TECHNISCHE UNIVERSITÄT MÜNCHEN

TUM School of Engineering and Design

Safety Critical, High-Performance Systems based on COTS Multicore Processors for Industrial and Aerospace Applications

Lukas Frank Steinert, M.Sc.

Vollständiger Abdruck der von der TUM School of Engineering and Design der Technischen Universität München zur Erlangung des akademischen Grades eines Doktors der Ingenieurwissenschaften genehmigten Dissertation.

Vorsitzender:	Prof. Dr.-Ing. Oskar J. Haidn
Prüfer der Dissertation:	1. Prof. Dr.-Ing. Florian Holzapfel
	2. Assoc. Prof. Peter Chudý, Ph.D.

Die Dissertation wurde am 13.09.2021 bei der Technischen Universität München eingereicht und durch die TUM School of Engineering and Design am 08.02.2022 angenommen.

Abstract

Highly automated systems in the transportation and industrial domains face significant challenges during the design phase of their embedded computing platforms and control systems. First and foremost, certification is challenging, because they will require substantially more computational power for extensive automation. Prohibitive costs for tailored components require the use of off-the-shelf, highly complex system-on-chip devices without specific safety evidence or support by the manufacturer. As of today, the system and component architecture design for very high performance, coupled with stringent safety goals is not well covered and fails to address the rising challenges in industrial and transportation markets. Also, system functions which were previously allocated to a human operator are now gradually automated. These new system functions are considered highly critical in terms of certification and must be implemented with adequate levels of redundancy and fault detection.

To support and enable future generations of highly automated systems, this work provides three important building blocks and introduces an effective approach for certification under current standards and guidelines. We first propose a set of system requirements for future computing cluster generations considering changing requirements and market preconditions. We derived a classification scheme to distribute, allocate, and analyze critical / non-critical system functions by their safety impact level on the physical system. Three different levels are introduced, which can be reused in different application domains, in conjunction with the already existing safety or assurance levels. Second, we contribute two cluster unit architectures, suitable for any system and all criticality levels, and show the integration of proven concepts for software determinism into our novel architectures. The unit and system level concepts address the need for a highly integrated, high-performance platform. They employ varying degrees of redundancy and dissimilarity to support even the highest safety claims in the aerospace domain efficiently and scale down for cost sensitive industrial applications. While our architectures are based on off-the-shelf devices, added architectural safety nets with inherent dissimilarity ensure certification under current guidance and standards. The architectures are analyzed from the aerospace and industrial certification perspective. To this end, we use fault-tree and Markov analyses and a walk-through through current certification processes. A conclusive application example is given, which enables key industries to drive their designs forward.

Zusammenfassung

Hochautomatisierte Systeme im Transport- und Luftfahrtsektor stehen hinsichtlich ihrer Rechnerplattformen für Steuerungs- und Automatisierungsfunktionen vor großen Herausforderungen. Die Zulassung dieser Systeme wird Zusehens aufwendiger, da der enorm steigende Bedarf an Rechenleistung nach immer komplexeren Prozessoren verlangt. Da aus Kostengründen keine anwendungsspezifischen Komponenten zum Einsatz kommen können, ist man gezwungen, auf Standardkomponenten auszuweichen, bei denen die Hersteller keine Sicherheitsanalysen oder die notwendige Dokumentationskette liefern. Heutige Konzepte für die Geräte- und Systemarchitektur bilden dies nicht ab und sind für zukünftige Systemgenerationen ungeeignet. Zudem werden Systemfunktionen, die aktuell als reine Unterstützungssysteme mit niedriger Kritikalität konzipiert sind, durch den Wegfall der menschlichen Eingriffsmöglichkeit und dem steigenden Automatisierungsgrad ebenfalls sicherheitsrelevant, weshalb auch sie mit entsprechenden Redundanzen und Fehlererkennungsmechanismen implementiert werden müssen.

Um zukünftige Steuerungsplattformen für hochautomatisierte Systeme zu ermöglichen, trägt diese Arbeit drei entscheidende Bestandteile dazu bei und zeigt gleichzeitig einen effektiven Weg der Zulassung auf aktuellen Normen und Richtlinien. Wir definieren wiederverwendbare Anforderungen für zukünftige, leistungsstarke Verbund- und Geräte-Architekturen, unter Berücksichtigung der domänenspezifischen Besonderheiten. Zudem führen wir ein neues Klassifizierungsschema für kritische und unkritische Systemfunktionen ein, basierend auf ihrem Einfluss auf das Gesamtsystem. Dazu definieren wir drei Klassen, die in vielen Anwendungsdomänen wiederverwendet werden können, zusammen mit den bereits existierenden Sicherheits- und Assurance Klassen. Weiterhin entwickeln wir zwei Gerätearchitekturen, die auf beliebige Systemgrößen und Sicherheitsklassen skalieren und zeigen die Integration von bestehenden Konzepten für Softwaredeterminismus in unsere Architekturen. Diese bestehen hauptsächlich aus leistungsstarken Standard-Mehrkern-Prozessoren, verfügen über verschiedene und mehrschichtige Redundanzen mit gezielter Dissimilarität und können von kostensensiblen Industrieanwendungen bis hochkritischen Luftfahrtanwendungen eingesetzt werden. Wir analysieren die Architekturen nach gängigen Methoden in der funktionalen Sicherheit, wie Fehlerbäume und Markov-Analyse und klären die marktspezifischen Besonderheiten bei der Zulassung. Abschließend diskutieren wir Anwendungsbeispiele.

Danksagung

An dieser Stelle möchte ich all jenen Personen meinen ausdrücklichen Dank aussprechen die diese Arbeit stets unterstützt und bereichert haben.

Besonderer Dank gilt Univ.-Prof Dr.-Ing Florian Holzapfel für die ausgezeichnete und all-umfängliche Betreuung über die lange Laufzeit und natürlich die Möglichkeit diese Arbeit anfertigen zu dürfen. Prof. em. Dr. Peter Hartlmüller möchte ich an dieser Stelle ebenfalls danken, der mich als fachlicher Mentor durch die Arbeit begleitet hat und mit zahlreichen Diskussionen zur Konzeptschärfung beigetragen hat.

Für die vielen fachlichen Diskussionen rund um funktionale Sicherheit die diese Arbeit sehr bereichert haben und die Einführung in die Gremienarbeit bei der VDE/DKE möchte ich mich sehr herzlich bei Matthias Ramold bedanken. An den gemeinsamen Vortrag auf der SafeTech erinnere ich mich gerne zurück.

Besonderer Dank gilt auch der MicroSys Electronics GmbH die in der Anfangszeit dieser Arbeit ein Drittmittelstipendium stellte, welches diese Arbeit erst möglich machte. Außerdem hat sich durch die viele Erfahrung des Entwicklerteams das technische Konzept gerade im Hinblick auf die Umsetzbarkeit nachgeschärft - daher gilt mein Dank besonders Dieter Pfeiffer und Richard Loeffl für das entgegengebrachte Vertrauen und die Risikobereitschaft dieses Unterfangen anzugehen. Natürlich möchte ich auch dem ganzen Entwicklerteam danken - Walter, Erich, Christian, Kai, Kay und allen anderen, die bei der Umsetzung unterstützt haben.

Prof. Dr. Marco Caccamo und Assistant Prof. Renato Mancuso, Ph.D möchte ich insbesondere für die Diskussionen rund um ihre Themenschwerpunkte danken, die die Softwareaspekte dieser Arbeit beeinflusst haben.

Für den großartigen Einblick und die vielen Informationen, und deinen wirklichen deep-dive in die Halbleiterwelt möchte ich mich ausdrücklich bei Geoffrey Waters, Rolf Schlagenhaft, sowie Andy, Brad und Alfredo von NXP Semiconductor Inc. bedanken.

Ohne die viele Geduld, den Zuspruch und moralische Unterstützung während der Entstehungszeit durch meine Frau und meine ganze Familie und Freunde hätte ich es nicht geschafft - daher auch an dieser Stelle vielen Dank euch allen.

Table of Contents

List of Figures

List of Tables

1

Introduction

1.1 Motivation: A shift in safe system design methodology

When Gordon E. Moore, back in 1965, projected that the number of transistors and likewise the density and complexity in integrated circuits will double every two years, nobody could have foreseen the tremendous impact on the daily life of millions of people by the following generations of microprocessors. The vast increase of computational resources not only gave rise to many new forms of communication, such as mobile phones or the internet, but also empowered the control of embedded systems. Over the decades, mechanical control systems were, and still are, replaced by electric and electronic units in the form of sensors, actuators, and control units. They augment, secure, protect and connect more and more physical systems, transforming them into cyber-physical systems, see for example [Lee08].

Among those systems are a special sub-category, forcing the highest requirements on both engineering and development processes. The safety-critical systems. They cover many entirely different application domains, such as aviation, automotive, medical equipment, infrastructure or automation of machinery. Specific standards and guidelines guard these domains, with regulatory agencies overseeing the development activities, leading to a strongly regulated, difficult, but also fascinating and prestigious market.

Since their first appearance almost 50 years ago, safety critical embedded systems were subject to specialized components, custom designed or selected to fit the purpose. Electronic parts and integrated circuits, especially microprocessor units (MPUs), were always selected conservatively. Since the absence of component design errors, stability of a manufacturing process and the long term failure modes are hard to prove during design time, the common practice is to either rely on the service history or a device in other safety applications, or special (and costly) long term experiments to collect statistically relevant data on error and complete failure rates. As a consequence, the technology actually used in most safety applications is lacking 5 to 15 years behind the current state of MPU

development, depending on the criticality of the application and the openness of the certification authority for new technology. For example, a still used, high-end processor in some applications in the aerospace domain, the NXP Semiconductor MPC8349, already left the extended 10-year availability program the manufacturer offers.
Up until now, this practice was fine for safety related equipment in many application domains. The application specific, highly specialized workloads required little computational resources, resulting in old and proven technology satisfying the needs with some headroom. Each (safety-related) system function is allocated to a special piece of equipment, like mechanical components, that serve a specific purpose. Such units are being named, for example, a "flight control computer", "autopilot control unit", "surface control computer", "flight management computer", etc. to clearly state the specific function they execute. As a consequence, the individual computational resources required on each line replaceable unit (LRU) are low, however, when the whole system is considered, a substantial amount of computing power is needed to compute all system functions in parallel, with the required degrees of redundancy. The employed system architectures are classic, multi-layer sensor-computer-manual bypass-actor designs. They are based upon the human as the primary system controller and augment the human beyond the limitations of his motion apparatus (speed, precision and repeatability of movements) and sensory capabilities (vision, acceleration). In the past years, however, one can observe several changes in the environment of safety critical systems and at the electronic component level:

- Growing demand by industrial customers for higher degrees of automation to reduce the costly workforce required to operate those systems or provide better service for their customers.

- Small unmanned aerial systems are a large future market to monitor for many application scenarios, starting from recreational use, over farming, monitoring of construction work and infrastructure, public safety all the way to agile logistic delivery systems.

- Highly automated or autonomous systems in the aerospace, automotive, industrial, naval, or even land moving equipment where the human executes only very high level control and supervision tasks.

- Academic advancements gave rise to lots of (complex) algorithms to mimic human cognitive functions such as planning or complex decision-making based on certain constraints, allowing to substitute the human as the main system controller

- Complexity, integration and compute power of even the smallest MPUs expanded significantly with smaller feature sizes while reducing thermal design power. This results in a very strong discrepancy between the classical certification approach, where safety analysis was executed down into the MPUs, and the complexity and

effort required to conduct these analyses. With new generations of devices, even small microcontroller will be considered highly complex devices with enormous efforts required to justify their use with classical certification approaches.

- Safety application specific MPUs will remain a niche product [1], due to the tremendous development cost involved in designing branch specific solutions, when quantities are low to almost non-existing compared to non-regulated markets such as networking/telecommunication, consumer devices. One must acknowledge the fact that the business case for most semiconductor manufacturers is hard to make, and will result in a reduced portfolio over the years, lacking behind in technology.

- Driven by the continuous manufacturing advancements in the semiconductor industry, the classic, small and simple microprocessor will effectively no longer exist. Instead, one is faced with highly complex, densely integrated, difficult to deal with system-on-a-chip (SoC) devices. Manufacturers integrate more and more function units to cover as many application domains as possible with their devices to increase the return on the costly development process. As the name SoC already indicates, we are dealing with a system in the system, with limited insight due to lots of confidential intellectual property (IP). As a result, a purely hardware or ASIC driven certification approach might no longer be the best suited option for certification.

Because of these new constraints, future system architectures and embedded computing equipment must change in many ways. While offering more computational resources than what is currently considered high-end to accommodate increasingly complex algorithms, they must also be lighter, draw less power (electrical, battery powered systems), stay within certain limits of power dissipation (active cooling may not always be an option) be highly modular and expendable and offer lots of flexibility in terms of interfaces. The removal of the human fallback layer in unmanned or highly automated systems implies, that system functions that are currently rated with low criticality levels will rise in their criticality, requiring redundancy and special unit designs where up to this point simple, non-redundant configurations are being used.
Also, a swift change in the mindset of many safety engineers is needed to in cooperate new technologies in system designs with slow changing regulation and well established development processes. The redundancy explosion with current system architectures would be a showstopper in future development efforts, introduced by the common misconception that safe systems must be build out of safe parts (proven to be error free and only producing limited, environment introduced failures at known rates during lifetime or failing completely), instead of deriving a system or unit architecture, that is explicitly capable of mitigating any single, or even multi-device failures at the system level.
New system architectures for safety critical systems must be compatible with upcoming generations of commercial-off-the-shelf (COTS) devices that are not tailor made for the

[1] compared to standard devices, from networking, general computing, or small embedded MCUs

embedded safety market to keep up with rising computational demands and minimize the gap between the current state of technology and the components used in safety rated equipment at the moment. Likewise, the total cost for electronic control equipment in the overall system must not be affected by taking the next step towards more computing power and new generations of devices. The risk of not achieving certification, or substantial project delays due to an unclear certification workflow for current development projects must be minimized, to raise the technology readiness level for future real-world projects.
One way of solving these issues, is the use of COTS multicore MPUs (Multicore Processor, MCP) as the main computation elements, together with a new board level architecture within the LRU. Offering multiple central processing unit (CPU) cores per physical device, MCPs can execute different computational tasks in parallel. Unlike in the desktop or server environment, where applications itself are executed in parallel over multiple cores, embedded safety applications target running many system functions in parallel, each spanning only a single physical core. Physical LRUs can be combined into one single unit, which in turn can be part of a redundancy cluster. This not only reduces the number of physical LRUs by a significant amount, but also the interfaces required to connect and power those LRUs. Besides the low price point of COTS MCPs, they offer outstanding performance per watt figures and are quickly moving to even higher integration densities, with more and more cores being integrated. Today, typical designs may offer from two, up to 16 physical cores for typical embedded COTS MCPs. These benefits however come at a cost. While single core devices can be predictable and, up to certain degree, behave deterministic, the interference and cross-coupling between multiple cores with equal rights and permissions in the SoC pose great risks for a successful certification effort. Consider the following examples:

- The internal function blocks of the SoC must be supplied with a clock source to execute their functions. This also holds for the different cores. Usually, a clock tree is used to supply the core clock, which may be fixed for all cores, or offer per-core dividers to permit clock gating for power saving reasons. For certification, can it be shown, that a single-event latch-up in one core's clock tree does not affect other parts of the clock tree? Will a latch-up in one core propagate to other cores, the common bus matrix or other SoC parts? These questions can only be addressed with manufacturer level knowledge, and even if the vendor is willing to share this information, which is at very heart of his IP, it is highly unlikely that special circuitry has been added to the device to prevent error propagation, because these circuits are complex, require space and power and are not required in the actual target market the device was designed for.

- Imagine a multicore device executing software of different criticality levels concurrently on different cores, core 1 and core 2. Suppose the software on core 2 exhibits

a severe design error, due to its criticality level being rather low, and the testing effort was therefore legitimately reduced, resulting in a failed loop-end statement in a memory copy routine for a large data array. The added memory transactions of core 2 effectively reduce the available memory bandwidth of core 1 in the shared memory controller. Core 1, executing highly critical software, exhibits a prolonged execution time, resulting in a deadline miss. An easy solution would be to introduce per-core memory priority systems into the SoC, which no device, core architecture or memory controller peripheral currently offers. Priority levels exist for multi-bus-master devices with other high bandwidth masters like graphic or special signal processing accelerators, but not on a per-core basis. In addition, suppose the above failure mode was not triggered by a design error, but a single-event upset in the register file, the pipeline or a function unit if the core. Just like in the first example, a per core priority scheme is useless for the server workloads present in the primarily targeted application domains of MCPs. It is unlikely that a device manufacturer will spend the DIE area and development effort in such a feature.

- A single event upset in the memory controller, the shared bus matrix or other vital peripherals such as clock generation, power management, etc. results in a changed configuration and renders the MCP inoperative, greatly reduces the speed of the device or results in false data present in the device's memories. The shared peripherals inside the SoC each present a single point of failure when used during normal operation. How can a failure at the bus matrix be detected and/or mitigated, when it affects all cores? Will a single event effect in the memory target a specific core, all cores or a subset of the available cores? Not all, but many telecommunication MCPs offer some degree of memory error protection via error correcting codes (ECC), but the memory controller and the common bus matrix remain as a single source of error in those devices.

- A core exhibits a single event upset in its pipeline, resulting in a false target address for a store operation. It overwrites certain critical configuration sections of shared memory, or inter-core-communication scratchpads. Can the other cores detect the failure of said core, or will it remain hidden until the error manifests in wrong, and potentially dangerous output data? Will the error lead to a wrong timing of another core, executing critical software?

- Can true electrical independence of the cores be justified? Each core will likely be constructed with independent seas-of-gates for its core functionality, but does this also hold for the electrical interconnect layers? From discussions with semiconductor manufacturers, it is unlikely that they are able to show true electrical independence of different cores on the same silicon DIE, without electrical isolation barriers as they are found in special safety microprocessors based on lockstep-architectures. The usage of such barriers greatly reduces the overall achievable clock speeds due

to added signal lengths and gates, which is again neither useful nor desired by customers in the primarily targeted application domains of MCPs.

Solving problems like those stated in the above examples, must be done on the system level. The lack of very detailed insight into those types of devices, due to confidential IP blocks at critical SoC levels make it virtually impossible to successfully conduct most types of classical certification related verification or validation activities. Examples include, but are not limited to, the shared bus, the memory controller(s), the cache and different protection and rights management function units as well as virtual memory subsystems, clocking, power and physical silicone features like interconnect spacing or electrical interference between adjacent circuits. Certification legislature and standard definition are adapting slowly, and cannot be transferred easily to new technology. Therefore, a solution for using MCPs in safety critical equipment must consider the current state of safety standards and guidelines and define new ways, to adapt proven, stable development and validation processes at the system and LRU level. This not only holds true for MCPs, but also for many other upcoming COTS devices, for example high-speed bus transceivers and peripherals (like Ethernet Switches or Physical Layer Interfaces), which must also be considered highly complex in the certification context with the above constraints holding for certification.

1.2 Mission Statement and Scope of Work

The overall goal of this work is, to design a high integrity, high performance embedded computing platform, based on COTS MCPs. A certification strategy must be worked out, in order to demonstrate the systematic capability for given certification levels in the aerospace and industrial to domain, and to justify the use of the COTS MCP in these application domains. The concept must be suitable for future system architectures, in highly automated or autonomous systems. During the course of this work, requirements will be defined for an LRU architecture, which fit into a defined set of system architectures. Next, multiple LRU architectures shall be derived from these requirements, based on at least one MCP as the main processing element. Industry-Standard safety verification techniques must be applied coupled with an analysis of current regulatory frameworks to justify the applicability and analyze the suitability for certification. Lastly, an application example will be shown, in order to illustrate certain design features to support the claims made in the design and safety analysis. Overall, this works follows to some degree the V-Model approach (see [Cla09] or [MM10]) as required in most relevant standards.

While many MCP architectures exist today, this work will focus on COTS devices only. Application Specific Integrated Circuits (ASICs) are therefore not within the scope of this work. Configurable logic, in the form of SRAM- or flash-based FPGAs may be employed where necessary, to accomplish certain interface tasks, but not to implement

any form of custom processor or hardware device that is beyond the scope of data bus communication, interface- or glue-logic. Therefore, custom on-chip hardware units, that permit to execute certain functions in hardware which monitor or supervise the SoC, are also out of scope. Such devices may be better suited to accomplish certain certification goals, but are generally not an option for real-world development projects, due to their extensive development costs (when considering highly complex, high performance devices), long development time, and project risk. We center our scope of usable devices on common processors, found in the telecommunication, mobile computing, or non-certification critical application areas, where MCPs have been employed for years. This thesis is neither focused on a specific family of devices, nor on a specific manufacturer, but rather on a general, n-core microprocessor with no special features included on purpose for certification within a special application domain. However, certain constraints like power consumption or thermal power dissipation still apply, which narrows the scope of applicable COTS devices.

Likewise, lockstep architectures are explicitly not considered a MCP in this thesis. Although this is technically not correct (a lockstep microprocessor already contains at least two cores), the fact that the cores are used to validate one another does not permit benefiting from the inclusion of a second core, reducing the available computational power to that of a single core device. Also, these architectures are yet again domain specific designs, mainly designed for the automotive industry to fulfill a special use case in some applications. One may use these devices at some point in the design, but they do generally not fulfill the performance requirements of a high performance platform and only offer a small benefit as the central processing element.

Beyond the system and hardware perspective, the software executing in the LRU plays a major role in overall system safety. Therefore, a software architecture, and special considerations regarding mixed criticality, separation of critical- and non-critical software functions, etc., shall be within the scope of this work. We will not consider low-level software aspects, as they are highly device, programming language and tool specific, but rather address a high-level, more general perspective on certain software functions and the software architecture necessary for successful certification. Whenever possible, the designs shall not be reliant on specific, in-depth design data by the semiconductor manufacturer. Such information is not only hard, or in some cases impossible to obtain, but is also an indicator for a failed system architecture, that does not manage to mitigate certain type of device failures, but instead tries to claim that they are either nonexistent or internally mitigated. We will try hard to overcome this design philosophy for the benefit and simplicity of future generations of safe systems.

1.3 State of the Art and Related Work

The state of the art can be divided into three main categories, e.g. architectures on the system level and hardware- or software-centric publications. In the past, several large scale joint research projects tried to address the issue of multicore certification, like ARAMIS [BB18], RECOMP [PTV+13] or EMC^2 [Web17] under the EU ARTEMIS initiative, with minor success for practical applications. None of these projects succeeded in providing a viable path for real-world certification. Back in 2012, a study conducted for the European Aviation Safety Agency, see [JXM12], examined the current technology readiness level, as well as system, hardware and software related topics surrounding the use of multicore devices in aerospace applications. The study evaluated different multicore architectures and identified potential pitfalls for certification, like lack of determinism due to contention, common-mode failures, lack of insight into the complex devices, etc.
Examining the current state of system architectures and design principles applied in the industry is difficult. One is faced with closed system designs, subject to non-disclosure agreements due to the fact that company internal or product specific intellectual property is closely related with the overall safe system architecture of embedded electronic equipment. Publicly available information is often only illustrative, like [Yeh96] or [Gou11]. In several discussions with industrial certification authorities, it was found that currently no certified system is using multicore processors with more than one core activated at runtime (mainly x86 machines, SIL4 triplex railway systems use x86 multicore devices with all but one core deactivated with efforts being made to argue for a second core). For general system architectures, not addressing multicore processors but design patterns in general, contributions from academia can be found in [Arm10] or [JG11]. A promising board level architecture, focused on multicore processors, was presented in the MUSE project [FOK]. It was primarily targeted at space applications and did not address industry and aerospace related certification issues. In aerospace applications, some manufacturers offer rugged computing equipment based on multicore processors for non-critical applications like mission and flight management, for example [Gmb]. Currently, two major movements can be identified in terms of certification with complex COTS components. The first one advocates to continue the long-serving approach of using custom safety parts. While these devices offer lower performance and less features, with a very bad cost/performance ratio, they make certification straight-forward with manufacturer level support and a rich, tailored documentation. The second movement is abandoning the “safe system by safe parts” philosophy and moves the argument for safety entirely to the system level and into a gray area in current standards. In this approach, is up to the applicant to demonstrate to the authorizes that sufficient architectural mitigation is included on the system and equipment level, together with some device specific safety precautions, to mitigate permanent and transient faults on the individual components. This implies a much stronger relationship between the individual unit architecture and the higher-level system architectures, leading to a much stronger entanglement between OEM and subcontractors or even

bringing the design and engineering of equipment back in-house due to the level of system knowledge required to engineer the individual unit. Up to this point, no architectural concepts exist to lower the dependency on SoC internal device function units in terms of certification by suitable system level safety argumentation.

While the industry started to engage certification authorities in the aerospace and factory automation domain to define a practical methodology for using multicore processors, academia is investigating hardware and software solutions to overcome the determinism and core-separation issues with current device generations. Albeit being valuable contributions, the literature referenced in the following does not address specific certification topic, nor analyze their concepts in terms of certification within a certain application domain. Custom on-chip hardware units together with certain software functions, special fault tolerant cores or rather SoC function blocks are employed for example by [PGN+14], [DBG10], [MHPS96], [LPO15], or [MDB+12], to ensure device level determinism and fault detection. Although this related work is slightly out of scope, since it will result in ASICs, it is of importance since it might lead to future COTS devices featuring special function units to aid in the process of spatial and temporal isolation of different cores on the device. The interference of multiple bus-masters, operating on shared resources, results for example in contented SoC busses, that are extremely difficult to analyze and validate (relevant for certification). Timing analysis and prediction has been subject to many works, trying to provide software solutions for worst-case execution time (WCET) analysis methods for multicore devices or use existing device features together with operating system level measures for mitigation. See [SBM+08], [NPH+14], [NPB+14], [AE10], [DNA11] or [NP12]. An outstanding concept for spatial and temporal isolation has been developed by Caccamo, Mancuso, Chen, Yun, et. Al, with their "PALLOC", "MEMGUARD" and, most notably, "Single Core Equivalence" approaches, see [YMWP14], [YYP+13] and [MPC+15], with additional work presented in [Man17]. The main drawback to their concept, in terms of certification, is the usage of special tracing units at the SoC level to monitor memory transaction. These on-chip units are not developed according to a certified process. Therefore, they do not provide test, fault-injection or other means to ensure their correct behavior at run-time and cannot be loaded for justifying certification claims. We will provide system level measures in this work, to aid in this respect.

Since the promising "Single Core Equivalence" concept on the software side has just been published at the time of writing this thesis, the support by commercial operating system vendors is currently non-existing. Most pre-certified real-time operating systems (RTOS) do currently not provide adequate measures for spatial and temporal isolation on the multicore device and may not be available as an asymmetric multiprocessing (AMP) port, which is able to execute multiple fully isolated instances on each core or distribute tasks in an isolated fashion to slave cores. The lack of proper software support is one of the major project risks for future development projects in cooperating multicore processors.

In summary, the current state of the art is very promising in terms of software level con-

cepts for multicore processors, as well as in terms of concepts for safe multicore ASICs. However, the applicability of these concepts to real world projects is limited to some extent, since currently no overall system and board level architectural strategies exist, allowing to demonstrate the systematic capability for equipment based on multicore SoCs in various application scenarios.
We already published parts of this work in a working paper [Ste15], and presented the work at the safe.tech conference in 2016 held by the TÜV SÜD Rail GmbH together with Matthias Ramold, TÜV SÜD Rail GmbH and an industry forum (Safety and Security Forum 2016, Munich by WEKA FACHMEDIEN GmbH). Both presentations can be obtained from the conference owners.

1.4 Contributions

This work contributes to the ongoing efforts to use COTS devices in safe embedded applications in several ways:

- (C1) High performance LRU cluster requirements for highly automated systems

  Currently, no set of generalized high level requirements exists as a common baseline for complex control systems based on high-performance processors. This work provides a substantial set of requirement aiding in the general design process. The requirements are also used to prove the real world applicability of the derived architectures.

- (C2) Novel system function classification scheme

  System functions need to be classified in many ways during the development process. Not only with respect to technical, business and project management aspects, but also in terms of their criticality and safety attributes. This work introduces a novel fault impact level classification scheme, derived during the course of this work in order to cope with functional degradation paths and different redundancy strategies.

- (C3) Low internal redundancy LRU Architecture (fail-safe) for less critical or highly physically redundant applications

  The main contribution of this work is the development of a novel system architecture and certification approach for electronic hardware based on COTS complex, multicore SoCs. Due to the diverse nature of real-world applications, this work presents two architectures. The first, low internal redundancy architecture is either suitable for larger systems with high degrees of physical redundancy or less critical applications with lower certification levels. The software level is also covered as well as system function allocation schemes.

- (C4) High LRU-internal redundancy Architecture (fail-operational) for critical or low physically redundant applications

The second architecture is specifically designed with a high internal degree of fault tolerance. It is by design at least single fault tolerant and sustains even certain multi-point faults. It is mainly applicable to high certification levels with low physical redundancy possible due to cost or size constraints. Like the first architecture, it is discussed thoroughly. Both architectures support a vast set of legacy and high-speed interfaces for current and future communication architectures.

- (C5) Certification path for industrial and aerospace applications

  Both architectures are examined in the context of industrial and aerospace certification standards and analyzed for their suitability in the respective context. This work then provides a stable path to actual certification up to the highest safety levels based on the current issues of relevant standards and guidelines.

- (C6) Application Example

  A prototype example forms a possible real-world implementation and concludes this work based on currently available high performance SoC devices. To prove the practical feasibility of the presented concepts.

1.5 Structure of this Thesis

This work is divided in four main chapters. First, we discuss possible future system architectures, focused on automated and autonomous systems and derive requirements for the individual LRU within the context of these architectures. In the second chapter, we present the LRU architecture designs, their inner workings and interface behavior. Afterwards, in the third chapter, the LRU architectures are being analyzed in terms of safety, employing fault-tree and Markov analysis. We will also discuss certain standard and domain specific aspects for certification which are necessary to comply with certain safety levels in different domains. Topics like multi-point faults or common mode failures will also be addressed in the third chapter. In chapter four, we discuss practical architectures for different domains based on currently available hard- and software. Finally, we summarize this thesis and conclude its findings.

2

System- and Board-Level Architecture

In this chapter, we will be focusing on deriving new system-level architectures for safety critical systems to enable the usage of COTS multicore processors. We will therefore first look at current and future cyber-physical system architectures for safety critical systems and define system requirements for an individual LRU in these contexts. Based on these requirements, two different LRU, or board-level architectures will be derived which address the use-cases in different types of systems. After presenting the system design and system architecture (alongside with some common hardware aspects), we focus on the software aspects inside the multicore processor subsystem and the concepts for achieving determinism and proper isolation of critical functions on the device.

2.1 LRU Requirements

Deriving requirements for generic, safe compute systems is a hard task. When this thesis started, it was mainly driven by future aerospace application with small and medium-sized autonomous vehicles in mind. With recent developments in the industrial market, namely the efforts for more intelligent, more efficient and fully connected machinery and infrastructure, the requirements expanded to also include those application domains with the according regulatory frameworks for safe machinery. Looking at these domains, which at first seem like entirely different disciplines with their own mindsets, one may find that the current development standards are closely related and almost lead to identical development processes and technical solutions for system safety. The level of verification and validation however differs between those two domains, except for the highest certification levels where the time and effort which has to be spent in both domains is almost identical. We will nevertheless rely on aerospace examples of current and possible future architectures, since they clearly reflect the ongoing shift from functionality-based LRUs in classical system architectures to federated, modular and highly integrated designs. The

same shift can be observed in industrial applications where more powerful and centralized control units, alongside with smart actuation and sensor nodes are brought to the market by different vendors.

2.1.1 Targeted Future System Architectures

Today's system architectures for safe systems can best be described by a classical multi-layer architecture. Multiple independent layers form dedicated safety nets, aided by layer internal redundancy (with dissimilar components) and zonal safety of the whole control system. However, a slightly different system topology can be found on the most recent multi-role fighter aircraft F-35 (Joint-Strike-Fighter, JSF) by Lockheed Martin. The innovative JSF program introduced not only flight-worthy high speed data bus communication via a particular physical layer adaption of the commercial FireWire standard, a comprehensive and highly appreciated C++ coding standard [Cor05] to cope with growing software complexity and maintainability by using an object-oriented programming language, but also a modern, non-layered avionics architecture.

As shown in figure 2.1, the JSF uses three fully redundant branches, each consisting of a central Vehicle Management Computer, connected to independent data bus groups (three per unit). A cross-channel data link connects the three branches only via the management computers which also carry out all processing related to flight control and vehicle management. In each data bus group, input and output elements, as well as mission management equipment is connected via dedicated interfaces and forms in this particular example, with the IEEE-1394 "FireWire" bus used, a self-reconfiguring tree network which is tolerant to leaf failures due to the self-reconfiguration capability of the IEEE-1394 standard [Bai07]. For example, the FADECs (Full Authority Digital Engine Control) are redundant for this single-engine aircraft and each unit is connected to all three management computer branches via their individual data bus group one. This allows to tolerate up to two full branch failures before the control augmentation by the management computers (executing the flight control algorithms) is lost. Likewise, actuation control, sensors, external communication equipment and mission equipment is also connected redundantly. The mission system integrated core processor is nevertheless separated from the vehicle management computer.

Although being very innovative and future proven, the JSF design should be regarded as an intermediate step between the classical "one-unit-per-function" federated architectures of today's systems and future fully integrated designs. The designers of the JSF introduced the term "integrated-federated", see [WJKLMC, Slide 12], to describe their vehicle system architecture, which best describes the intermediate nature of this approach. A full integration of computing LRUs is not yet achieved, as multiple large processing systems still exist. As stated in the introduction, this leads to an unnecessary high number of LRUs in the system, especially when more and more former non-critical functions become critical due to the lack of human oversight in autonomous systems (see figure 2.1, mission

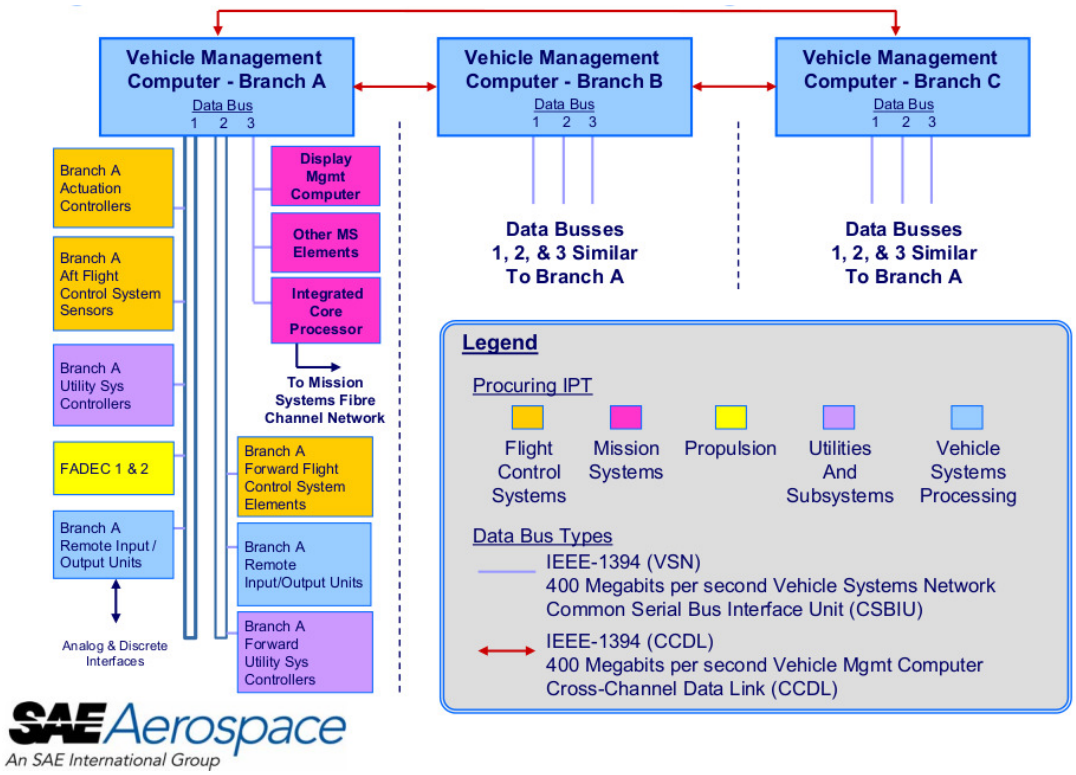

Figure 2.1: *JSF High-Level Vehicle System "Integrate-Federated" Architecture [WJKLMC, Slide 19]*

equipment is replicated at least three times, dedicated bus trees for mission equipment). This also implies that in order to provide somewhat reliable mission computer functionality (not even considering autonomous navigation and complex data fusion) three times the cabling, connectors, physical LRUs, storage spaces, power wiring, battery backup and power generation, development effort (interface definitions, designs, requirements, V&V, etc.), not to mention the overall added maintenance due to added units which might fail over time, or the total system cost and complexity. It is clear, that with current architectures (even considering the modern, rather radical approach of the JSF), the introduction of certified small/medium-sized autonomous vehicles is not possible, due to evident size, weight and development/certification cost issues.

To overcome these issues, one must completely abandon these classical design approaches for a new system architecture design philosophy, which provides adequate levels of safety and availability. Since the needed computational resources in those systems will grow significantly in the next decade, as stated in the introduction, highly capable individual LRUs are a practical solution to combine the currently distributed physical units into one single LRU. Usually, combining many safety functions of a system, together with non-critical or mission critical functions in a single device, is no wise idea. But, provided the individual units can ensure adequate determinism and isolation between those functions and a particular, defined failure behavior, a redundant cluster of such LRUs is

capable of providing multiple independent layers of safety (even diversity by-design with equal LRUs as we will see later on) in a small and cost-effective form factor which can be easily scaled with the physical system. Beside the redundant compute clusters, other key points in these new architectures shall include:

- Redundant communication paths

  Interconnects between a centralized compute cluster and its inputs and outputs, or other system elements, are and will be one of the most critical system components. It is vital to provide modern, redundant data paths within the system which are truly single fault tolerant, or even multi fault tolerant with added zonal safety. This not only holds for the physical cabling and connects but also for the data flow itself. Given the enormous amounts of sensor data required for autonomous systems (vision, radar, optical measurements like light detection and raging, high frequency inertial and global navigation satellite system (GNSS) data, etc.) there is also a strong need for bandwidth and minimal transmission delay. We will discuss this point later when we examine some architectural examples.

- Smart Sensors and Actuation Elements

  Innovation must not stop at the computing platforms. In recent years, manufacturers of electro-mechanical actuators and sensing equipment already integrated more and more functionality into their devices. This trend must give rise to new device generations, featuring not only the same high speed interconnects as the computing platform itself, but also complex algorithms to precisely determine the devices health and trustworthiness. Voting of multiple redundant input data sources can be carried out in a distributed fashion on those elements, without the need for centralized voters or intermediate layers [1].

Since overall control system architectures are closely tied with the physical system itself and the intended use case, finding a common and abstract design architecture is difficult. It is clear that the avionics architecture of an Airbus A380 would not be suitable to be used inside a 10kg UAV, nor inside the powertrain system for a bullet train. In order to still provide some examples at this point, which will be used later to derive our set of high-level LRU requirements on which the LRU design itself is based on, we will now discuss two different system architectures. In general, the first architecture is suitable for small and medium-sized (Small System Use Case), the second one targets mostly large or highly critical use cases with very high certification requirements (Large System Use Case). We will discuss possible use cases and constraints for each architecture individually in the following. Note that these examples are explicitly kept at an abstract level, since the details of each system and application domain pose very specific implications in real-world applications which must be considered very carefully to not over-complicate the final

[1] Centralized voters are tough in certification and result in very high assurance levels for those devices

system design or miss the chance to achieve safety and availability by simpler or more efficient means (for example, secondary cut-off paths in industrial systems, or emergency stabilization followed by remote control in autonomous systems).

2.1.1.1 Small System Use Case (UC1)

Small and medium-sized systems are driven by strong size/weight and cost constraints. The first use case is therefore intended to be used in these types of systems, with the aim to minimize the total number of physical units. Zonal safety or high degrees of redundancy are either not emphasized by cost constraints or not physically possible. However, a certain degree of redundancy and low failure probability must be present in order to guarantee certification up to a medium level. Single fault tolerance is also an issue that must be addressed, especially in the industrial context for current machine safety regulation (e.g. IEC13849).

As shown in figure 2.2, the computing cluster consists of two equal units (Channel A and Channel B), interconnected via a cross channel data link (CCDL). The CCDL in this figure is drawn as a single line, implying a non-redundant connection. Depending on the physical design of the unit, for example, if a back-plane is used to interconnect the units in or if they are physically separated at different system location or in a common electronics bay, the CCDL can be enhanced to a redundant interconnect to offer a single fault tolerant connection between the two units if needed by the system design. This brings us directly to the sensor and actuation elements. External communication interfaces (Data Links, either line of sight, beyond line of sight, or other types current in a particular application domain) and sensor units, like inertial measurement units, air-data measurement systems, etc. are connected to both units via redundant data busses. If dissimilarity is required, a secondary emergency data bus can be added in case both busses exhibit a common mode failure. The depicted configuration protects from single hardware failures on the physical layer only. The same arguments hold for the output elements, which also feature redundant data bus connection to the computing cluster. These actuator control electronics (ACE) are internally redundant, smart actuation elements in order to further reduce the need for redundant physical units. Since adequate computing performance is present in those systems, each ACE can perform the voting between the set of redundant output data generated by the computing cluster in each discrete system time step. This negates the need for a dedicated voting layer between the compute cluster and the actuation elements, or complex, time-consuming distributed voting strategies. Due to the bidirectional communication between the smart actuators and the cluster, the current health status, as well as voting results and further real-time data like actuator position, is fed back and can be considered in vehicle management algorithms. Driven by the autonomous system background, an emergency control system is present with dedicated external data links, data busses and sensor units to allow for human intervention in case a severe system failure renders the computing cluster inoperative. Note that this abstract overview ne-

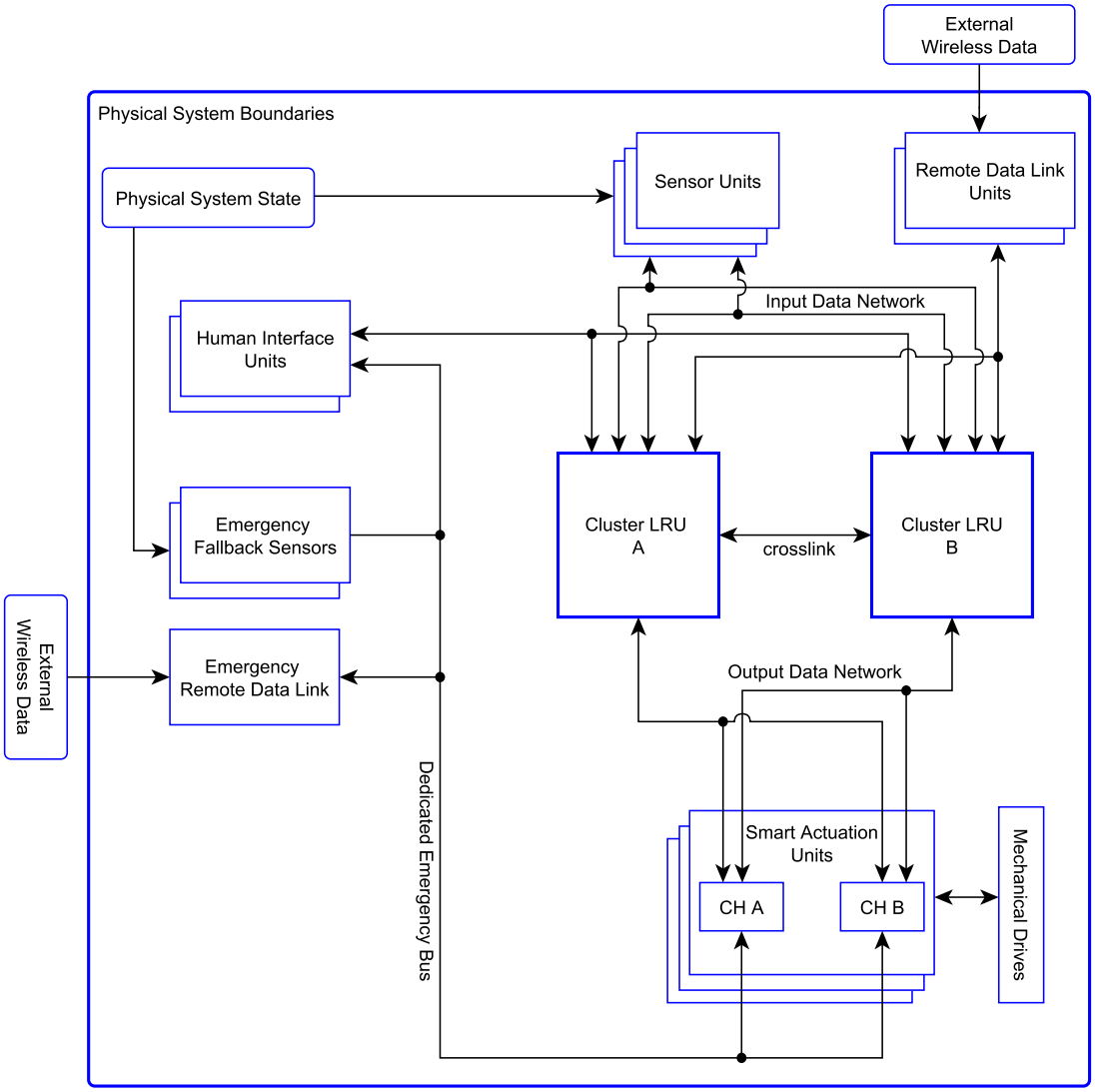

Figure 2.2: *Example architecture for small and medium-sized autonomous systems or industrial use-cases*

glected some aspects and thereby electronic units that may be present in a real system like power distribution and management units, battery management units, gear and engine management units (although they can be regarded as actuation elements), etc.

A computing cluster LRU consists of a safe and secure [2] multicore subsystem and the usual hardware components to provide redundant power and bus interface connections with domain specific protection against the environment like electromagnetic interference (EMI) or under/over-voltage events and lightning strikes. Being comprised of two equal units, common-cause failures are critical in this configuration, such that internal mechanisms must be implemented for dissimilarity and fault tolerance to offer intrinsic common

[2]Safety and security already go hand in hand in many projects, although we will not address security actively in this work

mode failure protection. This also holds for the bus interfaces (data and power). We will address these points later on in the LRU requirements.

The redundant data bus interconnects on system level are not strict bus topologies. The designer is free to choose an appropriate communication channel for critical data that suits the real system application best, might also be point-to-point-, star- or ring-bus topologies according to the actual physical and link layer employed.

The failure mode of the compute cluster LRUs is either fail-silent or a fail-degraded failure mode where a failed unit does either no longer communicate over the attached data busses or enters a degraded operating mode where minimal, and but still trustworthy output data is send in order to still provide additional data for the distributed voting on the actuation elements in the system. Note that the fail-degraded mode is somewhat special and might not be suited for all types of systems, but may provide additional operational capabilities, for example, to safely return to a predefined location or to bring the system into a safe state through a series of complex state transitions where redundant output data to execute some degree of voting is still valuable. When a unit flags itself as degraded depends on the internal design of the secured multicore subsystem. The same holds for the trustworthiness of output data in the degraded state which will be discussed later on in the safety analysis of our derived LRU architectures alongside with the functional degradation scheme proposed in this work.

2.1.1.2 Large System Use Case (UC2)

For larger systems, the simple duplex architecture presented in the last section will not able to achieve the desired failure rates, nor a high degree of fault tolerance. After a single unit failure, the duplex architecture is in a degraded state — and while still able to guarantee safe control the system, the availability is limited by the single LRU failure rate for nominal operation. In order to greatly expand the availability of the cluster and tolerate single (or multiple) compute LRU failures without losing the capability to provide multiple, independent output data results for voting [3], the large system architecture features an expanded cluster with at least three units and zonal safety.

