer memories utilizing smaller technologies, the distribution of the error is very likely to be different (i.e., more multiple-bit errors are to be expected).

In the context of safety-critical systems, the application of specific fail-safe architectures with hardware redundancy is very common. The next section will introduce a widely used fail-safe architecture and show how the application of simple EDAC codes can further improve a system's availability.

B. Hybrid techniques

While the previous section investigated purely software-based strategies, another option to increase a system's fault tolerance is to actively integrate the underlying architecture and components together with a software strategy.

In $1oo2D$ architectures with redundant memories (Section II-B), if an error appears it is not clear which memory was affected. Therefore, an error can be detected but not corrected and it will result in the system transitioning into a safe-state. A solution for overcoming this problem is to add mechanisms on top of the existing architecture that allow the recovery of faulty data and to extend the up-time of the system. Recovery mechanisms in this context are usually EDAC code based.

Adding additional hardware to the system is not feasible, as this would require redesigning the system, and an alternative option is to apply software-based EDAC code approaches.

1) Software Redundant parity: Given that 1oo2D already provides the possibility to detect memory errors, the question arises how existing architectures (i.e., 1oo2D) can be combined with software-based approaches.

A method for enhancing existing 1oo2D hardware architectures was proposed in our work [52]. This method constitutes an extension for mixed-critical real-time systems with an underlying 1oo2D architecture. We refer to it as Redundant Parity (RP). Figure 16 explains the basic concepts of the RP method. The method relies on 1oo2D's ability to detect soft errors and uses parity bits to establish the location of the error. Initially, the method generates parity bits for data that need to be protected (i.e., data in redundant memories). When bit flips occur and the 1oo2 comparator detects different bits in redundant data, the usual consequence is to generate a signal that will trigger the safe-state of the device. In contrast, our proposed RP method calculates new parity bits for both protected parts of the memories. In the next step, old parity bits are compared to newly calculated parity bits to establish the fault source. If the algorithm distinguishes between healthy and faulty data, the recovery phase is activated. Recovery is performed by simply overwriting the faulty data with the healthy data. Summarizing, the method uses the inherent capability of the 1oo2D architecture to detect bit flips. With the additional parity bit, the faulty redundant words can be determined and by means of redundancy, recovery is possible.

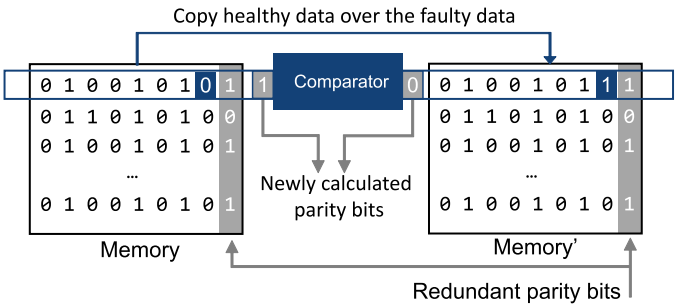

Figure 16. Redundant Parity method.

The method enables the correction of single-bit soft errors, which constitute the majority of soft-errors that occur. Odd multiple bit soft errors can also be corrected and even multiple bits can be detected. In the context of the recorded error data presented in Section V, this method would detect 100% of the errors and correct 99.4% of them. Memory overhead would be doubled and complexity would increase by twice the complexity of the parity bit.

Furthermore, the RP method provides separate detection and recovery phases, leading to less recovery time than in other EDAC methods. In addition, the proposed method is completely independent of the software architecture as it focuses on the memory's word level rather than on variables or structures [44]. However, the results also show that the application of the approach is limited to a 1oo2D architecture, which already provides the required data redundancy as well as self-tests to detect errors in the data.

2) Hardware redundant parity: As mentioned in section II, several affordable MCUs already integrate parity checks for SRAM memory. If HW-based parity checks are available, the Redundant Parity (RP) method explained in Subsection V-B1 could be implemented even more easily. This would help to overcome the main drawback of the RP method, i.e., an on-demand software-based calculation of the parity bit whenever a protected word is accessed in memory. Given the appropriate hardware, the calculation could be done automatically, minimizing the impact on the existing code. Using dedicated hardware would also relieve the CPU of the calculations required by PB. In case of discrepancy detection between redundant memories, the parity bit can easily be accessed and compared with the newly calculated parity bit, allowing a fast recovery procedure (i.e., overwriting a healthy over the faulty word) to be performed. While several MCUs (e.g., the STM32L4x MCU family) already provide inherent parity calculations, they often do not allow direct access to the calculated parity bits, which are calculated and saved internally. The only information provided by the system is a highly prioritized interrupt to the CPU, without any information about which of the memory addresses the error occurred in. A potential solution would be to scan the entire memory, but this not acceptable due to timing reasons and it would also defeat the purpose of using hardware.