As shown in figure 2.3, the remainder of the architecture is equal to the small system use case, with the aforementioned smart sensors and actuators. The CCDL internal to the compute cluster has been expanded to multiple point to point interconnects. While this is a viable option for three units, interconnecting four units already requires three high speed interfaces, which might lead to a redundant bus-type interface in order to reduce the number of physical interfaces required back to two. Note the network gateway and routing units, which build a dual-redundant network to the sensor, actor and computing ressources spread throughout the system for zonal safety. The cluster can directly interface with the redundant backbone network in the vehicle.

[3] to guarantee safe operation regarding possible single event effects during the computation of these results

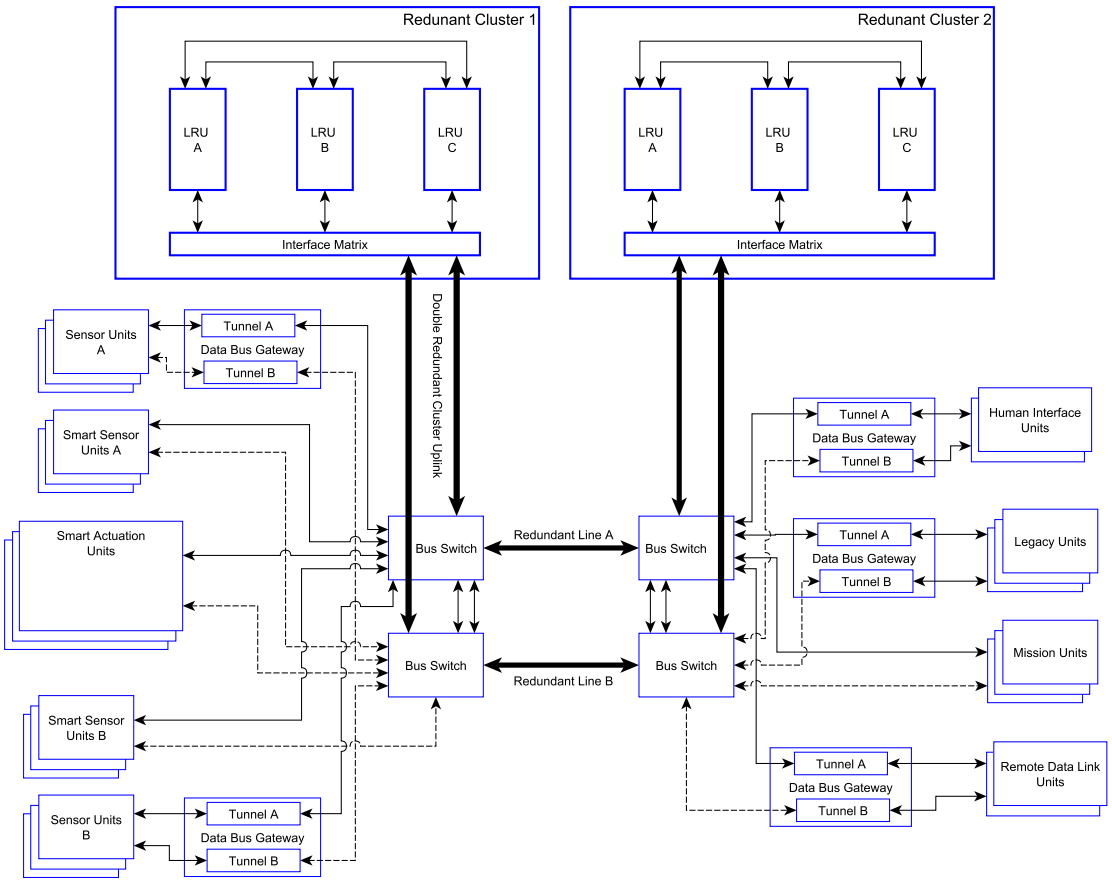

Figure 2.3: *Architecture for large sized autonomous systems or industrial use-cases*

The computing cluster itself can be composed out of entirely separated physical units. But it is also possible to employ a backplane-based solution to further cut down the physical size of the cluster, by using smaller sized individual LRUs and inter-LRU routing moved to the backplane, as well as the protection measures against the potentially harsh environment. Such solutions are currently widely adopted in the industry, and are labeled as an "Integrated Avionics Rack" [New94, p. 16]. On one hand, physical units tend to provide better isolation against zonal failures in the cluster, for example when single units fail spectacularly (single unit fire, burning/melting electronics components), since their housing provides some degree of containment. On the other hand, backplane based solutions are generally more flexible and can scale with minimal space requirements (if the individual units are of responsible size) and offer the possibility to easily be duplicated within large systems for zonal safety. However, the fault containment, especially when considering failure modes that involve potential fire hazards, may not be an issue overall due to the required flammability rating in most regulated markets, which require burnable substances to either have self-extinguishing or highly flame retardant properties while not producing gasses for example (like demanded by application domain specific standards, like CS 23.2325 in [EAS17]). As every so often in the field of safety critical systems,

there are seldom true black and white comparisons or recommendations that truly hold for real world applications. Final solutions are mostly application specific. It is therefore hard to make general statements about the suitability for a wide variety of different application scenarios, even in a single application domain like the aerospace or industrial field. Nevertheless, we will address the high level hardware architecture of the cluster again in 4, with further considerations for modularity or reuse for cost-effectiveness in possible hardware architectures.

2.1.1.3 Additions and Common Topics

The two exemplary architectures described above can be used in future autonomous systems, where computer vision is heavily employed for context and environment detection and classification. The digital pixel data of one or multiple camera systems (with or without internal preprocessing and filtering in the camera LRU), which must be fed into the main compute cluster, requires very high bandwidth data busses. Unlike other sensor types with very limited output data per time step, like inertial measurement units or simple distance sensors (optical-, ultrasonic- or radar-based via time of flight measurements), they produce massive amounts of raw input data at a high input rate. This leads to very specific data bus solutions, and often times special physical layer implementations, particular for this application. One major issue is the presence of single points of failure with the point-to-point interconnects for camera data. This can either be solved by redundant connections to multiple LRUs within the compute cluster from a single camera LRU, but this requires numerous high speed interfaces, especially when multiple camera systems are employed for stereo or surround-view applications, leading to an unreasonably high number of high speed interfaces for a single sensor group. One possible solution might be to develop new multi-drop, bus-like interfaces for direct camera LRU connections or to tunnel the camera data over other data bus standards, for example standard or industrial Ethernet variants. Over other link-layer standards, one can employ the already existing physical and link layer protocols to enable multi-drop communication to broadcast to multiple receiving compute cluster LRUs or to connect a larger number of cameras via a single data bus. For better error detection and system level redundancy, a redundant connection may be employed in order to form redundant networks of high-speed sensor input data, which offers a lot of flexibility but does not require an extensively high number of high speed interfaces at each compute cluster LRU which are solely allocated to a single sensor input. The same argument holds for the back-channel from the cluster LRU to a camera, or more general high-bandwidth data LRU for control data, acknowledgments or status data. The high bandwidth cross channel data link within the compute cluster should not be used to interchange high bandwidth sensor data, in order to keep the actual bandwidth during runtime low, leading to very short response times over a lightly used high speed link. However, if the cross channel link features very high bandwidth, beyond several 10 or 100GBit/s, it may also be considered to move high bandwidth sensor data.

We will however not focus on such interconnects in this work, due to the current lack of appropriate standardization efforts on such interconnects in the industrial and aerospace domain. The high speed data bus interfaces will be discussed during the LRU architecture presentation with greater detail.

When combining many system functions, each with their own safety criticality rating, we are inevitably faced with a mixed criticality scenario on the compute cluster LRU. In current designs, system architects try to avoid the combination of different criticality levels in one device, especially for the software, since it leads to a whole new set of requirements for the spatial and temporal separation of system functions with different criticality levels. Since we are targeting multicore COTS parts as the main computing element of a cluster LRU, we are faced with the partitioning issue for a different reason. As stated in the introduction, the individual cores cannot be regarded as truly electrically independent of each other, and thus guarantee an interference-free operation of individual system functions running in parallel on different cores. Thus, proper spatial and temporal isolation must be provided anyway, to ensure that a sufficient degree of independence, or freedom of interference can be provided between the different cores on a multicore COTS device. This also guarantees a certain degree of determinism alongside with worst case execution times of the system functions allocated to software that are being executed in parallel on each core. It is therefore clear that space and time partitioning is a necessary must-have on those devices for safety critical applications. But having adequate means for isolation system function also enables us to explicitly use this new degree of freedom, which is to execute all those functions in parallel on one device, to make the next move and combine all system functions allocated to software in one central compute cluster, as shown in the two architectural examples from the previous section. The presence of a redundant computing cluster offers further possibilities and a large flexibility to allocate different software functions, and offer on-unit or cluster-based redundancy for each critical system function according to their required criticality level and desired availability figures.

On one hand, while offering lots of possibilities, the explicit mixed-criticality scenario also results in added cost and complexity at the device level. As with single core systems, validating the strict requirements to ensure complete function isolation, with respect to shared device resources, is difficult during software low-level and software integration tests. This is mainly driven by the low-level, highly platform specific implementations of partitioning solutions which quickly reach high complexity levels due to software solutions for cache, memory, bus bar, or shared peripheral unit spatial and temporal partitioning. In addition, these codebases require very in depth programming language knowledge, compiler expertise, a great level of hardware knowledge from the software designer and software architect and can therefore only be designed by true experts which are hard to find for a specific platform or must be trained specifically for a dedicated platform. Overall, the development effort significantly increases whenever mixed-criticality is involved in an equipment design, especially in the lower software layers where an operating system or

embedded hypervisor may reside.

On the other hand, the total cost in the integration phase on the top system level is lower. This is achieved by the greatly reduced number of physical units and integrating them into the high-integrity cluster, for which early integration hardware-in-the-loop tests offer the potential to pass integration tests on top system level effortlessly. The integration of real physical units is always difficult and time-consuming, due to nearly endless sources of design errors which are hard to track down, since they may reside in the hardware, data busses, data bus protocol implementations, operating system configurations etc. In addition, the specification effort for system function interfaces is shifted from real physical interface and communication protocol specifications into the compute cluster and the cluster LRU. There, they are handled by software-driven inter-task communication functions (via memory, inter-core-communication, or inter-LRU protocols within the cluster), which are more flexible, easier to adapt (no hardware changes needed) and easier to integrate, due to the lack of actual physical communication interfaces. The actual remaining integration work on top system level is reduced down to the cluster, and the real physical sensor and actuation units, which were simulated in the hardware-in-the-loop environment beforehand.

Besides technical implications, there are also impacts on project planning and the general business strategy. The lapse of physical units results in a shift from specifying and allocating real physical units to subcontractors, testing and integrating them, to specifying purely software-defined system functionality fitting into a defined scheduling pattern inside the cluster. This basically busts all previously established subcontractor structures for large cyber-physical systems and shifts most attention to the high-integrity platform and the software function providers, which might be tier 1 suppliers, but also the OEM itself, due to the very high business criticality of the computing cluster and the software functions. These functions directly include very sensitive and valuable system level knowledge and will directly represent the core competence of an OEM in the years to come.

While the presented architectures were designed and justified with small and medium-sized autonomous air vehicles in mind, they can also be adapted for industrial application scenarios. In the industrial context, a true fully fail-operational, high-availability system design is seldom required in classic applications like factory automation, or machinery in general terms. In order to comply to current regulatory frameworks, a safe state must be defined in any case for a certifiable system, which can be entered by a defined, timely and deterministic series of mode transitions. This also holds for ground-based transportation systems which can be brought to a safe stop condition where a severe control system failure is no longer critical. In turn, most applications will most likely not require an n-times redundant, high-availability cluster for these applications. However, this statement only holds, if the single compute LRU offers a distinct failure mode behavior, as we will see in the next sub-chapters, in order to guarantee a certain failure mode behavior

for different categories of system functions. A reduction of redundancy in the top level system architecture is especially desirable, where tight cost constraints are involved. Since we will not discuss further domains, like railway or automotive applications, it is up to the experienced reader to transfer the presented high-level concepts to his or hers specific physical system configuration. It should be clear however, that the architecture fully scales from a single unit up to large redundant, distributed clusters easily, when the individual computing LRU fully complies to the LRU requirements we will define in section chapter 2.1.3 . Note that the presented architecture and the core contribution of this work, the internal LRU architecture for COTS multicore processors, are closely tied and cannot be discussed independently, since board and even hard- and software considerations directly affect the top level architectural structure due to safety and reliability considerations (see chapter 3).

In order to sum up this part, one must note the implications in the top level system architecture when taking the human as a control element out of the loop. The resulting redundancy explosion with classic, function-based LRU architectures for previously non- or semi-critical system functions strongly contradicts the use cases in small, yet certifiable autonomous systems. One possible solution, when COTS multicore processors can be used in a certified LRU, is to combine large numbers of currently dedicated LRUs into a single high-integrity cluster. This computing cluster can easily be scaled to application requirements and top-level system function criticality levels, in order to support the desired safety claim and platform availability. Alongside with smart sensors and smart actuation elements, a highly integrated architecture is formed, with centralized high-availability computing, but also distributed redundancy elements by shifting some system functions (like output data voting or emergency stabilization and control), as well as the integrity monitoring to the distributed, smart final elements and sensor units. The number of redundant cluster LRUs is driven by the actual application. This design decision is driven mainly, but not only, by size, weight, power and cost constraints, as well as the desired availability for the computing platform. Also, considerations like zonal safety or the choice between separated physical units and a backplane-based solution for the high-integrity computing cluster will affect the realization in real world systems significantly and will directly impact certification arguments. In order to continue the development for autonomous systems with certification in mind, one must acknowledge the very different application context without the human fallback layer. The need for more and more computing power at the higher level system functions will certainly affect LRU design and as a result, also the top level system architectures needed to cope with the different design approach employed for new COTS device generations.

2.1.2 System Function Classification by Impact Level

System functions in cyber-physical systems can be classified and categorized in various different ways. When considering certification, the classification as a safety function is

certainly among the most important ones. Along with the classification itself, usually comes a further graduation in different levels of criticality, driven by safety and risk assessments, which are defined in the applicable standards, certification specifications and assessment guidelines. Parallel to these existing classifications, one can derive further system function attributes based on the observation of current system functions and architectures and future autonomous system considerations. As previously stated, system level functions which are currently evaluated at a low or medium criticality level, must be considered to be highly critical when moving from semi-automated (human-in-the-loop) to fully autonomous or unmanned. We will present examples for these types of functions shortly with the impact level definition (Contribution C2). The possible failure mode for each type of system function, leads to a classification based on the system-level impact of an incorrect output result (either fed into lower-level system functions in the hierarchy or into actuation elements) and the impact of the full loss of a function. Considering today's common failure and degradation mode and the common nomenclature $Fail - Mode_1 ... - Mode_n$ (where $Mode_n$ denotes the resulting mode of a system function after n failures occurred [4], which is ensured by the underlying system architecture, system design and LRU design) classifying system functions according to their failure mode behaviour can be beneficial in many ways. We will rely on the following classes of system functions, for later on distributing system functions within the individual compute cluster LRU and the cluster itself, while ensuring that the desired failure mode of a system function is met and the standardized criticality levels satisfied by adequate per-function monitoring and per-function redundancy. Note that the primary application of this classification are the system functions allocated to the compute cluster. It may also be used in the context of smart actuation elements and sensors, which is not directly the scope of this work. We define three impact levels (IL), starting from IL0 up to IL2, implying different system function failure modes:

- Impact Level 0, $IL0 \implies Fail - Fail$

  Definition: A system function that is a comfort function, add-on or payload functionality. A wrong, untimely or unavailable output result has no impact on system safety.

  Examples: Passenger entertainment systems, active payload subsystems like optical measurement systems or other sensor units not involved in system control loops, data links solely serving live mission payload data.

  Failure Mode / Safety Measures: A failure leads to a wrong output result / behavior or failure of the system function.

- Impact Level 1, $IL1 \implies Fail - Safe/Silent$

[4] The final condition for the last failure leading to a loss of function is usually omitted.

Definition: A system function, part of higher level control functions, which serves lower level control functions. A wrong output result is critical, whereas an untimely or fully unavailable output result shall be mitigated by depending on lower level system functions.

Examples: On-line mission planning and optimization, Path planning and optimization, autopilot and autopilot mode transition logic, status, control and mission data links, advanced environment perception with detection and classification of surrounding scenery and objects.

Failure Mode / Safety Measures: All single failures are detected by at least one redundant instance of the system function. A mismatch between comparing units results in a transition to the safe state of the system function, which is either a safe output state or the shutdown (silencing) of the system function. Depending on the nature of the function, an acknowledgment-based output verification might be adequate (e.g. Data links). Common cause errors resulting in multiple failures render the function unavailable immediately.

- Impact Level 2, $IL2 \implies (Fail-Operational-Safe/Silent) \vee (Fail-Operational-Degraded-Safe/Silent)$

  Definition: A system function providing low-level control to directly control or stabilize a physical system, who's outputs are interpreted by actuation elements. A wrong output result, as well as a loss of function is critical and directly leads to a catastrophic failure mode of the physical system, resulting in an unpredictable, uncontrolled and therefore uncontrolled state.

  Examples: N-degree of freedom control functions which provide actuation element set-points while observing sensors distributed in the physical system, state machines or mode control logic for critical function elements, supervision and monitoring systems which protect the system from critical conditions based on sensor readouts.

  Failure Mode / Safety Measures: After a single failure, which must be detected in all operating conditions, the system function is still fully operational. The failure is indicated to an operator or other system functions which may perform adequate actions. After a second independent failure, the function enters a safe state, by either remaining silent (no further output results provided) or defaulting to a predetermined safe state. If this cannot be tolerated (like in airborne systems, or nuclear power plants), a degraded operating mode is entered after a second independent failure. This ensures that the system function is able to transition the physical system into a safe state through a series of complex state transitions by focusing on the essential functions to ensure system safety and correct actuation element set-points (fly to safe crash site, return to base for operator hand-off, orderly reactor shutdown). This can be described as a degraded operating mode, which effectively

must require three independent critical failures in the same window of operation. [5] Common cause failures must be considered at design time, with adequate measures of dissimilarity in hardware and software elements involved in the execution of a system function.

The impact level classification aids the design process, especially the distribution of system functions with respect to their physical computing location within a specific compute cluster LRU or multiple LRUs which are part of a cluster. Due to the diversity of the different application scenarios, there is neither a single nor an off the shelf solution for the distribution of functions with different impact levels. Modularity is key in this regard, with distribution solutions offering as many degrees of freedom as possible, while still being able to be tailored to a specific certification effort. The high availability cluster offers a multitude of possible system function distribution scenarios. Even if there are enough computing resources on one single LRU, due to the high-performance multicore processor as the core processing element, fault tolerance can (or must, depending on the impact level) be accomplished by distributing multiple instances within and even outside the cluster. The latter is possible via the aforementioned smart sensor and smart actuation elements. Since redundant computing channels in form of the internal cores cannot successfully be claimed in a certification process for the multicore device, the execution on processing units other than the multicore processor itself is necessary.

As a result, the desired fault tolerance and failure mode behavior, as stated above in the impact level definition, is not bound to the compute cluster LRU itself. Consider for example an IL2 system function, with a Fail-Operational-Safe failure mode. Since a single failure shall not render the function inoperative, a single execution of the function on a cluster LRU would not be sufficient since a failure of this LRU renders the function inoperative. Therefore, one must provide at least one additional instance of the IL2 system function on a physically different cluster LRU. On the individual LRU, the function may now fail silently, since another redundant instance continues operation. I n order to satisfy the fail-safe requirement in case of an additional failure, at least three instances throughout the system are required.

A reduction in the criticality level, or in this case the impact level, of an allocated system function is common to many certification standards. The applicant is allowed to reduce the assurance rating by one step (SIL3 to SIL2, DAL B to DAL C) if a safety function is composed of two parallel subsystems, given sufficient redundancy (function-wise), a common cause failure analysis is provided and sufficient independence in verification/-validation, internal component choices, etc. is given. This also holds true for the impact level classification:

[5] In aerospace, no single failure or likely combination of failures may lead to a catastrophic condition (see [EAS17]), same holds for industrial applications, but latent faults have to be considered starting from SIL3.

The impact level of a subdivided system function may be reduced by one on an individual subsystem providing the functionality, if and only if architectural mitigation is provided on higher system levels to ensure the failure mode to be mitigated does not manifest in the overall (physical) system.

To continue the above example, note that the impact level on each individual cluster LRU executing the IL2 system function is reduced to IL1. This leads to a Fail-Safe or Fail-Silent behavior, which can easily be reached with simple cross comparison means on the individual LRU itself. On the level of the computing cluster we can still guarantee that no false output result is provided, and that the Fail-Operational character of the system function is preserved. A common cause failure mode analysis is still necessary, especially in case the compute cluster LRUs are not hardware dissimilar. We will address this issue later on in the LRU architectures with intrinsic hardware dissimilarity for complex components. As we will discuss later on, a higher order of redundancy and multiple degradation strategies of IL2 functions can be established with a particular LRU architecture, which also aids to reduce the number of compute cluster LRUs required to perform the IL2 system functions. This in turn frees resources, required for highly intense IL1 functions, like image processing, adaptive and non-linear algorithms, or knowledge-based / artificial intelligence decision systems or additional IL1/IL2 functions.

Also note that the impact level classification is not (per definition) related to a safety or design assurance level defined in the applicable standard. There is however an obvious relationship for IL0 (non-critical) and IL2 (high/very high criticality) classes with safety criticality due to their definition, which might lead to the belief that this relationship can also be easily established for IL1 functions. IL1 functions are difficult to classify due to their strong dependency on other IL1 functions, IL2 functions, and a large variety of ways to strike the argument for a particular safety level based on architectural mitigation strategies and particular implementation details. When we consider today's systems in general, IL1 functions own a medium safety level (SIL1/2, DAL D/C) due to the human as the monitoring and cross-checking instance. In future autonomous systems, most IL1 functions will own a higher (or the highest) safety level, due to the lack of the human monitoring. A full functional monitoring for these functions is certainly possible, which is already a possible architectural mitigation scheme to be executed within the high availability computing cluster. Therefore, if a relationship between criticality and impact level can be established, it should be considered particular and for a given, very specific system or design as an exception to the general rule that there is no such relation in order to prevent false conclusions.

We will address further possibilities and future work regarding the impact level classification at the end of this work, see chapter 5.2.

2.1.3 Derived LRU Requirements

After the intended system level configurations and the notion of impact levels just presented, we will now define the set of high level system requirements for the individual compute cluster LRU (Contribution C1). To designate the origin of each requirement, we will refer to the small system use case presented in chapter 2.1.1.1 as Use-Case 1 (UC1) and the large system use case, see chapter 2.1.1.2, as Use-Case 2 (UC2). Each requirement is identified by a numeric ID, given in table 2.1 along with the requirement description.

ID	UC1	UC2	Description	Domain
RQ1	x		The compute platform shall satisfy a safety claim of up to SIL2 / KAT3 (Pl.a-d) with a single LRU, Safety Claim of SIL3-4/KAT4 (Pl.d-e) depending on system architecture and other failure mitigation means on system level	Industrial
RQ2	x	x	The compute platform shall satisfy a safety claim of SIL4/KAT.4 (Pl.e) with two LRUs or more	Industrial
RQ3	x		The compute platform shall be certifiable in accordance with DAL C using one single LRU	Aerospace
RQ4	x		The compute platform shall be certifiable in accordance with DAL B with two LRUs maximum	Aerospace
RQ5		x	The compute platform shall be certifiable in accordance with DAL A with at least three or four LRUs (1oo3/1oo4/2oo5/ etc.)	Aerospace
RQ6	x	x	The LRU shall provide on-board fault detection / diagnosis / functional monitoring to detect wrong results of IL2 functions. A faulty computation result of any IL2 function shall not be distributed by the LRU.	Industrial and Aerospace
RQ7		x	The LRU shall provide a fail-passive / fail-silent safe state in a redundant configuration.	Industrial and Aerospace
RQ8	x		The LRU shall provide means to reach a safe state via separate system mode transitions requiring complex actions.	Industrial
RQ9	x	x	The LRU shall be able to maintain the safe state.	Industrial and Aerospace

ID	UC1	UC2	Description	Domain
RQ10	x		The LRU shall provide a fail-operational fallback operating mode, in order to execute complex system mode transitions required to reach a system safe state.	Industrial
RQ11	x		The LRU shall satisfy ISO13849 KAT3 single fault tolerance.	Industrial
RQ12	x	x	The LRU shall detect latent faults in its nominal compute channels by online monitoring or cross comparison.	Industrial and Aerospace
RQ13	x	x	Fault accumulation may only lead to a unsafe LRU state if multiple, physically independent subsystems of the LRU are subject to failures (nominal and monitoring failing at the same time).	Industrial and Aerospace
RQ14	x	x	Complex, multi failure scenarios which disrupt the nominal and monitoring compute channels of an LRU shall be mitigated on the system level via redundancy.	Industrial
RQ15	x	x	The LRU shall be designed such that system functions of different criticality and impact level can be executed side by side on the multi-core platform safely. The partitioning of different functions must be guaranteed at all times.	Industrial and Aerospace
RQ16	x	x	The LRU shall provide legacy interfaces such as CAN, RS232/485, ARINC429 or similar data-bus communication interfaces to support existing infrastructure, sensors or actors.	Industrial and Aerospace
RQ17	x	x	The LRU shall provide at least two pairs of redundant modern high-speed interfaces such as Ethernet (AFDX, EtherCAT, Powerlink, etc.), TTP, SpaceWire, with at least 50Mbit/s average throughput per interface, as well as a redundant, high-speed cross channel interconnect with at least 100Mbit/s average throughput per interface.	Industrial and Aerospace
RQ18	x	x	The LRU shall be designed to mitigate common mode failure, with regards to common complex COTS parts used within the LRU.	Industrial and Aerospace

Table 2.1: *High-Level System Requirement for the individual compute cluster LRU in the context of industrial and aerospace certification*

In the following, we will discuss the origins (use cases, certification background, etc) and give some descriptive examples as well as possible implications on the requirements in table 2.1:

- RQ1-RQ5

  The requirements originate from the certification levels to fulfill in common application scenarios. Basically, a single multicore-based LRU should be able to achieve the same rating as today's safe LRU designs, which is usually around SIL2 and DAL C. SIL3 may be reachable by appropriate top level system architectural precautions, such as secondary, dissimilar shutdown paths for industrial applications. Two units, shall be usable in the highest possible level for small systems, which is DAL B up to the CS-23 aircraft class and SIL3 for industrial applications. As the system size and moving mass and danger for passengers and personnel increases, SIL4 and DAL A is necessary, which can no longer be fulfilled by just two units, let alone by the required dangerous failure rate of at least 10^{-8} (industrial) and 10^{-9} (aerospace), as well as multi-point and common cause failure considerations.

  This will lead to small systems with a two-unit compute cluster, where size and weight are precious. Larger systems, with higher requirements and special consideration at the top-level system architecture for the given application domain, will at least feature a three-unit cluster which permits full operation capability after one LRU failure. A reduced-capability, or emergency operation mode may be established with one LRU left, in order to transition the physical system into a safe, controlled low-energy state.

- RQ6, RQ7

  In order to claim the SIL2 / DAL C rating, the LRU must at least feature onboard fault detection during runtime, to detect single failures with high levels of confidence, since the multicore itself cannot perform such monitoring tasks while also executing the nominal IL2 system functions. In addition, we explicitly deny the transmission of faulty compute results via external data busses. This basically resembles a Fail-Safe or Fail-Silent failure mode behavior. On the top system level, the fail-silent mode implies that whenever a result is emitted by an LRU, it can be trusted because it has already been independently validated by a second party in the LRU itself. Thus, the top level voting, or the voting performed via smart actuation elements, is eased and no further error detection on the logical signal level is necessary.

  While IL2 functions are covered inside the LRU, no checks are performed for IL1 functions. Since IL1 functions will be extremely compute-heavy, we will present an adequate redundancy and monitoring scheme in the next section along with the LRU architecture. IL0 functions are not monitored and may fail at will.

- RQ8, RQ9

  In case of a single failure inside the LRU, the safe state must be reachable. In some situations, the transition to a safe state requires several steps, for example, to execute a controlled breaking operation or to indicate a failure visually/audibly while executing an appropriate action to ensure machine operator safety. Larger, more critical systems (e.g. aerospace, reactors) involve even more complex state transitions and control to reach the safe state and may require emergency operating capabilities. Performance may be reduced during in this operating mode, but the ability to control the system is essential. In a redundant cluster, the remaining LRU(s) fulfill this task, or do not require the safe state transition at all when enough LRUs remain to guarantee safe operation until maintenance.

  When the safe state is reached, a failed component or function inside the LRU shall not be able to compromise the safe state, by issuing faulty, dangerous commands via external interfaces. This basically maps to the fail-passive/silent behavior, but requires a different set of countermeasures at the board level to ensure that failed components remain unpowered or reset until maintenance can be performed. The LRU itself may be operating in an emergency state during this time. Also, a controlled reset of a component inside the LRU may be performed to clear transient failures (mainly due to single event effects) and resume nominal operation after internally ensuring that the reset device performs as intended.

- RQ10

  Ensuring that RQ8 and RQ9 can be fulfilled with a minimum number of LRUs, the single compute cluster LRU effectively needs a fail-operation-silent failure behavior, with degraded operational capabilities after a single failure inside the LRU. In large systems, with a larger compute cluster or multiple clusters due to zonal safety requirements, this may be not necessary given the number of available physical units. Cost sensitive or space constraint applications, however, may require that for example IL2 or some IL1 functions are still available to ensure that the system remains controllable or can still determine its surroundings with only one or two LRUs in the compute cluster. It may also be required that the single fault is fully mitigated in order to reach the desired dangerous failure occurrence rate required with a minimal set of physical units. For example, most industrial systems may only be allowed to require two units due to commercial requirements, like localized stock size, overall cost for the customer or similarity to current solutions where only two units were required to achieve a medium safety level.

- RQ11

  Like in the aerospace domain (starting from DALB), from KAT3 onward in the context of ISO 13849, a single fault shall not lead to a catastrophic system failure.

This entails, that sufficient redundancy and monitoring at the system or LRU level must be present, to ensure that a single event (or root cause, originating from fault tree analysis, see [ADC96]), triggers a failure path through various layers of the system eventually leading to catastrophic failure of the entire system. While a large variety of countermeasures may be performed at the top system level, we want to provide an architecture, that fits into a new era of systems which are inherently single fault tolerant, without special considerations other than using COTS equipment as intended. We will discuss this topic during the architecture presentation in sections 2.2, 2.3 and later on the safety analysis in chapter 3.

- RQ12, RQ13

  As previously stated, the individual compute cluster LRU shall not emit wrong results for IL2 functions. This requires at least a full monitoring of IL2 functions, which, if possible, may also be present for IL1 functions. As such, the internal, nominal compute channel (most likely at least one multicore processor) is accompanied by at least one additional compute channel (physically separated microprocessor) to execute the functional monitoring.

  If faults accumulate inside one channel, they may not lead to an unsafe state, meaning that the safe state shall be reachable and maintained if only one channel has failed. If both channels fail independently at the same time, despite being hardware-dissimilar for example, then the LRU will exhibit a catastrophic failure condition where wrong output results are emitted. Those scenarios however, are sufficiently unlikely that most standards to not require an analysis of them up until the highest criticality level. But since large systems requiring such high criticality levels will feature at least two units in a cluster (likely more than 3), a catastrophic LRU failure is mitigated at the compute cluster level.

- RQ14

  If all LRU-internal means fail, the redundant compute cluster will provide fault detection and mitigation. The failed LRU remains silent. This is only relevant if the certification context requires measures against multi-point and common-cause failure modes.

- RQ15

  A very important requirement. If we do not manage to find an architectural, hardware and software solution, that enables us to execute software in a mixed-criticality scenario on the multicore processor, the entire concept and use-case of the multicore device is no longer valid.

- RQ16, RQ17

To ensure a smooth transition to the new top level system architectures based on the compute cluster, legacy low-speed interfaces must be supported. This is more or less a commercial requirement. In addition, the architecture shall provide enough throughput to enable the use of high-speed data bus standards for future system generations. The architecture itself shall not limit the bus bandwidth (within reasonable technical and financial limits of course) available on all interfaces. To support the single point of failure mitigation on the top system level, all interconnects should be redundant.

- RQ18

  To ease the argument in high criticality levels (KAT4, SIL4, DALA), where a common-cause failure analysis is mandatory, we shall design the nominal and monitoring channels with hardware-dissimilarity where possible. Depending on the LRU architecture, hardware dissimilarity may be given by design (if different devices are used in the nominal and monitoring channel in the LRU) but this requirement ensures nevertheless that the compute cluster LRU is not just 10 equal devices forming a high availability structure but complies to good design practice to in cooperate different technologies, manufacturers or architectural mitigation to deal with common-mode failures as much as possible.

With this small set of only 18 requirements, we already constrained the final board-level LRU architecture considerably. Most of these requirements are very familiar to the safe system designer but force us, to design a reliable embedded computing platform which is, by design, adequate for light, medium and high certification scenarios. Some of these requirements did not arise before the architectural design, but we introduced after the first design iterations as an implicit result of the components and strategies used.

2.2 Centralized Monitor Architecture

In the last sections, we concluded that most requirements can not be addressed on the complex multi-core processor alone and that architectural mitigation side the LRU and on the LRU cluster level are necessary. Our set of measures allows the use of multi-core processors in safety critical applications, despite being COTS and high complexity devices. After defining the surrounding top level system context, the high-availability compute cluster and a set of high-level requirements for the individual cluster LRU, we will now step into the cluster LRU. In the following two subsections, we will define two possible board-level architectural configurations, in order to meet our requirements and also discuss possible design choices for the individual board-level components and board-level subsystems involved (Contribution C3). While we present the technical point of view, the reader is referred to chapter 3 for the certification perspective. However,

some aspects for certification will be briefly addressed, due to their close relation with component choices, interconnects, or data-flow between the individual parties.

2.2.1 General Architecture

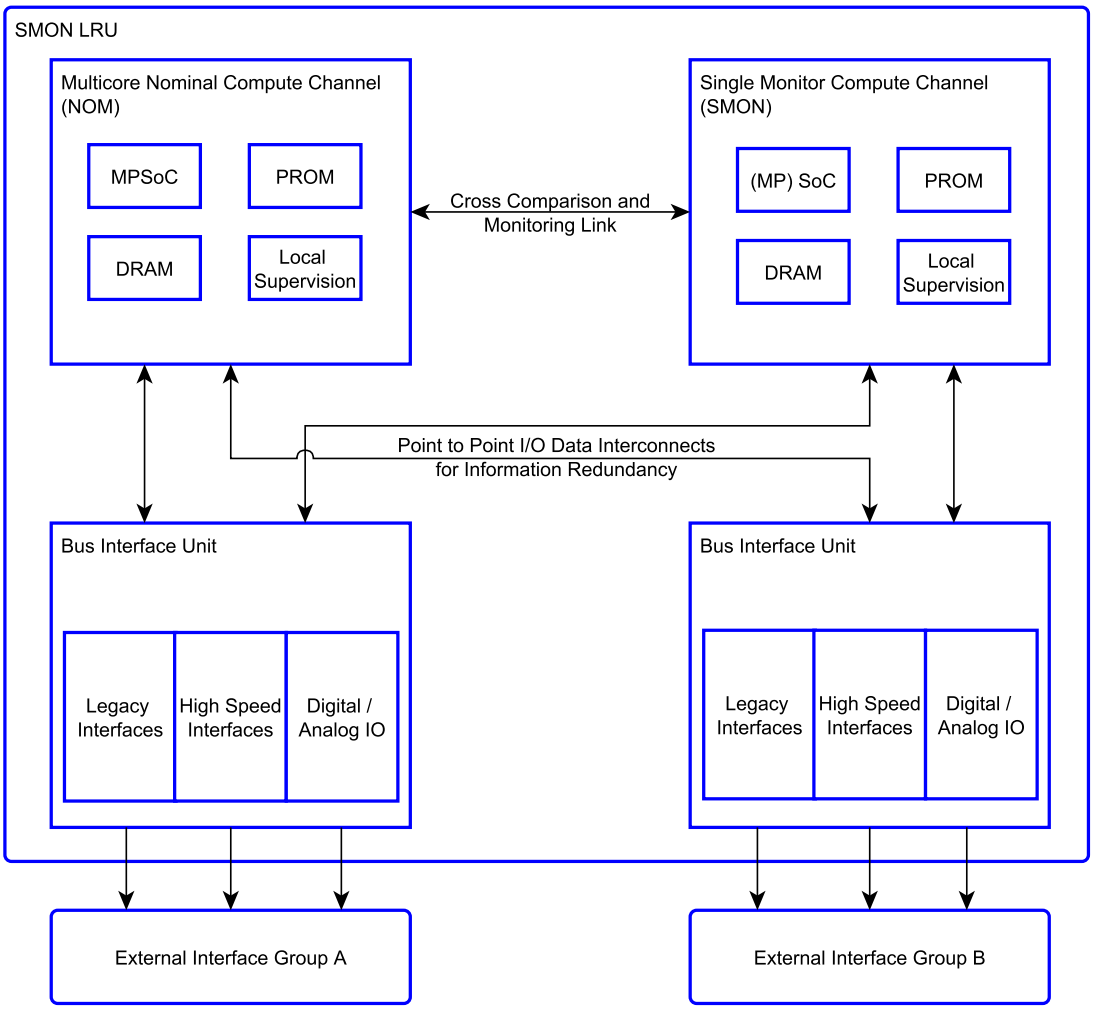

Figure 2.4: *Single monitor cluster LRU architecture high level block diagram*

The first architecture offers a single, centralized monitoring to the nominal compute channel. As shown in figure 2.4, a nominal compute channel (NOM) is the primary processing element. It is based on a multicore processor with its surrounding components like RAM, ROM, local power management, and required circuitry for operating the device. The NOM is monitored by a single on-board monitoring channel (SMON). The device used to monitor may be a less complex device, like a single-core microprocessor, but a multicore processor may also be used regardless of the component choices in the NOM. In any case, NOM and SMON must be hardware dissimilar to avoid common-cause failure scenarios in terms of hardware devices. The input data and interface processing is carried

out by redundant interface elements, here shown as FPGA-based bus interface controllers. Microprocessor or FPGA-Soc based interface controllers are possible alternatives, if the required interfaces and processing capabilities are met. The outside-world interfaces are split in two redundant data bus groups. Along with the system architectures presented earlier, this offers a fully redundant input path, starting from the top-level interconnects (cabling, connectors, etc.) right down to the lowest level inside the cluster LRUs (bus-transceiver, processing elements, PCB traces, etc.). By eliminating all possible single points of failure along the interfaces, true input and output data redundancy can be claimed. After some degree of preprocessing, like protocol decoding or error detection and correction, the input data is forwarded unmodified (Store-Forward-Element) by the bus interface controllers, via dedicated data busses to the NOM and SMON. Therefore, the input data is available by both independent sources (e.g. the bus interface controller) at the NOM and SMON. A cross comparison data bus link between the NOM and SMON is used to interchange output data and execute a cross comparison of the computed results on a per time step basis. The internal interconnects between the individual subsystems NOM, SMON and bus interface controllers must be capable of handling all input and output data, as well as the cross comparison data exchange in a timely fashion, such that they do not dominate the input processing delay. As a result, we choose standard Ethernet in a one-to-one, non-switched configuration as the physical layer in this architecture – simply by the fact that it is readily available in almost any current SoC or microprocessor with at least one interface (some offer up to 16 dedicated peripheral units), its speed (100Mbit/s or 1GBit/s nominal), good software support and COTS higher level safety protocol stacks like openSafety, EtherCat, Powerlink, AFDX, etc.

The nominal information flow and processing sequence is shown in appendix A.2. At first, a challenge-response signature watchdog check is executed between all parties. This ensures that every component is still operative and able to process further requests. Note that the watchdog check does not make any statement on the correct functioning of any device involved. Asynchronously to the regular, fixed system cycle in the computing elements (NOM and SMON), the input stages collect all incoming data packets and store them for retrieval. After the watchdog check, but still at the beginning of each time cycle (also called time step, or system time step in this work), the NOM requests the input data collected. An adequate time offset may be required by design to allow for all sensors to stream their input data to the cluster LRU. The input stages consecutively forward all collected data to the NOM and SMON. This aspect is important, because if and only if the input data is available to both parties in the nominal / monitor configuration (information redundancy), can true independent monitoring be claimed which is by definition in today's standards already a form of hardware fault tolerance or systematic fault tolerance. Based on the input values, both the NOM and the SMON compute their set of result for the critical system functions. The result of each computation is exchanged and compared between the two, which may happen at a defined handover point in time, or also in an

asynchronous fashion. The latter would most likely be the adequate choice if the SMON is also based on a multicore processor. If the comparison is successful (bit-perfect for discrete states, range comparison for continuous output variables), the results are packed into the appropriate data bus messages and signals, and transferred to the output stages. If the check fails, an appropriate action is carried out, based on the desired failure mode of the device and other criteria, like the criticality of the false output result (see chapter 3). In order to secure the transfer to the output stages and avoid double transmission, either the SMON or the NOM could only provide a higher order checksum of the output data stream (simple cyclic redundancy checks or others, see [HP10] or more sophisticated, cryptographic checksums like SHA, WHIRLPOOL, etc. see [2718]) which is compared at the bus interface controller to the checksum he computes for the output data stream. If double transmission is permissible in terms of bandwidth and timing, a simple bit-wise comparison may be employed to detect erroneous data streams from the NOM or SMON. The time step ends with the transmission of the computed output results by the bus interface controllers.

2.2.2 Board-Level Function Allocation

Based in the impact level classification introduced in section 2.1.2, we now distribute the IL0 to IL2 functions on the board level. Since we discuss the individual LRU here, which is part of a cluster, a certain set of IL0, IL1 and IL2 functions have been allocated beforehand to this specific LRU. Note that in most cases, IL2 functions will be executed on all cluster LRUs to reach a high level of fault tolerance and availability. An IL1 function might be allocated to a subset of units within the cluster, depending on the desired degree of redundancy for a specific function. IL0 functions could be allocated based on remaining free computing capability (after allocating IL2 and IL1 functions) without any special constraints.

IL2 function allocation is driven by the requirements for fault detection, single fault tolerance and application domain specific certification aspects which we will discuss later in chapter 3. As shown in figure 2.5, we allocate the IL2 functions to both the NOM and SMON. Together with the input data being available for both the NOM and SMON, this already provides true dissimilar hardware fault tolerance. We did choose this allocation specifically for several reasons:

- Re-use of algorithms for nominal and monitoring channels. This avoids the development of monitoring algorithms. Usual methods for software dissimilarity should be applied when demanded by the applicable certification standards to claim freedom of common mode errors related to software.

- A hardware fault tolerance based configuration is stronger than a monitoring based configuration for industrial certification. One of the major problems of certification in the industrial domain is to determine the diagnostic coverage provided by a

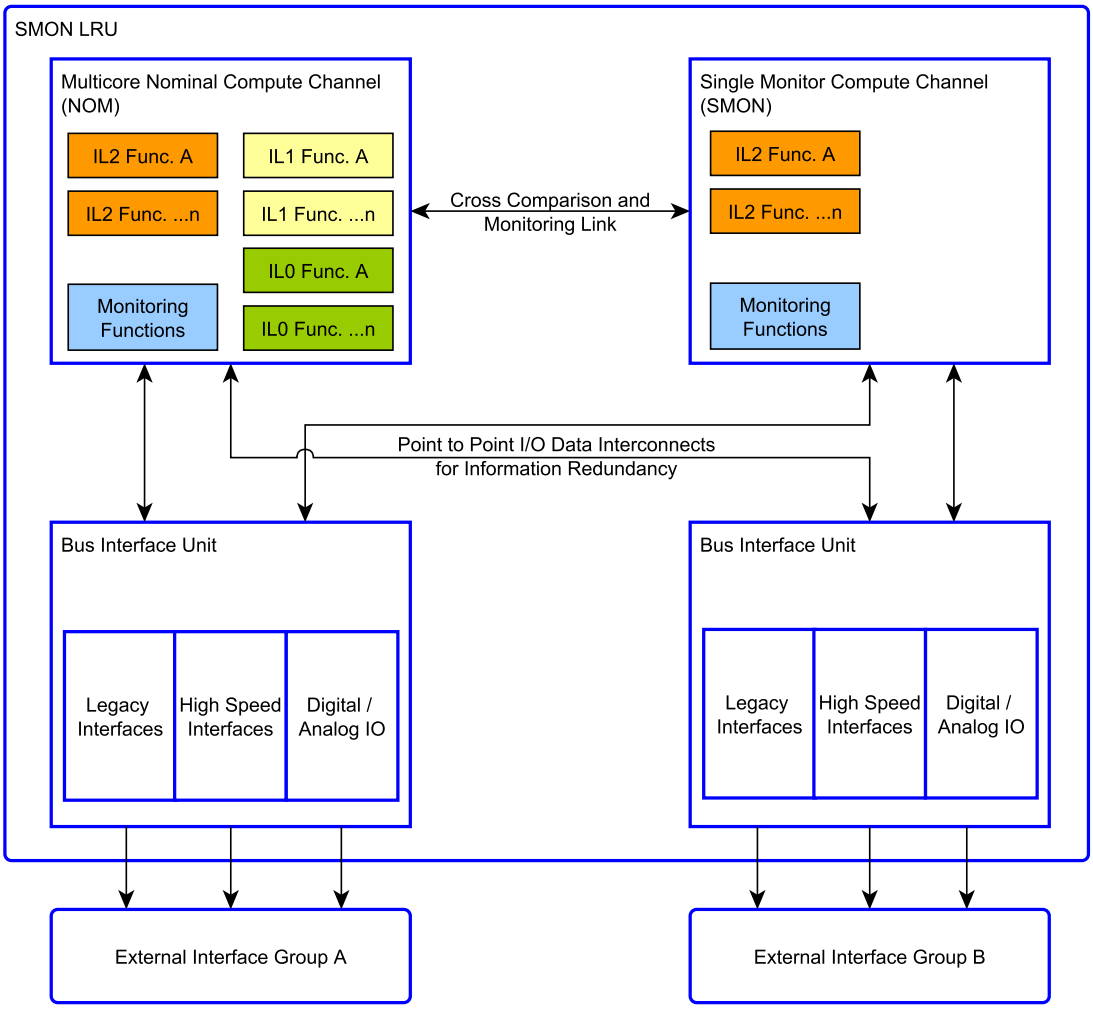

Figure 2.5: *Single monitor cluster LRU function allocation and distribution*

monitoring solution for a highly complex COTS device with limited/none design data access. When providing hardware fault tolerance with information redundancy, we avoid this pitfall. See chapter 3 for more details.

- Straight-forward monitoring information flow. NOM and SMON follow the same basic timing cycle of input, compute, cross-compare, and output. This results in a less complex configuration and better maintainability over the unit's lifetime.

- Either the NOM or the SMON is able to provide a functional degradation path, when one of the two units failed transiently or permanently. The permissible degradation scheme is dependent on the overall redundancy and availability concept of the high-availability cluster. The SMON could, for example, continue to provide output data when the NOM has failed or is about to restart due to a transient fault, flagging the data as "low trust" for the smart actuation elements or actively drive the system into a safe state. Holding up one channel when the other one has failed aids in

re-synchronization with the cluster, which is faster on a board-local level when only the restarting NOM or SMON needs to resync to the LRU internal cycle of the remaining second channel (either NOM or SMON, depending on which one has been restarted). This reduces downtime or the time in a degraded state, which in turn raises the availability of the whole cluster.

- The IL2 functions are replicated throughout the cluster. This results in a high-availability configuration, with numerous output results per function in each time step for cluster-wide voting or voting at the smart actuation elements in the system. If the number of LRUs within the cluster (or multiple clusters) is large enough, the cluster may be divided into several IL2 function pools, resulting different sets of IL2 functions allocated to a pool and the cluster LRUs participating in each pool.

IL1 are quite different to IL2 functions and therefore require special care in terms of allocation and distribution within the cluster. Il1 functions can be very resource heavy (consider mission algorithms, algorithms that involve map data, environment detection with object classification, etc.) which make it very difficult to provide on-board redundancy for IL1 functions. Doing so would result in very high resource demands on the SMON. So far, the only functions allocated to the SMON are the IL2 functions, which are functions predominantly in the form of control algorithms and state machines with limited demands. This permits a low-power or small processor solution for the SMON, depending on only the IL2 computing demands as the main driver (and of course protocol decoding and the monitoring we will cover shortly). We therefore decided to not move any IL1 functions onto the internal monitoring path, resulting in:

- Monitoring and redundancy of IL1 functions is provided via the cluster, not the individual compute cluster LRU. Note that the single LRU is able to run multiple IL1 functions, depending on the processing capabilities of the NOM (e.g. number of cores, memory bandwidth, interface bandwidth, etc.). The compute cluster then provides multiple instances of a critical IL1 function, resulting in redundant output results for cross comparison or voting within the lower level IL2 functions which depend on the output results of the IL1 functions. This already applies to a cluster of only two units (cross comparison only) and scales well up to high numbers of LRUs in a cluster for large, complex systems.

- Asynchronous compute path, without involving the internal monitoring. IL1 functions are usually less time sensitive, with higher cycle times compared to the high frequency, hard real time IL2 functions. Separating the IL2 and IL1 in terms of monitoring allows for the SMON to feature a smaller, more deterministic software architecture with less overhead, resulting in a simpler and less costly monitoring solution. This is especially true when considering the safety analysis and safety requirement specification.

- IL1 functions are offered advanced degradation strategies by the cluster. If for example, the number of LRUs in the cluster is reduced to failed compute cluster LRUs, some IL1 functions could be shut down or being executed in a degraded mode with voting downgraded to cross comparison, while other, more essential functions, are still allowed to be executed redundantly with voting. IL1 functions can also be scheduled dynamically within the cluster by activating or deactivating them on the individual cluster LRUs, effectively moving the physical computing location of IL1 functions within the cluster (common in distributed systems, see [Gho14]).

- While IL2 functions are mostly single core applications, with diminishing returns when executing them over multiple cores [6], IL1 might benefit from a multicore scenario. This is another reason for not moving any IL1 functions to the monitor, to keep resource demands on the SMON as low as possible. Another aspect here is the usage of non-certifiable operating systems for IL1 functions in a hyper-vised or virtualized environment on the NOM for future applications which provide a better infrastructure and COTS algorithm suites for many applications like machine vision or knowledge-based decision-making.

Lastly, IL0 functions are not part of any monitoring or redundancy strategy and are scheduled as needed to fill up remaining computing resources in the NOM. They may fail at will when the NOM of a specific compute cluster LRU fails. This is tolerable, since IL0 functions are non-critical, comfort functions. Alternatively, IL0 can be executed on a different set of LRUs throughout the system to remove their workload completely from the high-availability cluster. But in the spirit of canceling out as much LRUs throughout the physical system as possible by the compute cluster, we allocated the IL0 functions on the NOM.

The monitoring functions, also shown in figure 2.5, are part of all programmable components in the LRU and split up into the LRU supervision and inter-device monitoring and the internal device monitoring. As mentioned in the previous section, a signature challenge-response watchdog test is executed in each time step. This holds for all devices, in order to verify that each major system component is alive at the beginning of each time step. The system supervision is distributed among the NOM, SMON and the bus interface controllers, and will be detailed in section 2.4, together with the internal device monitoring, as part of the software architecture description.

Not depicted in figure 2.5 are all device internal function modules, like the RTOS or the important IO processing modules, whose allocation and distribution in the multicore

[6] The scheduling and operating system overhead vs. performance gain (with a more complicated cache and memory management in a certified setup) might consume the speed gain, especially since modern pipeline architectures already include the possibility to execute more than one instruction in parallel on a single core, even in small ARM Cortex-M devices like the Cortex-M7 micro-architecture implementation. Distributing a high criticality application over multiple cores can also imply certification issues with new failure modes and added complexity for their mitigation.

will be discussed in section 2.4 as well. As far as the implementation on a possible FPGA-based bus controller interface is concerned, they will be treated as black-boxes, since many possible solutions already exist as COTS code solutions for many industrial and aerospace bus interface standards.

2.2.3 Coverage of Requirements

The centralized (single) monitor architecture complies to many of the requirements defined in section 2.1.3. In table 2.2, we discuss the compliance. The design can be fully compliant (FC), partially compliant (PC) or not compliant (NC):

ID	Compliance	Rationale
RQ1-RQ5	FC/PC/NC	Will be discussed in chapter 3. SIL2 / DALC possible, higher levels depending on the surrounding system and the cluster.
RQ6	FC	IL2 functions are subject to on-board cross comparison and are executed redundantly within the cluster. The output result is only emitted if the cross comparison is successful. IL1 functions however might produce wrong results since they are compared / voted on the cluster level.
RQ7	FC	The LRU can passivate itself if a cross comparison error is detected or if the internal monitoring in the NOM or SMON flags an error.
RQ8	PC	Only possible via the cluster with multiple units. The unit is only able to provide a silent failure mode with one processing channel remaining, and only single, non-checked output results. If this is permissible on the top system level (safety assessment), a single LRU may initiate state transitions into a safe state.
RQ9	FC	LRU is able to shut down safely after a severe fault has been detected in one of the processing channels or trigger a restart of the failed channel while maintaining the safe state with the second channel.
RQ10	NC	The single monitor architecture cannot provide a true fail-operational failure mode, since no cross comparison can be carried out with only one processing channel remaining. The output results are therefore no longer trustworthy after a single failed internal channel.
RQ11	PC	In essence the architecture is compliant, but only with certain environment considerations. See chapter 3.
RQ12	FC	Full monitoring for IL2 present, internal monitoring for error detection as far as possible by the COTS device.

ID	Compliance	Rationale
RQ13	FC	Hardware dissimilar NOM and SMON reduce common mode failure probability.
RQ14	FC	Provided by the compute cluster when multiple units are present.
RQ15	FC	See section 2.4.
RQ16	FC	Flexible bus interface controllers can provide large numbers of legacy interfaces.
RQ17	FC	Flexible bus interface controllers can provide many high speed interfaces, depending on their capabilities. Modern FPGAs and even microcontrollers already offer enough resources.
RQ18	PC	This is only fully fulfilled, if not only the NOM and SMON are dissimilar, but also the redundant bus controller input stages (different manufacturers for FPGAs, different Toolsuites, etc.).

Table 2.2: *Single Monitor Architecture Requirements Compliance Matrix. Compliance denoted with: fully compliant (FC), partially compliant (PC) or not compliant (NC)*

In order to finalize the discussion of the centralized monitor architecture, one should note that this minimalist approach is intended to be used in large cluster scenarios. Due to the inherent fail-safe/passive failure mode behavior, this architecture does not provide sophisticated on-board degradation paths for IL2 functions. For large systems, with more than four compute cluster LRUs or even multiple clusters due to zonal safety considerations, this is absolutely acceptable and already provides a very high resilience against single component failures. A single failure might be tolerable if the system-wide voting strategies can cope with less trustworthy sources, for example in a distributed scenario with smart actuation elements. Most transient errors can be resolved by a controlled reboot, further enhancing the availability due to a reduced downtime of the single cluster LRU because of the second channel (either NOM or SMON) issuing a reboot command for the failed channel and maintaining a safe state until the failed channel is back on-line.

We already covered most of the requirements with this architecture as shown in table 2.2, but nevertheless, this architecture is not suitable for small, space constraint or cost sensitive applications where huge high-availability clusters are not feasible. The single monitor approach lacks the feature of a fail-operational failure mode behavior, which is necessary to guarantee a very low dangerous failure rate in configurations with only two to three LRUs in a single cluster.

2.3 De-Centralized Monitor Architecture

The second board-level architecture is focused around a fully fail-operational failure mode behavior, where a failed single element has no effect, and double failures mostly result in a degraded operating mode (Contribution C4). Raising the failure mode of the independent compute cluster LRU to become fail-operational allows us to reduce the number of LRUs required in a high-availability cluster. The overall failure mode is not affected in this case, since each unit has to exhibit multiple failures in hardware dissimilar internal channels, to render the LRU inoperative. While large clusters with more than three units do not need this high on-board fault tolerance (and should use the single monitor architecture from section 2.2), space, weight and power constrained applications will greatly benefit from the reduced number of LRUs required to justify a certain safety level. Note that for most applications with no secondary safety path or additional safety nets, we need at least two units in a cluster to overcome common mode failures introduced by the printed circuit boards (board cracks, broken tracks and vias, de-lamination, etc.), the physical enclosure, and environmental effects.

2.3.1 General Architecture

In order to provide the required fail-operational behavior for IL2 functions (RQ10), the architecture has to provide at least three independent sources of output results per time step for these functions. If only two sources are present, checking of output results is no longer possible with a single failed computing channel on the unit. For a sophisticated on-board fail-operational mode however, verified output results are a must, especially after a single fault has occurred. When looking back at the single monitor architecture from section 2.2.1, one should note that it features four independent computing devices: the multicore-based NOM, the monitoring channel (SMON), and two (likely) FPGA-based output stages. From there on, we perform the natural conversion from a single monitoring channel, to a distributed monitoring on all independent computing devices (with sufficient computational resources) on the platform. This cancels out the dedicated monitoring channel, and provides a hardware fault tolerance equal to the number of channels, which execute the IL2 functions besides the NOM. Depending on the application environment, and considering current developments in SoC integration (especially FPGA-based SoCs with a hard multi-core cluster) we propose the architectures depicted in figure 2.6 and figure 2.7.

Note that the second DMON architecture is more future oriented, and offers outstanding modularity due to the FPGA fabric used for implementing bus interfaces inside the distributed monitors, alongside with the dedicated hard IP cores for board-internal data links.

As shown in figure 2.6 and figure 2.7, the nominal compute channel (NOM) is provided by a multicore processor (along with its volatile and non-volatile memory, power

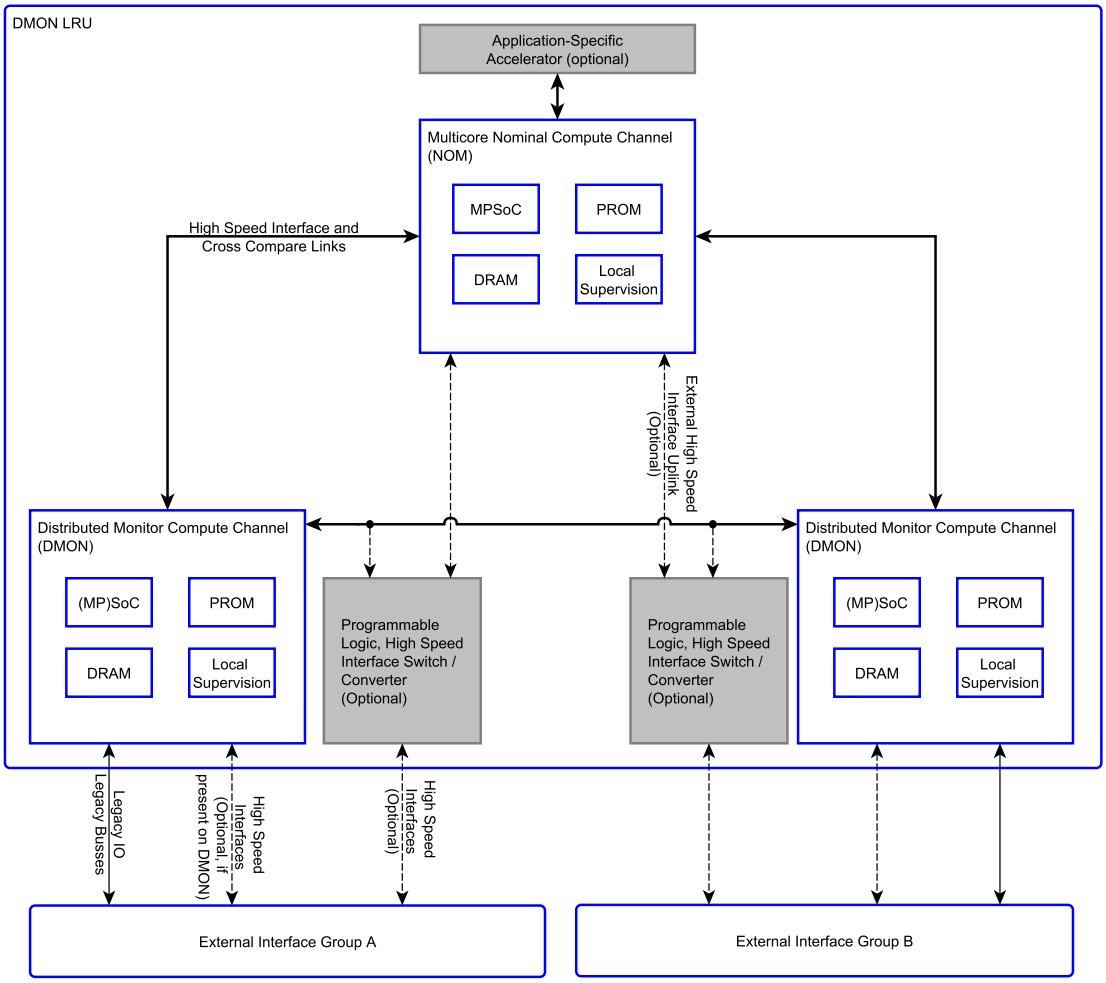

Figure 2.6: *Distributed monitoring architecture high level block diagram, based on COTS microcontrollers and optional high-speed interfaces*

management circuitry, etc.). Instead of a single, dedicated monitoring channel, we moved the processing of the redundant output results into the bus interface units. This leverages the fact that future microprocessor generations and also FPGA-based SoCs already offer enough computational resources for the system functions we will allocate to them. While they still provide the bus interface characteristics, like protocol decoding and the store-forward behavior of input data, we transformed them into distributed monitors (DMON) for the nominal channel. In order to be able to interface with legacy equipment and current top level system designs, they offer various low-speed interfaces (like CAN, low speed serial busses, special legacy busses like different ARINC flavors for example, via dedicated on-chip units). The LRU internal busses are dedicated high speed links, established between each DMON and the NOM. A cross-link between the two DMONs ensures that IO data can be exchanged, in case the NOM becomes unavailable or one of the interconnects to the NOM is lost.

The major difference between figure 2.6 and 2.7 lies in the way the external high-

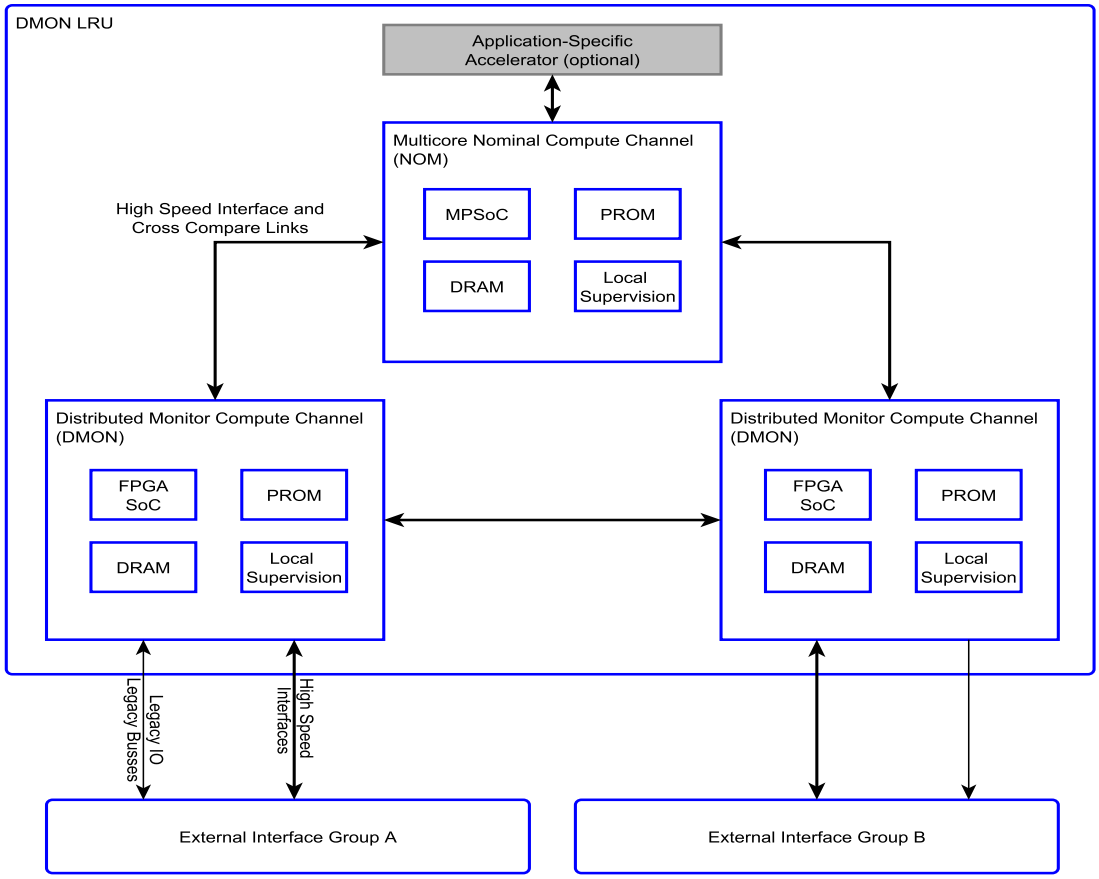

Figure 2.7: *Distributed monitoring architecture based on re-programmable SoCs*

speed interfaces are handled, which also impacts the data-links between the NOM and the DMONs, and between the DMONs itself. The addition of redundant FPGA-based high-speed interface controllers is driven by the fact that today's solutions either offer a lot of legacy field bus interfaces [7] or large numbers of high speed interfaces [8], but not a considerable combination of both in a single package. In addition, processing (floating point) control algorithms on an FPGA is not an ideal device usage, and considering the substantial efforts for the certification of configurable logic in various application domains (namely the aerospace industry), we consider a hard-IP solution in the form of a COTS microprocessor a better choice for this task. Additional topics like maintainability, modification costs should also be considered. Therefore, we split the high-speed interface processing from the legacy field bus interface and functional monitoring.

However, the architecture presented in figure 2.6 is somewhat depending on the interface between the DMONs and the redundant FPGA output stages in order to be a

[7] Those devices originate from the automotive or industrial domain where high integration with many lower speed field bus interfaces is required to lower cost.

[8] High counts of on-device Ethernet and other high speed busses are found in networking/telecommunication processors where typical field bus interfaces are not needed.

practical choice for real world applications. If the DMONs offer only one high-speed bus interface as depicted, the interface between the DMONs, potentially via the high-speed output stages) becomes critical, in the sense that it limits the capability of the LRU to fully operate all external high-speed interconnects (which may transfer safety critical data) in case of a NOM failure. Connecting the DMONs and the FPGAs via interfaces which do not feature enough bandwidth requires a functional degradation strategy (or emergency operating mode), which either limits the bandwidth on the external interfaces, neglects non-critical traffic, or triggers a shutdown of certain high-speed interfaces while the unit is in a degraded state. The exact nature of the DMON-DMON interconnect required and the possible choices for the high-speed interface controllers is strongly dependent on the application context and will result in some trade-offs. However, if the DMON microcontroller offers enough high speed peripherals, this issue might not arise after all. Note that the addition of further complex devices (more complex monitoring devices, and still two FPGAs for external interfaces), effectively lowers the mean time between failures (MTBF) of the architecture by design. Also note that the architecture does not impose restrictions on the processor choice within the DMONs. COTS multi-core processors, COTS safety specific processors, or simple microcontrollers are all valid choices depending on the application context and processing requirements for the function allocation described in section 2.3.2.

One way to overcome the internal interconnect issue is to fuse the DMONs and the external high-speed interfaces, as shown in figure 2.7. SoC with dedicated hard IP-cores (fixed, non-modifiable logic) coupled with a medium to large SRAM FPGA fabric began to show on the market around 2010, and have since then evolved in performance. These devices are commonly used for tasks like image processing, embedded networking applications or sensor fusion applications, where the integrated (multi-)core cluster fetches data from the FPGA fabric which provides the physical interfaces and preprocessing like decoding, filtering or transcoding. The beauty of these devices lies in their configurability in terms of interfaces, while also offering the fixed core cluster (mostly based on application cores like the ARM Cortex-A architecture) for higher level operating systems and vastly improved software support. As an addition, they also offer hard-IP peripherals, like Ethernet, serial ports and DRAM/ROM interfaces directly connected to the hard core subsystem. This internal architecture allows plenty of possible board-level architectural configurations. As shown in figure 2.7, the NOM and DMONs are now connected via dedicated high-speed interconnects, which are not constrained by the DMONs and can be chosen based on application requirements in terms of bandwidth and complexity. Note that we encourage to use hard-IP peripherals for the board-internal interconnects, since the fabric may be disabled or its configuration erased in order to remain in a safe state (true fail-silent by erasing the logic required to communicate). If only hard-IP peripherals are used, the connection between the NOM and the DMONs may still be kept alive in order to perform orderly resets or conduct for example internal fault logging. The external

interfaces, either high-speed or legacy field busses, are connected in two redundant data bus groups like in the centralized monitor architecture (see section 2.3.2) via the SRAM FPGA fabric. In programmable logic, the particular interfaces for each required data bus standard can be implemented and is often times already available as COTS pre-certified soft IP-cores (solutions exist for example for AFDX, EtherCAT, Ethernet, CAN, etc.). When the overall system architecture is centered around Ethernet-based switched busses, one could also opt for only COTS SoCs with numerous on-board Ethernet interfaces. See chapter 4 for an implementation example.

Note that some critical board-level modules, like the power supply, clock generation or interface protection circuitry, is not shown in the general overview figures. In order to not compromise the function allocation and functional degradation strategies described in section 2.3.2 and to overcome added single-point of failures, these shared modules shall be redundant as well. The NOM and DMON modules already include the necessary auxiliary circuitry for their processor to operate, for example the local power regulation (core supply, SoC domain specific supplies) via dedicated integrated power management controllers, clock oscillators or local DRAM and ROM, resulting in zero overlap between the three functional modules in these respects. As a result, off-the-self modules are likely available and offer a convenient solution and true modularization of the final LRU hardware platform. Using System-On-A-Modules (SOMs) for the NOM and DMONs also significantly reduces the complexity of the underlying carrier-board which holds these modules, since it merely contains routing, the carrier power supply (to divide the input power from higher bus voltages down to reasonable logic levels), external bus interface physical transceiver circuits, as well as hardware supervision logic like watchdogs or voltage supervisors with glue-logic. Adding redundancy to the carrier power supply finally reduces the remaining single failure points down to the carrier PCB-laminate and copper itself and is easily achieved by duplicating the input power supply (along with protection, filtering and input connector pins) with power OR'ing (hot-standby) or a fail-over logic (cold-standby). We concluded at this point, that the remaining common-mode failure source, the carrier PCB, should be compensated in the computing cluster and not in the individual LRU itself. Despite the high redundancy in the cluster (with at least two units in the small configuration, and likely more than 4 in larger systems), the PCB should be designed with proper design rules and common defensive layout strategies in order to offer some degree of resistance against laminate cracks, broken vias or de-laminated or broken tracks.

Also note that we do not recommend scaling this architecture by adding more DMONs. Increasing the number of monitoring and interface channels in this design should only be considered when more than two, physically independent and dissimilar external high-speed or field bus groups exist in the top-level system architecture. If this is the case (for example, due to zonal safety constraints) a third DMON might offer additional fault tolerance and functional degradation possibilities which could lead to better overall system

fault tree cut sets. However, one has to consider the trade-off between LRU complexity and complexity, LRU internal communication overhead, LRU component MTBF (also consider the common carrier PCB) and the complexity of the functional degradation schemes. We strongly believe, that the LRU architecture should be kept as simple as possible, with added fault tolerance provided by the computing cluster which scales very well with top-level system architectures and required dangerous failure probabilities. The same recommendation holds for the NOM. If more computing resources are needed, a higher-end multicore processor with more cores or higher clock frequencies should be used. Our architecture offers all possible freedoms in this regard, as long as the internal interfaces are provided by the implementers' SoC of choice and cooling/power constraints are met with respect to the surrounding environment context.

2.3.2 Board-Level Function Allocation

With the cluster-wide IL2 and IL1 system function allocation over the whole cluster and the basic allocation scheme presented for the single monitor architecture (both discussed in section 2.3.1), we now move on and allocate the system functions with different impact levels to the on-board units of the distributed architecture.

As shown in Figure 2.8, the allocation is very similar to the single monitor architecture. Only IL2 functions are distributed among the DMONs and the NOM, while IL1 functions are redundant via the compute cluster and no redundancy is present for IL0 functions. In section 2.2.2, the main drivers for not allocating the IL1 functions to the internal monitoring are lower performance requirements on the single monitor channel and better utilization of the numerous multicore NOMs within the compute cluster. While these two arguments still hold (also influencing the decision) the workload for the DMONs in the distributed monitoring architecture is slightly different than for the SMON in the single monitoring architecture. In addition to the allocated system functions, the DMONs also process all data transfers via the legacy field bus interfaces and, considering the FPGA-SoC variant, the high-speed bus interfaces. Depending on the actual bus load and the interface speed, the I/O packet processing may consume significant resources at the DMONs, which in turn reduces the remaining compute time (within each system time step) for allocated system functions and their internal voting. Even in the light of future FPGA-based SoCs with a powerful multicore subsystem, it is not beneficial to explicitly force the usage of very high-powered (computational wise, but also power wise) for most applications, since the effective tripling of the thermal design power of the involved processing elements makes passive cooling almost impossible in harsh environments. When considering the more conservative variant with dedicated FPGAs for handling the high-speed external interfaces, this problem persists. The DMONs employed in this case must feature lots of dedicated legacy field bus interfaces, which reduces the COTS microcontrollers available on the market quite significantly. The only devices on the market at the point of writing this work are domain specific safety controllers, mostly

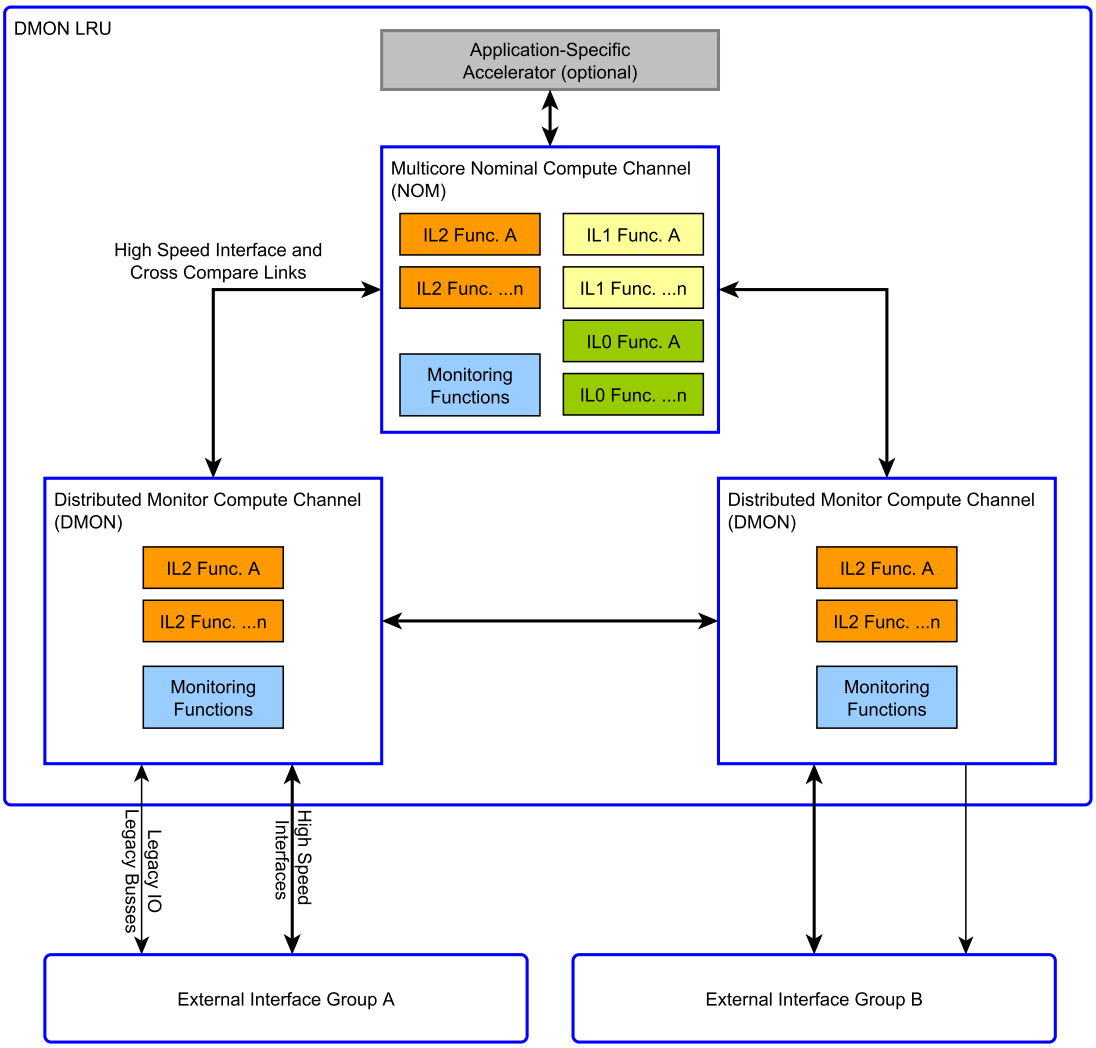

Figure 2.8: *Distributed monitoring cluster LRU function allocation and distribution (COTS FPGA-SoC variant)*

designed for automotive drive-train and motor control applications. These devices offer lots of features, especially in terms of reliability and inherent safety features, but offer very limited computing power (in the 200-500MHz domain) and are generally very resource constrained in terms of RAM and ROM. The expandability of these devices with external memories is very limited, since these features are not explicitly demanded by the automotive customers of chip manufacturers. Future expandability, upgrade paths and modularity is reduced, since one binds to a specific family of devices with an ecosystem attached, like safety enabled power management controllers (also called "System Basis Chips" or "SBCs"). In this constraint environment, IL2 function allocation is also not possible, given the workload on the microcontroller DMONs due to the IL1 functions and the interface handling. Another issue is the I/O and communication data link between the microcontrollers and the dedicated FPGAs, as discussed in section 2.3.1. Note that

the FPGAs should not process any system functions, since the type of processing involved is generally better suited for microprocessors and the certification effort (especially in the aerospace domain) for complex FPGA designs is extremely time and cost consuming. The FPGAs should only contain a minimum set of functions required to serve the high-speed interface, with package decoding executed at the NOM and the DMONs.

While the microcontroller plus FPGA variant offers a temporary solution, and might introduce some reuse of microcontroller or FPGA software from previous system generations, we advocate for the FPGA-SoC based variant and will focus on it for remainder of this section and the rest of this thesis.

Alongside the regular system functions, the obligatory internal monitoring tasks for each computational device are executed locally. This includes device specific configuration and validity checks (configuration of peripherals, status register monitoring, etc.) as well as RAM and ROM checks carried out periodically to identify silent corruption and upsets, as well as persistent defects in the devices. Also, part of the monitoring tasks is an inter-device, windowed, challenge response watchdog, which is executed at least once per time step to verify that all devices are responsive and can be addressed. Note that the quality of a watchdog does not permit checking further device functions in the sense of certification, even if a complex interaction between device internal peripherals, bus matrices, a processing core, memories and software is needed to successfully complete the check.

Not depicted in Figure 2.8 are the additional I/O-tasks, which are allocated at the DMONs for interface data preprocessing and lower OSI layers. While each device, also the NOM, will execute some form of interface data processing [9], most the work is shifted to the DMONs since their direct connection allows for faster cycle times in the I/O processing. Some parts can also be shifted towards the FPGA fabric, depending on the actual field busses used. Hard coded frame decoding in the FPGA fabrics should be avoided whenever possible (this does not include support functions, for example data error detection and correction offloading (CRC, forward error corrections) or restructuring into meaningful data structures for the processing cores). A fixed implementation prohibits easy maintainability (future expansions or slight frame modification on the field bus) and reuse of the LRU in other application contexts. In addition, the verification effort and re-certification costs (due to the modification in the FPGA logic) may be higher a result in larger deltas compared to a simple software adaption of decoding routines executed in the processor core subsystem of the FPGA SoC.

Since the distributed monitoring architecture inherently offers multiple output results for IL2 functions in each time step, some form of voting must be executed in the system. Dependent on the top-level system topology, the degree of redundancy by the LRU cluster and possible smart actuation, one may perform the IL2 output result voting as following:

[9] To claim a simpler store-forward scheme for the input and output data at the DMONs, which in turn decode the input data themselves for their allocated IL2 functions

- Intra-LRU voting, resulting in a single output result per LRU

  Voting is executed locally in the individual LRU by evaluating a voting algorithm on the DMONs. The NOM passes the output results to the DMONs. The DMONs exchange their IL2 output results. Based on these three values, a single output value is computed based on the individual needs of the specific IL2 function. The DMONs again exchange their voting results, and check whether the results fall into a specific validity window. If the check is positive, each DMONs transmits its computed voting results independently via the external interfaces. If a mismatch is detected, no output result is transmitted. The mismatch is handled internally and may eventually result in a reboot of one device (NOM or DMON) or the whole LRU. One should consider the high communication overhead in the final DMON voting stage and only use this method if very little redundancy is given by the cluster (for example only two cluster LRUs) or if the top-level system architecture demands one single output result per redundant communication channel. Note that there are still two independent results transmitted over the redundant external data links (hot redundancy).

- Intra-cluster voting, resulting in a single output result from the cluster

  After the NOM results are passed to the DMONs, each DMON compares the NOM result against its own result. If both values fall into a specified valid window, the DMON calculates an application specific average of the value pair and transmits its output results via the CCDL to the remaining cluster LRUs. If a mismatch is detected, the DMONs exchange their output results and again perform the valid windows check plus averaging. If a valid combination is possible, the averaged result is transmitted to the remaining LRUs in the cluster. Each LRU in the cluster repeats this process, resulting in the exchange of output results between all LRUs. After a second check and averaging phase over the LRU results, a single LRU in the cluster is assigned in a deterministic way (fixed IDs, deterministic rotating scheme) to transmit the final pair of output results via the redundant external data bus interfaces to the actuation elements.

- Voting on smart final elements

  The individual cluster LRU does not execute any voting of output results. The NOM results are passed down to the DMONs, where the individual DMON checks the result against its own IL2 functions output results. If the check is successful, the result is passed on to the external bus interface. If the comparison does not fit in a specified validity window, the DMONs exchange output results to identify the false result and transmit their averaged output value (over the correct results) via the external interfaces. No voting is executed in the cluster, where each LRU also transmits their output results to the actuation elements. In each time step, the

current internal status is exchanged between all cluster LRUs, including origins of false results, internal error counters, etc. to provide premature failure warnings and cluster-wide status availability in case of sudden LRU failures. The smart actuation elements itself collect all available output results from the cluster and perform an application specific voting or averaging.

The voting on smart final elements offers the advantage that no distributed voting scheme is required and synchronization/data exchange delays within the cluster and the LRU are minimized. Note that the number of output results can reach from six (cluster of two LRUs with distributed monitoring, producing three results each) up to 10 (cluster of five LRUs with single monitoring, producing two results each) or even more when clusters are made up of distributed monitoring LRUs. Building large clusters out of single monitoring LRUs is currently the better option in terms of component cost and thermal considerations. The trend towards FPGA-based SoCs is promising and will surely enable future solutions based on the distributed monitoring solution for large clusters, however. While we established a solid base for IL2 functions, IL1 functions offer no onboard-redundancy at all. Instead, they employ a simplified intra-cluster voting scheme, where in each time step, output data is exchanged between the cluster LRUs via the cross channel data link, followed by a suitable compare algorithm leading to validated and maybe consolidated results.

2.3.3 Coverage of Requirements

The distributed monitor architecture complies to all the requirements defined in section 2.1.3. In 2.3, we discuss the compliance. The design can be fully compliant (FC), partially compliant (PC) or not compliant (NC):

ID	Compliance	Rationale
RQ1-RQ5	FC/PC	Will be discussed in chapter 3. SIL2 / DALC possible, higher levels depending on the surrounding system and the cluster.
RQ6	FC	Full on-board fault tolerance, which offers at least single fault tolerance, and up to dual fault tolerance with degradation for IL2 functions. IL1 functions are redundant in the cluster. No unverified IL2 results are emitted by the LRU in any case.
RQ7	FC	After multiple errors, the LRU passivates itself with the remaining good processor when the two other processing elements have failed.
RQ8	FC	Single faults do not affect the ability of the LRU to execute IL2 functions and check output result validity. After a second consecutive fault, the remaining processing element is able to serve output values for a safe state transition before passivating the LRU or maintaining the safe state.

ID	Compliance	Rationale
RQ9	FC	See RQ8.
RQ10	FC	The operational capability of IL2 functions are fully retained in case of single/dual failures and permit to maintain a safe state, even if operating in a single LRU environment in case all other cluster LRUs have failed catastrophically.
RQ11	FC	The unit offers at least single fault tolerance, in certain scenarios even higher degrees of fault tolerance. See chapter 3.
RQ12	FC	Full hardware and functional redundancy for IL2 function allows for enhanced on-board fault detection. Monitoring between processing elements, and on each individual element, ensures premature error detection.
RQ13	FC	At least two subsystems must fail until the LRU can no longer operate. DMONs are fully independent and can be designed hardware dissimilar without major efforts.
RQ14	FC	Provided by the compute cluster when multiple units are present. Only two LRUs provide adequate protection for most applications.
RQ15	FC	See section 2.4.
RQ16	FC	Full modularity given by FPGA-based output stages with COTS IP cores for legacy field bus interfaces.
RQ17	FC	High speed connection between the NOM and DMONs, as well as fast and direct FPGA based IOs (Gigabit SerDes Lines) provide enough bandwidth for lots of interfaces, even if cost-effective COTS devices are used – higher-end devices provide an easy upgrade path for large amounts of interfaces.
RQ18	FC	NOM and DMON are inherently hardware dissimilar. To fully cancel common mode failures, the DMONs should be hardware dissimilar as well. Since the current COTS devices on the market, at the point of writing this thesis, offer similar feature sets and capabilities between manufacturers, we consider this absolutely feasible with reasonable efforts.

Table 2.3: *Distributed Monitor Architecture Requirements Compliance Matrix. Compliance denoted with: fully compliant (FC), partially compliant (PC) or not compliant (NC)*

In order to finalize the distributed monitoring architecture, one should note that this approach was in the first place intended for small and medium systems. The ability to operate in a high availability scheme with only two units permits the use in most autonomous applications where the cost, weight and power footprint should be minimal.

Nevertheless, the exceptional modularity of the distributed architecture allows for easy future upgrades, as well as the usage in larger clusters for highly critical system architectures with three or four units. The on-board IL2 function crosschecks enable advanced functional degradation methods and follow our approach of known-good output results from each cluster LRU for IL2 functions. The allocation and distribution of IL2 functions within the cluster remains open for the system designer and does not restrict the design freedom in any way, while offering multi-fault tolerance by design. The modular output stages are not bloated by large IL1 functions, which are only executed on the NOM. They can be scaled by adapting the NOM to the needs of the specific application scenario to provide the resources needed for more or less complex algorithms.

Future designs based on the distributed monitoring architecture may be more suitable for larger clusters due to the expected price drop for higher-end SoCs (both standard and FPGA-based) and the decrease in fabrication sizes on the DIE, resulting in lower thermal design power and lower LRU power consumption. The added flexibility when the DMONs feature FPGA-based SoCs is especially useful for the future upgradeability and flexibility in terms of interfaces. With wise component choices on the board, the design is also inherently hardware dissimilar, eliminating most common-cause failure paths in the LRUs itself.

2.4 Software Aspects

In the previous two sections 2.2 and 2.3 we presented the high-level system architectures of the centralized and distributed monitoring approach, along with some advices on possible hardware choices. We will address further hardware topics later with the exemplary prototype in chapter 4 and focus on software aspects in the following.

Both presented architectures are included in the software discussion, because most software architecture aspects are not directly related to one of the two board level architectures. This is especially true for the nominal channel (always realized by a multicore processor), on which will in the focus in this section. We will not discuss high-level architecture aspects for single core processors, since we consider this to be state of the art for many decades in various application domains. On the multicore devices, will address the high-level software architecture and modules involved, however. This also includes operating system specific topics, like multiprocessing strategy (asynchronous / synchronous), inter-core communication, as well as internal device monitoring and interface data processing (section 2.4.1). In section 2.4.2, we will discuss the mixed-criticality scenario with possible ways to spatially isolate system functions (which we will refer to as “processes” for the remainder of this section) with different impact or criticality levels and build a temporal isolation between them to establish a known-interference scenario. Instead of claiming interference-free operation, which is not possible with COTS components, we accept the contention and conflict scenario inherent to the multicore and discuss solutions

based on commonly available internal units. The basis of this discussion is provided by Mancuso et. Al. [MPC+15], who developed a promising approach for de-conflicting the multicore from a software perspective. The software aspects will be finalized in section 2.4.3 by discussing the board-level system supervision (the only centralized / decentralized monitoring specific subsection) and inter-processor software modules. The implications on certification in conjunction with the system architecture will be discussed later in chapter 3.

2.4.1 High Level Software Architecture

As previously pointed out in section 2.1, the only reasonable application of multicore-based LRUs lies in the combination and centralization of many system functions on a single computing channel. The inherent mixed-criticality character, given by the side-by-side execution of these functions, has certain implications on the software level. From a certification and engineering standpoint, the high-level software architecture should therefore address:

- The general processing scheme – Asking for a decision between symmetric and asymmetric processing on the multicore SoCs, including the individual rights and privileges, as well as the allocated function set and the execution of RTOS kernels (single vs. distributed vs. master/slave kernel instance(s), see 2.4.1.4),
- The nature of inter-core communication – Defining what and how data is exchanged between cores and consequently between different system functions as well as the outside world via the external interfaces,
- Interface handling – Defining where and how the interface handling is implemented. Is a single core or RTOS instance handling the full I/O workload, or is the interface processing allocated to each individual system function resulting in a distributed I/O processing?
- Internal monitoring and local integrity checks - Defining what device-internal checks can be executed on a regular basis to ensure the integrity of certain configuration registers, memory areas in the ROM and RAM and proper functioning of shared and core-local function units with on-line tests.

Addressing the above points, we propose the high-level software architecture depicted in Figure 2.9. The figure shows an abstract multicore with n usable cores C_n. The cores are connected to shared resources (bus matrices, memories, intermediate multi-processing supervision units, etc.) and shared peripherals (high and low speed interfaces). Each processing core is controlled by a dedicated RTOS instance. Included in each instance is a defined set of tasks which are scheduled by the RTOS core, for example on a priority, deadline or time division basis. We suggest an asymmetric multiprocessing scheme (AMP,

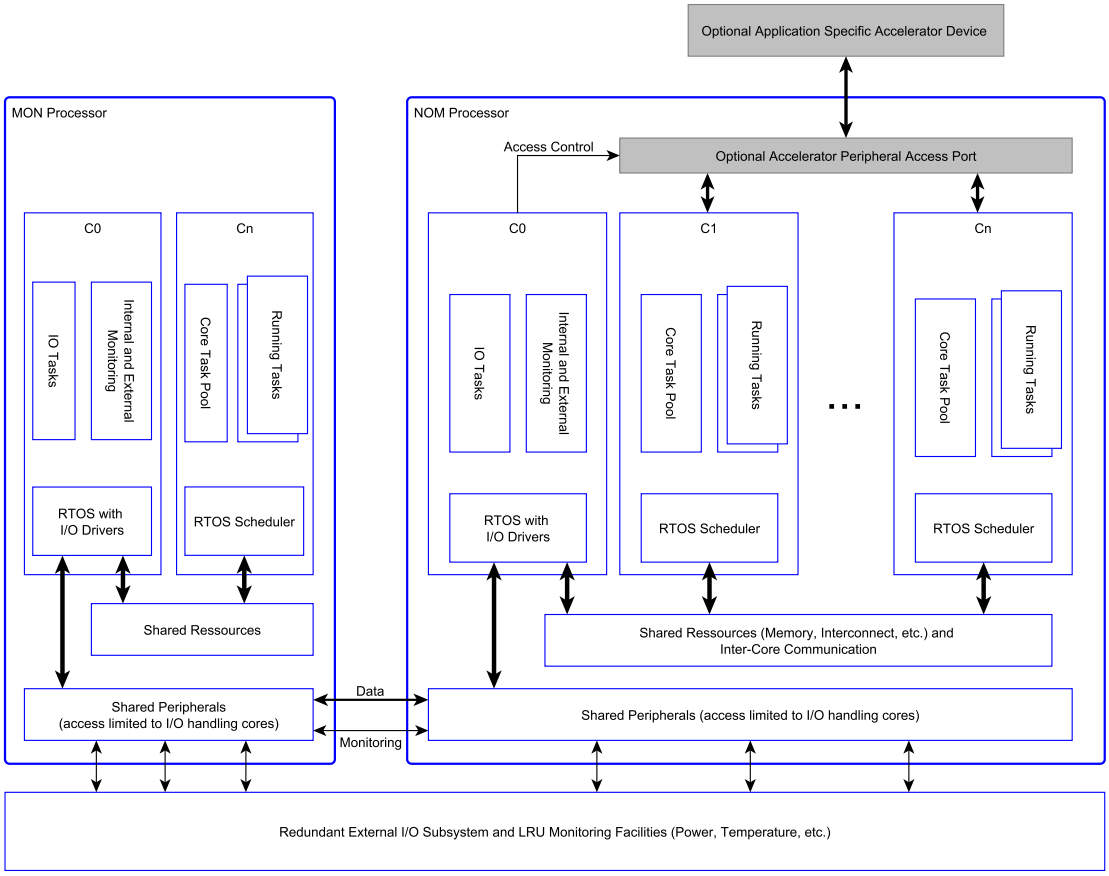

Figure 2.9: *Proposed multicore high-level software architecture for mixed-criticality systems; SMON configuration*

see [Mit16]), where not all RTOS instances can access the shared peripherals. Instead, the cores are divided in three functional groups:

1. Supervision/Management/Monitoring
2. IO-Processing
3. Application Core (Execution of System Functions)

As depicted, groups one and two have been merged on C_0. Depending on the effective number of cores which must be supervised and the workload generated by internal monitoring tasks, as well as the total interface bandwidth and number of data packages to be processed by the IO-Tasks, it may be necessary to distribute the first and second functional groups to separate cores to guarantee timely I/O processing. In Figure 2.9, the remaining cores execute only system functions (user applications, with different criticality levels). Splitting the application, IO and supervision workloads on different cores has several implications from an architectural and performance perspective:

- Better Determinism due to interrupt offloading

  Cores executing primarily application code are not interrupted by interface interrupt processing. This results in a more predictable and deterministic execution of application tasks

- Seperation of Task sets

  Each core can be assigned a clearly defined task set at design time. Assignment and scheduling of tasks can be further grouped and analyzed, in order to minimize for example inter-core communication overhead (execute small control algorithms which heavily exchange data locally on a single core, move higher system control functions to a different set of cores).

- Core Utilization

  Individual cores (or core-subgroups) can execute different tasks in parallel. Despite the nature of classic real-time embedded systems with their "reception, data processing (computations, filtering, extractions...) and transmission" [EAS18a, Footnote 6, p. 38] behaviour, the monitoring, IO-processing and application tasks can be executed in parallel on different cores. This is especially suitable, since these three functional groups can easily be designed with dedicated data transfer and synchronization barriers. For example, when the initial IO-phase is finished and critical control loops are executing on different cores, C_0 is free to conduct internal checks and tests. Untouched by the hard-real-time cycle time is an environment detection and classification suite executing on a different core which exchanges data at a much lower frequency with the functionally underlying control loops.