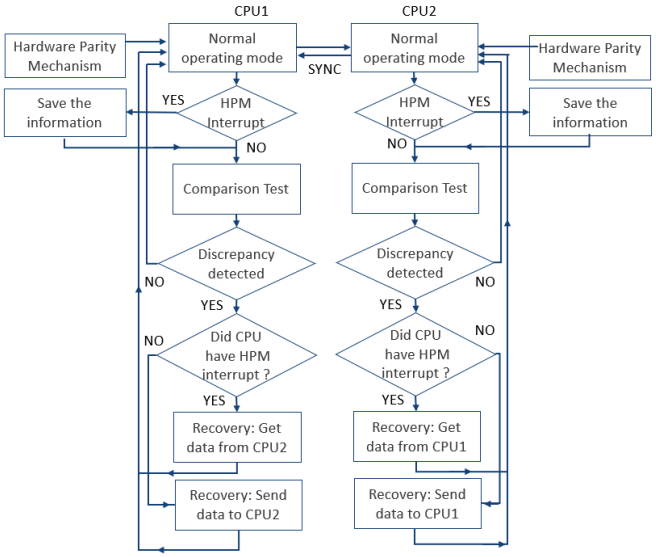

Figure 17. Flow chart diagram of Hardware Redundant Parity.

These limitations can be overcome with the following approaches: consider that a Hardware Parity Bit Mechanism (HPBM) detects an error in 1oo2 redundant memories. The CPU with the memory error will receive an interrupt. The information is saved, and CPUs continue with normal operations. Later, the 1oo2D comparison test detects a discrepancy between the redundant memories and its exact location. With the previously stored information about the location of the faulty memory (i.e., the saved interrupt), the fault can be pinpointed to one memory. If the interrupt is received on the CPU1 side, then data from CPU2 can be copied over the data of CPU1, otherwise, if CPU2 got an interrupt then data from CPU1 will be transferred to CPU2. If one of the CPUs gets more than 1 interrupt in the interval between two comparison tests, the safe-state should be activated, because the latest information about the faulty side will be wrong. Also, the safe-state will be activated if both CPUs receive an interrupt.

The flow chart diagnosis of this Hardware Redundant Parity algorithm is shown in Figure 17.

In contrast to its software-based predecessor, the approach does not require additional memory. Additionally, its complexity decreases as the overall code does not need additional changes, in contrast to the software RP algorithm, where the code to calculate parity bit needed to be inserted at every write request. The complexity of this approach is therefore very simple since no costly computations and no additional time is required, so overall system run-time is unaffected. In the context of safety, the functional safety assessment of the resulting system is easier. In other words, validation of the concept and showing that it has no false positives or false negatives is easier than in previous cases.

3) DMA based recovery: Direct memory access (DMA) is a method that allows an input/output (I/O) device to send or receive data directly to or from the main memory, bypassing the CPU to speed up memory operations. The process is managed by a chip known as a DMA controller (DMAC). The following mitigation strategy utilizes DMA method capabilities to protect memory with minimum changes to the operating code of the system.

Assume that a comparison self-test is done in slices as explained before. In the beginning, a copy of the safety-critical data is stored in the spare memory via DMA. As a result, both CPUs will have an original and a copy of the original safety-critical data. When an error occurs, i.e., a bit flip on the CPU1's memory, a comparison test will detect a discrepancy between the original data of two memories. Usually, this would lead to a safe-state but in this approach, recovery is possible and the safe-state can be avoided. After a discrepancy between memories is detected, each CPU starts a local self-test, comparing the original with the copied data. If the locally compared data is equal for CPU1 then data on CPU1 is intact and we can assume that the faulty memory is on CPU2. The recovery can be achieved by simply overwriting faulty data (CPU2) with healthy data (CPU1). If the locally compared data is not equal, then the assumption is that further corruptions occurred, and therefore, a safe-state will be activated. In general, the DMA method will theoretically cover all 1-bit errors. As shown in the example memory usage profile (Figure 12), although it is not possible to cover everything, a significant part of the memory will be covered. The previously described method's behavior for recovery and safe state handling, is depicted in Figure 18. The method can be applied in the same manner to CPU2.