- Clear and pre-definable privilege levels

  As shown in Figure 2.9, the RTOS instances on each core can be tailored for the specific function group allocated to a core. If no peripheral access is needed, driver modules can be removed. This inherently hardens the architecture against some failure scenarios, for example when a faulty application software illegally calls RTOS or peripheral functions, leading to a misconfiguration or wrong output data being sent. In addition, the allowed memory ranges on the shared RAM and ROM can further be narrowed down, with access to critical configuration data (peripherals, shared units, etc.) explicitly removed. Since today's devices offer sophisticated memory management, it is non-trivial to prove that a SEE or design error in an application software leads to a privilege elevation in the memory management, when the respective memory management unit (MMU) mapping tables have not been configured for critical memory regions.

- Interface Abstraction and Isolation

Dedicating the interface data handling to a dedicated core (or the local management core without applications) allows to fully abstract the interface handling from the cores which only execute applications. The communication with external interfaces, as well as the decoding of the actual protocol used, is centralized, resulting in less overhead in the individual application and a stronger separation between system-specific data bus implementations. Bus specifications often differ significantly from system to system and disqualify some application modules from being fully reused over different system configurations. The central IO handling might also be allocated to several cores, instead of one single core, when large numbers of cores exist (many-core devices). In addition, the access to all external peripherals can be restricted to only allow modification by the actual IO processing cores. This isolation is highly preferred from the certification perspective and provides another layer of safety and insurance, that sensitive configuration data within the SoC cannot be inadvertently modified by a rouge application core (Single Event Effects, Programming Errors, etc.). This includes the memory regions allocated to certain peripherals, as well as access to the peripherals itself, which can be controlled by so-called system or platform management units on a core and privilege-level basis (see for example [Sem16, Section 6.4]).

If a single system function can be paralleled (thus make use of more than one core due to its internal algorithms), it is possible to distribute certain sub-functions over multiple cores by creating separate light-weight tasks on different cores which are synchronized by inter-core signals or barriers. The RTOS should provide these synchronization primitives within the inter-core communication framework, based on core-to-core interrupts which trigger rescheduling within the RTOS. Note that we effectively declared C_0 to behave like a local master core (hence the asymmetric multiprocessing classification) in the multicore SoC, overseeing other cores and handling the I/O communication.

2.4.1.1 Inter-Core Communication

The inter-core communication is an important aspect, not only performance wise, but also for ensuring proper core separation and safe communication between the cores. When the same basic principles defined on the upper physical system levels for data bus communication between different LRUs are applied to the inter-core communication, one can easily establish a simple yet sophisticated data transfer mechanism. The basic properties of an adequate solution are related to the common principles of black-channel communication [10] and basically consist of:

- Known Data Frame Layout - Well defined identification fields to differentiate between different types of data frames, as well as the source and destination

[10] In the IEC 61508 context, a black-channel is an unsafe communication channel which must be secured by an overlaid safe communication protocol.

- Replay Protection – Special purpose fields in the header or footer of a message ensure that a certain message is only processed once and no old data is constantly replayed to the receiver
- Data Integrity Protection – Checksums or forward error correction are used to verify the integrity of the message. The quality of the integrity verification depends on the complexity of possible alteration scenarios (single/multi/burst errors), the payload amount and the performance / detection trade off (Are special hardware units available for checksum/error detection and correction offloading?)

Hence, the effective payload or cross-core RTOS system calls must be wrapped in a proper frame structure, addressing the above points. The fastest possible way to exchange data locally within the SoC between cores is the use of shared memory regions between cores. Certain core architectures also offer means to directly transfer data payloads with special interrupt request between cores [FS13, Section 3.4.11.4]. However, these means should be used with caution, since the underlying hardware does usually not provide special means to protect or check the given payload (and if so, these hardware units then become safety relevant. See chapter 3. Transmitting explicit links to a valid memory location where a properly secured inter-core communication frame is located via interrupts should be permissible in terms of certification, since the downstream interrupt processing functions can verify the identity and validity before conduction further steps. Note that the inter-core-communication can be supervised by the master core, if required regarding failed transmission, replays, etc., to provide premature failure metrics for functional degradation strategies resulting in core or device restarts to prevent dangerous failures.

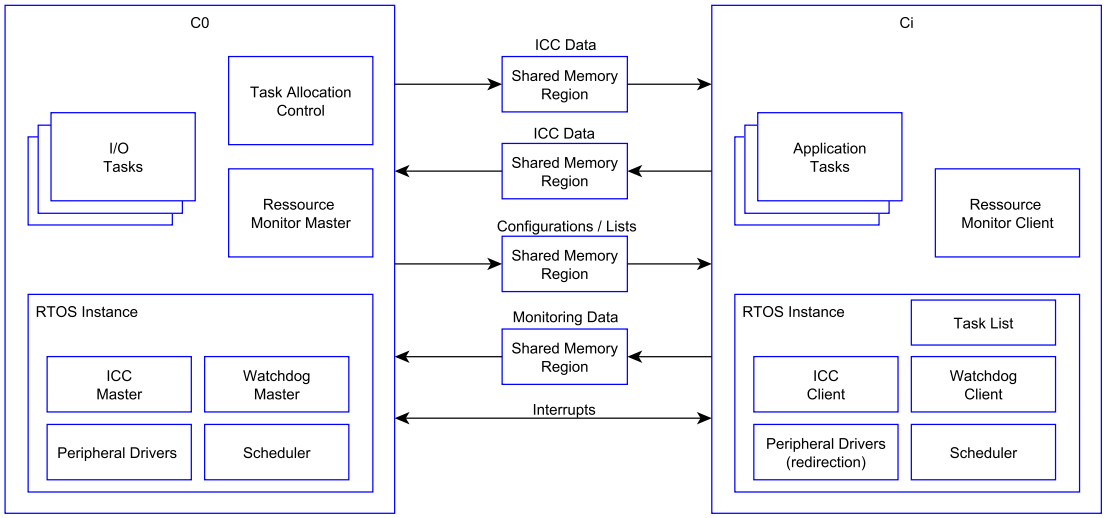

Figure 2.10: *Software Architecture, focused on the master and one computing core with system modules*

Figure 2.10 shows the high-level system architecture under a different light, with the

local master core C_0 and a slave core C_i. Like in Figure 2.9, each owns a local RTOS instance, which is driven by a core-local timer, sending time-tick interrupts to the instance (primarily intended for scheduling, but also timekeeping or time-stamping, depending on the use case). The inter-core communication takes place via a per-core shared memory region with the local master core. To serve the inter-core communication, the master possesses an ICC-master module, while the slaves handles communication via a client module. The difference between the two module lies in the supervision and configuration functionality added to the master. Both need to verify the correctness (integrity) of data packages for incoming transfers. A second shared memory region transfers watchdog and resource monitoring data from the slave to the master core. The data (see sections 2.4.2.2 for resource monitoring for temporal isolation) is processed by the watchdog master and resource monitor master modules on C_0 to not disturb task execution on the individual slave and allow central logging, as well as central handling of abnormal conditions. The on-device watchdog detects when slave RTOS instances become unresponsive or are unable to process interrupts (due to failures in the interrupt controller or associated core logic), a major issue when relying on distributed timing mechanisms to ensure proper hard real-time behavior. Especially the periodic time tick interrupt is crucial for orderly scheduling of the task set. For the resource monitoring, the back-channel from the master to the slave instances is provided via the bidirectional inter-core communication channel. As shown, the application tasks and the scheduling is controlled by an allocated task list for each individual slave core. All lists are managed by C_0 and can be changed, swapped or updated during runtime. Since the application tasks can run on any core, one can establish fail-operation degradation schemes where task lists are loaded dynamically based on slave core status. For example, a whole task set of critical tasks can be swapped from one slave to another to restart a failed or abnormal core, or to disable unneeded system functions when operating in emergency conditions with focus on impact level 2 functions with a reduced subset of impact level 1 functionality. The application tasks can access I/O data via virtual channels provided by the inter-core communication. The I/O access is transparent to the application and could be implemented per pre-existing standards which use software-defined ports, providing fast (cyclic) or slow changing data (configuration, status), for example [AEE10]. However, this could be undesirable for some application scenarios (for example, if existing software should be reused). In this case, one should consider implementing the optional peripheral extensions as shown in gray in Figure 2.10, and redirect the data access to the inter-core communication as a semi-virtualized peripheral stub. The control of the respective peripheral remains with C_0. Status data could also be forwarded if required by the application (link status, error counter, etc.) via the inter-core communication. In addition to the internal monitoring and supervision, C_0 also handles all incoming peripheral interrupts [11]. Besides the obligatory time tick interrupt required by the RTOS and, if required by the processing core architecture, inter-

[11] For higher core count devices, multiple cores can share the IO interrupt load

core interrupts for notifications between the cores (new data in shared memory, special events), the slave cores are freed from the I/O-interrupt load. Lastly, one should note that all internal supervision is coordinated by C_0, which in turn can be checked externally for its alive-status by other on-board processors, see section 2.4.3 for more. Note that depending on the board-level architecture of choice (SMON or DMON), the interface handling, for example polling sensor values or sending output frames to actuators at a specific point in time, can be handled by the interface units. Especially in the DMON architecture, the frames could be finally assembled at the DMONs themselves to reduce on-board round-trip delays, caused by circulating data messages before a store-forward stage at the DMONs emits the messages to the external equipment via the data busses.

2.4.1.2 Internal Monitoring

Albeit the sophisticated board-level architecture lifts a lot of certification weight from the multi-core device itself, we still must provide internal fault detection to some degree. This is mainly due to current regulation, but it is in general very advisable, since the internal monitoring and supervision can be executed at a much higher frequency (due to the local access) and has access to lower-level configuration settings within the SoC which is not possible by external monitoring.

The general goal with device-internal monitoring is on one hand a premature failure mitigation, which detects configuration, memory or other errors before they manifest as failures in the form of wrong or missing output data. This adds a layer of safety and helps to improve availability of the individual on-board channel (NOM, SMON or DMON) by reducing channel reboots or LRU reboots to clear a pending fault. The internal monitoring is usually specific to the individual device used, along with its hardware characteristics for detecting internal failures, protecting configurations during runtime (via register protections) or power, clock and memory error detection units.

On the other hand, internal monitoring is required by most standards for latent fault detection or as one mean to provide enough evidence to use certain on-chip units for safety related purposes (ensuring that one or more device characteristic is still met during runtime). In conjunction with an on-line monitoring, a startup/boot test is mandatory in almost any standards, which is part of the built-in self-test (BIST) or power-on self-test (POST). Depending on the nature of the peripheral and its complexity, power on-line and startup tests greatly vary in their complexity and significance. While configuration checks are rather simple to implement [12], verifying the correct behavior of transparent units on the device can be hard at times. For example, complex communication controllers, such as Ethernet MACs or internal accelerators, like GPUs or neural network accelerators require a distinct self-test strategy to conduct start-up tests. Some peripherals feature internal

[12]Memory mapped configuration registers of device peripherals can be compared at run-time against a set of expected values or can be check-summed (assisted by hardware) and also compared with a stored, pre-computed known-good value.

and/or external loop-back-test circuitry that may aid in these tests, or can be stimulated with special test patterns that verify basic functionality.

A special role is allocated to the RAM and its associated memory controller in most systems. In terms of hardware, today's devices feature a highly-integrated memory controller with dedicated transceivers to connect high-speed memory devices. The external memory devices are located off-device (either on the circuit board, or via package-on-package designs) and are not only simple memory cells in an array, but also feature configuration registers to configure the physical interface of the respective device (I/O drive strength, termination) and timing parameters like clock speed and a large set of protocol specific time delay values for data transmission and reception, as well as internal timings of the device. Most controllers do not feature special diagnostic features for connected memory devices, but rather include forward error correction and detection logic (ECC), also called single error correction dual error detection (SECDED). The ECC feature originates from the ever-growing server or high-performance computing market, where memory related errors (due to SEE or device failures) become relevant due to the amount of hardware installed. Since ECC memory is one of the key features for transparently mitigating sparse memory errors, one should in any case opt for proper hardware support in this regard, especially since the shrink in fabrication technology for SoCs and memory devices lead to a slightly higher upset rate than previous memory generations (see [HGG+11], [GHG+12], [KGB12]). If the chosen multi-core SoC does not support memory ECC, a triple modular redundancy scheme in software could help to mitigate memory related errors, but introduces a significant performance penalty due to the multiple stores, loads and compare operations involved with critical data stored in the memory. If the device does however support build in ECC and one relies on integrated hardware support, those units become relevant for certification.

Thus, the applicant must demonstrate that the integrated unit was designed in an appropriate process, execute on-line tests to ensure proper operation. Note that our board-level system design effectively mitigates per-channel errors on the board level, which also considers catastrophic memory errors and modified data in the individual channel. This does however not lead to a blank approval but rather significantly lowers the effort needed on the device level. Internal self-test measures become less critical in the sense of certification which helps in justifying the measures taken when the concrete hardware design of the SoC is unknown. For example, claiming a significant on-line test of the memory subsystem of a complex SoC is nearly impossible without deep design data access (not feasible in practice), resulting in a lot of assumptions about on-line test coverage and quality of an on-line test. Without system level mitigation, these assumptions do not stand up to certification and lead to tremendous project delays and cost increases in the past. During a certification eff:ort, the applicant could for example face the following questions: What integrated self-test features does the memory controller of the device come with? Do we want to use these features and rely on the process of the semiconductor manufacturer

and his ability to provide in depth evidence on request? If so, what costs are associated with the evidence? Can the included error detection and correction measures be disabled temporarily during runtime to introduce known errors in a certain memory region for an on-line test of this feature? These and further questions arising when in the process of evaluating potential SoC candidates should be answered as soon as possible in the design process. The availability and usability, not only in the technical sense but also in terms of certification, is one of the major pitfalls for future projects working with multi-core SoC technology. Software can only partially provide solutions for testing the critical function units associated with executing itself and storing data. After all, if false program code or data is loaded from the COTS memory, the question if software is under any foreseeable operating conditions able to detect and react is very hard to answer in general.

Luckily, our system level solution helps to effectively mitigate those types of errors where they manifest – resulting in added design freedom in terms of software and less stringent requirements for on-device error detection in general. This makes the software and hardware choice largely independent of the system-level safety assessment. But not all hardware features are irrelevant, especially for proper core and impact level based system function isolation in software, as we will discuss later in section 2.4.2.

2.4.1.3 Internal Supervision

C_0 internally supervises the remaining cores with different means. First, the status of each slave instance, or worker, is checked in each time step. As shown in Figure 2.10, each slave RTOS instance executes a watchdog client. Multiple data words are updated in a shared memory region, which serves as the back channel to the master core (in addition to the standard inter-core communication). The feedback consists of a multi-byte word (likely register width for convenience, 64bit in today's devices) which holds a pseudo-random number [13]. Each slave works on a different seed value for the pseudo-random number which generates a unique, but predictable sequence of digits, different for each slave core. This allows us to conveniently detect not updated values (replays), malicious overwrites (for example, when core n overwrites the region of core $n-1$, due to a design error or upset), as well as wrong updates due to program or memory corruption. The update can for example be driven by the RTOS time tick interrupt handler. While on-device software watchdogs cannot be considered a high-quality test in terms of certification, we can however guarantee, if the periodic check succeeds, that the slave instance was capable of serving the RTOS time tick interrupt (positive interrupt controller test) and was thus able to schedule accordingly since the time tick is the only trigger for the RTOS scheduler besides special task calls like yielding, or pending on a semaphore, spin-lock or barriers which are common to most RTOSs. The instance can therefore be said to be alive in terms of the RTOS. Note that an additional inter-core communication framework

[13] Since security is a minor concern at this stage, a simple generator can be used like a linear congruence generator [Rot60]

is likely to be able to detect missing output data when an RTOS scheduler gets stuck, providing additional detection means when the scheduler fails but is nonetheless able to serve the time tick interrupt handler. The monitoring via the very high-frequency time tick interrupt however allows us to execute the alive-monitoring at scheduler frequency (1kHz to 100kHz, depending on the RTOS and core used, trade-off between overhead and scheduler tick resolution needs). We are therefore able to prematurely detect instance failures within a time tick and respond timely when IL2 functions are concerned. The verification, executed on C_0, can be statically scheduled when the time shift between the slave and the master cores are known and consist of simple random number generations and checks or be served by the local time tick of C_0, if so desired. The pseudo random number generator must be simple and not computationally expensive to reduce the load generated on C_0. We will address the internal watchdog later in chapter 3, when discussing its value for certain certification cases.

Secondly, all critical SoC configurations and states are supervised by the master core, as mentioned earlier, with the help of checksums or reference copies of relevant register arrays. This also accounts for the task lists of the slave cores (Figure 2.10), which have to be accessible by C_0 but not between the slaves themselves. The master can then periodically verify the integrity of each task set, without interfering with the slaves in any way or disturbing running applications on the slaves. The task set can be viewed as a set of scheduler parameters, including pointers to tasks, parameters or arguments, as well as scheduling parameters like priority, time slot numbers, etc. As with the internal watchdog, the significance of the periodic configuration and parameter checks is also limited in the sense of diagnostic coverage for certification, especially for higher criticality levels. Based on our proposed board-level architecture however, the need for strong on-device tests is dramatically deduced for both industrial and aerospace applications, where they serve as a premature failure detection and add-on to the already strong system level measures. Architectural mitigation on the board level already detects the manifestation in wrong, missing, or untimely output results of complex electronic devices. On-device measures are also required by certain standards which justifies their presence. The abstract concept in our case is, that the internal master oversees the relevant shared resources and ensures that malicious actions by slave cores, due to transient or permanent errors, are discovered rapidly. A shared resource in this case is either defined as a function unit shared by multiple cores or data residing in the shared memory. Unfortunately, we cannot claim high confidence in these tests, due to the lack of hardware support. In theory, the core-local memory management units and system-wide management units (called Platform Management Units, or IO Management Units in some architectures) would solve the problem in its entirety, but they are not backed by sufficient evidence by the device manufacturer to justify claims based purely on them without additional tests. We will use them later in section 2.4.2.1 for spatial isolation, but their usability for certification claims is limited, and results in on-line checks of each unit (positive and negative tests

required) and periodic configuration checks. While system-wide units can be checked by the master core, core-local units can only be addressed by the individual slave itself. This leaves us with periodic checks on the slave for its memory management unit, cache management (for local caches) and local function units, like timers and other architecture specific function blocks. We will discuss the implications and claims possible with these "untrustworthy" internal units later in chapter 3 in greater detail, with the resulting implications to complete the picture why certain design decisions were made. Since the slave must report its internal check results back to the master core, the dedicated back-channel helps to consolidate the status indicators from all slave cores at the master. Based on them, we can introduce further layers of functional degradation and fault mitigation schemes, besides the strategies already discussed in 2.2 and 2.3, when a core has been detected as faulty:

- Core shutdown (and restart), without task reallocation

  A faulty core is shut down, and optionally restarted, and its tasks are lost. This also includes potential IL2 functions allocated to the core. This results in permanent (temporary) unavailability of these functions, which must be mitigated at board level alone, effectively resulting in a failed device on the board level.

- Core shutdown and restart with temporary reallocation

  A faulty core is shut down, followed by an immediate restart. Upon detection that a core has temporarily failed, its tasks are relocated. The relocation only takes place if at least one IL2 function, or depending on the cluster design, a critical IL1 function is scheduled on the core. The task set is relocated to one of the remaining tasks, if and only if the relocation target itself does not contain any IL2 or IL1 functions in its task list. This approach requires at least one spare core which executes only IL0 function in the nominal case, which could be impractical for smaller devices with limited core counts. After a successful restart of the failed core, and probably a small delay to regain confidence and execute the on-core diagnostics for some time steps, the original task set is reassigned and the relocation target also returns to its original task set. If a core cannot successfully execute a successful restart or if diagnostic measures fail, it remains shut down and the devices remain in a degraded state, focusing on IL2 (and IL1 if enough resources are available) functions solely.

- Core shutdown and restart with permanent reallocation

  Same as temporary relocation in general. However, the task set which has been relocated upon failure remains on the core targeted by the relocation. The task list of the relocation target is permanently assigned to the recovered core. This saves one task list transfer, but breaks the originally assigned task allocation, which might be an issue for certifying authorities if only a single task allocation is permissible by regulation. From a purely technical point of view however, the permanent and

temporary allocation are functionally equivalent on symmetric multi-core devices. Asymmetric devices require elevated validation and verification efforts to prove that the desired reassignment can be scheduled on cores with less performance or a different core architecture.

- Core shutdown and reallocation A faulted core is disabled and not restarted. Its task set is permanently relocated to a spare core or a core with enough headroom to host its functions. The multicore device is now degraded, possibly resulting in an overall degraded state of the individual LRU. This might lead to disabling IL0 and certain IL1 functions, to free up resources for relocated IL2 functions. Since no IL2 functions are lost, they can still be provided to the LRU cluster and are checked for consistency by the LRU internal, on-board cross comparison. Operating in degraded states and still providing output results can be considered where the system can not be easily maintained (low repair rates in the sense of a Markov model, difficult or costly repair) or where the system has to operate for extended periods of time without maintenance (long mission times). A full power cycle can remove certain types of errors (for example, memory errors, see [HGG+11] and [GHG+12]), which can be an option for larger LRU clusters, where an individual LRU can safely be fully power-cycled without compromising the overall safety of the LRU cluster. Note that a full power-cycle of the unit is required, which might entail external power-management units to cycle the unit or special measures inside the unit's power input / supply stage to do so.

Depending on the application context, control of a possible reallocation mitigation scheme right down to the individual task could be desirable, to provide a finer graded control mechanism. The resulting scheduling problem, how to redistribute the vital IL2 and IL1 tasks on the remaining slaves, is not necessary solvable in real-time. And in terms of certifiability (static defined task allocation, tested), the question arises if it is more suitable to pre-determine emergency task lists. These lists, based on design time scheduling for k out of n remaining cores, can then be scheduled as needed to provide a deterministic and predefined system behavior for the functional degradation paths. With large core counts, proper tool support with validation and traceability throughout the process is crucial for real-world projects.

2.4.1.4 Task allocation and Scheduling

This turns us to the general problem of multi-core task or process scheduling in general, for the nominal case. Scheduling of program tasks in hard and soft real-time environments is already well covered, see [Deu11] for an introduction and further references. Which specific scheduling mechanism (rate-monotonic, deadline-based, time sliced, etc.) is not at least depending on the capabilities of the RTOS selected for the individual use case. Employing the proposed asymmetric multiprocessing architecture, with less-privileged worker

instances, we are however in the comfortable position of adapting existing single-core only RTOS solutions. If the RTOS itself supports the targeted core architecture (of the individual core the RTOS is supposed to be executed on, could be different architectures in a heterogeneous multi-core device), knowledge of the other cores is not needed for the instantiated scheduler core. The only coupling in the kernel space is given by the inter-core communication module, which is fully independent and transparent for the kernel, since it only operates on shared memory and notification interrupts (if available on the selected device). C_0 includes further routines for initializing, monitoring and supervision of the slave cores on the device, but even these software functions do not entail a kernel or scheduler modification, since they operate only on special purpose function units or registers in the SoC, which is comparable to a standard device driver for controlling an on-device peripheral (state-of-the-art). The access to core-local function units, namely timers and memory management units, is fully equal to a single core device, and does not require any further modification, besides the design time definition of shared and core-local (private) memory regions. The high reusability of existing solutions provides a distinct advantage for project risk and up-front invest, a major issue for many early adopters of new technology in regulated market environments.

Fully SMP-capable operating systems on the other hand do not offer this advantage. When opting for an SMP operating system, one should at least consider and clarify ahead:

- Kernel complexity

  SMP kernels are inherently more complex than RTOS "scheduler-only" operating systems. Since one kernel instance manages the whole SoC, a single fault in the scheduler disrupts all tasks. The same holds for the interrupt controller which is coupled to the core executing the kernel. The central task and process management allows for easy core affinity management, but this is not required in most statically defined embedded systems. Does the application really require dynamic task reallocation? What other features of a complex monolithic kernel are required?

- Slave core management

  Depending on the implementation in the specific SMP kernel, is the operating system able to deal with core failures, lockups or misbehaving cores? How is the inter-core communication handled and how is the cross-core system call infrastructure implemented? Does the OS employ any sort of worker on a core to implement time slicing remotely or is the entire scheduling handled at a central location?

- Mixing of multiple operating systems and use of legacy software

  Core-local instances allow different RTOSs on individual cores, as long as they support the internal monitoring, supervision and inter-core communication. This is especially useful for large, legacy software based on a special purpose RTOS, which is common in many application domains. SMP operating systems usually require the

need for hardware virtualization to execute different kernels side by side, since they are built to execute exclusively on a device and reserve resources such as interrupt management or memory management for themselves.

- Certification cost

  Due to the added complexity and code size, certification cost for new features and modifications on large, complex SMP operating systems is a major showstopper.

Note that, at the time of writing this thesis, no commercially available (real-time) operating system with SMP support does offer the necessary features for spatial and temporal isolation, nor does it support most of the architectural aspects described in this chapter. Due to the enormous cost associated with modifying already certified RTOS-solutions, most vendors currently fear to invest until they can copy from proven solutions. There are many possible solutions, which are highly specific to the targeted application domain and the special requirements of the physical system there.

2.4.2 Mixed Criticality and Determinism

One of our main design requirements, defined in section 2.1, was the call for a mixed-criticality platform. The requirement was driven by the use case to concurrently execute tasks of different criticality (and impact) levels on the individual cores inside the COTS multi-core device. On one hand, this allows us to combine previously separated LRUs on the system level, into one single high-performance platform. On the other hand, since we operate on non-safety COTS devices, we cannot claim that the individual cores are sufficiently electrically independent. In consequence, we are not able to claim sound evidence for the internal core independence when a single system function is executed on multiple cores at the same time for redundancy. One could either disable all other cores while a critical function is executed, or provide means to spatially and temporally isolate the individual cores on the device, which finally leads to the mixed-criticality scenario. The problem of proper core isolation is closely tied to the SoC device architecture. See Figure 2.11 for an example of prominent device architectures, abstracted to a reasonable level.

Shown in Figure 2.11 are two multi-core device configurations found in today's devices. They both consist of a core complex, with n physical cores. Each core features its own instruction pipeline, function units, floating point units, memory management unit and at least one cache level (at least L1 [14], L2 possible for complex core architectures). The cores may also contain further peripherals, for example local timers and a local interrupt controller. To interface with the outside world, certain peripherals allow access to common

[14] Different cache locations in a device are named after their proximity to the processing pipeline. L1 being the closest, followed by L2 (per core or shared between cores) and L3 (usually shared between cores or not present).

data bus interfaces, while others provide power management, clock management, security and platform management, or interfacing with memory devices. The main difference between the two architectures is the location of the core (cache) coherency management and the location of the second cache level (or third level, if the device has two core-local cache levels). The upper architecture implements the second cache level as a shared cache, inside a central coherency unit. The coherency unit handles core interfacing, cache coherency management, as well as access to the primary interconnect fabric on the device. It also manages the cores in terms of power and frequency settings and interrupt routing. The lower architecture on the other hand features a so-called local backside-cache, which is local to the individual core. Coherency management is provided by the primary interconnect fabric, which notifies neighboring cores when common data values in the shared data memory change and a new value must be re-fetched by the local backside-cache. A possible third cache level cannot be implemented in a central unit (because there is none) an is connected to the switch matrix instead. Note that architectures are moving more towards shared cache architectures.

Shown as well in Figure 2.11 are the function units, where cores with equal rights and priorities in an SMP multi-core system interfere with each other during normal operation. The bandwidth of shared function units must be shared among the accessing cores. This leaves the door wide open for non-critical software, to effectively block or slow down critical software when two, or more, cores interfere on shared function units. Since this fact is unacceptable in classical single core systems, where functional separation is also mandatory for certification if a mixed-criticality system is anticipated, it is likewise one of the major issues for a successful certification of COTS multi-core devices. By design, our high-level software architecture already deconflicts the shared data bus peripherals, by dedicating a single core to each interface unit. This removes conflict potential and redirects interface-interrupts to the "I/O-core", which does not execute application tasks. The main interconnect matrix usually offers more than enough bandwidth to handle the connected cores and peripherals concurrently and is therefore not relevant for core to core interference. However, one should evaluate this claim with the device manufacturer beforehand when in doubt, although most high-performance devices are sufficiently optimized to not provide performance bottlenecks in their originally targeted applications (networking, high-performance computing) [15]. Remaining for a software-based deconflicting solution are therefore the shared cache (if existent) and the shared memory, namely the DRAM where data and program code is stored during runtime.

The interference on the cache and DRAM have different origins in the design of the two units. In the shared cache, which usually operates based on cache lines and cache ways (which point to certain locations in memory and cache the content, see for example

[15]Information on the internal bus matrix is spare and usually confidential. However, a manufacturer of networking processors presented evidence under NDA that their internal bus matrix is not a limiting factor.

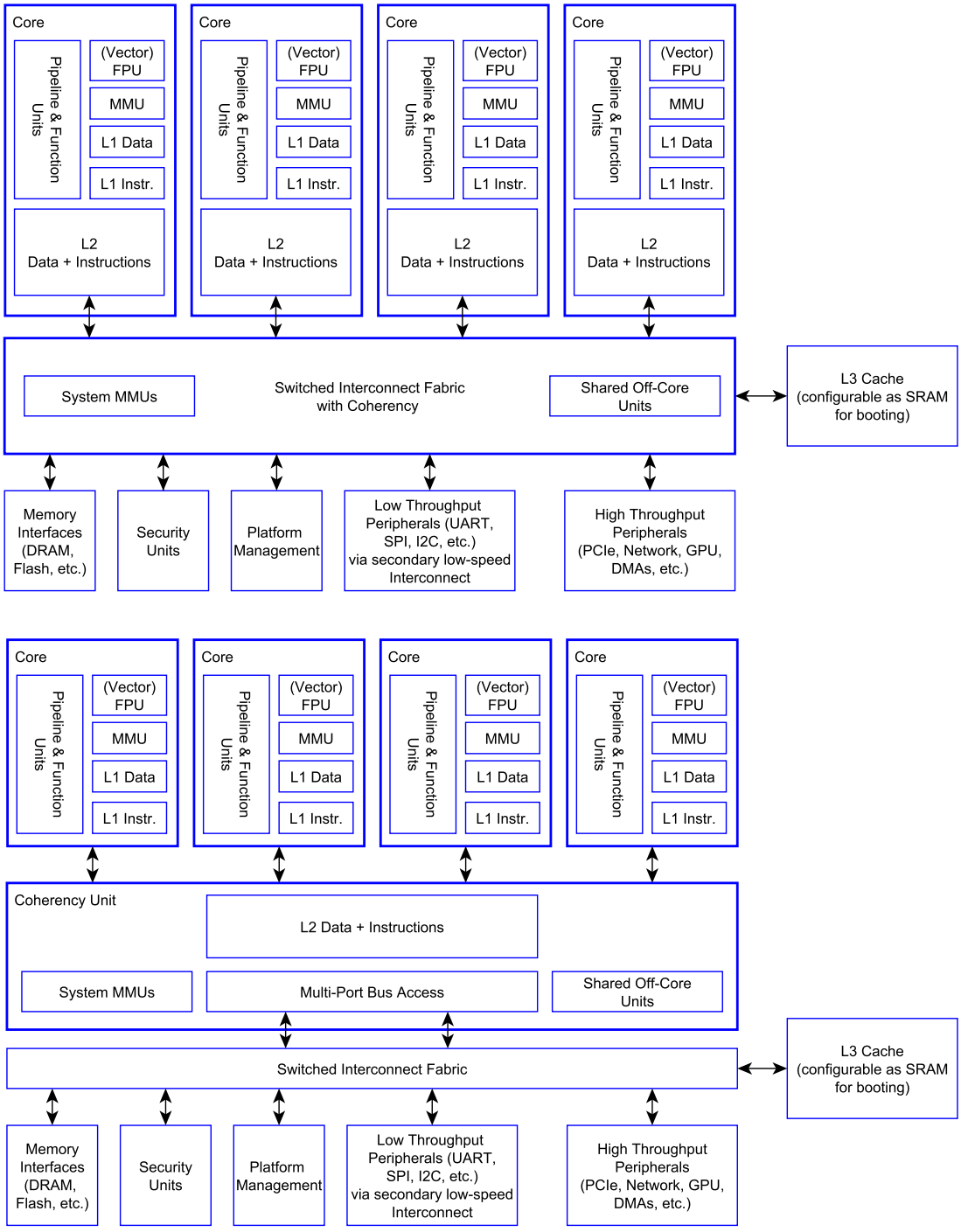

Figure 2.11: *Common multi-core SoC architecture example with core local cache (top) and shared cache (bottom)*

[Hol19, Section B2.3.1]), cores allocate lines when they access data in the DRAM. Old data is moved out of the cache, for example, after a certain timeout or when it has not been addressed for a certain amount of cycles and the cores demands access to new data which is not yet caches (this is a simplified description). Now consider for example: core 1 executes critical software, and actively uses about 256kByte of 2MB total shared cache. Core 2 executes non-critical software, which usually works on about 512kByte cache by its accesses. Due to a design error in the non-critical software, a memory-copy loop does not terminate properly and operates out-of-bounds. The second core now allocates more and more cache, since its continuously addressing new, uncached memory regions. Once a specific threshold is reached (this is highly architectural dependent), the cache flushes parts of the content of core 1, which usually remains cached, due to the requests by core 2. This breaks the determinism for core 1 and the critical function, when it tries to access memory addresses which are now flushed and no longer cached, resulting in non-deterministic execution times of core 1 due to the cache pollution by the second core. The scenario gets even worse with more than two cores, and the general question arises, if it is possible to partition or allocate a specific amount of cache per core. Note that the exact behavior of the cache is highly specific to the SoC architecture (ARM, PowerPC, x86, etc.), and even different between device generations and families. Also, the cache is highly configurable in his eviction, flushing, update, etc. behavior, which must be considered in the software architecture and the software requirements stage. A proper cache configuration for the given application scenario is vital for deterministic and high performance.

The shared DRAM does not suffer from the same type of interference as the shared caches, but adds to the problem due to its limited bandwidth. Let's first consider devices with one single DRAM memory port at first, since they make up most low-power embedded devices used today. With a single controller, the total DRAM bandwidth is shared among all cores and other bus-master peripherals (such as network controllers, GPUs, DMAs, etc.). As shown in Figure 2.12, the effective memory access speed per bus master dramatically decreases as more transaction requests reach the memory controller concurrently. Depending on the memory controller architecture, each bus master is usually allocated a fair share of the available bandwidth, which is achieved based on memory transaction queues inside the DRAM peripheral. From there on, the memory controller physically accesses the external DRAM ICs to load/store data words. The finite bandwidth and time delays to address, store and load data bytes is one of the major timing factors which influence the maximum memory bandwidth achievable. In current devices, there is no priority scheme to raise or lower the priority of an individual core with respect to its emitted memory transactions (in real systems, it is the last level cache emitting the memory transactions due to the cache miss, not the core itself, this is however fully transparent to the core). The fair bandwidth sharing results in a non-deterministic bandwidth reduction for critical software, when another core executes memory-intensive

software concurrently. Some devices implement a priority scheme based on the master port accessing the memory controller, in order to prioritize peripherals such as GPUs, which suffer even more than classical cpu cores when they stall due to limited memory throughput.

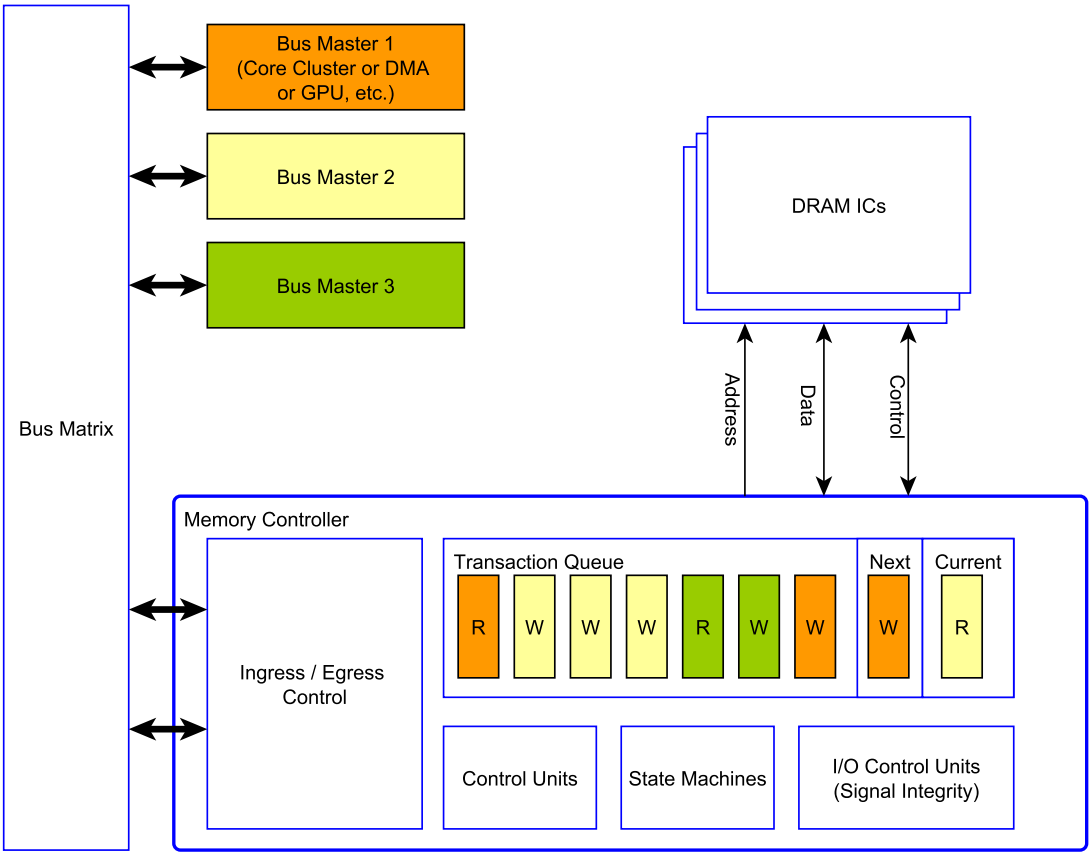

Figure 2.12: *Queued transactions from multiple bus masters leading to congestion and added latency in a shared memory controller*

Since we can realistically not hope for hardware support for COTS devices (full core priority settings, right to the memory controller via the interconnect matrix), and the fact that already existing on-chip units may be risky or very costly to certify (due to limited design data availability by the vendor or the cost associated when certifying on-chip units, as well as limited self-test capabilities, systematic capability of the development process of the semiconductor manufacturer, etc.), we have to develop a software solution, which provides effective and efficient deconflicting of shared resources with minimal on-chip function unit dependencies. For all units, the problem can be divided in spatial isolation, which provides separation of locality, and temporal isolation, providing deterministic bounds and low jitter for execution times. The resulting solution executes during run-time of the system and ensures that the clear separation is maintained, detects anomalies and constrains cores when they violate a certain boundary.

Our high-level software architecture (see section 2.4.1) and board-level mitigation strategies work closely with the following isolation strategies and are one building block in a sound safety case for low to very high criticality systems. Without proper system level mitigation, even the strongest separation, even with hardware support, would be useless since the non-present electrical isolation of the individual cores is still a major issue for COTS multi-core certification. In an imaginary certified system, spatial isolation software establishes a well-defined memory (and memory-mapped function unit) isolation by constraining individual cores to a set of bounded memory regions. Full temporal isolation of cores with equal bus matrix transaction priority is not possible without dedicated hardware support. The key in this regard is not to provide a fully deterministic system by hardware design, but to introduce sound monitoring and mitigation strategies which only permit a limited amount of transactions of a core in a specified time slot. This reintroduces maximum upper bounds for worst-case execution times and ensures run-time determinism. Note that certain hardware features offer clear benefits for certification, like a local backside-cache (as found on the e5500 PowerPC core [FS13]), but are in general not necessary for the concepts we present in the following. The specific implementations of these concepts will be device specific for real-world projects and must take the precise inner working of the device into account to be effective. Portability and reuse for other device families or other SoC architectures may not always be possible.

2.4.2.1 Spatial Isolation

In the following, we will discuss the spatial isolation with respect to the goals, hardware support and the proposed concept in relation to our board-level system architecture and consequences for it.

Goals The main goal we must achieve is to isolate individual cores at shared on-chip resources in terms of the locality of their write accesses. This leads to a spatial containment, where one core cannot modify another core's resources, if not explicitly allowed and specified. The relevant peripherals are the shared cache and the shared DRAM. Interface peripherals are deconflicted by the software architecture, however further hardware support is required to enforce the separation.

The concepts described in the following are based on [MPC+15], especially the cache lock-down as described in [MDB+13], which we rather included in the spatial isolation section than in the temporal isolation section. However, improper cache separation also leads to temporal cross-couplings between cores. We do not directly contribute to those principles, but rather use their work and put it into perspective in the overall certification effort, as one important building block for deterministic and conflict-free execution of different concurrent software tasks on a multi-core SoC.

Modern devices use memory-mapped peripherals. All resources on the SoC that can be addressed, possess a specific physical memory address (we will not use virtual address

space here, but rather physical address space). Software can write to a specific register of a device or a memory location by issuing a write operation to a specific address (e.g. "write register 3 to address 0xABCD", encoded in the respective instruction set of the machine). The request is than translated, cached and finally issued via the bus matrix to the respective peripheral as a read or write request. To deny the access to a specific location, a hardware unit is usually used to check the requested read/write address and either permits further traversal through the cache and the interconnect, or denies the request with some form of feedback to the software.

Hardware Support The mentioned hardware support is usually provided in the form of memory management units, local to each core. They process the emitted address by the core and decide based on specific MMU entries, whether the core is permitted to read/write from the address. Also, associated with a specific access region (set of memory) are additional flags, which specify if the region can be cached, which cache write-back strategy is used for that region, whether other cores should be notified that a write operation modified the content of that region (coherency), etc.

Most devices also provide platform management units which provide additional and finer graded control over special purpose registers and peripheral configurations, usually coupled to certain privilege or hypervisor extensions the core architecture offers. They are usually employed for security reasons in common operating systems, to deny a specific task access to a resource if it is not executing in a privileged (root or administrator) mode. On complex architectures, this concept extends beyond the device itself and includes for example devices connected via PCIe, which can be dedicated to certain virtualization and hypervisor levels (like the x86 IOMMU system).

While the MMUs manage memory very well, they usually have little influence on the locality of data in the cache and the allocation/deallocation of entries in the cache. Being transparent, the cache acts as a homogeneous memory pool for properly attributed memory regions (cache-able regions). To virtually partition a shared cache, hardware support is vital and software can not alone accomplish this task, even if virtual memory address space is used to hide actual physical addresses to the application software. Luckily, some architectures offer special cache instructions (or special purpose registers) to lock parts of cache. They operate either on the cache lines. When a certain memory location is cached, it can be marked as locked, which freezes the association of a cache entry to a memory location, even if the memory region or the entire cache is flushed, forcing the memory location to remain cached at any case. The respective line is then no longer available for other memory locations and can only be evicted when it has been unlocked again. Cache lockdown has been part of almost all architectures in recent years, but has been dropped in current ARM architectures for no obvious reason (see chapter 4 for an evaluation of current devices for our prototype). Since the cache is transparent and the upstream memory management and address translation has no direct access to the

address tags of the cache, the application software is not able to determine if a cache line currently points to a specific memory address or not. When no locking support is available, there is however the possibility that the cache might be execution level and hypervisor-aware, which can be used to provide a very coarse but working pseudo-locking when for example hypervisor lines remain cached or prefetch hints might be useful to guide the cache behavior, see [Hol19, Sections B2.3.5, D4] or [PS19], which is a subject for possible future work. The real-time operating system vendor DDCi Inc. has patented a different method which uses the memory management unit to allocate cache to specific memory regions by utilizing the virtual to physical memory mapping, see Patents [LRM11] and [HM15].

Possible Solutions and Consequences The different parts of SoC require individual measures for spatial isolation. First, the peripherals are isolated by shifting the access to a single core which handles the I/O access. Architectural features, such as IOMMUs, System-MMUs or Platform-MMUs should be used to further enforce the access-rights separation. Second, the read and, more importantly from the safety standpoint, the write access to the shared data and program memory must be isolated. This is only possible on a per-core basis by employing the local MMUs at each core. The MMUs of larger devices operate based on lookup-lists in defined memory locations. Since the maximum number of entries in these lists is usually finite and quite constrained, a lookup miss triggers a memory protection exception which is handled by the RTOS. All major RTOS on the market today support the use of the integrated memory management and protection features, and provide adequate configuration means. The granularity is adequate, and can be set per task (or process), which is a desired feature not only for a safety application. However, by stressing the MMUs in terms of safety, we are forced by regulation to provide certain process and runtime measures to ensure the correct operation of the MMUs, as well as to detect a failed unit during runtime. Since the COTS SoCs do not provide hardware fault-injection for memory management units (not required by applications other than safety), we can only provide weak on-line and development-time error detection measures. For example, the MMUs can be tested during runtime by injecting accesses to denied areas and checking for an appropriate memory access exception (this must be repeated for entries in the current MMU table and entries not present to check the reaction not only for the latter). Likewise, positive access must be tested by issuing requests for permitted regions, which should not trigger an exception (this must be tested for entries in the lookup table which result in no memory management exception, but results in an exception for an entry not present in the current lookup table where the correctness of the address causing the violation in the MMU registers must be checked). The maximum diagnostic coverage or credit for the certification effort must be clarified upfront with the individual authorities. Test frequency and the time slot of the tests with respect to other cores are application specific. Some applications may require a

high test frequency (multiple times per time step, to ensure fast detection within a short time period) or that the test is triggered by the master core. It is also possible that the master core modifies the MMU table of slave cores and thereby triggers a violation which must be caught by the respective slave and reported back to the master for the test to be successful. During design time, proper device tests should include testing the MMUs for desired operation to detect hidden device errata, as well as high-rigor [16] automated testing of the MMUs for as many use-cases as possible.

Third, the cache levels on the device, shared by multiple cores. Depending on the cache architecture, the core complex holds at least one shared cache level. Special architectures might offer only core-local caches, for example as a local backside second level cache, like the e5500 PowerPC architecture. If the caches are local to each core, no further deconflicting measures are necessary. If, however, shared cache is involved, we must provide a clear separation of the cache allocated to each core to prevent important (safety critical) data from being evicted in favor of less important (uncritical) data. This might happen when cores compete on the cache, and a non-critical software operates on large amounts of memory, while a critical program has a small set of working memory. To provide this separation, the cache must support a feature called cache locking or lockdown. Cache lockdown works by freezing a certain subset of the cache to a memory region, or individual cache lines (or ways) to a certain memory address. The granularity for the locking instructions is architecture dependent, but equals the size of a cache line in most architectures, since the locking is done by freezing the association via a cache way lockdown. If the hardware supports cache lockdown, Mancuso and Yun et. Al. provided a practically useful method called colored cache lockdown (see [MDB+13], [YYP+13] and [YMWP14]). The colored lockdown principle works by first analyzing the cache utilization of a task or process (either via RTOS-level support or special debug tools with memory tracing), followed by classifying the accessed memory regions by their access frequency ("coloring"). Once the often-accessed memory regions are known, they are locked in cache once the task is executed by the RTOS, to prevent other cores from forcing these regions out of the cache. This greatly benefits the determinism and WCET estimation, as stated in [MPC+15]. Runtime jitter is minimized and further spatial isolation between competing cores is achieved. In conjunction with a lockdown strategy, all modern architectures offer configuration options for exclusive or inclusive cache levels. When the data (and program) cache is set to behave exclusively, data is only cached by one level at a time. An inclusive cache would cache data at different cache levels simultaneously, leading to redundant caching over the different levels. The exclusiveness usually configures the L1 and L2 cache behavior, or more general from the last core local cache level to the first shared cache among multiple cores. In this case, data is only present in the upper cache

[16] In many standards, different "rigor levels" are defined which imply a certain set of validation and verification activities from very basic to complex and costly procedures; this is mostly dependent on the design assurance / safety integrity level

levels, which is especially useful for small functions or algorithms where the local data fits inside the core-local cache. In this case, the data is fully kept locally to the core and does not require space in the shared cache and is not subject to a potential removal by memory requests of an adjacent core. Cache exclusiveness should be considered as a potential optimization strategy to further reduce jitter for small routines and algorithms which fit inside the first cache level(s), but does not entail a fully core-local caching of program data. Note that, while the colored cache lockdown is a well-engineered strategy to deconflict the shared cache, it is fully dependent on proper hardware cache lockdown support. Since the cache operates fully transparent and cannot be addressed directly by software, there is no other mean to divide the cache or ensure that critical data remains cached over the runtime of an application. The lack thereof has serious consequences for the usability of a given architecture in a safety-critical context. Many architectures still provide special instructions to lockdown cache, like the x86 and the PowerPC architecture, but today's most commonly used embedded SoC architectures, the ARMv7 and ARMv8 architectures, silently dropped support for cache locking after the ARM Cortex-A9 multi-core. If the underwriting authority demands a conservative WCET analysis, one is forced to determine the WCET with an always-cold cache, in order to demonstrate that the deadlines of critical applications can be met under all foreseeable conditions (which is, that the data is permanently removed from the cache by memory requests of other cores), leading to the only reasonable conservative assumption of an always-cold cache. Since the memory bandwidth is reduced by several times (see for example appendix B), it may be difficult to meet these deadlines for certain critical applications with many memory accesses. To overcome this issue, we investigated the possibility to use modern hypervisor and security extensions (for example, ARM TrustZone) to effectively divide the cache into multiple sections or ensure that certain mappings persist due to different security and access right level flags present in modern cache designs. However, after discussions with chip designers and architecture specialists, we found that neither hypervisor nor security extensions have an impact on the behavior of the cache [17]. It is not possible to lock, or rather reserve, certain lines or ways to pseudo-lock them. ARM devices, with core-clusters other than the ARM Cortex-A9 MPCore should therefore be avoided when possible.

Last are optimization strategies at the DRAM level which lead to higher memory address speeds and make use of the physical construction and connection of the external DRAM ICs to the SoC DRAM memory controller. If applied correctly, these measures boost overall memory bandwidth when multiple cores access different sections of the DRAM. DRAM ICs are organized in ranks, banks, rows and columns, see [JWN10]. While the chip array connected to the SoC memory controller can only serve one request at a time, it is possible to minimize the latency between subsequent accesses and address banks in parallel. With the principles defined in [MPC+15], critical and non-critical

[17] Based on information obtainable under NDA from NXP Semiconductor

cores can be assigned different banks to spatially isolate the memory access requests and make use of the parallel addressability, which results in higher overall throughput and improved latency, as described in [YMWP14]. An ideal solution to the spatial separation of different criticality sets in DRAM is to use a multi-core SoC with more than one DRAM memory controller. In practical devices, the number of physical compute cores will be larger than the number of memory controllers. The cores are therefore grouped per criticality or impact level (actual grouping depends on application context and authority requirements) and each set of cores is then allocated to one memory controller. Note that multiple memory controllers not only isolate the cores in terms of memory access contention, but also provide true physical isolation of critical and uncritical data in the DRAM. Note that multiple memory controllers are often found in very high performance and high-power consumer-grade or server processors, but can also be found in embedded-devices like the NXP LS2088/84A or the NXP T4080/4160/4240 as well as the older NXP P4080/4040 and P5040.

2.4.2.2 Temporal Isolation

In the following, we will discuss the temporal isolation with respect to the goals, hardware support and the proposed concept in relation to our board-level system architecture and consequences for it.

Goals The required temporal isolation scheme should provide means to monitor and control the individual cores on a multicore SoC regarding their timing behavior, which produces contention and interference when software runs on more than one core. The main goal is therefore to develop a strategy to constrain each core to a specified timing behavior. In section 2.4.2.1, we discussed several spatial isolation methods to deconflict the shared peripherals. While these methods provide effective mitigation for fault or contention issues arising from the physical locality of data and program code, they do not address the time-varying behavior of software executing in parallel on multiple cores in the SoC. The latter must be constrained by a temporal isolation architecture which ensures that concurrent tasks, with their non-constant access to shared peripherals, do not conflict with each other, leading to nondeterministic deadline violations of critical tasks. Furthermore, the resulting system behavior when the temporal isolation mechanism is in place should be deterministic and predictable, up to a degree where certain upper bounds for the execution time for critical tasks can be guaranteed and determined. Determining upper bounds for WCET in multicore systems is a non-trivial task, see for example [NPB+14], [NPH+14] and [DNA11]. The main issue when trying to determine the WCET for a given task in a multicore system, with multiple cores executing software concurrently, arises from an unknown interference produced by the remaining cores producing interference (leading to stalls and delays in the software under analysis) which is not bound. The main driver of interference, when the shared peripherals are already deconflicted, and the cache has been

managed, is the shared DRAM memory and its internal controller. There, transactions are queued and issued sequentially to the external DRAM ICs, leading to severe delays when multiple cores fight over the available bandwidth. The bandwidth on the external DRAM interface itself is also rather limited, compared to internal caches (see appendix B for measurements with a e5500 quad-core and a more recent Cortex-A72 core). Note that is not only the nominal contention scenario causing unpredictable execution times and nondeterministic behavior. Especially if fault scenarios, where one (or even multiple) cores exhibit single-event effects during execution, this may eventually cause the faulty core to emit far more memory transactions, which in turn delays execution of critical software on other cores and undermines deterministic side-by-side execution in mixed-criticality systems.

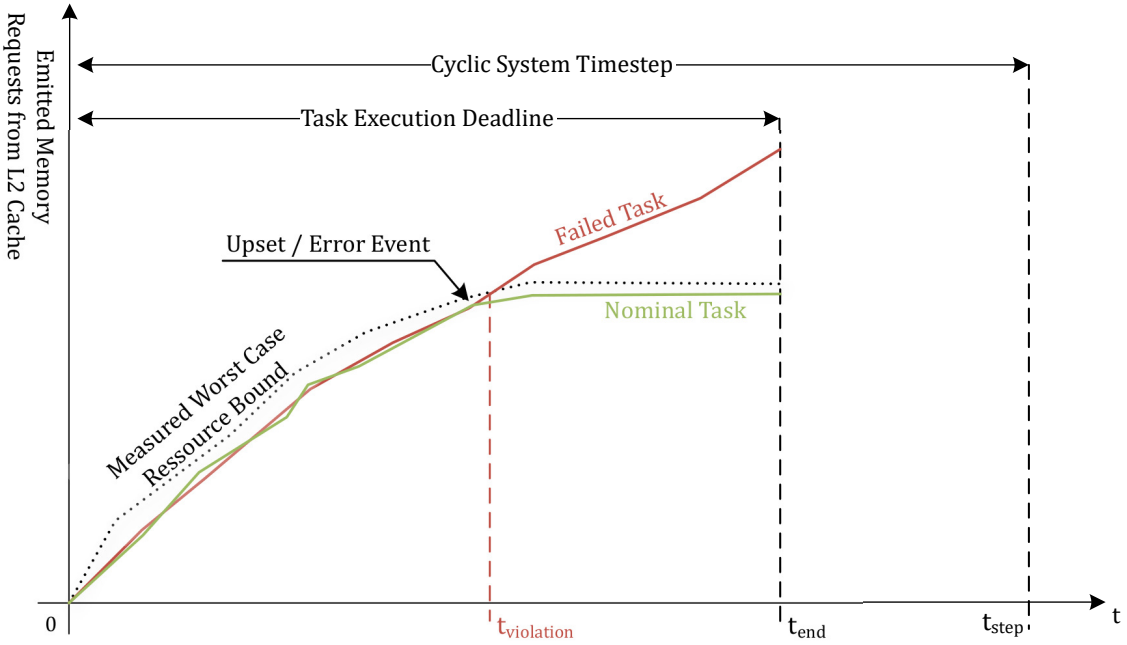

Figure 2.13: *Memory transactions graphen bild für tasks und so weiter*

Figure 2.13 depicts the aforementioned behavior as an example. Shown in green is the cumulative nominal number of emitted memory transaction by the last level cache mechanism (L2 in this example). The amount grows steadily and is always less or equal to the predetermined maximum worst-case transaction bound. After a certain time t_{end}, the task is scheduled out and terminated. As shown in the figure, a failed task may behave nominally at fist, but escapes the worst-case bound at $t_{violation}$ once, for example, an upset caused a malfunction in the execution or the task's data leading to an undefined state. Note that the violation does not immediately occur after the upset, but only after the task has passed the worst-case bound. Only then, it causes untested resource conflicts with other processes running on the same device. Without mitigation mechanisms, the task terminates when the it is scheduled out, after the task deadline has ended. Note that this assumes a scheduler where tasks can not yield unused time budgets.

This gears us towards our goals for a temporal isolation mechanism. To guarantee a certain degree of determinism, and certainly to comply with current and future regulation and guidelines, we must prioritize one core over another, or limit the usage of the shared memory bandwidth. At design time, the core complex in the SoC can then be set-up (priorities) or analyzed and confined (bandwidth allocation). This provides a guaranteed and testable timing behavior for the nominal run-state, which is also resilient against malicious behavior of cores, because critical cores are either not influenced due to their higher priority, or penalized malicious cores which are stopped or slowed down [18] and therefore do not affect deterministic operation on the remaining cores of the SoC.

Demonstrating that units of different criticality levels do not affect each other [19] is a vital part, already to be discussed on rather high system requirement levels in current certification projects. Often times, applicants fail to realize the importance of proper separation and isolation of function units, leading to very late architectural design changes in many projects, when issues are discovered at the hard- or software level. In the multicore device, the problems are amplified and are one of the major concerns with highly complex SoCs for certifying and authorized bodies. The following strategies integrate tightly with our board-level system architecture, and are not alone usable to justify a certain safety integrity or design assurance level. They provide however effective means to ensure determinism (but not fault detection or mitigation) and prevent overly pessimistic WCET assumptions resulting from the interference on the device.

Hardware Support (Low-Level) Management of memory transactions, emitted from the last-level cache via the shared bus matrix towards the memory controller, requires hardware support.

An ideal hardware could feature full transaction prioritization. Whenever a memory transaction is generated by a core, triggered by a miss in the last local cache level, it is assigned a dynamic priority. A core-local configuration or special purpose register could be used to hold the currently assigned priority level for the core. A local-only configuration could lead to potential safety issues, which can be avoided by adding special protection and check logic for these register(s), like ECC, hardware checksums, or transparent time-modular-redundancy (3 effective registers + voting, reads and writes are delayed by one or more clock cycles to counteract common mode effects by clock upsets). Each transaction is than flagged with the currently configurable priority, which can be assigned dynamically by an RTOS for each running task (per-function priority) or pseudo-static at boot time for a core (per-core priority). When the transaction enters the shared cache levels, the flags are preserved and remain present in bus matrix transactions emitted by

[18] Most modern SoCs employ frequency scaling or other mechanisms to control the per-core clock frequency for power savings and power/heat limitation. These features are driven by a combination of hardware (hard limits) and software (power saving).

[19] The simple workaround of disabling unused units can not be considered safe since their configuration might change due to SEU, therefore monitoring is needed

the shared cache levels for memory reads or writes. The priority is considered on the central bus matrix, where low-priority transactions are enqueued (therefore delayed) in favor of high-priority transactions of (safety-)critical routines, which resembles a classic quality of service feature. Once the transactions reach the shared memory controller, the priority tags are stripped and each transaction is enqueued. The memory controller either features multiple queues for different service quality (priority groups) levels, one queue for each core with re-assignable priority (this requires that also the core ID is included as a tag) driven by the transaction flag, or a highly efficient single queue with priority based insertion. Semi-fair transaction queues must still be provided, since we do not intent to stall a non-critical core for extended periods. A useful implementation could offer a per queue (or per priority level) configurable wait timeout, which bounds the maximum waiting time for lower priority messages when many high priority transactions are issued. The entire memory access flow can then be fully configured to specific application requirements. Sadly, no devices on the market today do offer any kind of transaction priority system. Implementations based on a master port priority quality-of-service on the bus matrix, like the CCI by ARM, offer the possibility to increase responsiveness when multiple high-bandwidth masters issue large numbers of transactions simultaneously. This is useful for devices with powerful integrated GPUs or additional accelerators sharing the same bus matrix as the core-cluster towards the memory controller. However, these priorities are not granular enough and are equal for all cores inside a core cluster connected to a single master port. Any future COTS SoC which implements a well-documented priority system would be of great value for system designers and the system safety case and solve the issue of temporal separation for the shared memory all together. But until such devices become reality, one is relying solely on present monitoring facilitates in current SoC devices.

One of these monitoring features are the integrated performance counters. All major modern core architectures (x86, PowerPC, ARM, etc.) offer per-core counters for events generated at different levels of the core and the shared and non-shared cache levels. They can be configured at runtime to count events generated by these units, like processed instruction count, or a cache miss event in a specific cache level. The latter is of special importance, because a miss in the last cache level directly translates to a memory transaction being generated by the cache, which we eventually want to track in a specific time window. Events which can be counted are architecture or device specific, to include special sub-units inside the core (like special vector processing units, or other accelerators), but general events, like the mentioned cache miss, are generally available. It may not be given that an event is generated on the other way around, when a value is written from cache to memory (eviction, or policy based - for example write-through), which must be considered in a monitoring and partitioning algorithm. To determine the number of events in each time slot, the timer is started (reset) and counts all events without disturbing the function units. One can then determine the number of events by a simple register read

and store the result.

In summary, one of two features must be offered by the SoC of choice. The device must either provide the mentioned priority feature for temporal isolation, or performance counters for counting generated events. The latter entails a monitoring solution, which enforces bandwidth allocation on the shared memory controller when one core emits more transactions than allowed in a narrow time window. If none of the two features is present, the SoC is not suitable for a safety application and should be avoided. Note that some device families, like the QoriQ or Layerscape Series by NXP Semiconductor, offers additional counters for SoC-wide events on peripherals, the bus matrix, etc., which might be helpful in monitoring the temporal behavior of these units for mixed-criticality applications. Also, special debug units may be present which might offer additional metrics and control possibilities. Information on these units in complex SoCs is under NDAs, since they in theory allow some degree of reverse engineering.

On devices with more than one shared memory controller, each core or group of cores (cores with critical vs. cores with non-critical software) can be statically assigned to the individual memory controllers. The resulting isolation is then given between critical and non-critical functions, but not between the cores itself, unless the device has as many controllers as cores, which is an extremely unlikely combination. Multiple memory controllers offer better and more deterministic performance and fault isolation, because they provide individual hardware units for different criticality (or impact) levels, but do solve the temporal isolation puzzle.

Possible Solutions and Consequences Since no device available today features proper hardware priority support, we must opt for the monitoring-based approach. This implies that memory transactions are monitored for each core, with per-core boundaries. A viable approach for this scenario has been defined by in [MPC+15] and [YYP+13]. Their approach employs the previously mentioned performance counters for each core, to monitor the last level cache misses. The monitoring and boundary enforcement is tightly integrated into the operating system kernel (Linux in this case), coupled to the currently running task. At design time, one measures the effective transaction throughput, which enables the designer to define an upper bound for the respective application on a single core, which can be guaranteed at runtime. The guarantee is given by not over-allocating the memory controller bandwidth with the set of tasks running in parallel on all cores in the SoC. As defined in [MPC+15], this enables the designer to re-define the WCET as a function of active cores with interference.

Depending on platform-level design decisions on the fault behavior when cores perform abnormally, a task or core can be temporarily or permanently disabled once the upper boundary for its nominal memory transactions has been hit. See 2.4.1.2, and Figure 2.14, which depicts the case where a task has hit his upper transaction boundary, is temporarily disabled and re-enabled once it is back inside the valid transaction budget and has hit a

lower hysteresis limit. Note that this behavior should only be implemented for non-critical application (IL1), since the task will no longer be able to meet stringent, hard real-time boundaries once it is scheduled out. For critical tasks, it is more appropriate to disable the task, and communicate its failure accordingly, since we provide sufficient on-board fault tolerance by our board-level system architecture to compensate for failed system functions. The failed critical function (IL2 and IL3) can then be properly restarted.

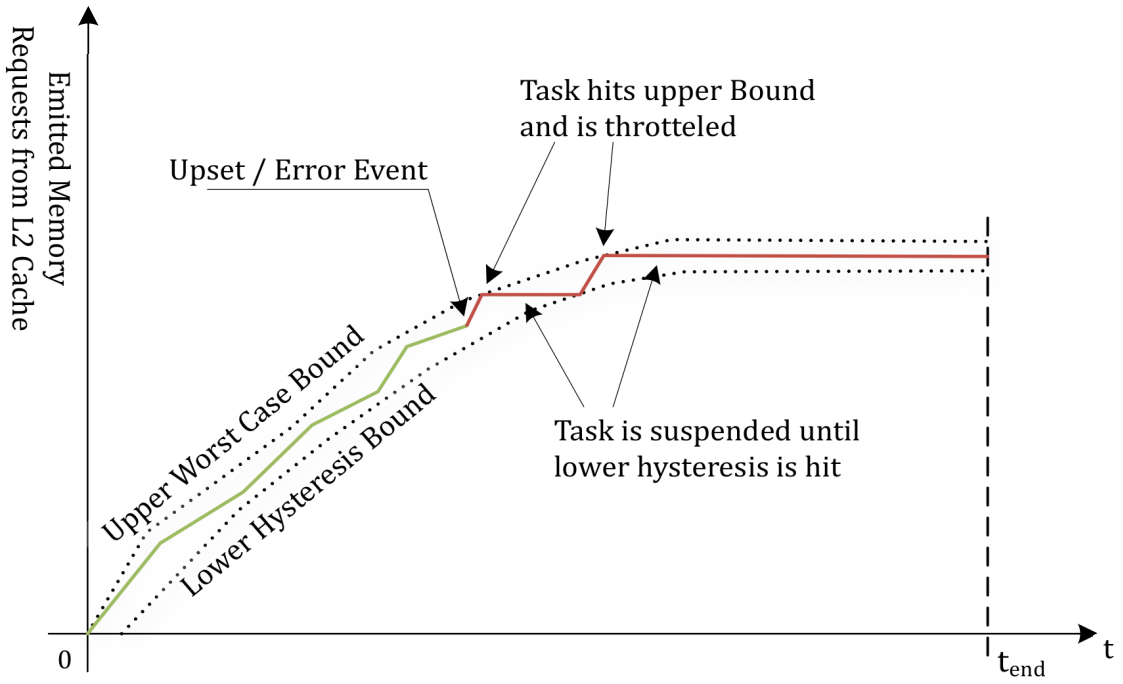

Figure 2.14: *Failed task handling when memory transaction boundaries are violated to ensure deterministic platform behaviour*

The SCE approach by Mancuso et. Al. allows us to re-establish a testable WCET boundary in the contention scenario, which is practically feasible in real-world development projects and fits nicely into our system- and board-level architectural concepts. SCE provides important software and determinism building blocks on the multicore devices, to comply to current regulatory requirements for aerospace certification, and higher industrial certification levels. Despite the needed performance counters, the concept is not depending on special hardware features, and can be extended when further monitoring facilities are present on a SoC. Note that, since SCE uses these performance counters, they are part of a monitoring function for a safety critical function. This implies that the performance counters must be periodically tested at runtime to detect latent faults in these units.

Both spatial and temporal isolation strategies are closely tied together, and, when correctly implemented for the respective device, are one corner stone for a successful mixed-criticality safety case. Our board-level architecture already provides strong mitigation for all possible failure modes, however, in the light of current regulation, these

on-device measures are still very important for successful real-world certification efforts. Only when sufficient isolation, in the context of these guidelines and standards can be demonstrated, one can provide the evidence for determinism (mainly the WCET) and freedom of interference between the individual criticality and impact levels. Albeit guidance is given in all application domains, the suitability of certain concepts is always subject to the discretion of certifying agencies and technical bodies. A singe approach, discussed in isolation, will never met the goals declared by those entities. Only a holistic approach, which directly connects the system-, board- and device-level can provide enough resilience against the critical technical reviews based on today's regulation.

2.4.3 System Supervision and on-board module interaction

After the short dive into the multicore SoCs, we now shift our perspective back to the board-level with our single and distributed monitoring architectures.

To supervise the board-level components (nominal channel, single or distributed monitoring channel(s)), we employ a layered diagnosis scheme. Within each layer, we address either specific features, starting from the device level (for example, checking RTOS instances on slave cores or SoC peripheral configurations at a high frequency) right to the highest board-level (for example, checking that a device is responding to a query or testing redundant system power paths). Each processing device executes local runtime diagnostic checks and build-in-self-tests at startup, which are overseen on the board-level by two different measures:

- Multiple sets of output data for each system function, which are compared for consistency or equality before leaving the LRU, or voted on the top system level at the smart actuation elements
- Distributed watchdog for internal check functions, ensuring that device-internal diagnostic checks are being executed

We will focus on the second point in the following, since we already discussed the first point in section 2.3.2. Figure 2.15 shows an example for the layered supervision scheme. On the bottom, the NOM and each DMON executes internal checks locally, which include for example the measures described in 2.4.1.2, 2.4.1.3, 2.4.2.1 and 2.4.2.2. These internal checks provide a reasonable detection level in the light of current standards, but do not fully protect against all possible internal failure modes. When the internal diagnostics fail however, we lose this capability. Even in a multi-core device, we need some external entity to ensure that at least the internal master core can respond to external queries. Given our board-level architecture, we can readily execute the periodic external check via the SMON or the DMONs, to gain the confidence that the device is in a specific state. What holds for the multi-core NOM, must also be true for a possible multi-core SMON or DMONs, which have to complete the periodic external check too, as shown in Figure 2.15.

For the check itself, we reuse the signature-based challenge-response watchdog discussed in section 2.4.1.3.

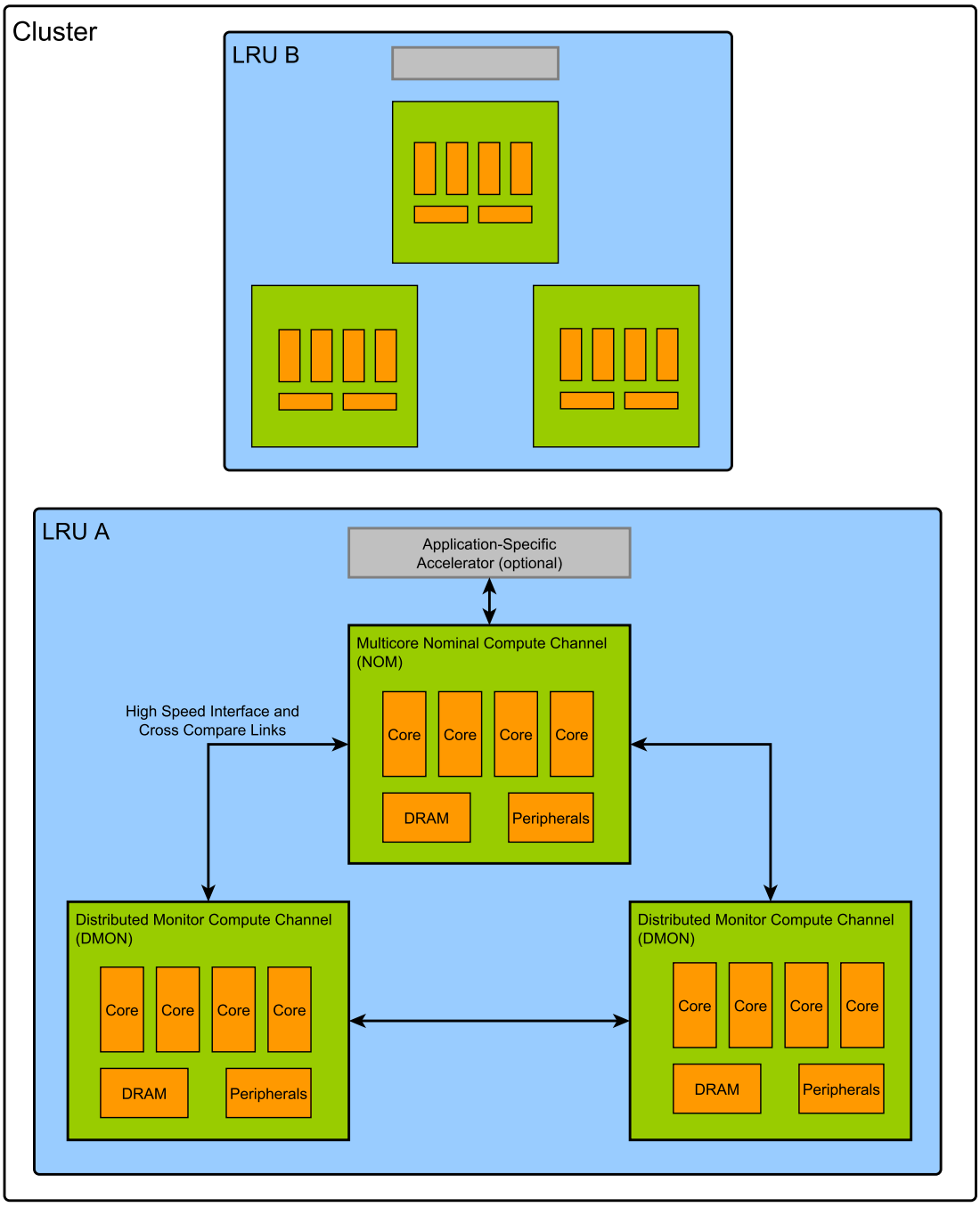

Figure 2.15: *Levels of monitoring, from inside the SoC and its memories (orange), inter-LRU (between NOM and MONs, green) and cluster wide (between LRUs, blue)*

In conjunction with the granularity of the self-tests and supervision, which decreases from the bottom to the top level, the frequency of the diagnostics likewise decreases. The distributed watchdog authentication between each device should only be executed once per cycle (location of the watchdog time-slots within a system time step depends on

the timing requirements of the critical functions [20]), whereas parts of the device internal monitoring execute at a much higher frequency (for example, with the RTOS kernel time tick). This is reasonable, in order to reduce the overhead involved with the cross-device communication and the occupation of the device-master-cores. After all, since we lack any hardware support for executing on-device diagnostic measures in discrete logic, we are not able to provide strong detection measures for latent faults inside the complex SoCs from a conservative perspective. However, when combining our board-level architecture, with its inherent resilience against any kind of wrong or untimely output result (including common-cause failures, if NOM and MONs are dissimilar), the measures proposed up to this point suffice to guarantee that stringent timing requirements for detecting failed (unresponsive) devices are met. Some errors (due to SEE, like memory upsets or rogue cores) on the device level can be detected before they lead to a critical board state. The simple, layered watchdog approach with added on-device diagnostics is lightweight and easy to implement and test, not only in terms of the actual software implementation, but also from a conceptual standpoint on the system level. Complex measures, relying on some (non-common) unique device properties, only present in a special variant of family would defy the purpose of integrating any COTS multicore processor with an acceptable level of process maturity and quality assurance. Note that special processors may nonetheless be used, especially when they offer internal error detection. The S32 Automotive Application Processor Series from NXP Semiconductor for example includes some features from past automotive powertrain processors, which included well-defined build-in-self-test measures and fault collection units with externally routed error outputs. The watchdog scheme can then be enhanced by monitoring these error outputs and forcing a device power cycle when they indicate an internal fault.

The interaction between the different board-level components for single and distributed monitoring architecture is detailed and described in appendix A.2 by message-sequence diagrams. The appendix A.3 includes examples for fault mitigation, in case one or multiple components fail. Appendix A.1 shows the on-device monitoring flows.

In this chapter, we discussed our board-level architectural concepts, along with a brief dive into the inner workings of today's multicore processors. First, we defined a set of LRU requirements based on two possible use-case scenarios. We concluded that a centralized, high-performance platform is best suited to exploit the characteristics of multicore devices. The resulting set of LRU requirements reflects not only an aerospace, but also an industrial application, up to the highest certification levels from a system perspective. We also defined a novel system function classification scheme based on impact levels, which can be used alongside classic safety integrity levels or design assurance levels. The concept takes specific properties for different classes of system functions into account, and

[20]Latency sensitive applications like control loops might not allow introducing additional dead time due to multiple authentication steps; however, this is highly application dependent and less sensitive functions may not be affected.

divides them into three impact levels. Based on the LRU requirements, we derived two board-level architectures. The centralized (or single) monitoring architecture features a nominal channel and one single monitoring channel, as well as output stages. The decentralized monitoring architecture includes at least three devices, with one nominal and two monitoring channels. All channels can be based on complex multicore SoCs without compromising system-level safety. For each architecture, we discussed the coverage of our LRU requirement set. After finishing the board-level architectural aspects, we shifted perspective towards the software (or behavioral) perspective and discussed the inter- and intra-device communication between different cores, system functions, monitoring functions and operating system parts. Next, we discussed mixed-criticality and determinism aspects on the multi-core device, which are essential for future applications. Based on prior work on the software side regarding WCET, we defined the integration with our system and board-level concept, as well as alternatives and desirable features on the SoC level to provide spatial and temporal isolation for system functions of different impact and criticality levels. Lastly, we revisited the different diagnostic measure levels, which are present on the device, channel, and board-level, as well as their interaction.

3

Safety Analysis

In the previous chapter, we discussed our board-level concept with some complex COTS SoC specific aspects from a mostly technical standpoint (with certification in mind, however). We will now focus on the actual safety analysis of the presented architecture and dive deeper into aerospace and industrial certification topics to prove, that the presented concepts hold in a real-world certification process and permit the applicant to justify the use of highly-complex COTS devices (Contribution C5). First, we will discuss system level and COTS certification in general, with aspects common to most application domains. Next, we will conduct a fault-tree analysis and a Markov analysis for the centralized and decentralized monitoring architecture and show, that even with overly pessimistic assumptions, safety goals are well met and even exceeded with our architectural designs. Lastly, we will discuss domain specific aspects and shine light on the application of current standards and guidelines from the industrial and aerospace domain, namely in the IEC 61508 (industrial) and CS/ARP/DO context (aerospace). Note that no actual certification has been conducted within this thesis. However, the early involvement and very positive feedback from notified bodies leave us optimistic that an actual certification is possible, at least in the industrial domain.

3.1 System Level and COTS Device Certification

Certification of critical technical systems has a long-lasting legacy in many application domains and has ensured operational safety for many decades in the past. When applying the current set of standards, one ensures that the risk of a catastrophic event, harming operators, third parties, the system itself or infrastructure is reduced to a certain level. More importantly, the risk is controlled and predictable - an essential argument for the acceptance of any technology within our society. The acceptable risk by the general population for a catastrophic event for any class of systems dictates the acceptable occurrence rate and has been determined by industrial committees or public authorities for each ap-

plication domain [1]. But the topic is even more complex, as even more parties have special interest whenever technical systems pose the risk of harm to people, such as employer's liability insurance associations, insurance (as well as reinsurance companies) and last but not least the legislative and executive governmental branches. On top, we have to deal with economic constrains and company politics, which try to force safety standards into a certain direction to support their portfolio and internal standards for obvious reasons. As a result, a good technical solution might not always be feasible and may fall short due to a lot of special interests involved, which were not taken into account during the design. We invested heavily into many discussions with industry partners, committee work [2] and public talks with leading certifying agencies to harden the concepts presented in this thesis to maximize its readiness level.

We found that newcomers often times mistake safety for availability. In short terms, a system which is safe does not need to be available in most applications (we will address fail-operational shortly). This statement implies that it is in general safe for a system to become inoperative whenever a failure is detected. The quality of the detection and the remaining risk for undetected, dangerous failures is subject to the certification level of a system, and is lowered at higher certification levels. In order to allow said behavior, a safe state must exist which the system can enter and maintain until the fault is removed or the system is repaired by a technician. However, there are (sub-) systems, where this safe state can not easily be established, maintained, or does not exist during normal operation, for example, in airborne aircraft, train braking systems, power plants or automotive steering systems. For these class of physical systems, the control systems (regardless if mechanical, hydraulic, electrical or electronic) need to operate safely and with a very high availability to guarantee an overall safe operation. This differentiation leads to terms like Fail-Safe and Fail-Operational, which indicate whether the control system (or subsystem) in question has a safe state (and may simply traverse into it in case of a failure) or if the system function must be maintained at any cost, to protect from hazardous or catastrophic events. Often times, only few subsystems of the full control system are subject to a Fail-Operational failure mode, especially in moving systems (usually for controlling certain degrees of freedom).

The governing standards we targeted our concepts at are the IEC61508 [65A10] (together with the ISO 13849 [ISO15] and IEC 62061 [4421] for industrial applications) and the various aerospace guidelines in the form of the ARP4754 [ADC10] and APR4761 [ADC96], as well as the DO-178C [SC-11] and DO-254 [SC-00] with the CS-23 [EAS17] as the referencing certification specification. We did explicitly choose to not include the ISO26262 in the main focus of this work, due to its similarities to the IEC61508 and some

[1] Comparing the highest levels: Aerospace DAL A $\leq 10^{-9}$ for catastrophic failures per flight hour; Industrial based on the EN ISO 13849: PLe/SIL3 $]10^{-8}, 10^{-7}]$ dangerous occurrences per usage hours, often 8760h per year.

[2] In the German committee working groups DKE AK914.0.3 and AK914.0.4 which mirror parts of TC65 on a national level for IEC 61508 associated work

domain specific aspects within the standard. The same holds for the EN 50126 (process aspects), EN 50128 (software aspects)and EN 50129 (system and hardware aspects) railway standards, which we do not address directly, but might fulfil due to the similarity to the IEC 61508 basic safety standard . Not all parts of the IEC61508 are applicable (also not normative) in the scope of this work, for which reason we only address sub parts IEC61508-1 to IEC61508-3 (IEC61508-4 is still normative but contains abbreviations and definitions), where IEC61508-1 plays a minor role due to its focus on the safety life-cycle process, project structures and roles. On the system level, the IEC61508-2, as well as the CS-23 and ARP4754A/APR4761 govern the assessment, safety process and general system level requirements. Below, on the hardware level, the IEC61508-2 and DO-254 are applicable [3]. On the software-side, IEC61508-3 and DO-178C are applicable. The certification agencies have both issued relevant Memorandums and Memos targeted at multicore processors or complex SoCs in general, which we will discuss later on in section 3.4.2. Note that every application domain has numerous other normative standards for environment definitions, EMI emissions and susceptibility, wiring and mounting, equipment sub-classes for specific applications, etc. Not also that it is not within the scope of this thesis to provide a guide on the application of said standards, nor any official explanation or interpretation. We expect the reader to be familiar with these standards in detail and its implications.

Developing system- and hardware-level safety solutions is fairly straight-forward when pre-certified elements, or elements with a certain legacy in critical applications (in-service or product service experience, to prove the maturity, quality, stability and experience with element failure modes), are used as the basic building blocks throughout the design. All standards detail the process and metrics required, and sometimes even provide examples, like [65A10]. When unsafe [4] commercial-off-the-shelf (COTS) elements are used, all normative standards start to become slightly vague, leaving room for innovation but also possible pitfalls and higher project risk. In some cases, for example, expert judgment or architectural mitigation are valid means to compensate missing evidence and device data by the COTS manufacturer. Current standards and past rulings of certifying agencies, however, make it clear that a single COTS element is currently not allowed to fulfill a higher-level safety function (larger than SIL1, or DAL D, for example) without special system level precautions, redundancies, or additional hardware aiding in the mitigation of potential faults in the safety function. The lack of detailed knowledge about the nature and occurrence rates of internal failures within the element, and the quality of monitoring facilities, either within a device or external, further complicates the usage of COTS

[3] The DO-254 primarily addresses complex programmable semiconductors and ASIC development, but the general process outlined in the guideline is applicable to general hardware development as well.

[4] Unsafe according to the relevant standard or guideline, where sufficient evidence from the device manufacturer can not be presented by the applicant to the certification authorities or notified bodies and the device qualification is therefore subject to the process and verification/validation activities of the applicant.

elements in safety paths. Our board-level analysis, presented in the next sections, will therefore provide base failure events, for example random and permanent device failures (based on pessimistic assumptions) and combine them in a fault tree and Markov analysis. The board-level results are propagated to the cluster level, to support our initial claim of fault tolerance and an architecture which is suitable up to the highest safety levels by providing architectural safety nets and mitigation paths. By the time of writing this thesis, a new school is slowly developing in the functional safety domain, focusing more on system-level fault mitigation than fully qualified safety elements. This philosophy breaks with the legacy mindset, where a "safe system is build out of safe parts in a safe combination", which was practical for low or medium complexity systems build by large cooperation in a protected, controllable ecosystem/environment. SOTIF [3219] drives this concept even further. For generations of legacy systems, safety functions had been realized, in many cases, by simple electro-mechanical or electronic systems. The increasing complexity, connectivity and high degree of automation in recent development projects leads to a new, very different kind of safety case. Many components are not qualified (meaning that no special precautions were taken for safety, and the manufacturer does not implement a compliant process) and their failure modes and failure rates are rarely known. Instead of controlling local failures in the elements themselves, we counteract on the system level with active fault mitigation techniques and selective redundancy, combined with dissimilarity in a flexible architecture. These new generations of safe systems do not only offer far more safety-critical computing resources than their predecessors in a fail-safe configuration, but are easily extensible to fail-operational capabilities as systems grow or use-cases change. The latter provides a significant benefit for overall risk reduction when interacting with these systems and raising the level of thrust for highly automated (and autonomous) technical systems to a new level.

This new trend has far-reaching implications for upcoming editions of normative standards. While it makes immediate sense to compensate the shortcoming of individual elements on multiple, system-wide safety nets, these kinds of systems are significantly harder to develop. They span multiple disciplines, groups within an organization or subcontractors and require deep technological insight into the technologies used by the system and safety engineers. For example, fault trees, spanning from the highest system level right down into individual computing units, are not yet fully established in the industry and must be tightly coordinated. Furthermore, this change in the very fundamental system design process emphasizes a holistic risk and safety management, centered on system functions rather than components (which can be made or bought). Do we really need to build safe systems out of safe components? Can we build safer, more reliable, highly available systems with multi-fault tolerance with a simple design? These questions came up early in the process of developing our architectural concepts and in discussions with certifying bodies, triggered by systems-of-systems nature of today's very complex SoC semiconductor devices. They are closely tied with the effort to use multiple internal

cores, to form an internally redundant SoC with COTS devices available on the market. The short answer to the above questions is: no. Due to the device internals required to achieve high levels of integration and clock speeds in the gigahertz domain, a lot of internal logic is reused, shared or electrically depending on one another, such that the freedom of interference (required by the standards) or electrical independence is not given (in COTS devices). This in turn defies, from a technical standpoint, any argument for redundancy which might be claimed by using for example two or three cores (in a single SoC), to compute the same critical function in parallel to provide fault tolerance due to common mode failures. This simple, yet intriguing fact has far-reaching implications on the possible use-cases, as well as the entire system level architecture for safe systems, requiring the usage of COTS devices. A single, highly-complex, COTS SoC is not able to provide single fault tolerance by itself. For low safety levels (up to DAL C, SIL2), this is no issue at all, since these levels do currently not require fault tolerance or very low failure rates. If sufficient failure mode and effects data can be generated, the applicant is free to use such devices without many issue in real-world projects. But for higher safety levels, only system- and board-level measures allow the usage of more capable devices in future systems. This in turn has implications on every part of the system architecture, since additional requirements, like mixed-criticality or fail-operational behavior for certain functions, which finally lead to our proposed architecture. The reduced diagnostic coverage for internal function units and software-based monitoring algorithms on untrusted silicon also lead towards multiple physical devices per board, eventually resulting in the more complex (and preferred) three device configuration. In large scale manufacturing, the increased cost for the individual boards can be easily compensated by the system-wide reduction of control units associated with the availability of a high-performance, mixed-criticality cluster. And from a certification point of view, it is no longer possible to disqualify the use of complex COTS devices, due to the dissimilarity by design and the high order of hardware and information redundancy with fault tolerance, which can be leveraged for the most critical system functions.

3.2 Fault Tree Analysis

Fault Tree Analysis (FTA) [ADC96], [SVD+02] has been used in the industry for many decades mainly on the system level. In recent years, it has been extended into the LRU-level with the aim to provide full system fault trees for upcoming generations of safe systems. Starting the FTA process, one must identify the basic failure events of all relevant internal parts of the equipment, along with the statistical occurrence rates (and function). The base events are then combined by and/or/M-out-of-N and other combinatorial blocks to form a tree which finally leads to the catastrophic/hazardous events identified in the functional hazard analysis (FHA). The resulting failure paths must satisfy certain goals depending on the criticality level of the top event. It may, for example, not be adequate

that a single event leads to a hazardous/catastrophic failure, or that certain intermediate events (wrong output vs. no output) must not exceed certain rates. Minimal cut set analysis (see [SVD+02] Section 10.2) can be employed to identify critical failure paths or even single events leading to a top level event, to test for adequate levels of redundancy or architectural mitigation for certain events. The results are fed back into the system design process, which adapts the system/equipment architecture until a certain failure rate budget is met and all additional goals, imposed by standards or system requirements, are met.

For our single and distributed monitoring architectures, we identified the base events stated in table 3.1 (Failure Mode Effects Analysis). The distributed monitoring architecture considered in the following is based on two monitoring devices and two additional programmable logic devices as a worst-case scenario for the failure rate and fault trees (maximum device count). Note that the overall failure rate of a unit may be reduced by using the second distributed monitoring architecture based on FPGA-based SoCs with a hard-coded processing subsystem. Note also that the underlying information for almost all random failure rates (due to device SEU) might be subject to non-disclosure agreements with the respective manufacturer. Each basic event is either non-repairable (permanent) or repairable (transient) and has been modeled based on the Weibull distribution in the "Functional Safety Suite" software [Bru], which was also used to compile the resulting fault trees and Markov charts. Accordingly, the events listed in table 3.1 specify the operating failure rate λ_{op} $[1/h]$, test interval t_{chk} $[h]$ and the repair time t_{rep} $[h]$ for transient errors.

For a non-repairable event, only λ_{op} is of relevance. Using the Weibull distribution given by its density function $f(t)$ and cumulative distribution function $F(t)$ from [PP02, p.162] with $t >= 0$ and $\beta = \lambda^{-k}$ as

$$f(t) = \lambda \cdot k \cdot (\lambda \cdot t)^{k-1} e^{-(\lambda \cdot t)^k} \tag{3.1}$$

and

$$F(t) = 1 - e^{-(\lambda \cdot t)^k} \tag{3.2}$$

we assume the special case of a constant failure rate with $k = 1$.[5] The special case transforms the Weibull into the exponential distribution. In the software used, λ is further split up into

$$\lambda = D \cdot \lambda_{op} + (1 - D) \cdot \lambda_{sb} \tag{3.3}$$

[5]Reliable failure rate data is hard to obtain for any complex device and testing by the manufacturer is usually carried out on limited device numbers. They offer a failure in time rate, but not a distribution when failures occurred during their testing. If more adequate data is required, very costly in-house qualification by the applicant is the only way to generate more data, if sufficient field data is not yet available

according to its documentation, with the operating duty cycle $0 < D \leq 1$ [1] (non-continous usage of the unit) and the standby failure rate λ_{sb}. For our calculations, we assumed the most challenging case with $D = 1$ for continuous operation. The failure rate is thereby equal to λ_{op}. The Functional Safety Suite evaluates the unavailability Q conservatively at the end of the component lifetime T with

$$Q(T) = 1 - e^{-(\lambda_{op} * T)^k} \tag{3.4}$$

which, in the special case of $k = 1$, is equal to the steady state unavailability $\bar{Q}$ by definition.

For repairable events, the software defines the occurrence rate w as

$$\bar{w} = w(t) = D \cdot \lambda_{op} + (1 - D)\lambda_{sb} \tag{3.5}$$

In contrast to the non-repairable event, which models component failures, the repairable model includes the check and repair time intervals t_{chk} and t_{rep}. This results in the mean unavailability $\bar{Q}$ of

$$\bar{Q} = \bar{w} \cdot 0.5(t_{chk} + t_{rep}) \tag{3.6}$$

since the event falls in between two checks for the steady state, constant rate.

A failure in one of the LRU subsystems (NOM, S- or D-MON or additional interface FPGA) is defined as a fault or error, leading to a failed program execution (no or untimely output result) or a wrong, but seemingly correct, output value exiting the subsystem. Repairable failures are removed by restarting the failed subsystem (soft reset or power cycle) upon detection of the failure condition. A failure of one subsystem is always detected on-board for IL2 functions, which is represented in the first set of fault trees. Later in this section, we will discuss additional fault trees for IL1 functions. A non-repairable failure (permanent), triggered by a non-functional subsystem (for example due to ageing or permanent damage by SEE), requires physical maintenance of the LRU to remove the fault. Note that transient and permanent failures are treated as exclusive in the following and a failure in time (FIT) resembles one failure per 10^9 hours. The failure rates have been estimated at an altitude of 60kft, resulting in very high transient failure rates which are not applicable to most ground based applications (despite mining where alpha radiation might be of importance).

For each subsystem, the root events presented in table 3.1 are further aggregated into subtrees in section 3.2.1, before the full LRU and cluster analysis is performed and analyzed in section 3.2.2.

Name	Model	λ_{op} $[1/h]$	t_{rep} $[h]$	t_{chk} $[h]$	Notes
NOM transient failure	Repairable	$5.2 \cdot 10^{-4}$	$2.78 \cdot 10^{-4}$	$2.78 \cdot 10^{-6}$	Upset estimation, based on NXP QoriQ T1042 (C28HPM process) data, with 415 [FIT] total soft error rate (SER, based on neutron and alpha particles) at NYC sea level. [Wil] suggests a flux scaling factor of 1258 at 60kft with 50% solar activity, which we apply linearly. We ignore the associated DRAM, since Gurmann et. Al. concludes that post correction, the word error rate can be approximated to $2.8 \cdot 10^{-20}$ $[word/day]$ [GHG+12, Table IV] at an 800km orbit, which can therefore be neglected even after scaling to full memory size compared to the possible processor upsets.
NOM permanent failure	Non-repairable	$1.5 \cdot 10^{-8}$	-	-	Averaged, based on manufacturer FIT estimates at 105°C junction temperature (worst case) over multiple devices, families and manufacturers (NDA data).
SMON transient failure	Repairable	$1.0 \cdot 10^{-4}$	$2.78 \cdot 10^{-4}$	$2.78 \cdot 10^{-6}$	Estimation, based on analysis of the NOM device, with slightly reduced complexity due to less cores and/or less memory attached.
SMON permanent failure	Non-repairable	$1.5 \cdot 10^{-8}$	-	-	Averaged, based on manufacturer FIT estimates at 105°C junction temperature (worst case) over multiple devices and families (NDA data)
DMON transient failure	Repairable	$1.0 \cdot 10^{-4}$	$2.78 \cdot 10^{-4}$	$2.78 \cdot 10^{-6}$	Estimation, based on analysis of the NOM device, with slightly reduced complexity due to less cores and/or less memory attached.
DMON permanent failure	Non-repairable	$1.5 \cdot 10^{-8}$	-	-	Averaged, based on manufacturer FIT estimates at 105°C junction temperature (worst case) over multiple devices, families and manufacturers (NDA data). Devices typically range from 5-20 FIT over their operating temperature range.

Name	Model	λ_{op} $[1/h]$	t_{rep} $[h]$	t_{chk} $[h]$	Notes
FPGA transient failure	Repairable	$4.6 \cdot 10^{-3}$	$2.78 \cdot 10^{-4}$	$2.78 \cdot 10^{-6}$	Estimation, based on Xilinx XCKU060 UltraScale 20nm device, 30% essential bits with full BRAM utilization. Single Event Mitigation IP enabled at 200MHz. 60kft above New York at 50% solar activity. Junction Temperature 85°C. Lower rates are expected for newer generation FinFet devices (UltraScale+) already in preproduction.
FPGA permanent failure	Non-repairable	$1.1 \cdot 10^{-8}$	-	-	see [Inc20] and the SEU estimator offered by the manufacturer. Predicted FIT Rate of Xilinx UltraScale-Series 20nm devices at 55°C junction temperature.