The drawback of this approach is that additional memory is needed to hold copies of the data. This approach has a minor effect on the code because it only requires the configuration of the DMA and implementation of the recovery routine. The effect on the overall run-time is minimal because copying a few slices of the data should not have a significant impact. This method is not restricted to specific parts of the memories as in the case of HW redundant parity. Additionally, DMA is now a standard method that is included in most MCUs, therefore, it is not dependent on the MCU type.

CPU1		CPU2		State	Result
Copy	Original	Original	Copy		
0	0	0	0	OK	No failure
0	0	0	1	OK	No failure (copy was modified, but is allowed so)
0	0	1	0	RECOVER	Failure (Local test at CPU2 detects that CPU2 has a failure, so it needs data from CPU1 for recovery
0	0	1	1	SAFE	Failure (Fault assignment not possible -> Transition into Safe State) => availability loss
0	1	0	0	RECOVER	Failure (Local test at CPU1 detects that CPU1 has a failure, so it needs data from CPU2 for recovery
0	1	0	1	SAFE	Failure (Fault detection in both CPUs ->Transition into Safe State) => availability loss
0	1	1	0	OK	No failure
0	1	1	1	OK	No failure (copy was modified, but is allowed so)
1	0	0	0	OK	No failure (copy was modified, but is allowed so)
1	0	0	1	OK	No failure (copy was modified, but is allowed so)
1	0	1	0	SAFE	Failure (Fault detection in both CPUs ->Transition into Safe State) => availability loss
1	0	1	1	RECOVER	Failure (CPU1 local test detects that CPU1 has a failure, so it needs data from CPU2 for recovery
1	1	0	0	SAFE	Failure (Fault assignment not possible -> Transition into Safe State) => availability loss
1	1	0	1	RECOVER	Failure (CPU2 local test detects that CPU2 has a failure, so it needs data from CPU1 for recovery
1	1	1	0	OK	No failure (copy was modified, but is allowed so)
1	1	1	1	OK	No failure

Figure 18. States of "DMA based recovery" operation within a 1oo2 memory architecture, regarding different perceived errors in CPU1 and CPU2's original data and copy data segments

C. Built-in hardware techniques

If none of the previous two categories fulfill the requirements for memory protection, then a redesign of the system should be considered. In this case, micro-controllers with built-in protection techniques should be used from the early design stages. These techniques are explained in Sections II-C and II-B. However, these techniques also have associated quality attributes, and thus, limitations have to be considered. For example, parity bit protected memories have a low mitigation quality (detection only), while ECC-Hamming code protected memories are better in this respect. But in some cases, run-time is affected or only some parts of the memory are protected. In general, when using built-in hardware, techniques will guarantee a better mitigation process but on the other hand, we are getting away from COTS MCUs and heading to safety-certified MCUs that are far more expensive. Nevertheless, as we stated in Section II-E, there are already some COTS MCUs availabe with built-in protection mechanisms. With advances in production techniques, we expect that the number of COTS MCUs with integrated measures will increase.

VI. Conclusion

With decreasing transistor sizes, soft errors induced by external environmental factors increasingly constitute a problem for memory operation and provide challenges to ensuring a system's safety and availability.

The main goal of this work was to review mitigation strategies for 1oo2D safety architecture, which are applicable in different development phases of a system, as well as to identify the challenges which need to be considered during the design of soft-error mitigation strategies.

Today, several safety certified MCUs with integrated measures against radiation can be found on the market. COTS MCUs, on the contrary, are not always equipped with such protection measures and often only utilize the most simple protection techniques. As safety certified micro-controllers are becoming more expensive, industry often utilizes COTS micro-controllers in different safety architectures. These architectures rely on redundancy, i.e., the multiplication of systems, which can lead to even more expensive production costs and an increase in the overall system's complexity. Therefore, there is a need for solutions that utilize simple safety architectures together with additional techniques built on top of existing available architectures. Such approaches intend to keep safety at the demanded level but at the same time increase availability and reliability with minimal additional costs.

To increase availability and reliability within COTS memories, a certain level of fault tolerance is required. Current safety-critical applications rely on simple fail-safe architectures such as 1oo2D. The reliability and availability of fault-tolerant systems can be further improved, if such architectures are extended with additional software-based recovery techniques such as EDAC codes, which does not require additional hardware or a redesign of the underlying architecture.