Table 3.1: *Subsystem Failure Modes (Fault Tree Base Events)*

3.2.1 Subsystem Fault Trees

The subsystem fault trees, presented in figure 3.1 to 3.4, were generated in the Functional Safety Suite v5.1.0 software based on the previously shown tables. The system lifetime (mission time, T) is defined as 30 years, 100% duty cycle (262800 hours) and is taken into account in the occurrence number $N(T)$ calculation (upper left-hand corner). The steady-state results, mean occurrence rate $\bar{h}$, mean unavailability $\bar{Q}$, as well as the unreliability, evaluated at the end of the system lifetime $F(T)$ are also shown at the upper left-hand corner. Each base event includes the event name (matching the definitions in table 3.1), steady state occurrence rate $\bar{h}$ and event unavailability $\bar{Q}$. The Functional Safety Suite employs the standard logic gate notion as defined in [SVD+02] Section 4.1. Preventive maintenance has not been included, yielding a more conservative analysis. The top event of each tree does not affect the analysis and is used as a transfer event to the LRU fault tree to link the two.

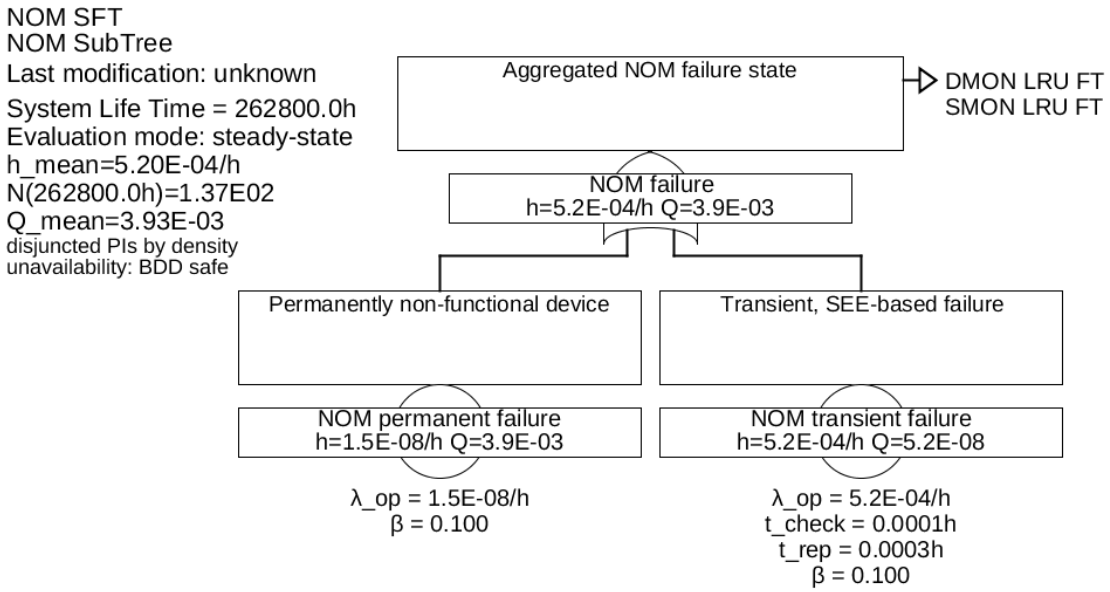

Figure 3.1: *Subsystem Fault Tree for one NOM channel*

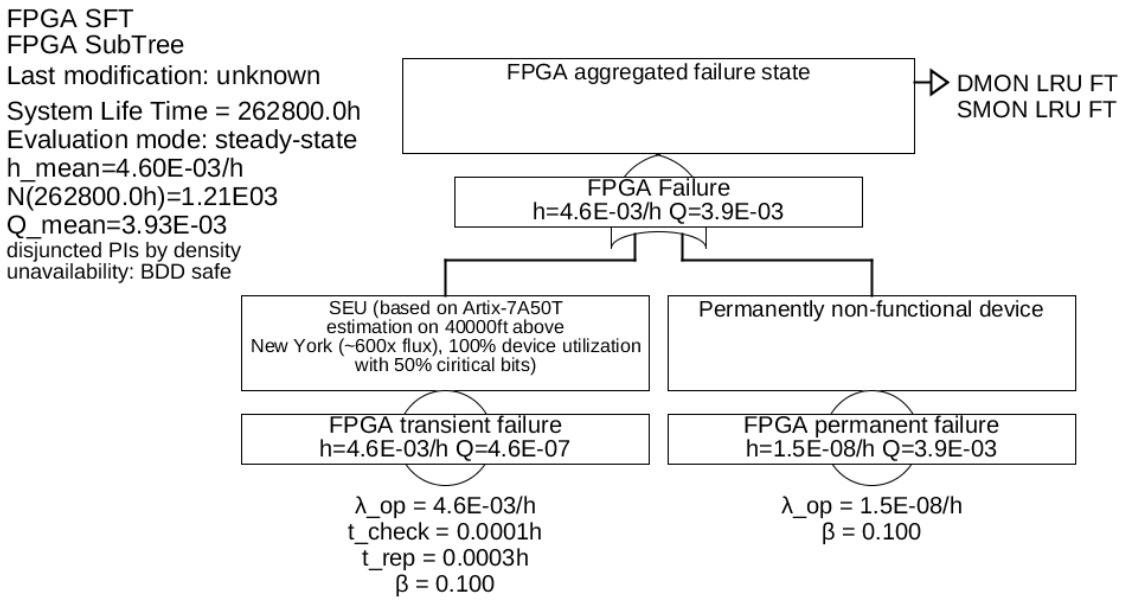

Figure 3.2: *Subsystem Fault Tree for one interface FPGA*

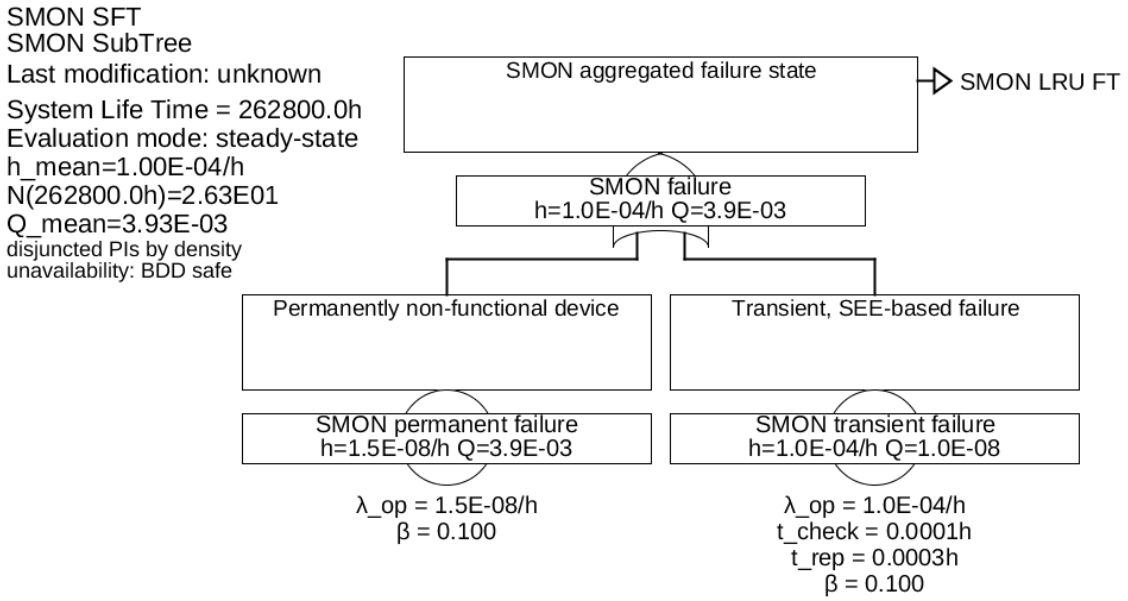

Figure 3.3: *Subsystem Fault Tree for one monitor as a SMON channel*

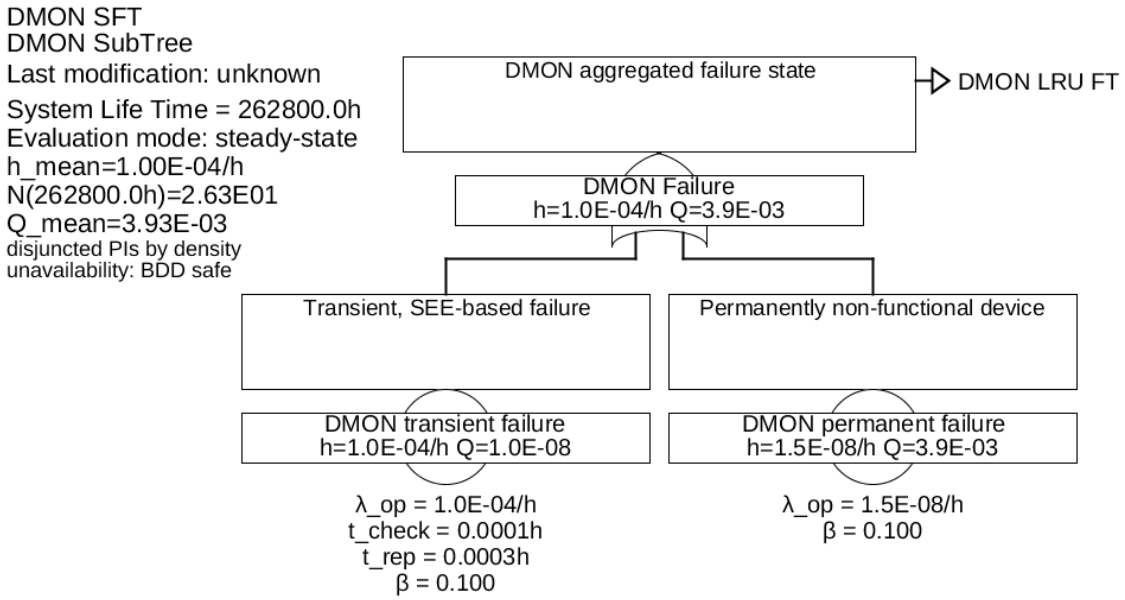

Figure 3.4: *Subsystem Fault Tree for one monitor as a DMON channel*

3.2.2 LRU and Cluster Fault Trees

A LRU based on the single monitor architecture fails, when either the NOM, or the SMON fail. This includes either a "no output result" condition, or a "wrong output result" (which is handled internally by cross comparison). The failure of both internal processing element is detected by the output stage (interface FPGAs), which enters a fail-silent state. If both FPGAs have failed, no more data transmission is possible, and the unit has failed silently. One FPGA failure is not critical, since the transmission path is fully redundant. The fault tree in figure 3.5 yields a mean failure rate of $6.6 \cdot 10^{-4}$. Note that the unit always enters a safe state (no dangerous failures for fail-safe conditions), but is unsuited for single operation in fail-operational environments, due to its high failure rate (when a silent failure or unavailability of the LRU is dangerous).

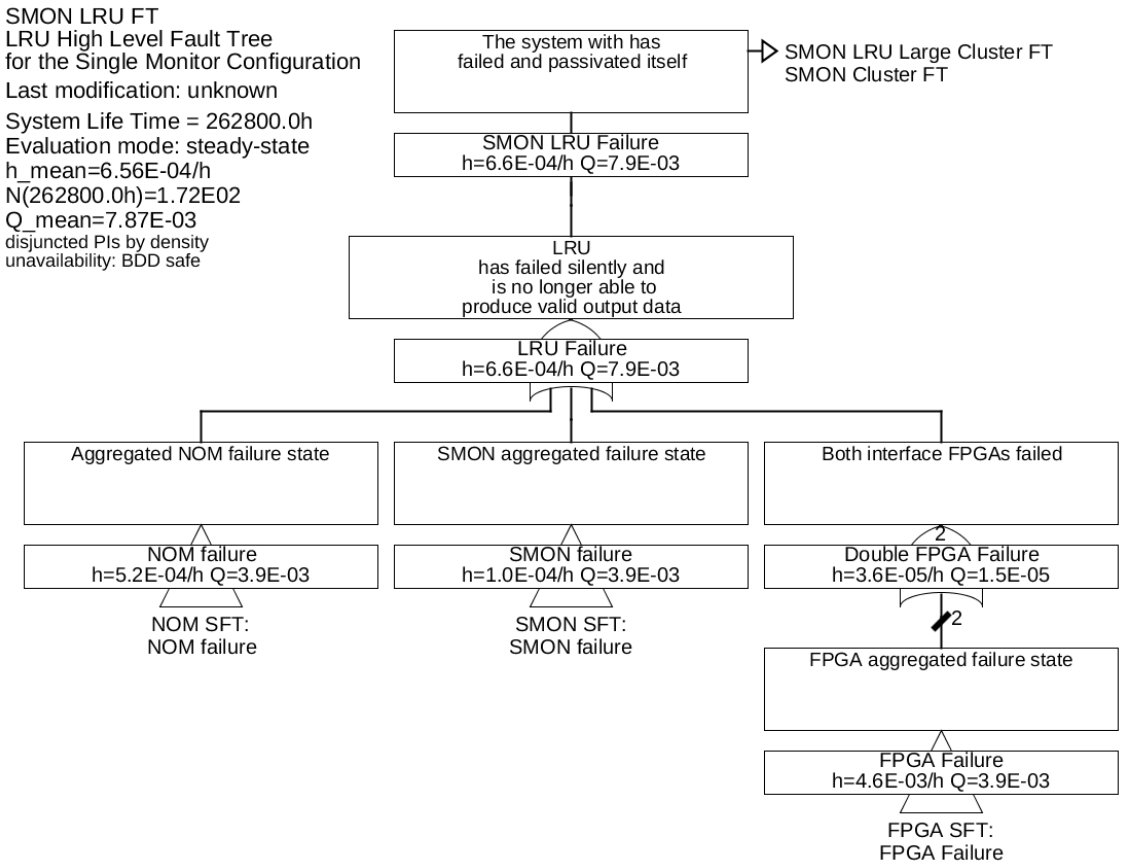

Figure 3.5: *Fault Tree for a single monitor architecture LRU*

The distributed monitoring architecture improves the per-unit failure rate by almost two orders of magnitude, see figure 3.6. Instead of the direct coupling to the subsystem failure events, a LRU failure is only triggered by combined failures of at least two subsystems. For the distributed monitoring architecture, this is the case when either the two monitors [6] fail at once, or the NOM and one MON fail. A working unit requires

[6] We focus on a design with two monitors. Additional monitoring devices are possible and would

internal monitoring, a failure is thereby given even if an unchecked output result can still be provided, although the cluster design may permit a degraded LRU without internal monitoring. We do not consider the latter case for a more conservative analysis. If both additional FPGA output states fail, the LRU fails as well, as shown in figure 3.6. Despite our high subsystem (and component base event) failure rates, the LRU performs exceptionally well with a total failure rate of $3.9 \cdot 10^{-5}$.

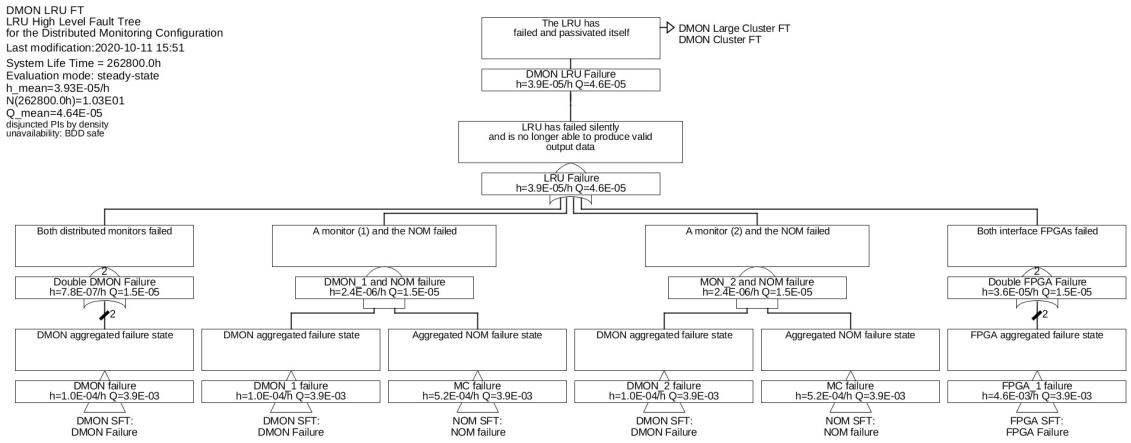

Figure 3.6: *Fault Tree for a distributed monitor architecture LRU*

We will use the NooM-notation in the following. In this work, NooM implies that IF N out of M units have FAILED, the control system is no longer available. For example, a 3oo4 architecture consists of four units, where a single LRU is not sufficient to keep the cluster operative and leads to a full loss of the control system. A 4oo4 configuration remains operative until the last LRU has failed. Based on the cluster fault trees for both architectures shown in annex C, the failure rates for IL2 functions for different cluster topologies and both LRU architectures are shown in table 3.2. It lists different degrees of redundancy (total units, minimum numbers of units required in large configurations). When single LRU operation is permissible after a already large number of LRU has failed, both architectural configurations are suitable for small clusters and require two (DMON, 2oo2) or three (SMON, 3oo3) units for medium safety levels (dangerous failure rates between 10^{-6} to 10^{-9}). The distributed monitoring architecture already exceeds the minimum failure rate required for even the highest levels in all relevant standards. It is however required by typical system designs to provide at least one or two orders of magnitude headroom for further components (associated mechanics and electronics, actuation elements and sensors). Large systems with additional requirements like zonal safety, operating on larger clusters, show a significantly lower failure rate, especially when the distributed monitoring LRU is considered. There, failure rates drop below the usual cut-off point for the aerospace domain (10^{-16}) for 4oo4 and 4oo6 configurations. The latter

reliability. However, the cost offset would hardly justify this decision since common mode failures remain and multiple have to be present in any case to mitigate those.

still provides a fully redundant system for continued operation, even if the cluster exhibits major failures (for example one zone failed and one failed unit in another). Emergency single unit operation either reduces the number of required cluster LRUs or lowers the failure rate well below 10^{-20} without considerations for application specific, common-cause failure modes. LRUs based on the single monitoring architecture perform less well, but may provide equally low cluster failure rates in two-zone 4oo4 configurations (not shown here). For less stringent operating environments, the single monitor architecture in medium cluster sizes (two to four units) may already provide adequate metrics for most ground-based systems. While IL2 functions are covered by the internal monitoring, IL1 functions are only computed redundantly at the cluster level. Their failure rate can be directly obtained by evaluating the NOM total failure rate (transient and permanent) in the desired cluster configuration. Scaling and different redundancy configurations are also possible for IL1 functions in a single topology, by varying the degree of redundancy within the cluster for each function or IL1 function group, for example, some functions are provided by all LRUs, while others are only computed in a dual-redundant or triple-redundant fashion, for example in a 5oo6 cluster.

LRUs failed	Architecture	$\lambda_{cluster}$ $[1/h]$	Notes
1oo1	SMON DMON	$6.6 \cdot 10^{-4}$ $3.9 \cdot 10^{-5}$	Single LRU base failure rate
2oo2	SMON DMON	$1.0 \cdot 10^{-5}$ $3.7 \cdot 10^{-9}$	duplex, no remaining units
2oo3	SMON DMON	$3.1 \cdot 10^{-5}$ $1.1 \cdot 10^{-8}$	triplex, single LRU operation permitted
3oo3	SMON DMON	$1.2 \cdot 10^{-7}$ $2.6 \cdot 10^{-13}$	triplex, no remaining units
3oo4	SMON DMON	$4.8 \cdot 10^{-7}$ $1.0 \cdot 10^{-12}$	quadruplex, single LRU operation permitted
4oo4	SMON DMON	$1.3 \cdot 10^{-9}$ $1.6 \cdot 10^{-17}$	quadruplex, no remaining units
3oo5	SMON DMON	$1.2 \cdot 10^{-6}$ $2.6 \cdot 10^{-12}$	pentaplex, at least two LRUs operational at all times
4oo5	SMON DMON	$6.3 \cdot 10^{-9}$ $7.9 \cdot 10^{-17}$	pentaplex, single LRU operation permitted
5oo5	SMON DMON	$1.3 \cdot 10^{-11}$ $9.2 \cdot 10^{-22}$	pentaplex, no remaining LRU
4oo6	SMON DMON	$1.9 \cdot 10^{-8}$ $2.4 \cdot 10^{-16}$	hexaplex, dual LRU operation at all times
5oo6	SMON DMON	$7.5 \cdot 10^{-11}$ $5.5 \cdot 10^{-21}$	hexaplex, single LRU operation permitted

Table 3.2: *Failure rates for IL2 functions over different cluster topologies and LRU architectures*

3.3 Markov Analysis

The Markov analysis and models in this work are using the terms and definitions from [ADC96], Appendix F. The repair of transient errors (resolvable by soft/hard reset of a device) are modeled by the return rates defined by the basic events defined in table 3.1. The resulting states are deduced from the previous fault tree analysis. Figure 3.7 and 3.8 show the resulting state models (IL2 function perspective) from the Functional Safety Suite Software. Since the transitions are very convoluted, the reader should focus on the computed state probabilities. The states are color-coded-coded, where

- green signifies a zero failures state,

- purple signifies a degraded operating state, where operation is still possible but may not be adviced due to multiple failures, and
- red signifies a non-functional operating state, where too many internal subsystems have failed.

$\bar{w}$ (named $\bar{h}$ in the software used and in the figures) and $\bar{Q}$ are shown in the upper left hand corner. Only DU-states (shown as red, IEC 61508 nomenclature, dangerous undetected - a safety-compromising failure mode which can not be detected by systematic measures or the safety function is lost entirely) and DD-states (shown as purple, IEC 61508 nomenclature, dangerous detected - a safety-compromising failure mode which can be detected by systematic measures) contribute to the LRU failure occurrence rate and unavailability of the unit, where either no internal monitoring (or redundancy for IL2 functions) or a loss of the data interfaces is present. SD-states (shown as blue, IEC 61508 nomenclature, safe detected - a failure mode which does not compromise safety and can be detected by systematic measures) result in a degraded system, which may still be operated depending on the cluster. They do not contribute to the LRU failure occurrence rate but are accounted for in the unavailability calculation, since a the unit is not fully functional. Note that a true dangerous undetected failure can only be introduced by the interface element (DMON, FPGA, FPGA-SoC DMON), when output data is internally corrupted before the final packet for data bus transmission is assembled with added checksums or forward error correction data. We assume that all LRU-internal data transfers are properly secured and provide sufficient error correction mechanisms for the respective application domain. Furthermore, if at least a two-unit cluster is considered, dangerous undetected failures (wrong or no LRU output result) are mitigated at the cluster level and no longer remain undetected in the scope of the cluster. We omitted the final failure states of the DMON architecture (more than three independent faults), due to the already low probabilities and improved readability of the diagrams. DMON-based LRUs without additional interface FPGAs exhibit a slightly lower occurrence rate and unavailability due to the reduced component count. The repair rates in the following figures are approximations for power cycle and warm reset conditions, or a unit replacement after a permanent failure.

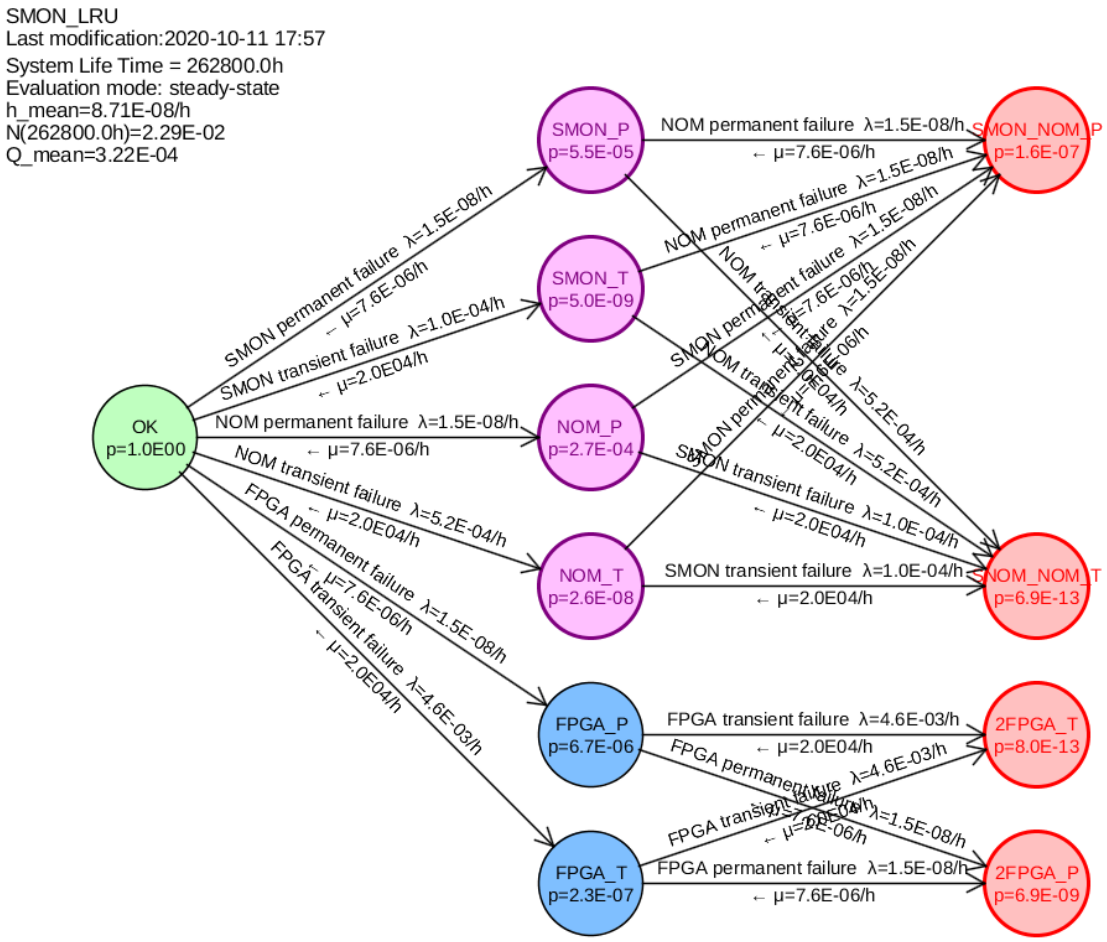

Figure 3.7: *Markov state space model for the SMON architecture*

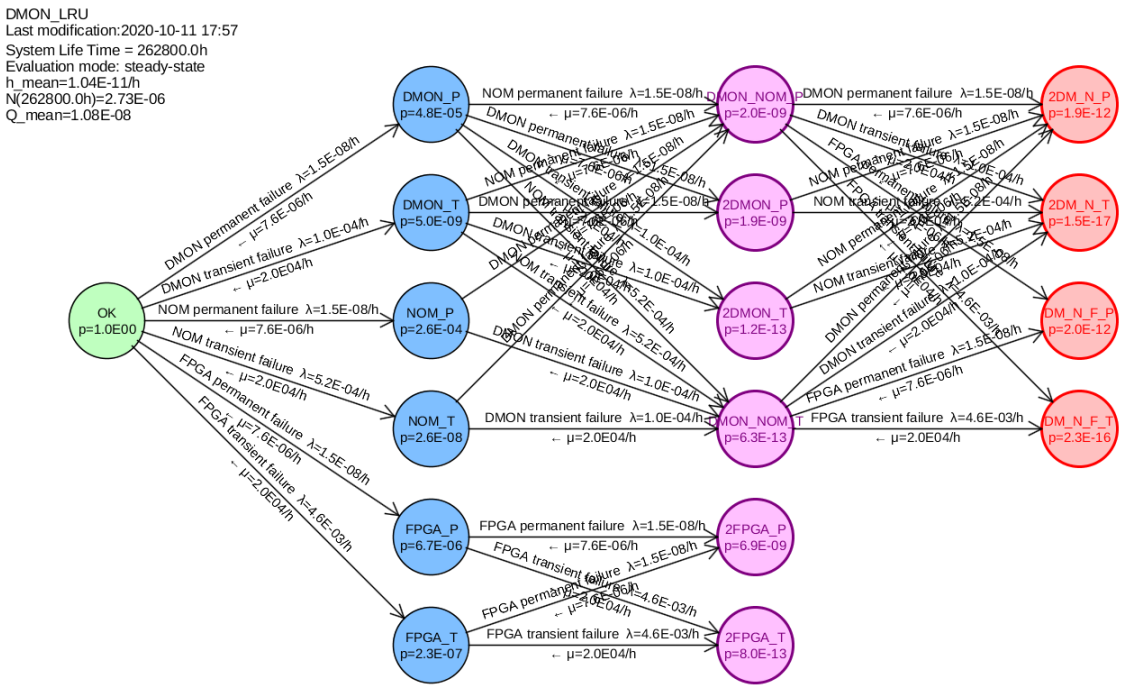

Figure 3.8: *Markov state space model for the DMON architecture*

3.4 Domain-Specific Certification Considerations

To finish this chapter, we will turn towards a domain specific discussion of certification aspects in the following sections. 3.4.1 is devoted to the industrial domain, in the scope of the IEC 61508 / EN ISO 13849. The preferred system architecture in this domain is fail-safe, due to cost/complexity constraints and less stringent application requirements. Note that special cases, like for example customer requirements for high availability or hot-swap support, or the highest certification levels may also lead to fail-operational architectures. In 3.4.2, we will discuss certification in the aerospace domain with a strong focus on fail-operational platforms for flight-critical applications.

3.4.1 Industrial Certification

The IEC 61508 [65A10] serves as the governing safety standard for general industrial machinery. Related standards, either for specialized domains such as chemical, nuclear or railway or regional standards have been harmonized since its introduction (or are in an ongoing harmonization process) to minimize development efforts between these domains. The standard is divided into seven major subparts. Parts one to three define the safety lifecycle (1), system-level requirements (including hardware) (2) and software requirements (3). Parts four to seven include definitions, examples and guidelines, as well as tables referenced by parts one to three. We will therefore focus on parts two and three. The second relevant standard for controlling industrial machinery is the EN ISO 13849. It is divided in two parts (requirements [ISO15] and validation [ISO12]) and defines for example risk assessment strategies, equipment classes with certification levels, preferred architectures, and many other machinery-related safety figures. The EN ISO 13849 will be of interest during the discussion due to its demand for single fault tolerance in higher certification levels, paired with very low occurrence rates. Since it does not address electronic hardware or software certification in great detail, the IEC 61508 is used in most projects to cover these aspects. We expect the reader to be familiar with the IEC 61508 and EN ISO 13849 for the upcoming sections, since quoting from the standard is prohibited by copyright and this work does not intent to provide a exhaustive introduction to both standards.

A well-defined development and lifecycle process is the cornerstone of every safety related development. Establishing such a process in grown company structures is not an easy, nor an inexpensive task and requires significant planning and resources. The latter also involves significant cost for the corporation seeking the relevant approvals by certifying bodies, hence they tend to only provide the bare minimum required for a given domain. The IEC 61508-1 therefore defines the minimum requirements for a compliant safety lifecycle, including process phases with their objectives and some requirements. The interested technical reader should consider sections one to four (of part one) for the scope, structure and how to demonstrate conformance. We will not further address part

one since the general development process is not our primary focus, even if this work was strongly influenced by existing certification processes around the industry to maximize real-world usability of the concepts already presented and the following considerations for certification.

Together with the EN ISO 13489-1, the IEC 61508-2 is the primary standard used on the system level for safety critical systems in industrial applications. In section 2.1.3, we defined that a single LRU based on our proposed system architecture must be certifiable up to SIL3 / KAT3(PL.d) while supporting a SIL4/KAT4 claim starting from two distinct LRUs (RQ1 and RQ2, see Table 2.1). RQ1 and RQ2 were driven by the fact that SIL1 requirements are fairly low and can be achieved with high-quality hardware in a controlled process - therefore a SIL1-only rating (or KAT1 in the EN ISO 13849 sense) for a single LRU would neither improve the state of the art nor future system designs (limited practicality for real-world projects). Reaching categories SIL3/KAT3 with a single physical LRU is difficult/impractical from multiple reasons:

- On a single printed circuit board, the design and verification effort is quite substantial to prove that PCB failures do not affect redundant topologies on the same physical circuit board and are tolerable up to a certain degree (channel isolation).
- Very few system functions in industrial applications actually require a SIL3/KAT3 (and beyond) level. The trade-off between cost and usability and the actual use-cases where a single board SIL3/KAT3 design would be beneficial is limited.
- Since I/O interfaces and are required to be independent for multiple on-board channels (13849 KAT.3 single fault tolerance - a single fault may not disrupt the safety function and is mostly detected), one would only gain the benefit of eliminating one LRU housing. The latter can even be achieved when it is possible to combine multiple units in a single housing with adequate separation.
- Availability of a non-redundant SIL3 LRU is questionable (or even below limits required by the standards, 13849 PL.d for example is hard to reach with a single unit), due to the inherently higher failure rates as designs get bigger and more complex. Note that a high-performance SoC requires extensive support circuitry, memory, etc.

We therefore conclude that the ideal certification level for the single LRU is SIL2/KAT3, since we're at least able to meet SIL2 with the SMON and DMON architecture. KAT3 is also possible, since, on a logical level, we're able to detect failures in safety functions accurately and are able to trigger a reaction even in the presence of latent faults (not only true for the DMON architecture, but also for SMON when proper cross-device monitoring is in place). If more availability is required by the application (customer will not tolerate that a single LRU failure leads to the unavailability of the whole system) or functions which

require SIL3/KAT3 (or higher) are present, one is easily able to include a second LRU. A high-availability system with very limited failure modes is possible with three or more LRUs, making an ideal fit for SIL4/KAT4 applications which can not be fulfilled with a 1oo2 or 2oo2 DMON architecture. The result is a very scalable, highly modular solution which even supports retrofitting and in-field expansion when requirements change over the lifecycle of a system or due to changed laws and regulation. By not relying on an individual part with extensive evidence and field-proven reliability at the core of our design, but rather the system-level approach of detection all possible failures due to their effects on the element level (element in the sense of IEC 61508), we claim that the amount of per-component evidence needed for a successful certification is drastically reduced. Since we lack this evidence anyway for our complex SoCs, moving the justification for a given safety case to the correct level in the system hierarchy is vital for any future certification effort.

If we look closer at the EN ISO 13849 KAT3, we notice the aforementioned single fault tolerance requirement which differentiates KAT3 from SIL3. Although SIL3 indirectly implies a redundant architecture (or ar least partial redundancy with high quality on-line tests) by lowering the $MTTF_d$ compared to SIL2, it does not explicitly force that the safety functions must still be operative after a single fault in the elements providing the safety function. Furthermore, the EN ISO 13849 also proposes a designated architecture for each class defined in the standard. For KAT2, a non-redundant approach is preferred with a dedicated monitoring unit alongside the main compute path (see Figure 10, [ISO15]). For KAT3, the authors of the standard envisioned that, in order to cope with all possible component failures, a fully redundant architecture is required. Note that KAT3 does only require fault detection where effort is reasonable, due to the redundant nature of the system, whereas KAT2 must detect that a safety function is no longer available. Although this apparent loosening of fault detection requirements might seem odd at the first look, it is adequate since the argumentation for system safety changes from KAT2 to KAT3. KAT2 follows a "safety by evidence and high-quality fault detection" (diagnostic coverage) approach, while KAT3 shifts towards redundancy and logical, per-channel fault detection with the added benefit of a second control mechanism, still capable to uphold the safety function even if one failure (or a series of failures) has disrupted one compute channel (also accounts for sensors and final elements, since the full chain should be redundant for KAT3). Successfully arguing for a KAT3 systematic capability claim with only a single logical channel (or information flow) is nearly impossible, since failures of connectors, cabling, circuit boards, etc. are also discussed during certification and do lead to undetected, unrecoverable failures of an channel without redundancy. It is certainly possible to duplicate all internal part on an LRU in one single housing with redundant physical interfaces, but cost and complexity constraints often don't permit these designs outside very special (and most likely not cost and space constraint) use-cases. Furthermore, establishing a safety claim for SIL3/KAT3 on modern, highly complex SoC devices

is very unlikely to be successful due to the level of diagnostic coverage required at these levels, if only one single compute channel (without further hardware measures taken) is used in a design. The architectures proposed in this work therefore follow an on-board, "mixed-redundancy" and diagnostic coverage approach, combining the best of both worlds into a semi-redundant, yet fault tolerant architecture with duplicated information flows and fall-back strategies. The diagnostic coverage is at a medium level for the individual SoC in the SMON and DMON architecture, but when combined with the additional compute channel with information redundancy, the overall result on the board level is already single fault tolerant (SMON) and, with the extension to the DMON architecture, tolerant of latent and multi-point faults anywhere in the LRU's flow of information. The levels of diagnostic coverage, as defined by the EN ISO 13849, are "none", "medium" ($\geq 60\%$), "high" ($\geq 90\%$) and "very high" ($\geq 99\%$), whereas the ISO 61508 defines the levels "low" ($\geq 60\%$), "medium" ($\geq 90\%$) and "high" ($\geq 99\%$). Both standards use the same basic definition of diagnostic coverage, which is defined as the ration between detected and undetected dangerous failures see [65A10, Annex C, C.1, G] and [ISO15, Annex E, E.2]. In order to claim a high diagnostic coverage for a measure undertaken by the designer or a COTS element (with documentation provided by the vendor), one must ensure that (either continuously at runtime, or at a specified interval) it is able to detect at least 90% (better 99%) of all failures determined in the FME(D)A analysis for a given element. For example: The DRAM of a processor is protected by error correcting codes (ECC, single error correction, double error detection in most cases) built into the memory controller of the processor. Does the ECC-logic support online/offline self-tests (undetected latent faults in the ECC logic might lead to undetected real memory errors not being discovered)? If not, is an adequate software mitigation strategy in place, where the ECC logic can be tested at runtime or in fixed intervals (if permitted by the use-case of the system)? Are there memory error scenarios not detected at all by the ECC logic (due to the polynomials used)? Due to the complex nature of most on-device peripherals and the usual lack of safety related features in device families originally designed for other application domains, the diagnostic coverage may be limited to only "medium" (EN ISO 13849) or "low" (IEC 61508) without further software countermeasures to detect latent faults at runtime. From experience, a "high" or "medium" (EN ISO 13849 and IEC 61508) diagnostic coverage value is usually reachable with additional efforts in the software and hardware design, which permits even complex SoCs in designs up to SIL2 with a corresponding safety analysis. With our SMON and DMON architecture, we lift parts of this process burden and shift it towards the architecture itself, which is by design inherently resilient against latent faults manifesting into true failures (when dangerous errors and failures are no longer detected due to a failed diagnostic measure or a systematic shortcoming of the diagnosis). By removing the constraint of proven-in-use components to achieve state-of-the-art performance for upcoming designs, we are fixed to Route 1s according to the IEC 61508-2 Section 7.4.2.2, leading to the demand for systematic, software and hard-

ware measures to control faults in these categories. Furthermore, we enter Route 1H and choose a hardware fault-tolerance (HFT) approach and do only rely as much as needed on the failure rate and freedom of errors of the individual element (Annex E and F of the IEC 61508-2 are out of context for this work, since we work with COTS components only). This choice then requires a system-level argumentation based on the safe failure fraction of each element (and the associated SIL level) while discussing the decomposition of these elements who make up the top-level system architecture on a block level together with common-cause, mixed-criticality and other analysis to support the claims made. If we again choose the pessimistic estimate that only SIL1 can be achieved per element (a NOM channel, MON channel, etc.) since we are working with Type B elements [7], the maximum level which can be reached (without any diagnostic measures, functional degradation paths and other system level measures, etc.) is limited to SIL2 for a single DMON/SMON LRU. Since we substantially lower the single element's (inside the LRU or its subsystems) dangerous undetected fault rate, the safe failure fraction of these elements will in most cases be high enough to reach SIL2, effectively raising the upper, per-LRU certification level to SIL3 in the DMON and SIL2 in the SMON case (SMON is limited due to the HFT=1 [8], while DMON can be classified as HFT=2 or even HFT=3 when implemented to the full extend described in this thesis). With the constraints from Sections 7.4.3.2 to 7.4.3.4 from the IEC 61508-2, and the assumption that a Systematic Capability (SC) of SC 1 can be demonstrated, a single DMON LRU is able to reach SC 2 when both MONs are similar. With hardware dissimilarity in the MONs, SC 3 is possible. Since multiple physical LRUs are likely to be used in a system (to increase MTBF, redundancy or zonal safety), it is sufficient to achieve a systematic capability of 2 for the individual unit to accomplish SIL3 with smart actuators and sensors on the top system level. With multiple units and zonal safety, multiple LRUs greatly enhance the availability and raise the SFF even further.

From the hardware safety integrity perspective, the concept takes route 1H (hardware fault tolerance, route 2H is based on component reliability data), since reliability data is not available in general for the rather new, high-performance multicore devices present in the NOM (and also possibly the MONs) in both the SMON and DMON architecture. These devices will be classified as type B elements, because they lack an in-depth failure mode definition on the silicon level and dependable failure data for both detectable and undetectable failures. Since on-chip diagnostics (if present) can not be regarded as a high quality failure detection measure in general, we assume that the safe failure fraction of the individual element (NOM, MON) will be low, and only suffice to comply to a low SIL / SC level. However, with the inherent hardware fault tolerance of 1 or 2 (for

[7] Type B elements are unsafe COTS components in the sense of the IEC 61508, where failure modes, fault condition behavior or failure rate data/evidence is not well-defined or lacking.

[8] In the IEC 61508 context, HFT=1 implies one degree of hardware fault tolerance, e.g. redundancy or (dissimilar) backup system. HFT=2 implies the system can tolerate two losses or dangerous failures before they manifest as a loss of the safety function.

DMON, in case both MONs are dissimilar), even a safe failure fraction of less than 60% is adequate to reach SIL 2 with HFT 2 and SIL 1 with HFT 1 according to Table 3, IEC 61508-2. With further investment in the analysis of the nominal and monitoring channel devices, and a safe failure fraction above 60%, SIL 2 and SIL 3 are reachable with HFT=1 and HFT=2. Since the individual LRU does not contain a serial combination of those higher-risk components, a higher per-element SFF and SC directly improves the whole LRU safety rating. Note that Annex E and F of the IEC 61508-2 do not apply, since we do not claim evidence based on on-chip redundancy or custom ASICs. Also note that internal, on-chip diagnostic measures are only regarded as a possible way to detect internal device failures in the COTS multi-core devices faster, but are not directly related to system safety. Fault detection, mitigation and fault tolerance is accomplished by the combination on the LRU and system level and with the architecture presented in chapter 2. Full information redundancy over all processing elements within the LRU ensures that all faults (also random, like single event upsets or other radiation and environment effects) leading to a wrong, untimely or missing computational result are discovered internally and do not leave the LRU.

One additional topic of importance in the presented architectures is the combination of safe and non-safety functions. All standards require that a non-safety function shall not interfere with a safe function and cause dangerous failures, see for example sections 7.4.2.3 to 7.4.2.5 IEC 61508-2. We already discussed several measures in hard- and software in chapters 2.2, 2.3 and 2.4. When properly implemented, they provide adequate separation as well as spacial and temporal independence. Multi-point faults may, however, lead to a loss of this separation, especially when internal hardware units in the NOM or MON fail undetected and additional errors in the remaining on-board channels fail to detect a wrong output result. These multi-point faults require that at least two independent, hardware dissimilar elements exhibit the same or different faults leading to the same wrong output result. As shown in 3.2 and 3.3, even under pessimistic assumptions, the remaining dangerous failure probability is exceptionally low and only exists as a theoretical possibility for most use cases. Furthermore, since multiple LRUs are present in a real-world scenario, these remaining dangerous undetected failures of a single LRU can be discovered easily by redundant units and do therefore not pose a real thread in an actual implementation. Note that the loss of isolation between safe and unsafe functions does only impact the NOM, not the MON. In both the SMON and DMON architecture, only the NOM is executing non-safety functions alongside safety functions.

3.4.2 Aerospace Certification

The certification of aerial vehicles on the top level is governed by different certification specifications (CS), usually issued by a some type of government agency with a given scope. Depending on the usage type, number of passengers, maximum speed, etc. the CS defines certain requirements and characteristics the aircraft must achieve in order to

be flown in a particular airspace. For this work, we will only focus on the aspects for on-board electronics. There, the CS define acceptable means of compliance (AMC) and alternate means of compliance (AltMOC). An applicant can for example show that he adhered to a special set of guidelines (CS among others reference guidelines issued by the Radio Technical Commission for Aeronautics, RTCA) which in turn define a special set of requirements depending on certain environment conditions, equipment type, the power system used on the aircraft, etc. The most relevant standards for developing airborne electronic components are by far the RTCA DO-160 [SC-12] (environment qualification, e.g. mechanical shock and vibration, EMI, voltage injection, lightning strike), the RTCA DO-178 [SC-11] (software qualification, microprocessors and complex electronic hardware) and the RTCA DO-254 [SC-00] (complex electronic hardware, programmable hardware). But besides those specialized guidelines, general process standards exist, usually defined in the CS by referencing for example the SAE ARP4761 [ADC96] for how safety assessments should be conducted on different system levels, right down to the individual unit, and the more overall process-centric SAE ARP4754A [ADC10]. The ARP guidelines also define the design assurance level (DAL, highest DAL A, lowest DAL E) which can then be used in the scope of the CS to establish a certain dangerous failure rate goal for each subsystem or LRU. Note that, due to copyright reasons, direct citations form the standard are not possible in this work, like with the industrial certification standards. This section is not intended to provide an introduction to those standards.

For each function the system must perform in order to stay operative or to complete a certain mission, a functional hazard analysis (FHA) must be performed, together with a fault-tree analysis (FTA) or Markov-analysis (MA). These analyses lead to a function-based design assurance level (FDAL) which is focused on the function level, not on LRUs or software. Once all system functions are known and categorized by their criticality and failure mode (fail-op, fail-safe/silent), each system function is then allocated to hardware and/or software. This allocation is not strictly a one-to-one relationship (e.g. one LRU per function). Prominent examples are primary flight control (PFC) or a full-authority digital engine control (FADEC) where the loss of functionality inevitably leads to a catastrophic scenario (which directly implies the highest DAL for the respective class of aircraft). Alongside the FDAL allocation, further aspects like zonal safety and common-cause failures must be taken into account in the FTA and MA. Once the FDAL assignment is finished, each resulting piece of electronic equipment (like a LRU) is assigned an equipment DAL (IDAL), based on the functions to be executed on this unit. The ARPs define allowed methods to compute the resulting IDAL based on the FDAL, degree of redundancy, dissimilarity between redundant channels, etc. Once the IDAL is known, each unit can be developed in a well-defined and strict process in terms of hard- and software, accompanied by a safety process to compile the necessary certification evidence for the targeted IDAL. This evidence is eventually presented to the certifying agency and is part of the type approval process of each aircraft. Later modifications to this specific aircraft

type then require supplemental type certificates, based on units which have been developed to a certain IDAL based on a technical standard (resulting in a technical standard order, TSO, approval) and have been integration-tested on this type.

On the equipment level, this process basically repeats for internal sub-assemblies and even individual parts which play a vital role in the design of the unit. The approved method of FTA and MA is used to compute the failure rates of all critical functions, where the designer shows that a failure is less probable than x per usage hour or event. Note that this is final evaluation is done on a functional level. To support these numbers, the FTA's base events originate from FMEAs, FMEDAs or FMECAs which directly translate right down to the hardware design of the unit. The software IDAL is achieved by the development process, which guarantees certain freedom from errors based on requirements based testing with coverage. The type of coverage needed highly depends on the IDAL.

On the level of a cluster, comprised of LRUs based on either the SMON or DMON architecture, different system functions are allocated. Each of these functions holds a certain FDAL, which results in an IDAL once the allocation to the cluster is done. Based on this FDAL/IDAL, the resulting software function and physical flow of input and output information for this particular function is evaluated. If the allocation is done correctly (dependent on the FDAL, on the NOM and MON, or only the NOM), one can show with the cluster fault tree presented earlier and the top level system FTA and FHA, that a single unit failure/loss within the cluster is no longer catastrophic, if the unit meets a specific IDAL. The failure of a certain function is mostly tied to element executing the function, which will be the NOM in most cases. In a classic LRU design, the processor in the NOM would be a well-tested, fully understood, very simple MCU in order to conduct a detailed FMEA to support the claim for a high IDAL. However, in our LRU the NOM is a low-trust, highly-complex, multi-core device, where an FMEA can only be conducted by the manufacturer of the device, and a lot of complex, coupled failure modes complicate a safety claim even further. With the SMON and DMON architecture, and the software concept presented earlier, the dependency on a trustworthy microprocessor is fully canceled, by applying architectural, systematic mitigation. Since each architecture is inherently hardware-dissimilar, critical functions (equal to or higher than DAL B) are at least once duplicated inside each unit on dissimilar hardware. Once a mismatch is detected, the fault mitigation strategy depends on the cluster and the architecture used (SMON or DMON). While most failures can be mitigated effectively, one common-case failure mode remains in the special case, when in a SMON cluster with similar NOM and similar MON devices in each LRU. There, a common-cause failure of all NOM devices renders all cluster units inoperative, if the wrong fault mitigation strategy has been chosen. Instead of the immediate fail-silent strategy, it is vital to first evaluate the state of all other cluster elements first, in order to decide whether the cluster should stay operative and use the results of the MON while all the NOM carry out a controlled reset.