As demonstrated in Section V potential mitigation strategies can be evaluated in terms of their overhead and complexity, as well as the different system development phases they apply to. Such a categorization of strategies highlights their individual cost and requirement trade-offs, their limits, and allows for the identification of suitable methods for specific application scenarios (e.g., when retrofitting existing brownfield automation devices).

When deciding on a method to be implemented on existing hardware, one must be aware of the associated overhead costs, as it will likely increase run-time and/or reduce available memory space. This aspect can be incorporated in strategy design by directly addressing mixed-criticality of data within correction and detection strategies, and differentiating among memory regions. This article demonstrated how such an assessment could be performed, by calculating and comparing memory consumption and run-time performances of different strategies, which can then be linked to the existing requirements of existing safety architectures, such as 1oo2D.

Software-based measures are rather difficult to use as they require implementation and integration into an existing system. If there is no other option, however, software-based measures must be implemented. In this case, two points should be considered: i) The usage of redundancy or coding theory (EDAC codes), where parameters such as quality and overhead (see Section IV) need to be taken into account. ii) The implementation has to be targeted at towards the usage profiles of the memory. Taking these profiles into account helps to reduce memory overhead and reduce implementation and integration overhead.

A thorough analysis of chosen strategies that are to be deployed in industrial controllers must be planned, in order to i) identify their limitations in the context of the system and ii) analyze the overall effect of the methods on the system regarding the associated challenges (see Section IV). Moreover, a detailed evaluation of a strategy's impact on a system's availability and reliability must be investigated in detail.

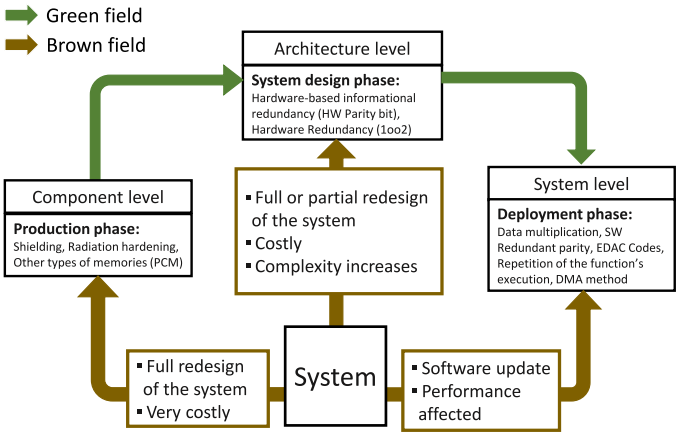

Figure 19. Deployment of mitigation strategies for greenfield and brownfield devices

A summary of this study's findings is presented in Figure 19. The green line presents different ways to mitigate soft-error for the different stages of system development. This option concerns greenfield systems (i.e., when designing a system from scratch). The brown line, on the other hand, represents options relevant to implementing additional soft-error mitigation strategies for brownfield devices (i.e., existing systems). Three approaches are possible for retrofitting brownfield automation. One is a complete redesign of the system, including measures such as shielding, hardening, or the selection and application of different, more resilient types of memory. This option might require more time and costs than expendable for an existing system. The second approach concerns a partial redesign, by adding additional components that increase the redundancy of the system. Although this approach is less expensive than a complete redesign, it is still associated with significant costs and effort. The last approach is to deploy software-driven approaches. While this approach is associated with the least costs, it requires extensive testing of non-functional parameters in order to make sure that the strategies are indeed applicable in the system context.

ACKNOWLEDGMENT

The authors gratefully acknowledge the support of the Austrian Research Promotion Agency (FFG) (#6112792).

REFERENCES

[1] A. Kajmakovic, K. Diwold, N. Kajtazovic, and R. Zupanc, "Challenges in mitigating soft errors in safety-critical systems with cots microprocessors," in PESARO 2020, The Tenth International Conference on Performance, Safety and Robustness in Complex Systems and Applications. IARIA, Feb. 2020, pp. 13–18.

[2] J. Vankeirsbilck, H. Hallez, and J. Boydens, "Soft error protection in safety critical embedded applications: An overview," in 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC). IEEE, 2015, pp. 605–610.

[3] H. Iwashita, "International standards adopted by ITU-T to address soft errors affecting telecommunication equipment," ITU-T International Telecommunication Union - Telecommunication Standardization Sector, Geneva, CH, Standard, 2018.

[4] H. Forsberg and K. Karlsson, "COTS CPU selection guidelines for safety-critical applications," in 2006 IEEE/AIAA 25TH Digital Avionics Systems Conference, Oct. 2006.

[5] V. THATI, J. Vankeirsbilck, J. Boydens, and D. Pissoort, "Data error detection and recovery in embedded systems: a literature review," Advances in Science, Technology and Engineering Systems Journal, vol. 2, no. 3, 2017, pp. 623–633.

[6] M. Duncan and P. Roche, "Paving the way towards autonomous driving—tackling soft errors to security challenges," in 2017 IEEE International Reliability Physics Symposium (IRPS), 2017, pp. 2E–1.

[7] D. Elena, Fault-Tolerant Design. KTH Royal Institute of Technology, Krista, Sweden: Springer, 2013.

[8] A. Mukati, "A survey of memory error correcting techniques for improved reliability," Journal of network and computer applications, vol. 34, no. 2, 2011, pp. 517–522.

[9] E. Fujiwara, Code Design for Dependable Systems: Theory and Practical Application. New York, NY, USA: Wiley-Interscience, 2006.

[10] S. Jeon, E. Hwang, B. V. Kumar, and M. K. Cheng, "LDPC codes for memory systems with scrubbing," in 2010 IEEE Global Telecommunications Conference GLOBECOM 2010. IEEE, 2010, pp. 1–6.

[11] B. Tahir, S. Schwarz, and M. Rupp, "BER comparison between convolutional, turbo, LDPC, and polar codes," in 2017 24th International Conference on Telecommunications (ICT). IEEE, 2017, pp. 1–7.

[12] G. Mayuga, Y. Yamato, T. Yoneda, M. Inoue, and Y. Sato, "An ECC-based memory architecture with online self-repair capabilities for reliability enhancement," in 2015 20th IEEE European Test Symposium (ETS). IEEE, 2015, pp. 1–6.

[13] R. Santos, S. Venkataraman, A. Das, and A. Kumar, "Criticality-aware scrubbing mechanism for sram-based FPGAs," in 2014 24th International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2014, pp. 1–8.

[14] M. Restifo, P. Bernardi, S. De Luca, and A. Sansonetti, "On-line software-based self-test for ECC of embedded RAM memories," in 2017 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT). IEEE, 2017, pp. 1–6.

[15] E. Peter and P. Salvatore, "Error detection in sram," Texas instruments, Application Report, 2017.

[16] N. Maruyama, A. Nukada, S. Matsuoka et al., "Software-based ECC for GPUs," in 2009 Symposium on Application Accelerators in High Performance Computing (SAAHPC'09), vol. 107, 2009.

[17] D. Dopson, "SoftECC: a system for software memory integrity checking," Ph.D. dissertation, Institute of Technology. Dept. of Electrical Engineering and Computer Science, Massachusetts, 2007.

[18] Intel® Embedded Memory User Guide, STMicroelectronics.

[19] MWCT101xS Safety Manual, NXP Semiconductors.

[20] F. Handermann, "Process safety architecture system neutral solution comparison," Chemical Engineering Transactions, vol. 48, 2016, pp. 499–504.

[21] R. Mariani and P. Fuhrmann, "Comparing fail-safe microcontroller architectures in light of IEC 61508," in 22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007). IEEE, 2007, pp. 123–131.

[22] S. Poledna, Fault-tolerant real-time systems: The problem of replica determinism. Springer Science & Business Media, 2007, vol. 345.

[23] OpenStax, Chemistry. Rice University: OpenStax, OpenStax Chemistry, 2014.

[24] F.-X. Yu, J.-R. Liu, Z.-L. Huang, H. Luo, and Z.-M. Lu, "Overview of radiation hardening techniques for ic design," Information Technology Journal, vol. 9, pp. 1068-1080, 2010.

[25] H.-K. Lim and J. G. Fossum, "Threshold voltage of thin-film silicon-on-insulator (SOI) MOSFET's," IEEE Transactions on electron devices, vol. 30, no. 10, 1983, pp. 1244–1251.

[26] T. Nakamura, H. Matsuhashi, and Y. Nagatomo, "Silicon on sapphire (SOS) device technology," Oki technical review, vol. 71, no. 4, 2004.

[27] M. Karagounis, D. Arutinov, M. Barbero, R. Beccherle, G. Darbo, R. Ely, D. Fougeron, M. Garcia-Sciveres et al., "Development of the ATLAS FE-I4 pixel readout IC for b-layer upgrade and super-LHC," Proceedings of the Topical Workshop on Electronics for Particle Physics, TWEPP 2008, Jan. 2008.