Under the assumption that the units within a cluster do not share a common-case

failure mode (impossible with DMON, SMON requires a special mitigation strategy), we can based on the size of the cluster (which is relevant to reach the desired cluster failure rate if the individual LRU failure rate is too high) determine the maximum FDAL level allowed on the LRUs: If it is possible to show DAL C compliance for the NOM and MON (which is possible even with complex hardware), the internal architecture and FTA analysis easily permits a DAL C approval which has no single points of failure. The combination of these units in a cluster, combined with the degree of zonal safety required for the application, then allows for at least FDAL B to be executed. If enhanced functional degradation and fault mitigation strategies are employed, it is possible to further raise the individual unit's DAL to B, because only multi-point failures, at the same time on dissimilar devices can render a function inoperative. Zonal safety, used to decouple the environment to some degree of redundant units, can then be used to argument of a DAL A classification. The DMON architecture inherently offers this capability and only fails-silent in a trivial fault mitigation scheme after two dissimilar devices have failed.

Note that the voting of function output results can either be executed on smart final elements, or inside the cluster. The latter requires a more sophisticated approach on channel synchronization and the voting algorithm. Output result voting on the final elements further reduces the coupling between the NOM and the MON, while easing fault mitigation. The distributed monitoring/supervision between the nominal and monitoring channels in both the DMON and SMON architecture accelerate the on-board failure detection to support the claim for a true fail-silent LRU design. These actions aid in a safety case based on architectural mitigation, which we will discuss in the remainder of this chapter.

Also note that in the DMON architecture, a classification of the single computing channel failure (either NOM or MON) as Minor or even "no safety effect" would be possible, since it allows for a full detection of all possible channel output result faults if at least two good devices remain. Coupled with the fast detection of a channel failure and a proper, application specific degradation and mitigation scheme, the failure of a NOM or MON does no longer play a relevant role in the LRU's fault tree. If NOM and MON devices are similar in each LRU within a cluster, the common-cause failure of all NOMs at the same time (even if zonal safety decouples the environments between LRUs, but still, this failure mode might be a concern) is negligible if the degradation and mitigation scheme allows the cluster to continue operation. For example, if the cluster operated nominal, the cluster votes by itself and only sends one known-good actuator command set to the final elements. In a degraded state, the actuators are being send a set of command values and carry out the voting by itself, since the cluster is now degraded and can no longer decide in the individual LRU which results are valid without significant communication and algorithmic overhead. The hardware independence for higher criticality functions is vitally important in both the SMON and DMON architecture. By establishing fully redundant information paths right down to the computational devices inside

each LRU, we decouple the non-trustworthy microprocessors (in terms of certification, not enough evidence) and given the hardware dissimilarity, the base events in the LRU fault tree analysis do no longer suffer from a single root cause which leads to unit and cluster failure. The redundant flow of information through the LRU and the cluster is essential in both architectural concepts and requires fully redundant data bus interfaces, in order to maintain the independence and dissimilarity of each channel inside the cluster LRU. In extreme scenarios, or when degradation paths can not be established, additional dissimilarity between the cluster LRUs NOM can be established (different manufacturers, instruction set architectures, etc.) to harden the cluster against common-cause failures at device or software level (the latter requires dissimilar software, which is implicitly given when different core architectures are used).

While the FDAL and resulting IDAL of the system functions allocated to a cluster along with its fault tree dictate the level required for the individual LRU, the software responsible for the strict separation of software functions on the multicore NOM or MON must always be certified to the highest level required by any system function. It may for example be possible to limit the overall unit LRU to DAL B or even DAL C depending on the application, while the spatial and temporal isolation software must be certified to one or two levels higher, as required by the RTCA DO-178 for any multitasking operating system or function scheduler. We extend the spacial and temporal isolation between the individual cores in the NOM and MON by a high-rate, cross-device monitoring in order to significantly reduce the time to repair in case a catastrophic software failure renders one of these devices inoperative. The goal is to detect and indicate an error even before the set of actuator command values is due, restart the violating device or LRU and always continue operation for high criticality functions. The fail-op behavior is either inherited by dissimilarity between the NOM within the cluster or by a degradation path. Upholding also the lesser or non-critical functions when failures occur is possible when the MON channels offer sufficient computational resources. Levering software dissimilarity in a cluster made out of similar LRUs with dissimilar NOM and MON channels or DMON LRUs is another way of adding additional resilience and harden the cluster even further.

Both major certification authorities in the aerospace industry, the Federal Aviation Administration (FAA) and the European Aviation Safety Agency (EASA) have issued relevant information material for complex off-the-self hardware components, namely the Position Paper CAST-32A [Tea18] and the CM-SW-CEH-001 [EAS18a]. Both documents define a set of tasks/actions which have do be carried out depending on the IDAL involved. We will now discuss the these items in order to show either direct compliance or the alternate means of compliance for the SMON and DMON architecture given by it's design. Note that we assume a IDAL B LRU in a cluster serving FDAL A functions.

3.4.2.1 EASA Certification Memorandum SW-CEH-001 Activities

Activity numbering and description from [EAS18a], chapter 9.3, see reference for full activity text.

- (1) - Device Classification

  The NOM and MON devices are both highly complex microcontrollers since they have more than one core on the same bus with a shared memory controller. Several complex peripherals are used for data bus communications which operate on microcode or dedicated accelerators.

- (2) and (3) - Device and Manufacturer Data

  For all complex devices used, the usual documentation including manuals, errata sheets, etc. is usually available. For some devices, dedicated NDAs which are required for further reaching documentation on internal function units or errata documents. All manufacturers that the author worked with in the past have a strictly controlled, very formal development process due to the complexity and cost of semiconductor manufacturing today. Especially the AEC Q-100 [Com14] automotive component quality standard offers the needed quality documents to justify the use of complex COTS microprocessors. One should not seek the availability of design data from the manufacturer, since this information will not be available.

- (4) and (5) - Usage domain aspects

  The final application dictates the used/unused internal units of the NOM and MON. Unused functions are deactivated on most of today's devices by power and clock gating. The clock and power distribution inside the SoC for the respective peripheral is disabled in order to save both power and thermal dissipation budget. This is vital for many application fields (mobile, industrial, small scale rack devices, etc.) and fully disables the unit. The management of such functions is usually controlled by a dedicated platform controller inside the SoC or per peripheral in a bus interface unit. We check periodically at a very high frequency that the units are still disabled. All vital static device configurations are periodically read and compared against their desired values. This is easily done by simply computing strong cryptographic hash functions over the register contents and comparing the outcome to a known good hash value.

  We heavily discussed the usage of internal function units, as well as shared resources throughout this thesis with the goal to demonstrate the decoupling of internal and LRU failure modes. Internal function units usually can not be fault-injection tested nor can they be otherwise sufficiently verified. We do explicitly not rely on those units to provide the overall LRU or cluster-wide low failure rate. We do however use some internal units to provide the spatial and temporal independence within

in the SoC (like memory management units, performance counters, etc.) which are checked dynamically at run-time. Failures which translate to failures of certain critical functions are fully mitigated by the LRU and cluster architecture.

As discussed in section 2.4, the single core equivalence framework provides the possibility to still conduct WCET analysis in a non-interference free SoC by establishing upper bounds through memory and shared cache partitioning. The concept also helps to provide additional determinism. Our general multi-processing concept does only support static resource allocation without dynamic task reallocation (which is possible for degradation paths to compensate failed cores, but not during nominal operation). We therefore advocate a pseudo-AMP approach on the SMP hardware, to dedicate one or more cores to certain functions to aid system verification and determinism validation during functional testing. Since the processing platform offers a significant boost in computational performance while most control algorithms remain fairly light-weight, the issues with computing time and WCET from old, low performance platforms will vanish. They are replaced by the difficulties in allocating the software functions within the complex cluster and ensuring that the true information redundancy and degradation paths are well engineered and validated.

- (6), (7), (8) - Errata, Past Experience

  These points are per default covered by a compliant errata management process as already required by the DO-254. This also includes the errata safety impact assessment. Additional experience during the development process and prototype phase, hardware-in-the-loop testing, etc. is beneficial and also counts towards product service experience.

- (9) and (10) - Configuration Management

  From previous experience with several semiconductor manufacturers, all device changes are adequately documented and distributed in a formal and controlled manner via product change notifications ahead of all new die revisions. The die revision is always directly readable on the device with special match-codes.

- (11) and (12) - Hardware and Hardware/Software Integration

  As shown in this chapter, the device failure modes do not play a significant role in overall system safety once the SMON and DMON architecture is applied. The individual failure does not lead to dangerous failure scenario. The hardware strapping used to select initial boot modes until a first stage bootloader or initial parameter section in the program memory configures the device can be directly tested during prototyping. The loaded device configuration by a predefined data structure in the program memory which gets loaded first (usually for memory, clocks, power, etc.) can be dynamically verified at run-time by hashing the configuration registers as

described before. A simple development time test is also possible without special instrumentation.

- (13) and (14) - Product Service Experience

  Product service experience is usually very difficult to obtain, if not generated by the applicant himself. Even if the device has been used in safety or aerospace applications before, it is unlikely that a competitor hands out delicate information about unit in-field runtime estimates, logged errors and returned units, since these figures are also business critical. In other, non-safety related fields, the exact use case and environment is often not known or not representative. It is best to produce sufficient service hours in a controlled hardware-in-the-loop environment with many units in parallel. For example, with two clusters of 5 units each in a test setup, it only requires around 13 months with 24/7 simulation runtime to claim "Sufficient Product Service Experience (PSE)" for a IDAL B according to the certification memorandum". The required test time can be significantly lowered if some data of non-safety applications can be obtained (down to around 42 days for the described setup).

- (15) Architectural Mitigation

  Our SMON and DMON architectures are essentially architectural mitigation measures to compensate for the low-trust COTS multicore device. Instead of the strict reliance on manufacturer information or deep fault injection in the devices, we actually gain additional confidence way beyond the level currently possible in a technical sense (today's confidence is often carried by process assurance). This is due to the "it will fail" attribute for each sufficiently complex, combined hardware and software building blocks of the NOM and MONs, which is mitigated on the LRU and finally the cluster level. The end result is a true single fault tolerant LRU with a multi-fault tolerant cluster.

- (16) Partitioning

  As discussed in 2.4, software partitioning is present at both the NOM and MON. In addition, the cross-device monitoring hardens the LRU architecture and provides additional error detection facilities. Since the MON only computes a limited function set dissimilar to the NOM, additional decoupling can be claimed by the mentioned external mitigation to the COTS component in the certification memorandum.

3.4.2.2 FAA Position Paper CAST-32A Objectives

Objective naming and description from [Tea18], chapter 7, see reference for full Objective text. The objectives align with the SW-CEH-001 Activities.

- "MCP Planning 1"

  The system uses the architecture described in chapter 2.4, an asymmetric multiprocessing approach, with a wide set of mixed-criticality software components. The cluster can not be characterized as an IMA platform, since we execute similar software on all cluster LRUs and do not support dynamic reallocation inside the cluster by default. The spatial and temporal partitioning as described in chapter 2.4 provides adequate isolation.

- "MCP Resource Usage 1" and "MCP Resource Usage 2"

  As in regular systems, the configuration parameters and settings are documented and tested. Periodic checks at run-time ensure that the relevant configurations are not altered.

- "MCP Planning 2" and "MCP Resource Usage 3"

  See discussion of "Usage domain aspects" from the SW-CEH-001.

- "MCP Software 1" and "MCP Software 2"

  Chapter 2.4 discussed the aspects of interference channels, WCET and mitigation measures. The data and control coupling needs to be verified on a per-application basis.

- "MCP Error Handling 1" and "MCP Accomplishment Summary 1"

  This objective is fulfilled by our SMON and DMON architecture. The accomplishment summary needs to be compiled in the individual project.

In this chapter, we analyzed our proposed architectures in the light of industrial and aerospace certification. We first discussed COTS device certification in the light of our concepts and presented fault tree and Markov analysis. The final domain-specific considerations complete the picture and show, how we intend to bring our concepts into new generations of safe systems while being compliant to current standards and guidelines. The reader should take away that the SMON and DMON architecture enable well-designed safety systems, where the individual failure of the individual complex COTS devices used in the NOM and MON does not lead to a catastrophic LRU failure. Both architectures do not violate today's standards and can be applied to future system architectures in highly automated industrial, aerospace and transportation systems. Future architectures can no longer suffer from the strong bond to a specific manufacturer or device, like in many commercial systems today, due to the system level measures we translate into the hardware

and software safety process. The shift in the mindset from proven parts, towards more fault tolerant and redundant system architectures is actively discussed in many application areas and might be reflected in future standard revisions. It will ultimately lead to very resilient, low-failure rate systems which are desperately needed for future industrial and mass transportation systems, where the sheer number of systems in the field requires even lower risk factors than current architectures can provide.

4

Practical Board-Level Implementations

In this chapter, we will discuss four implementations (two DMON, two SMON) based on actual COTS hardware components (Contribution C6). These designs can be used immediately in upcoming LRU designs featuring the DMON or SMON architecture. All four designs are based upon the findings from previous chapters and are directly suited for certification. All COTS devices are state-of-the-art with long term support, built by companies with a significant certification heritage. The application domains for the implementations we will show are not constrained and include for example industrial, railway or aerospace. Domain specific certification aspects apply when used in a real-world project, see the previous chapter. We will focus on the components used at the board level with their specific interconnects and also address external interfaces. The surrounding system architectures showcase possible future designs in transportation and industrial systems based on high-speed interconnects and legacy data busses. The overall goal for this chapter is to show that both the DMON and the SMON architecture are feasible and ready to deploy with the COTS devices available today, leading to future proof and also backwards compatible LRUs for a high-performance central computing cluster, running highly critical system functions in a mixed criticality environment.

We will start discussing the SMON variants in the first section and continue with the DMON variants in the second section of this chapter.

4.1 SMON Implementations

The architecture of most embedded systems, especially when controlling vital system functions, is always very closely tied to the full system architecture, e.g. its surroundings, interfaces, sensors, final elements, etc. We developed the SMON board-level architecture for larger systems with higher degrees of redundancy, due to domain specific aspects like zonal safety, very low failure rates or fail-operational characteristics. Hence the architec-

ture shown in figure 4.1.

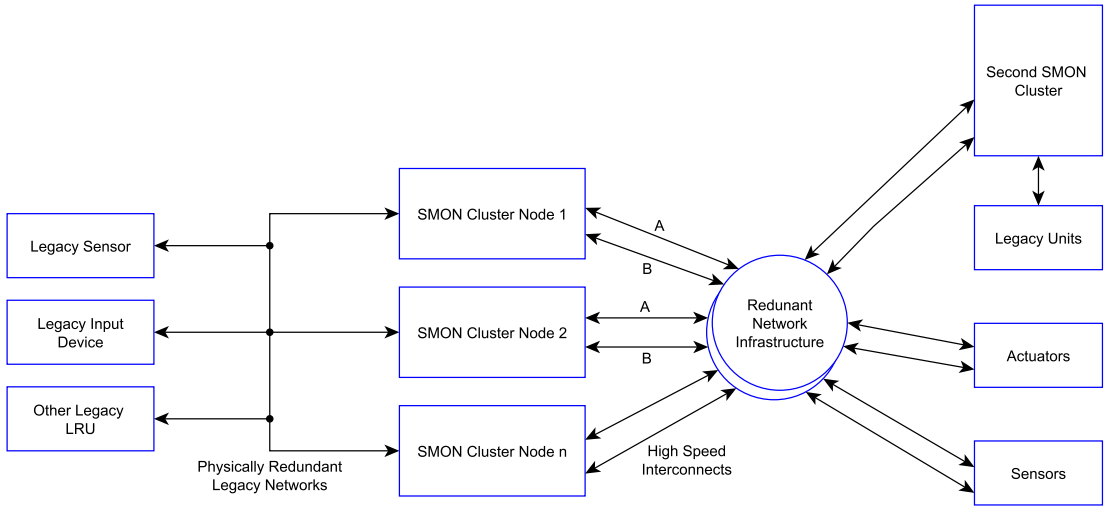

Figure 4.1: *Fictive example architecture for a large scale system integration with SMON-based LRUs as the central control cluster. Zonal redundancy partially shown.*

The system context in which the two architecture variant we will discuss next is shown in figure 4.1. Information about the physical system state is obtained from sensors on the left-hand side and fed into a centralized computing cluster which computes new set point values for the final elements (right-hand side) in a cyclic fashion. In most actual projects, the sensors and input devices will be standard off-the-self components which only offer legacy, low-speed interfaces (like RS232/422, CAN, ARINC429, Flexray, etc.) since they have been in the market for some time and reduce project risk due to certification evidence and in-service experience hours. If newer input devices are used or specially developed, they will most likely feature the same I/O structure as the final elements, e.g. modern high-speed interconnects which come in either point-to-point or switched networks. The latter is not depicted in figure 4.1 since we also want to show the possibility to integrate our new architectures in systems with a relevant legacy, e.g. where overhauls are carried out step-by-step over multiple design generations. If modern interconnects are used, it might be possible to join the sensor and final element network into one large complex, if possible in the application domain and under other system design aspects. For example: How many ports do COTS network switches offer? Are redundancy concepts too complex if sensors and actors are connected on the same physical network? Is fault tolerance and functional degradation for fail-operational use-cases still given, etc.? Note, while depicted as single interconnects, it is likely that physical medium redundancy is given for sensors and final elements, as well as for vital networking elements (switches, couplers, etc.) and yield even more required interfaces on the cluster side. For small numbers of final elements in a switched networking topology, it could be advantageous to move redundant switches into the cluster to save physical units and reduce cabling. However, as we do not want

to discuss very specific full system architectures in this work, which vary vastly even within application domains, we will not engage in further explanations at this point and shift focus towards the cluster, where the example from figure 4.1 defines some interface requirements for the needed I/O.

As far as the interface requirements for our SMON LRU are concerned, we need to address legacy interfaces as well as modern high-speed interconnects. From the example architecture, it can be seen that each LRU shall offer at least four (2 with physical layer redundancy) legacy interfaces. We will settle for CAN and ARINC429 (substitute ARINC429 by RS422/485 or even RS232 for industrial and by LIN or Flexray for automotive applications) in this example. We will also discuss scaling possibilities for more interfaces. Note that we assume that it is safe to combine multiple input interfaces into one I/O handling device inside the LRU, but not redundant hardware interfaces. This will result in at least two fully decoupled input paths, keeping the true hardware and logical redundancy right to the computing element for maximum separation, coverage of possible faults (fault mitigation by redundancy) and freedom of interference (which is always difficult to accomplish). For the final elements, the LRU shall offer at least four 1GBase-T interfaces to connect to a redundantly switched Ethernet network via a duplicated physical layer. Aside from digital data bus interfaces, the LRU is supplied by two redundant and independently generated power rails. For the sake of argument, we assume a 24VDC supply, since different supply voltages (e.g. 12VDC, 28VDC, etc.) do not significantly impact the structure of the power supplies of the LRU. Internally, a stabilized intermediate voltage (either highest voltage needed by components or slightly higher to power localized second- or third-tier regulators). Further requirements, which we will address further when discussing component choices, are for example: Industrial temperature nominal operating range ($-40°C$ to $+85°C$), passively cooled unit, LRU certification to SIL 2 / DAL C, and many more which we will be important in a real-world design like connectors or housing (sealing, pressure resistance, salt/humidity, etc.) but are not addressed here since they are too application specific and can not easily be generalized.

Within the board-level architecture, the most important aspects are the components used, the data paths between them and additional hardware aspects which greatly influence safety aspects such as power and clock domains or component supervision circuitry.

Shown in figure 4.2 is the first implementation variant for the SMON architecture based on COTS multicore SoCs and a re-configurable MPSoC. See the next example for another version without FPGA-based components. Depicted are several parts and subassemblies which we will discuss in the following including their interconnects. Note that we are not promoting devices or manufacturers - if a specific device is mentioned, it should be seen as an example for a range of devices from different manufacturers.

- (1) NOM Processor Subsystem

  The nominal channel processor subsystem consists of several components, required for the NOM processor to function. First is the processor itself (1.1), a local power

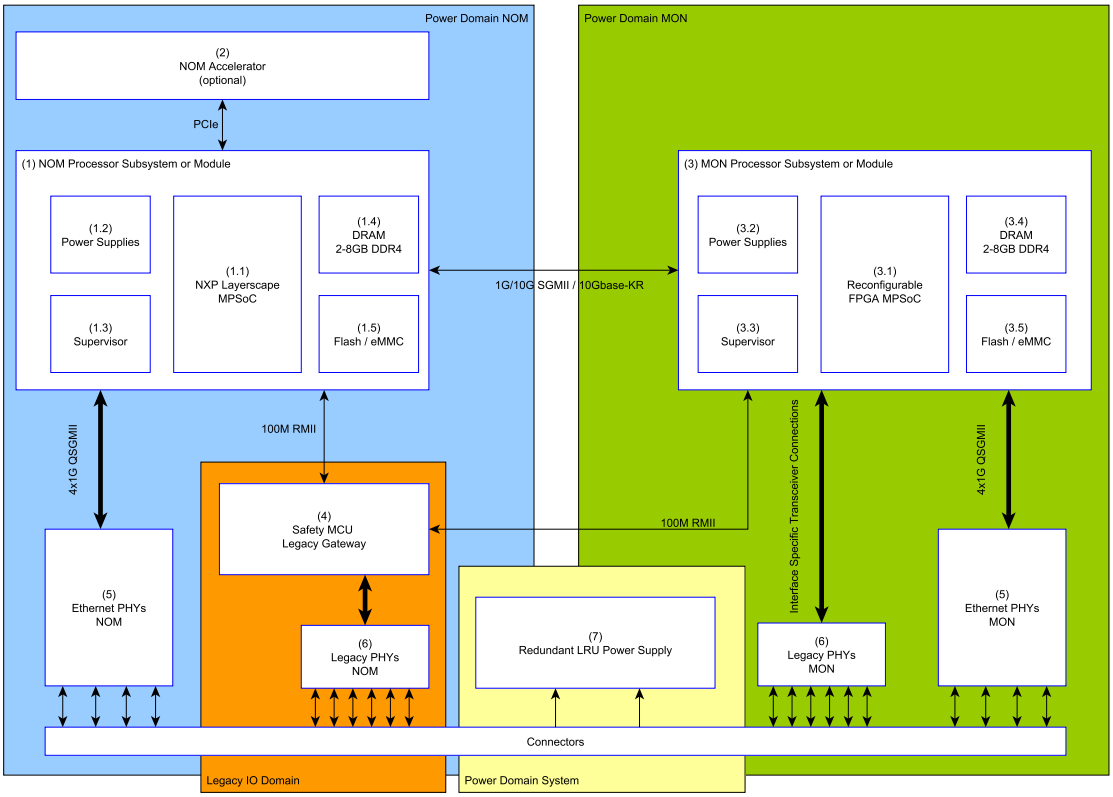

Figure 4.2: *Board-Level Implementation and power domains of the SMON architecture with shared interface processor*

supply (1.2) to generate the voltages / local power domains required to drive the processor core supply, I/O supplies and memories. Various manufacturers offer integrated solution for such tasks in a single package, but designs may also feature separate down-converters for each supply rail. A power and processor supervisor is required for the closely-coupled monitoring of the internal supervision and management core in the MPSoc (1.1). The supervisor could be an integrated microcontroller, conducting the voltage rail monitoring (checking for valid voltage levels) which must be connected via a digital interface to 1.1 in order to perform the high-frequency challenge-response watchdog. For the power rail supervision alone, a programmable controller is not required and different integrated solutions are offered by renown manufacturers. For the challenge-response watchdog functionality however, special power-management-integrated-controllers are necessary, which are rare and mostly intended for special automotive controllers. We therefore strongly advocate the use of a dedicated, low-complexity microcontroller with limited software. Most modern processors also require a precisely timed reset and power sequence to start into operation, where the microcontroller also offers a benefit and allows for a complete coverage of all supervisor tasks in an easily controllable subsystem. In order to store persistent / non-persistent data, the processor subsystem also includes a

volatile DRAM (1.4) and non-volatile flash memory block (1.4). Note that the exact type of memory is highly dependent on the processor decision. Modern devices, at the time of writing, support third and fourth generation dual data rate (DDR) interfaces for non-volatile memory where run-time data and program code is stored [1]. The program code as well as non-volatile data (configurations, parameter sets, tables and databases, historical data on states and faults, etc.) is stored on either controller-less memories or integrated flash memory solutions. Numerous memory types exist today, which are at the time of writing available as flash-based solutions for larger storage sizes. When a pure flash device is used without an integrated controller, the software must take tremendous care of the flash device, since writes and also reads wear out the flash cells over the lifetime of the system. Defective cells must be detected and avoided / repaired by moving data to functioning cells. An over-provision-management is therefore needed. Memory solutions with integrated flash controllers, often sold as eMMC devices (the board-level variant of an SD-Card), include the very challenging cell management and also feature on-line error correction and bad cell management. This does not free the implementer from program code verification via additional image checks, like cyclic, hash-based or cryptographic (authenticity + correctness) checks, since the implementation of the flash supervision is often not known and can therefore not provide sufficient evidence for certification in higher assurance levels.

One possible choice for the main processor in 1.1 are the Layerscape Processor series from NXP Semiconductor. They offer either A53 or A72 ARM Cortex IP-Cores based on the ARMv8 64bit architecture. More importantly, these devices are long-term available (at least 10 years according to the manufacturer) and contain many high-speed interfaces, most prominently the proprietary Ethernet accelerator IP, the data path accelerator architecture (DPAA) in different revisions. Since the processor series has a significant legacy in aerospace, military, industrial and railway applications, they are ideal for critical units. Since the DPAA is microcode driven, custom Ethernet implementations like ARINC-664 [ITC19] or support for the recent time-sensitive network standard additions to regular Ethernet are possible.[2] Furthermore, the Layerscape SoCs (and their predecessors, PowerPC-based SoCs) are supported by all major vendors of pre-certified real-time operating systems with support for time- and space-partitioning, like DDCi DEOS or Windriver VXWorks-Cert. The processor series offers, among others, high-speed SGMII (serial gigabit media independent interface) interfaces, ranging from single up to QSGMII

[1] Even though modern flash has progressed in terms of operating speed, executing programs from flash results in a penalty on program execution time. Therefore, program code is loaded into RAM before/while executing a program.

[2] Both the FAA and EASA are discussing that device microcode may fall under the COTS manufacturer responsibility and is out of scope for DO-178/254 certification, see EASA AMC 20-152A [EAS18b]. The FAA plans AC 00-72 (similar to AMC 20-152A) for 2020

(quad serial gigabit media independent interface), standard RGMII (reduced gigabit media independent interface) and also other high-speed serialized interfaces like PCIe or SATA. The SGMII and QSGMII interfaces are especially useful, as they introduce less electromagnetic interference as standard RGMII interfaces and can be directly connected to most modern physical layer interface ICs (PHYs) or Ethernet switches. From a single, low-power NXP LS1046 processor (see [Sem16] and [Sem19] for further details), we can directly generate for example 6 dedicated Ethernet channels by using one QSGMII PHY and two SGMII PHYs without intermediate circuitry required. Most MPSoCs offer so called high-speed serializer/deserializer (SERDES) lanes, which can be reconfigured in terms of internal routing to different backends (Ethernet, PCIe, SATA, etc.) as needed. To recall the LS1046, at least one PCIe x1 interface is still possible after the described Ethernet assignment on the SERDES lanes. This leaves us with a possibility to either connect even more Ethernet-Peripherals, or for example an accelerator device (2) as depicted in Figure 4.2.

- (2) Optional NOM Accelerator

  In order to meet very special requirements of certain applications, a specific accelerator device may be added as an optional addition to the NOM. Depending on the application this device could for example handle sensor data processing from raw radar or other rf-sensors or provide a direct connection to GNSS or data link communications, as well as satellite uplinks for remote health, status and operations monitoring. Also recent neural network accelerators, or FPGA-based solutions can be attached.

- (3) MON Processor Subsystem

  The monitoring submodule features an equally powerful multicore SoC, compared to the NOM. As before in (1), the subsystem features the same building blocks to supply the processor and its memories, as well as supervising the SoC supply voltages and the internal program flow of the monitoring core with an external watchdog supervisor. There are numerous possible choices for the MON channel. The choice set here for a hardware dissimilar solution is not strictly required for low to medium certification levels, but necessary to reduce the overall number of required LRU on the system level for higher criticalities, starting from SIL3/DALB when fail-operational behavior is required. One possible choice for the monitoring channel are the programmable MPSoC families from Xillinx and Intel Programmable Solutions, which feature not only a hard-ip subsystem with multiple cores and fixed hardware IP cores, but also a large re-programmable FPGA inside a single device. The added cost is rater low in larger quantities for these devices, compared to a host processor + I/O device solution and lowers the overall component count on the monitoring channel. In addition, to entirely different fabrication process, technology

and also the hardware dissimilarity in the I/O modules implemented in the FPGA lead to a truly dissimilar solution in all aspects. Here, the legacy data bus interfaces are implemented inside the re-programmable MPSoC as needed for the application. Additional high speed interfaces (Ethernet in this case) can also be implemented in the FPGA, as far as the MAC-layer or an additional third layer switch is concerned. Most devices also feature some hard IP cores for Ethernet, can or serial data busses which reduce the number of interfaces to be implemented via the FPGA fabric. The variable I/O portion allows for future upgrades to the connecting external data busses which make these devices very flexible and future proof. Fast SerDes lanes are also available in variable numbers through the device families, allowing for direct SGMII or even XFI (10GBase) to internal and external interconnects which benefits the overall system design (no physical layer interfaces required for direct connection MAC-To-MAC) and radiated emission budget.

- (4) Legacy Interface Gateway Processor

  Besides higher speed serial interfaces, low speed legacy links are required to interface to existing (qualified) equipment such as sensor or interface platforms. Since the complex MPSoCs used in the design do not support large amounts of the required simple asynchronous or synchronous interfaces (like SPI or UART) and can not effectively handle them (due to for example the high interrupt overhead compared to smaller MCUs) we added an intermediate gateway layer in form of a dedicated MCU. For today's automotive and industrial embedded system markets, a number of manufacturers already offer pre-qualified devices which are intended to be used in safety applications up to SIL3 or ASIL D. They are, in most cases, lockstep-processors with protected memories and a large suite of supporting evidence such as chip-level FMEDAs, detailed safety manuals and quality evidence regarding the manufacturing. Examples which can be used in the depicted architecture are for example the MPC5748G/MPC5777 (NXP Semiconductor), the TMS/RM-Series (Texas Instruments) or the Aurix-Family (Infineon). They all can be connected via standard ethernet to the NOM and MON processor (for information redundancy). Interfaces such as ARINC-429 are typically connected via special integrated transceivers via serial and parallel chip-level interfaces or are directly implemented in the device, such as ARINC-825 and only require a physical layer transceiver. The example device families feature modern core architectures in the 200-600MHz range, enabling them to handle many concurrent interfaces at once. Since they are not needed for computational tasks in the SMON architecture, all of these families are capable of handling the interface tasks. From a certification perspective, they can be black box tested since they only store and forward information without modification.

- (5) Ethernet Interface

  Ethernet based data busses are the defacto standard in many application domains,

and have also taken over safety applications with adaptions like (Safety Over) Ethercat or ARINC-664. The physical layer, up to 100GBit/s by now, is mostly equal over many application domains, and has been enriched lately by the automotive domain, driving a lower-cost, single twisted pair interface for autonomous driving applications. This new physical layer, while not being galvanically isolated like the 100 / 1000 Base-T standard common before, offers the advantage of reduced wiring cost/weight. Standardization of connectors and cabling is ongoing at the time of writing, but the underlying specification, 100Base-T1 [Com15] and 1000Base-T1 [Com17] is final with speeds up to 10GBit/s in the standardization, see [Com20]. The common MAC interface allows for a flexible connection to the host processor. Our processor choice can either be connected via the parallel media-independent interfaces (MII) or via the more modern serialized interfaces, where the MII communication is modulated on a differential receive/transmit path (SGMII). This interface is especially welcome in industrial and aerospace applications since it greatly reduces radio frequency (RF) emissions originating from high-clocked, parallel interfaces. Furthermore, with higher-end processors one has the opportunity to leverage faster transceivers which allow modulating more than one physical interface on one serialized/deserialized (SERDES) pair, resulting in interface standards like QSGMII or even faster standards for 25, 40 or 100GBit over a single or multiple SERDES pairs. Special quad-port physical layer transceivers are readily available in an industrial temperature range from several manufacturers. Note that, however, the coupling to a single host interface as well as the shared transceiver may not be desirable for some applications which require independence between the individual interfaces. For these use-cases, we recommend the usage of single industrial transceivers. These parts often feature enhanced internal built-in tests and proper loop-back-test means which include the whole front-end up until the external interface. Also note that serialized interfaces require a protocol/speed negotiation phase at power-up, which has to be considered during software and system design. Our board-level architecture allows the use of multi-port transceivers, since the monitoring channel is used as a fully redundant and independent external path. Couples with a redundant switching architecture on the system level, the high-speed interconnect between units to final elements, sensors or interface units is no longer a single point of failure. Especially with classic 100/1000Base-T signaling, with full galvanic isolation by design, a very robust, fully decoupled and very resilient data transmission can be accomplished with COTS components.

- (6) Legacy Interfaces Physical Layer

  This block contains all legacy interfaces which have to be supported for a standalone usage. Still common over the industries are simple serial connections like RS232/RS422/RS485 and others, as well as CAN, FlexRay (mostly automotive) and of course aerospace specific low-speed serial links like ARINC-429 or 1588. Care

should be taken to ensure the interference paths between the individual interfaces are kept low, by employing single-string transceivers for each interface, coupled with separated protection circuitry per channel. Otherwise, physical effects like voltage spikes or lightning might traverse interfaces and impact data communication over the whole system. A dedicated MCU is necessary to bridge the gap between the MPSoC and the legacy interfaces, since these processors do usually not offer the amount of serial interfaces required to drive the transceivers. Also, the interrupt-driven nature of these interfaces disrupts the efficient execution on complex processor cores and is best executed on smaller microcontrollers with a much shorter processing pipeline or field-programmable gate arrays. Very suitable microcontrollers can be found in the automotive line-up of several manufacturers which are either ARM or PowerPC based, with a decent amount of serial and parallel interfaces and an Ethernet (mostly RMII) uplink to the NOM/MON processor. They also offer programmable timer units (for example the TPU in NXP processors or the HPET in Texas Instruments MCUs) which can be used to implement custom handling for interfaces like ARINC-429. The choice between a microcontroller and a field-programmable gate array (FPGA) is mostly up to the implementer and the available resources in terms of established development processes and tooling. Both solutions are equally valid and offer great performance for a large amount of slow legacy interfaces. Note that, depending on the full system architecture, legacy interfaces in the computing cluster may not be necessary, since dedicated gateway or data interface units connected to the high-speed network can abstract the legacy interfaces and provide the protocol traversal. We integrated them nonetheless, so showcase the completeness of our solution, even in the light of legacy requirements.

- (7) Redundant System Power Supply

  The power supply of a combined unit with internal monitoring is of vital importance for the overall design. To properly decouple the failure modes of the NOM and MON, the power and voltage domains of either one must be decoupled, such that a single component failure in one of the domains does not immediately bring the whole system down catastrophically. This can either be reached by a fully redundant design or local point-of-load regulation, with localized small (integrated) power supply solutions. Most modern devices require multiple power rails at different voltages (for the internal core logic, IOs, high-speed interfaces, etc.) so the localized point-of-load concept is elegant and decouples the different domains of the design. These can be found not only between the NOM and MON side, but also within each channel considering the high-speed and legacy interfaces as well as support logic like level translation and the required voltage domain isolation buffering. The whole unit should be supplied by two independent power inputs, originating from different (even better, dissimilar) power sources. This is not only required on the level of the SMON unit, but also to power redundant switches and data

bus gateways or data concentrators throughout the system, to safeguard against a common-mode failure scenario originating from a common, failed power bus. Depending on the quality of the bus and the cabling and connectors used, the two inputs can also be combined onto a single bus without compromising the internal single fault tolerance provided by two input paths for specific applications within a well-known environment. Both power inputs are combined internally by simply or'ing them through adequate means (either diodes or special MOSFET-based designs) after providing common and differential mode decoupling and filtering. This establishes one common internal rail per channel, which is then fed into the local regulators. Care must be taken to provide protection against shorts occurring locally, so intelligent fusing with modern electronic fuses (which measure the current and disable a pass element once a fault has been detected) may be required to satisfy the safety goals. Another way of reducing the complexity and power supply requirements is the usage of COTS processor modules which integrate the NOM/MON along with its memories and power regulators on a small PCB with board-to-board connectors. There, the routing, power capacitors and regulators are fully decoupled from the carrier and isolate failure modes originating from the complex power requirements of modern MPSoCs and memory devices from a simpler, reduced carrier power design. The filtering and protection circuitry at the redundant power inputs is highly dependent on the application domain, but can be seen as a modular block which can be easily redesigned and adopted to different scenarios without modifying the overall design. Short power loss protection is also required for most safety applications and can be accomplished by a providing storage capacitance at the or'ed internal rails. To reduce the size of these capacitors (since batteries often can not be used to temperature, ambient pressure or maintenance constraints), the voltage can be stepped up to dramatically decrease the amount of capacitance required, leading to a smaller sized solution. With processors like the aforementioned LS1046 or FPGA-based MPSoCs, the power budget for the processor assembly should be in the range of 10-15W maximum, while a quad-port QSGMII Ethernet PHY requires around 4-5W max. With additional budget for the legacy interfaces, we end up at around 50W absolute maximum for the whole assembly of an SMON unit. Note that this power usage (also generating heat, which has to be dissipated) is a maximum value which will not be present at long duty cycles since modern processors feature precise power/thermal management and the power draw in the interfaces is highly dependent on the length of wiring, bus load and other factors like number of nodes on a shared bus. Cooling with peak values around 50W of heat generated can be challenging, especially in a fully passive design. The size and type of heat sinking required is dependent on the application domain, and can be augmented by heat-pipes and vapor chambers to provide better heat spreading across a large, finned' area.

- Interconnects and Intra-LRU Data Communication

  As shown in figure 4.2, the different function blocks are interconnected by dedicated links. Between (1) and (5), multiple SGMII or QSGMII links can be used to establish a low-EMI solution for Ethernet connectivity. The accelerator (2) can be connected via PCIe. PCIe is a complex one-to-one data bus with many features, such as link training and speed negotiation, with data transfers established by the master root complex down to peripherals (slaves). Most PCIe root complex implementations feature internal microcode, which made them difficult to certify in aerospace applications in the past. We can therefore not directly recommend the use of PCIe based communication in these applications, if the chip vendor (NXP in our example) is not willing to share implementation details. In industrial and other domains however, the black-channel approach allows using PCIe when a suitable higher-level safety protocol is in place to catch possible transmission errors or loss of communication. It is more likely that higher level system functions implemented with artificial intelligence algorithms will be (or are) implemented in the automotive and industrial domain first, so current accelerators based on GPUs or FPGAs, as well as specialized ASIC accelerators such as the Myriad X or Hailo-8 can readily be connected for AI acceleration with either PCIe or other serial busses such as USB from protocol level 3.0 onwards. The interconnect to the legacy gateway processor (4) is dictated mostly be the safety MCU. Ethernet based solutions are highly proffered, in order to reduce the interface diversity in the final implementation. Even small microcontrollers already offer at least a 100MBit/s interface, which offers low enough latency with minimum sized Ethernet frames. From a latency standpoint, gigabit Ethernet would be preferred, but we have not come across a device supporting more than two 100MBit/s links in the MCU class of devices yet. As a last resort, if only few legacy interfaces are needed, the implementer can resort to UART- or SPI-based communication. The same as above is applicable to the links between the MON (3) and his PHYs (5), with the exception that the legacy IO controllers are connected internally in the re-configurable SoC. The monitoring/redundant connection to (4) can also be established via Ethernet (a switch can be used, since the safety MCU is already a single failure point in the NOM channel and does only suffer from a MTBF perspective) or lower speed UART/SPI communication. Note that it might be required to electrically decouple these signals from the NOM to the MON power domain via galvanic or functional isolation, to prevent power sequencing or fault propagation effects. The cross processor link between (1) and (3) should ideally be as serialized (multi-)gigabit Ethernet link, which is especially suitable due to its packet-based nature and low latency. The serial links in the MII interface family can be easily decoupled and isolated with simple capacitive coupling, which not only helps to simplify the isolation between (1) and (3) but also reduces possible failure modes at this point. The interconnect speed and exact type of transceiver standard is mostly

dictated by the interface availability at both ends. Possible candidates are SGMII or for example X-Base-K (backplane Ethernet). Note that, not included in the drawing are possible connections to supporting circuitry such as power controllers or system supervision processors in the power supply. These can be necessary to ease voltage or power monitoring or provide supply sequencing. From a software perspective, our choice of proven and long available Ethernet technology results in a large reuse between the different interconnects and reduces certification risk. The latter is possible since the Ethernet-based interfaces share a common software stack which can be reused throughout the implementation. The packet-based nature of Ethernet allows an easy integration with tailored safety protocols in higher levels of the stack to further ensure data integrity, or encryption to satisfy security goals.

The presented example for a practical high-level architecture is build upon a re-configurable MPSoC for the monitor channel. This results in additional certification effort, especially in the aerospace domain, where this implies the DO-254 for the programmable logic part of the MON. We could also have chosen to use a re-configurable part for the NOM instead of the MON, but monitoring functions can, depending on the exact use case, sometimes be reduced in the criticality level which might lead to a more realistic effort of the MON side. The now following architecture, presented in figure 4.3, avoids programmable logic devices.

Our second SMON example is very similar to the first one, with one notable difference. The MON channel does no longer feature a SoC with a programmable logic portion. This reflects the fact that developments with the added verification and validation activities for programmable logic devices in the aerospace domain may be impractical from a cost/effort standpoint. Instead, a second, dissimilar multicore processor is used. As a result, the MON is no longer able to address the legacy interfaces on its own and a second legacy gateway microcontroller has to be added in order to satisfy the information redundancy goal for these interfaces. Since the internal information redundancy claim implies that both processors have access to both legacy input paths, the gateway MCUs need to be connected to both the MON and the NOM. All other internal and external interconnects, as well as the power supply considerations are not affected from this design decision. The added gateway MCU with its interface transceivers can be part of the NOM power domain. If the legacy gateway features only one Ethernet-based uplink, the interface to both host processors can be established via a dedicated switch per gateway, since higher level safety protocols ensure data integrity over the internal black channel link.

4.2 DMON Implementations

The system context for the following DMON implementations is very similar to figure 4.1, presented for the SMON implementation. For the DMON case, once should consider only two units, since we already showed in chapter 3 that very low dangerous failure

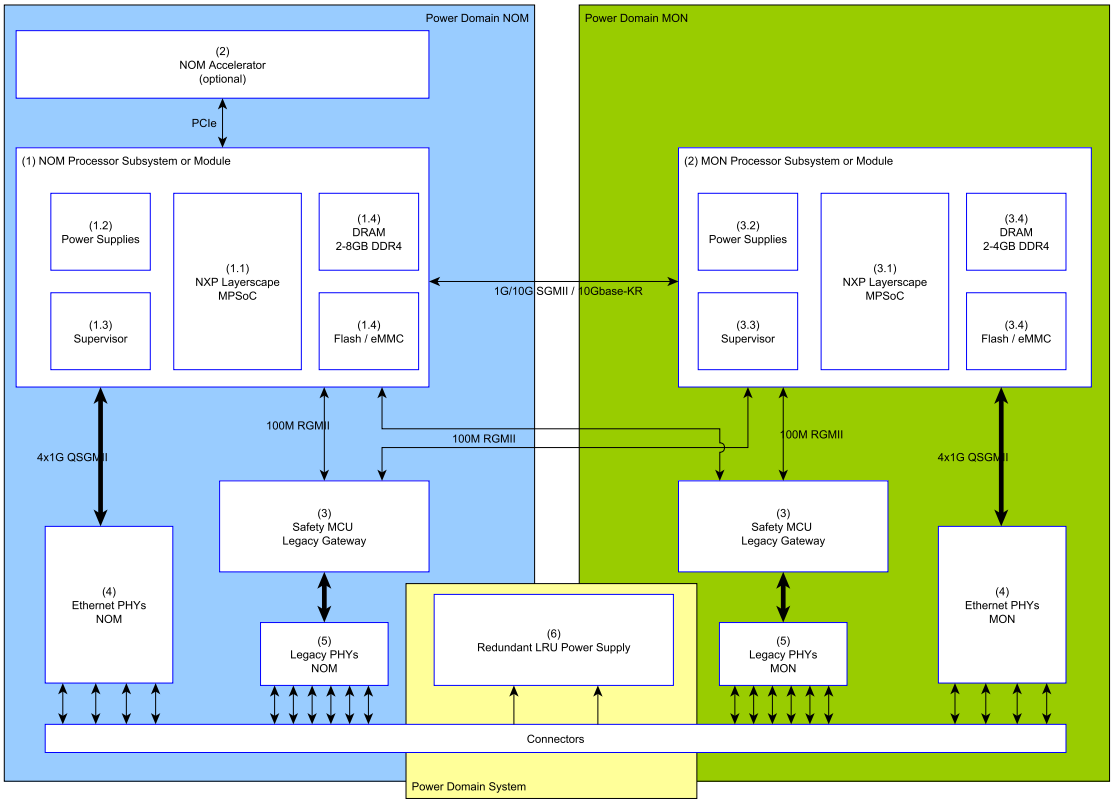

Figure 4.3: *Board-Level Implementation and power domains of the SMON architecture with dedicated legacy IO processors*

rates can be obtained starting from two redundant units. More units can be deployed for zonal safety requirements or availability reasons, but we will skip these considerations at this point, since they only affect the port count of the switched network. All inter-LRU transfer (like cross-channel communication or voting traffic) is routed via the switched high-bandwidth network, resulting in no special cross-channel communication links between the DMON LRUs. All relevant links should be physically redundant in a real world system, which is easily achieved due to the fully redundant high-speed I/O portion of the DMON architecture. This leaves us with at least four high speed connections required to build up two physically redundant, independent links to the redundant network at the LRU. The number of legacy interfaces is very application specific, so we will again present different implementations of the DMON architecture to cope with the high variability.

The first DMON implementation, shown in figure 4.4, is split in three internal sections which also resemble the larger internal power domains of the LRU: One NOM and two duplicated DMON sections. For the internals of the individual building blocks, refer to the above discussion of the SMON architecture, since we will only discuss the differences and specifics for the DMON architecture in the following.