[28] F. Bedeschi, R. Fackenthal, C. Resta, E. M. Donze, M. Jagasivamani, E. Buda, F. Pellizzer et al., "A multi-level-cell bipolar-selected phase-change memory," in 2008 IEEE International Solid-State Circuits Conference-Digest of Technical Papers. IEEE, 2008, pp. 428–625.

[29] N. An, R. Wang, Y. Gao, H. Yang, and D. Qian, "Balancing the lifetime and storage overhead on error correction for phase change memory," PloS one, vol. 10, no. 7, 2015, p. e0131964.

[30] R. Forchhammer, "Automotive MCUs in28nm FD-SOI with ePCM NVM," STMicroelectronics, 2018.

[31] A.V.Kolobov and J. Tominagaand P.Fons, "Phase-change memory materials," in Springer Handbook of Electronic and Photonic Materials. Springer, 2017.

[32] L. Forbes, "Fully depleted silicon-on-insulator cmos logic," Dec. 14 2004, uS Patent 6,830,963.

[33] V. Technologies, "HARDSIL® integration & component design for foundries," MoPac Expressway, Suite 350, Austin, Texas, 7874, 2019.

[34] Product manual VA108x0, Vorago Technologies.

[35] "Iso 26262 - road vehicles – functional safety, part 1–10. electrical and electronic components and general system aspects," International Organization for Standardization, Geneva, CH, Standard, 2011.

[36] IEC, "International Standard 61508 Functional safety: Safety related Systems," International Electrotechnical Commission, Geneva, CH, Standard, 2005.

[37] F. Nocha, "Ecc handling in tmsx70-based microcontrollers," Texas instruments, Application Report, 2011.

[38] F. Reichenbach and A. Wold, "Multi-core technology–next evolution step in safety critical systems for industrial applications?" in 2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools. IEEE, 2010, pp. 339–346.

[39] C. Preschern, N. Kajtazovic, and C. Kreiner, "Built-in security enhancements for the 1oo2 safety architecture," in 2012 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER). IEEE, 2012, pp. 103–108.

[40] STM32F4 Series safety manual - user manual, STMicroelectronics.

[41] C.-W. Wu, "Chapter 8 - memory testing and built-in self-test," in VLSI Test Principles and Architectures, L.-T. Wang, C.-W. Wu, and X. Wen, Eds. San Francisco: Morgan Kaufmann, 2006.

[42] A. J. Van De Goor, "Using march tests to test srams," IEEE Design Test of Computers, March 1993.

[43] Handling of soft errors in STM32 applications, Intel.

[44] A. Kajmakovic, R. Zupanc, S. Mayer, N. Kajtazovic, M. Hoeffernig, and H. Vogl, "Predictive fail-safe improving the safety of industrial environments through model-based analytics on hidden data sources," in 2018 IEEE 13th International Symposium on Industrial Embedded Systems (SIES). IEEE, 2018, pp. 1–4.

[45] A. Burns and R. I. Davis, "Mixed criticality systems - a review," in Department of Computer Science, University of York, York, UK, 2015.

[46] J. S. Miguel and N. E. Jerger, "Data criticality in network-on-chip design," in Proceedings of the 9th International Symposium on Networks-on-Chip, 2015, pp. 1–8.

[47] L. Botler, N. Kajtazovic, K. Diwold, and K. Römer, "JiT fault detection: Increasing availability in 1oo2 systems just-in-time," in Proceedings of the 15th International Conference on Availability, Reliability and Security, ser. ARES '20. New York, NY, USA: Association for Computing Machinery, 2020.

[48] K. Itoh, "Embedded memories: Progress and a look into the future," IEEE Design & Test of Computers, vol. 28, no. 1, 2011, pp. 10–13.

[49] Reference manual for STM32 applications, Intel.

[50] L. Zhengrui, L. Sian-Jheng, and H. Honggang, "On the arithmetic complexities of Hamming Codes and Hadamard Codes," 2018.

[51] H. Caleb and B. Vipin, "Error detection and correction on-board nanosatellites using Hamming codes," Journal of Electrical and Computer Engineering, 2019.

[52] A. Kajmakovic, K. Diwold, N. Kajtazovic, R. Zupanc, and G. Macher, "Flexible soft error mitigation strategy for memories in mixed-critical systems," in 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). IEEE, 2019, pp. 440–445.