As with the SMON architecture, we focused on standard Ethernet interconnects as the high-speed/high-bandwidth network. Internally, the NOM and DMONs are connected via

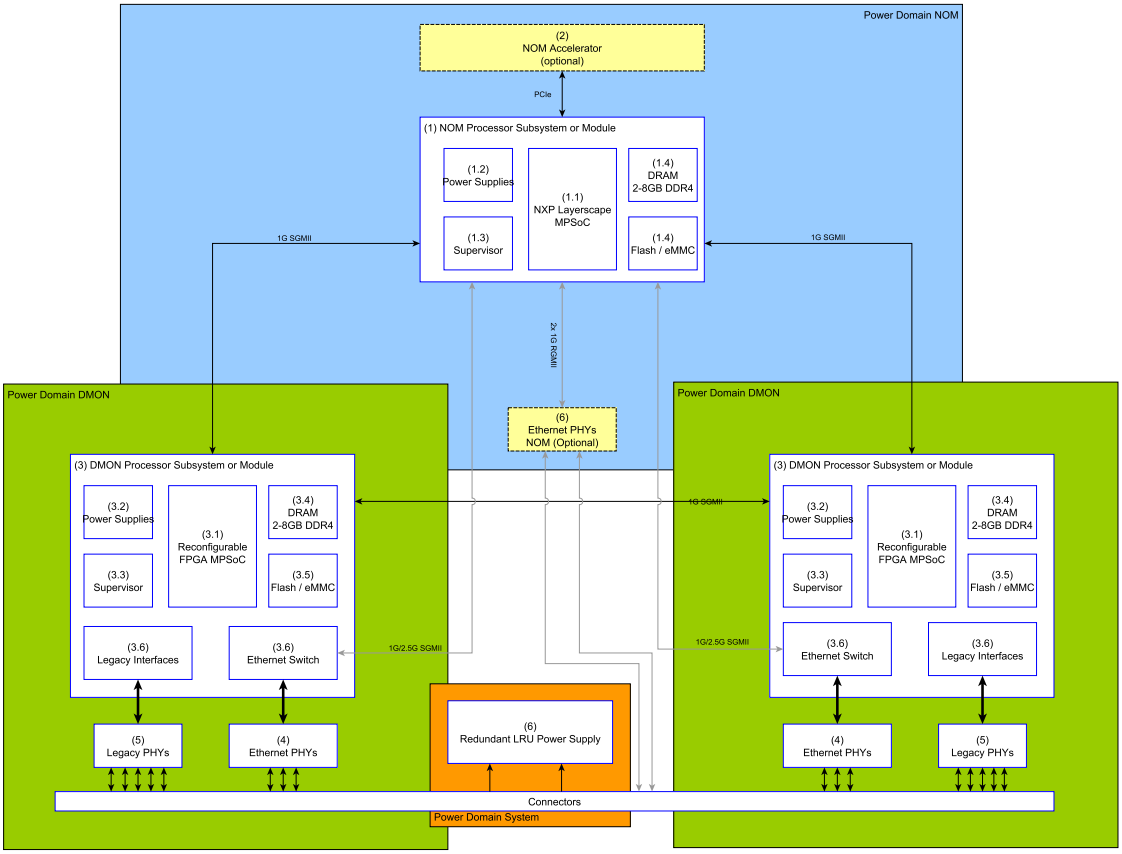

Figure 4.4: *Board-Level Implementation and power domains of the DMON architecture with optional links and programmable SoC DMONs*

dedicated links, which can offer up to 2.5GBit/s depending on the SERDES lane speed of the processors used. A processor like the aforementioned LS1046 is a good choice for the NOM offering sufficient high speed connectivity and a good core count and power efficiency. In larger deployments, SoCs as the LX2160, a 16 core device with eight dual-core clusters is the ideal choice and offers the possibility the memory controller contention (the device offers two dedicated memory controllers) for different impact levels. For the DMON, a LS1028 for NXP Semiconductor would fit into to design and has some unique advantages. It offers very high per-core performance due to the Cortex A72 cores used, and has sufficient cores (two) to have one dedicated I/O and management core, while leaving one free for computing according to our function allocation discussed in chapter 2. It offers four external Ethernet ports, internally switched, with IEEE time sensitive networking support, including recently added features like frame preemption. These ports can be connected to a single four port PHY via QSGMII, leaving additional SERDES lanes open. The second DMON can therefore be interfaced by either PCIe (which offers an easy memory mapped path between the two devices) or SGMII Ethernet, if PCIe is not desirable. The NOM is connected to both DMONs via dedicated, PHY-less links to provide galvanic isolation between the power domains and reduced component count. The

optional accelerator port is also present. As an option, additional Ethernet PHYs can be connected to the NOM, to provide an independent output path to the external network in case both DMONs fail. This is only relevant if functional degradation permits this reduced capability, degraded mode, where only the NOM in a LRU is left, while the remaining LRU is still fully functional. It could then still participate in cluster voting actions, based on the redundant input data of the second LRU. The interface to legacy data busses is provided in each DMON section via a safety MCU (see SMON above). As before, a single 100MBit link easily satisfies the bandwidth needs of the legacy interfaces and is available as a separate peripheral on the LS1028. Since the two DMONs do not share any parts of the I/O path and operate on separated power domains, the interfaces are fully independent (they still reside on the same PCB, but are electrically isolated). The redundant input supply provides stable power to all domains from at least two input sources. If the safety features of the legacy gateway MCU are not needed for a given application, one can choose from a very broad (and also low-cost) portfolio from different manufacturers, since we require only a single Ethernet uplink. Due to the eight external Ethernet interfaces (optionally ten), the unit can be connected in a dual-redundant fashion to the switched network, while still leaving four ports free for future extension, or a dedicated cross-channel link (also physically redundant, and connected to both DMON for internal fault protection and redundancy) to remove load from the interfaces, leaving more bandwidth and lowering latencies for I/O messaging. If the optional direct Ethernet interfaces to the NOM are used, it is wise to connect them directly to one of the free DMON interfaces each. Together with a physically redundant cross channel link, all four interfaces are then occupied on both LRUs.

Shown in figure 4.5 is the second architecture variant for DMON, based on re-configurable FPGA SoCs. Notable differences compared to the first DMON architecture are of course the DMON subsystems. As before with the SMON, the FPGA SoCs offer the advantage of combining the I/O portion which was previously externally to the devices into the monitoring channels. The integrated Ethernet peripheral can either be a managed switch (we depicted a four-port switch, but this is only limited by the logic resources of the device and I/O pins) or application specific IP for standards like EtherCAT, ARINC-664 or Time-Sensitive-Networking (TSN). Since most re-configurable SoCs also feature hard-IP peripherals (in silicon, not in the FPGA portion of the device) associated to the hard-IP multicore section, we routed two Ethernet connections from the NOM to each DMON. One connection is mandatory, the second one, attached to the soft IP Ethernet section, is optional and can also be replaced with dedicated Ethernet channels connected to the NOM as before in the first implementation variant. This second link offers the possibility to bypass the DMONs and establish a direct path to the NOM for NOM-to-NOM inter-LRU traffic. Note that the FPGA-fabric might not be available in case the hard-IP section reboots or the device power-cycles. Redundancy claims based on the second downlink might therefore not be an option. For these considerations, the independent

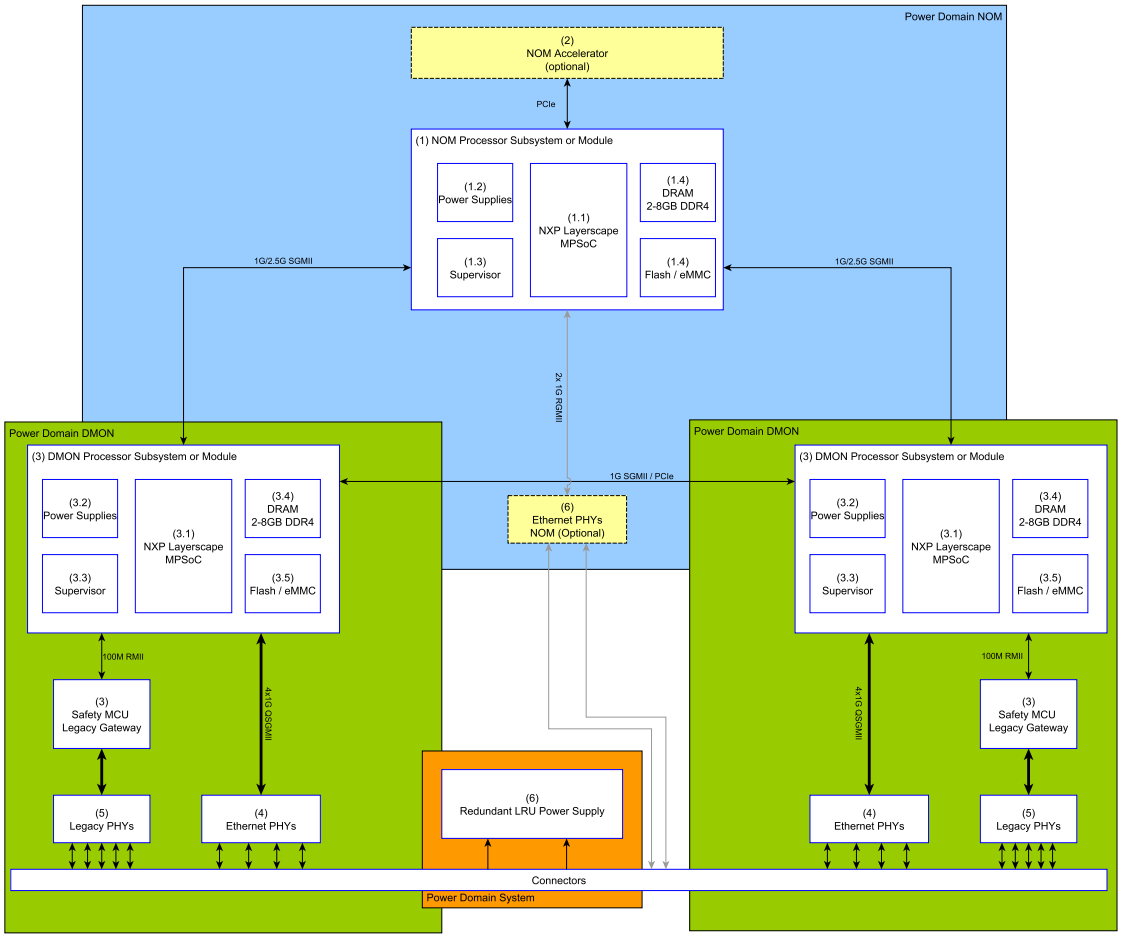

Figure 4.5: *Board-Level Implementation and power domains of the DMON architecture with non-programmable DMON devices*

output path via dedicated PHYs from the NOM should be the preferred solution if additional functional degradation paths need to be established in the system. Note that, as with the first DMON architecture, we still only need three power domains. Separating the I/O portion (bus specific PHYs) would not yield additional failure resilience, since a failure in the DMON computing side would also render the interfaces useless and vice-versa. We therefore only electrically separated the NOM and DMONs via the convenient SGMII interfaces which can be decoupled via simple capacitive coupling on the data lines. Also note that this second implementation might be worse from a cost perspective than the first DMON implementation, since re-configurable FPGA SoCs tend to be expensive compared to standard MPUs and MCUs, while only offering a slight reduction of MTBF when the additional interface gateway MCU can be removed compared to the first DMON implementation. Especially the Ethernet soft-IP cores might drive FPGA size. Aside from these specifics, this implementation is largely equal to the first DMON variant with the building blocks already explained in greater detail at the SMON implementation descriptions. Refer to the above sections for additional information.

4.3 Common Considerations for SMON and DMON

In both implementation examples we mentioned the possibility of so-called System-on-Modules (SoM) for both the MON and NOM sub-assemblies in the design. These COTS components with the MPSoC, volatile and non-volatile memories, localized power supplies (in the form of integrated power management controllers or dedicated solutions) offer some advantages but also disadvantages compared to a more traditional flat design with all components integrated on a single PCB:

- Partial Fault Isolation

  As the SoM is lifted from the main PCB with one or more board-to-board connectors, the localized power domains required for the SoC, Memories, etc. are physically separated from the main PCB, together with the traces between the SoM components. This provides some degree of fault isolation against localized shorts or supply failures, in the case the base board design can cope with a shorted module and disable it or limit the current safely. The component count around the SoC and also the memories is very high, which is mainly driven by decoupling capacitors and support circuitry required by the complex semiconductors. 300-400 components on a 60x80mm area are not uncommon in today's designs and drive reliability figures due to the amount of capacitors and resistors required, while the complex semiconductors are only in rare cases the source of low MTBF figures. In the event of such a catastrophic failure on the SoM, the base board may continue operation unaffected and still operate the remaining NOM/MON with it's interfaces for additional availability (if permitted by the application and system design). A faulty module can also be swapped during maintenance, which saves the base board and other (expensive) components which reduces waste and maintenance cost.

- Ability to Up/Downgrade

  If a certain module family or standard has been chosen with different SoCs in the same physical form-factor and connector pinout, the module can be used to up- or downgrade the processor sub-assembly. This is especially useful during development, where the base board can be designed and manufactured while the final processing platform or the memory requirements are unknown for new platforms. The project can then start on a larger, memory rich platform and later scale down to meet the application requirements with a simpler or more cost-effective SoM. Likewise, the platform can be upgraded during its life-cycle with more computing power and/or increased memory when application demands change or the LRU is used in a different vehicle. Note that these up/downgrades always require re-qualification of the LRU as a whole, but these regressions are limited to the interaction between carrier and SoM, when the SoM is already qualified or COTS with adequate documentation. The re-certification efforts limit the usefulness of such hardware changes in

the life-cycle of the unit, but may also be necessary due to component obsolescence or supplier changes. The latter may be a positive aspect for itself, when different suppliers can be qualified as second sources for a standardized module.

- Thermal Decoupling

  If the SoC is the main heat dissipating component in the design, a SoM provides an easy measure to thermally decouple the remainder of the base board from this heat source. While the SoC can be thermally connected to a cooling solution, this might not be necessary for other devices on the base board which either use the PCB as a heat spreader (and heat sink) or require no cooling at all over the temperature envelope. The reduced thermal stress on other components may lead to higher MTBF figures and reduces the overall stress on the PCB and all components. Note that this only applies with a sufficient air gap between the base board and the SoM, as it is the case with most today's module standards. Nevertheless, some modules, especially integrated sensor units or RF transceiver units are sold on modules which are directly soldered to the carrier PCB. In these cases, a limited thermal decoupling is the result of the direct contact between the module PCB and the carrier board.

- PCB technology decoupling

  Modern SoCs are available in ball-grid-array (BGA) packages or land-grid-array (LGA) packages which are very fine pitched components. The required circuit board tolerances are therefore very elevated and might in some instances require laser drilled micro vias or via-in-pad technology. This not only drives cost of the PCB if the area is large and a mixed design (with lots of different matched impedance also for external interfaces such as CAN and Ethernet and the intermediate PHYs are required) but also failure rate, manufacturing yield and design complexity as well was quality assurance cost. Constraining the complexity in routing to the SoM enables a much simpler carrier board, since all other components are available in more forgiving packages with larger ball pitch (in case of BGAs) and reduced pad density.

- Module Connector(s)

  Since the module is COTS, the carrier must include a suitable connector, for either a mating part or an edge-card connection. The additional connector has a negative impact on signal integrity and may lead to signal degradation on higher speed interfaces such as PCIe or SERDES driven Ethernet or other high speed / parallel interfaces.

- Additional Cost

  As with most electronic COTS components and assemblies, their pricing drastically increases in small production volumes. This might not be noteworthy for aerospace

applications which always have to deal with high per-unit prices due to low order volumes, but especially in larger scale industrial projects the additional cost of a module-based solution is often unbearable and a single-board flat design is proffered for better cost scalability and reduced cost/manufacturing effort in high volume scenarios.

- Increased mechanical complexity and assembly height

  In the light of a high-vibration environment, the stacked board-to-board construction may be undesirable. Board-to-Board connectors are usually not tested against special requirements found in some domains (like rotary wing or other high vibration and shock applications like earth moving machinery) and require project-by-project qualification. This increases cost and project risk. If raised vibration requirements have not been taken into account during the design phase of the SoM, the physical mounting may not be sufficient and lead to physical damage to the SoM and/or the carrier board. Increased height resulting from the stacked SoM may be impractical in space constrained applications. However, in most designs, the external connectors drive LRU height and size in at least one (or even two) dimensions.

Whether a SoM is employed as the NOM or MON sub-assembly, the monitoring and supervision functions inside the LRU are import for the overall safety of the design. From a hardware-related safety perspective, localized monitoring of all power rails of the NOM and MON processor, as well as their supporting circuitry (DRAM voltages, I/O voltages, etc.) ensures that nominal operating conditions are met. Combined with a watchdog circuit, a monitoring device (either a dedicated microcontroller, system base chip or integrated power management controller) can trigger a local reset for the subsystem violating its operating conditions. Note that such devices should also address safety and offer a safety manual or at least quality data for failure rates and manufacturing process in order to support a certification effort of the software executing on them. Certification levels of these monitoring functions can be lower than the actual safety function, since they represent diagnostic functions only and do not actively participate in the safety function. In practice, a single small microcontroller (plenty of options exist, automotive qualification is preferred as stated before) or even a programmable gate array device (when reset and power sequencing requires very stringent timing) is located close to each larger SoC or on the integrated module, using mostly digital I/O, analog conversion channels and one or two communication interfaces for exchanging status information with the host processor or in-field firmware updates. While the local point-of-load rails for each major computing element are monitored in proximity, a global LRU supervisor function should be implemented. It controls the start-up, restart and shutdown process of the different sub-assemblies, monitors the external power input or manages redundant LRU power supplies. If for example the NOM has failed and should be taken offline or restarted, the MON would flag a restart request to the supervisor which then carries out the steps

required for a controlled restart or shutdown. Due to the degree of redundancy on the system level, a failed supervisor restarting good devices is an availability issue, but does not afflict safety. Nonetheless, the software and functions in the supervisor should be kept lean, to ease certification. Whether the supervisor function is allocated to a single device or distributed over the different local management MCUs is a design decision. If no central supervisor is used, all localized supervisors (including one for the LRU power supply) can be connected via a common bus (like CAN, I2C, RS485, etc.) with the SoCs in order to facilitate the required functions. If a dedicated MCU is included, it can control the local supervisors by simple digital I/Os or UART/I2C connections in more sophisticated designs. From a failure-mode point of view, the distributed approach should be preferred, since it does not introduce a single point of failure for the LRU (which should not be critical due to system architecture, but might introduce availability issues). If a single device shall nonetheless be used, we urge for a safety MCU with dedicated safety documentation or pre-qualification, in order to present accurate quantitative data for device failures and failure modes for the LRU FMEDA. Note that one could also hot-restart the supervisor in operation which leads to a brief unavailability of diagnostic measures if the LRU design permits such a behavior (e.g. when only digital interfaces are used for downstream connectivity and communication watchdog windows are long enough). Only permanent hardware failures can then lead to supervision failure which are more rare than random hardware or software faults. Also note that in designs where a safety-rated MCU is already present as a legacy-gateway processor, one can reuse this asset for LRU supervision, which is, by design, already redundant for NOM/MON and therefore optimal for such a task.

One additional design consideration, more relevant for the SMON than for the DMON architecture, is the physical separation of the NOM and MON channels. If each channel is self-contained, e.g. includes all local supervision and relevant I/O circuitry as well as a full power supply, one can fully split both channels into two dedicated circuit board assemblies. These "fat channels" (in comparison to "slim channels" where both are logically independent but physically located on the same board for common power supply, etc.) can be located in the same housing or even further isolated in dedicated boxes. This allows for an increased degree of zonal safety and fault isolation in large scale systems with hazards such as fire or high EMI concerns. In most applications however, the additional cost and system complexity might contradict the benefits and lead to a single, integrated unit for space and cost savings. LRUs based on the DMON architecture are hard to split, due to the internal triplex design and should be handled as a single physical unit in one enclosure.

5

Conclusion

In order to conclude this work, we will first provide a summary regarding the main contributions and finish with an outlook for future work.

5.1 Summary

- Requirements

  As a solid base for this whole work, we first derived a set of requirements for our unit (line replaceable unit) architectures, given a small and large system use case and additional considerations for future system generations. For current systems operated by humans with a mild degree of automation, these requirements are very ambitious. However, they will be present in future systems, where human operators become only system users, without the deep knowledge or skill set to operate the system or mitigate critical failure scenarios.

- Function Classification

  Next, we developed a novel system function classification scheme based on the safety or failure impact. The approach is domain agnostic and can therefore be applied in many application fields. It is vital for future generations of safety critical, cyber-physical systems and removes the cut between the system and hardware and software level that previously existed in most safety projects. While the impact level does not directly translate to the currently used safety integrity or design assurance level notation, it does not break current certification work flows but rather extends them substantially, in order to provide the means necessary to allocate critical functions into future high performance system architectures with many possible physical compute locations spread over many physical devices.

- Unit architectures

  Another core contribution of this work are two novel unit architectures centered around multicore system on chip devices. Since real world systems differ greatly

in their requirements and constraints like space and weight, two architectures were necessary to cover most use cases for future highly automated or autonomous systems with elevated computational needs. Both architectures lead to a unified, homogeneous computing platform, which is able to integrate legacy components and communication standards, as well as address current and future high performance needs in terms of compute performance, communication bandwidth and latency requirements. The centralized monitoring architecture is close to current designs with a single monitoring facility with information redundancy to cancel failure modes associated with highly complex commercial-off-the-shelf devices without prior safety evidence. Therefore, any processor may be used as the nominal or monitoring channel, with some constraints addressed in the safety discussion. The distributed monitoring architecture enhances the centralized monitor with multiple monitoring and gateway devices, and make it suitable for stand-alone or small cluster operation. Both architectures scale well for different application needs and reach very low residual critical failure modes with moderate degrees of redundancy. Especially the second architecture is ideal for very high integrity systems, which rely on a fail-operational control system to prevent catastrophic events. For both architectures, we also covered possible function allocation schemes based on our impact level approach and discussed how our initial requirements were met by the architecture, as well as their possible use case in many scenarios. The unique mixture of inter and intra-device monitoring builds a very strong defense against known and unknown failure modes (due to the device complexity, hidden features and the unknown internal design). The software aspect also plays a significant role in both architectures, since every aspect of the system, from top level requirements down to the operating system memory management have to work hand in hand and can no longer be treated as separate entities like in classical unit architectures. We matched existing work which addresses major software shortcomings in the management of complex multicore system on chip devices to our architectures and integrated these remarkable concepts together with our novel system level and architecture design. Parts of the high level software architecture reflect all major parts, like system functions, communication stacks, monitoring facilities (and their interaction with the rest of the system) as well as isolation and separation concerns and hardware supervision by software.

- Certification

  In the third chapter, we conducted a safety analysis of both architectures and derived commonly used fault tree and Markov models, in order to justify the design decisions made. Along with special, domain dependent considerations for the usage of complex off the shelf devices without (or with very little) design evidence associated, we provided a path to certification for aerospace and industrial applications based on their current applicable standards and guidelines. This work is therefore a

strong contribution to the ongoing efforts for new ways to handle the ever increasing complexity of off the shelf components in safety critical applications.

- Example Implementations

  Lastly, we provided example implementations for each architecture. Based on actual devices available today, these implementation examples can be directly translated to real units which comply to our stated requirements. We show that real-world units are feasible today and also discuss additional topics like thermal management and mechanical integration variants to provide the full picture for future development projects. The technology readiness level can be seen as very high. While the availability of a pre-certified software solution which implements the required architecture and components is not yet given, this work has addressed most system, hardware and certification efforts to a great extent.

5.2 Future Work

For future activities derived from this work, we identified the following topics and activities:

- Prototype Platform

  Start to work on a prototype platform based on currently available devices and establish a research prototype for software development and platform integration tasks. The prototype should be built around the distributed monitoring architecture, since it is more suitable for small scale use cases and also possible other application domains not touched in this work like automotive or railway. It should be coupled to a real world application or seriously funded research project, due to the substantial cost and effort involved.

- Development of a unified software platform

  Another possibility could consist of developing the concepts and architectures for a unified software platform to handle the board management and operating system interface in the form of a middleware or redundancy framework. This framework could control the software defined system functions while respecting their impact level and fulfill the relocation and redundancy management tasks required, as well as managing redundancy resources throughout the full system with a distributed algorithm.

- SOTIF and Certification of preexisting complex software or operating systems

  In recent time, a new theory of system certification arose called "Safety Of The Intended Functionality (SOTIF)" [3219, Abstract] with the goal to provide means

to certify complex algorithms and system functions where not every possible permutation of input/output data can be tested beforehand (as is true for AI-based functions). Future work could study the implications of SOTIF for the concepts presented in this work, for commercial-off-the-shelf devices, and implications on software certification of the operating system involved. This could entail the usage of COTS operating systems such as GNU/Linux in safety critical applications under this new form of handling residual uncertainties and failure modes.

- Impact level classification

  During discussions with certification experts and notified bodies, the impact level classification was seen as an important building block for future systems. With a future work, one could establish this theory well enough to be included in future standards and guidelines for general use in industry.

Appendix A

Sequence Diagrams

A.1 On-Device Monitoring

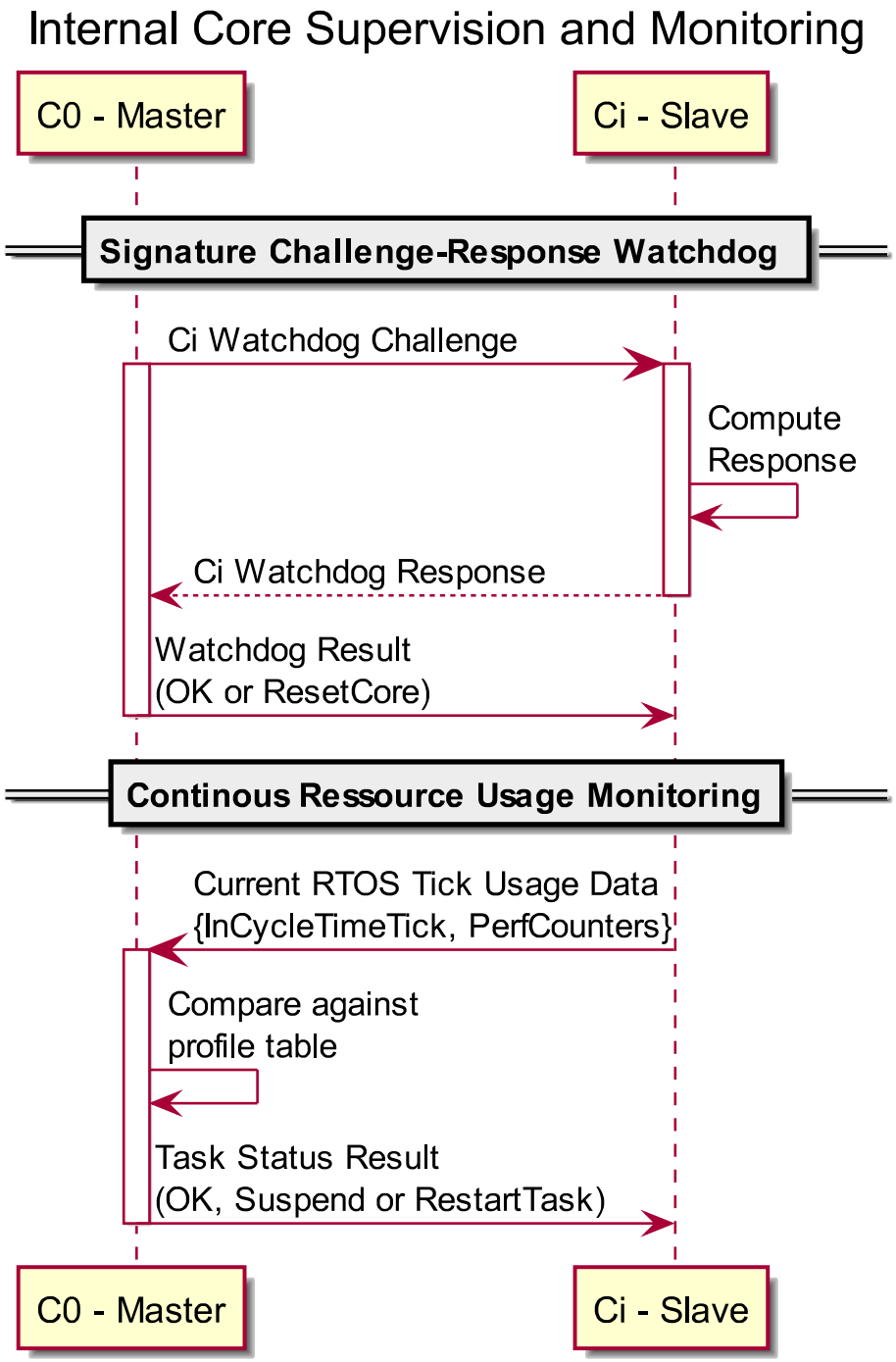

Figure A.1: *Overview of the device internal validation on the COTS multicore NOM Or MON*

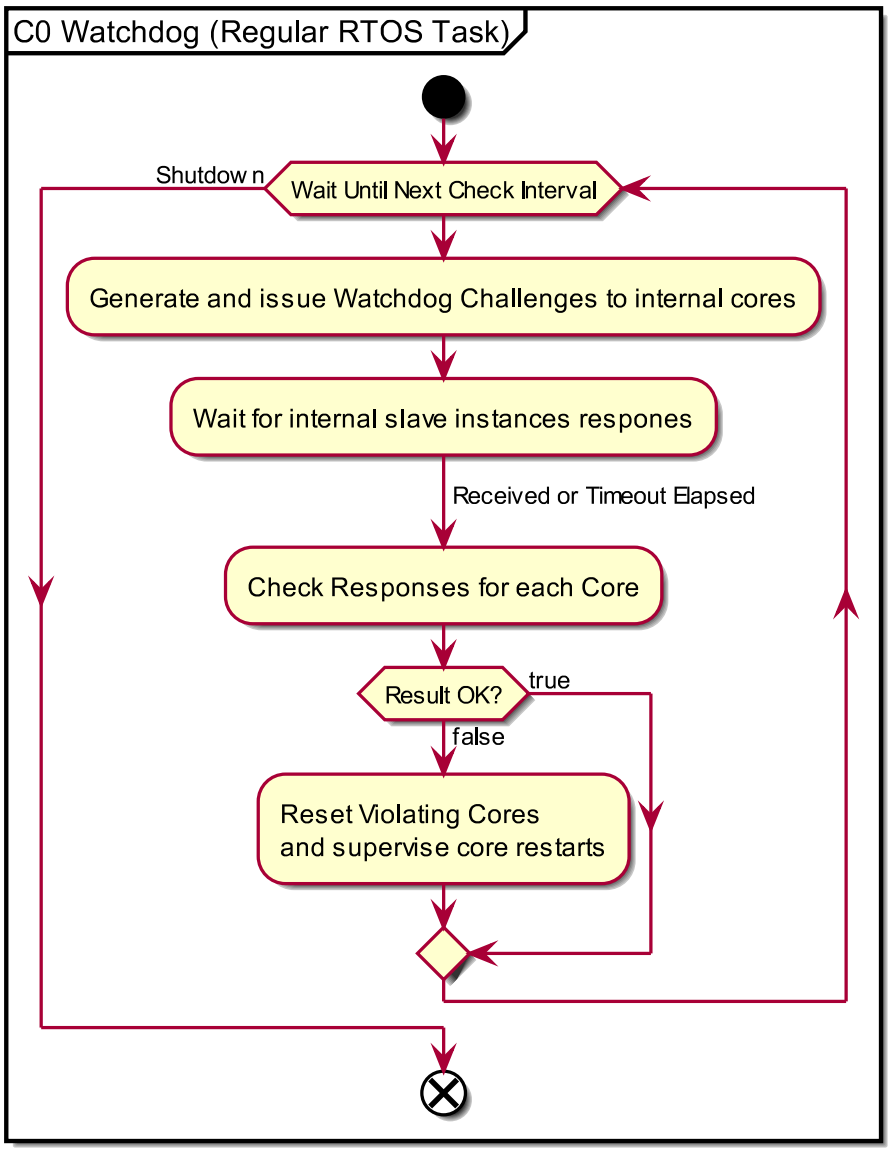

Figure A.2: *Inter-core watchdog flow for master core 0, periodic task in the local RTOS instance*

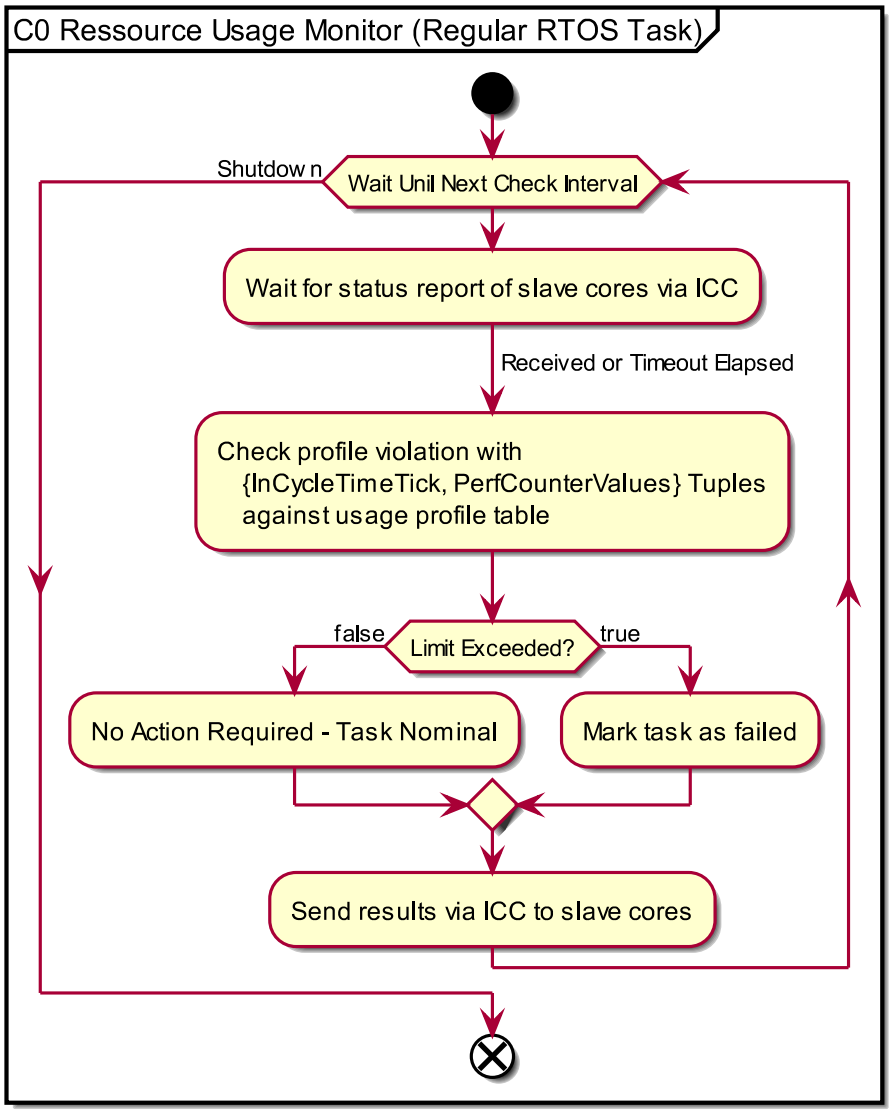

Figure A.3: *Inter-core resource monitoring master on core 0, periodic task in the local RTOS instance to validate resource bounds*

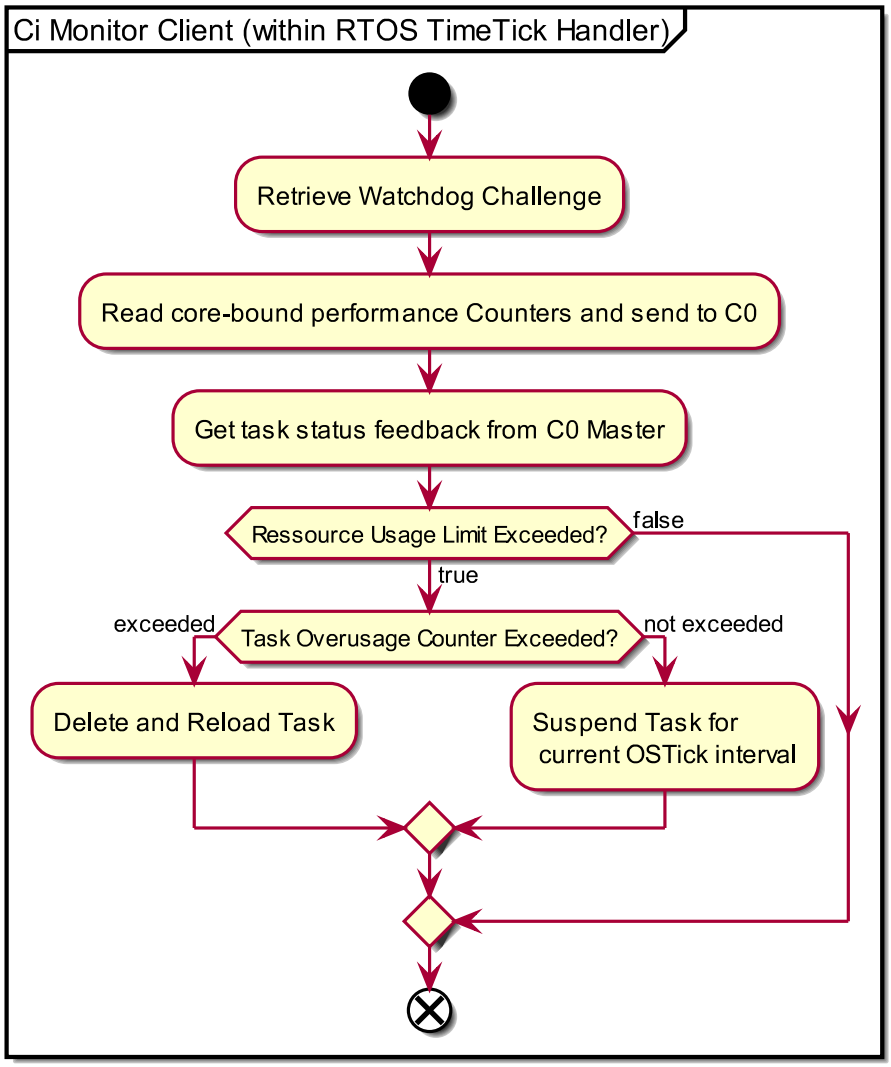

Figure A.4: *Internal device monitoring client flow running on each non-master core in the COTS device*

A.2 NOM/MON Communication

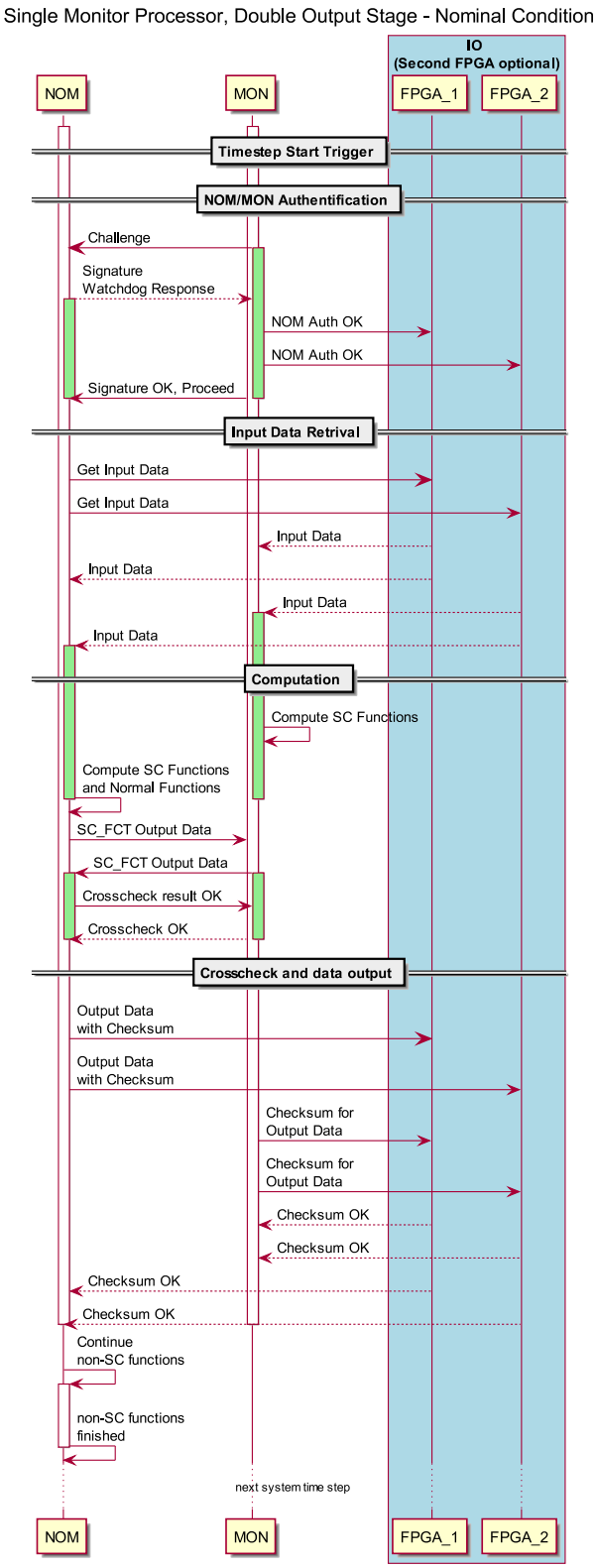

Figure A.5: *Nominal board-level interaction for one system time step of a SMON cluster LRU with FPGA-based IO stages*

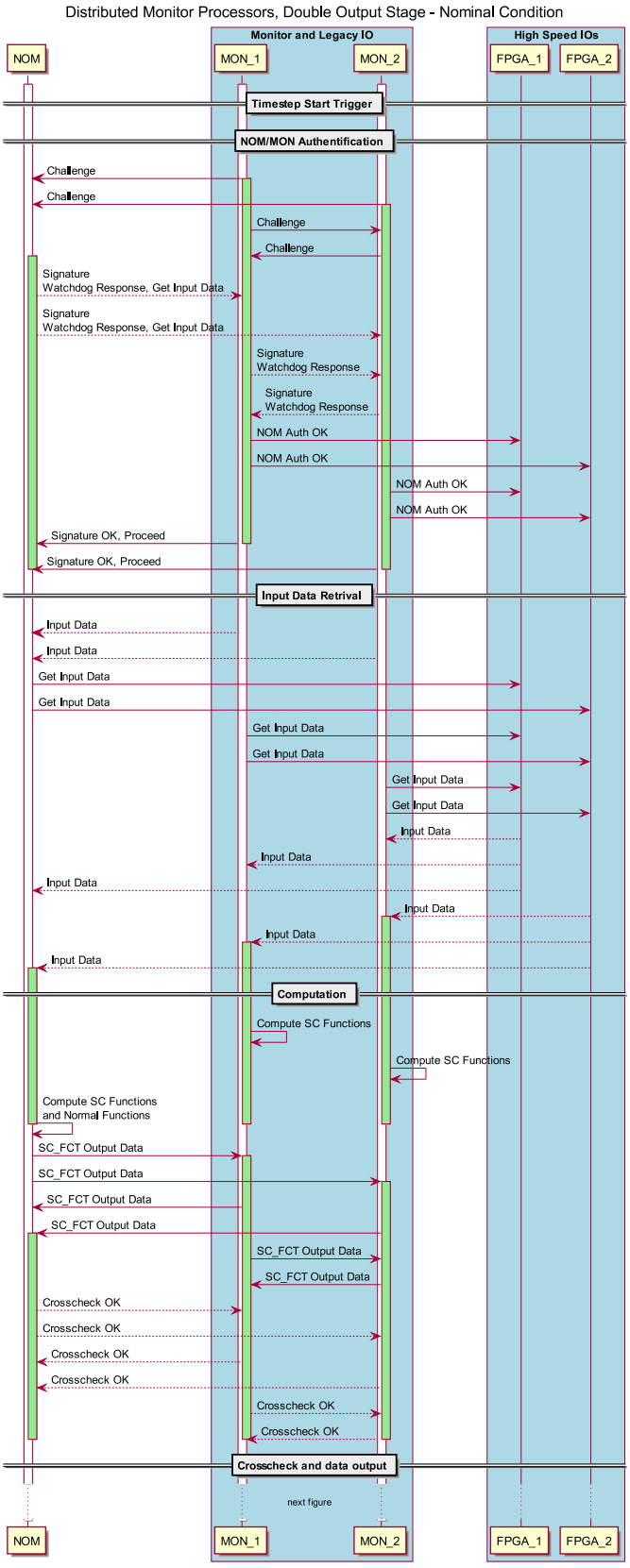

Figure A.6: *Nominal board-level interaction for one system time step of a DMON cluster LRU with FPGA-based IO stages - figure is split for readability, part 1 of 2*

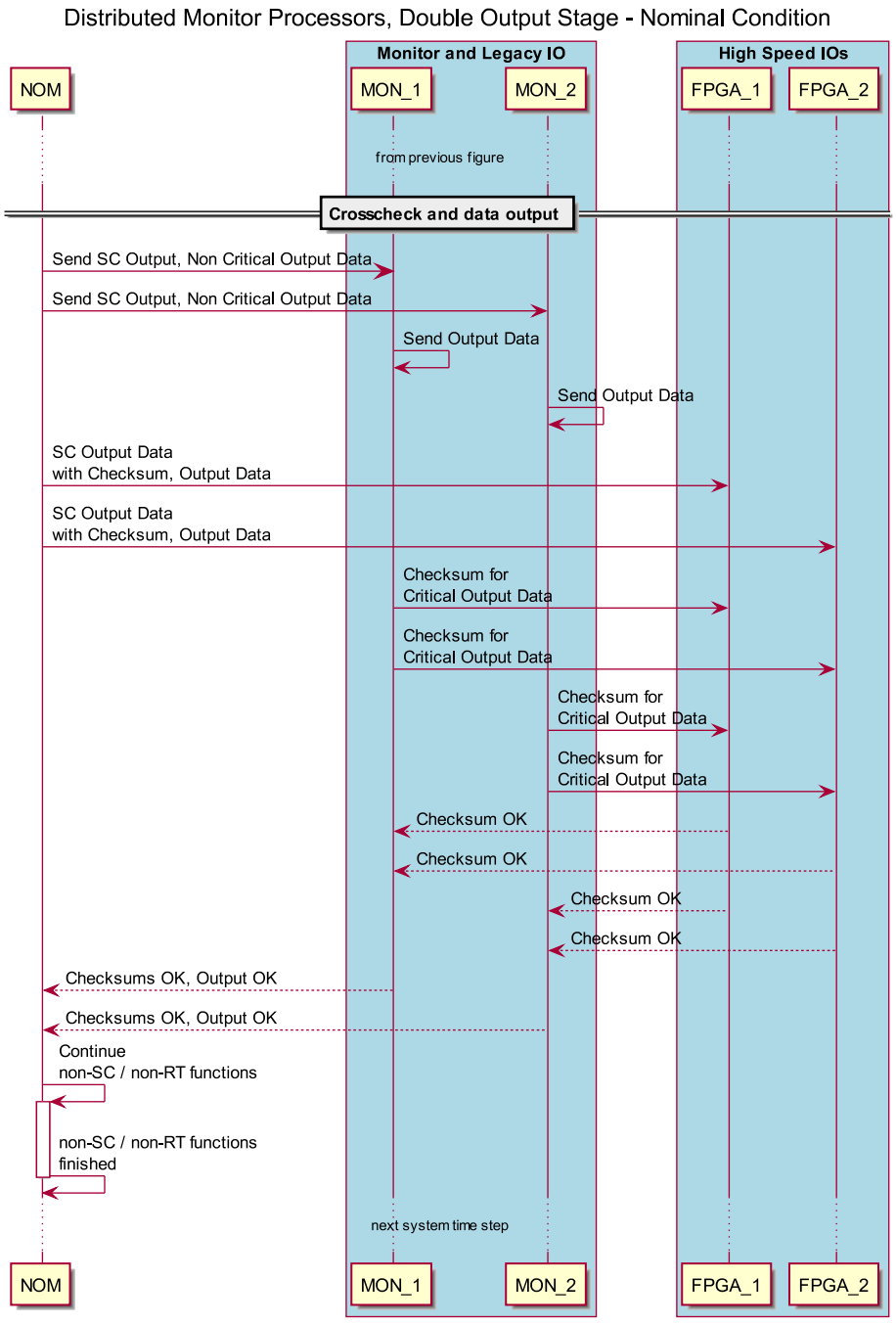

Figure A.7: *Nominal board-level interaction for one system time step of a DMON cluster LRU with FPGA-based IO stages - part 2 of 2*

A.3 NOM/MON Fault Mitigation

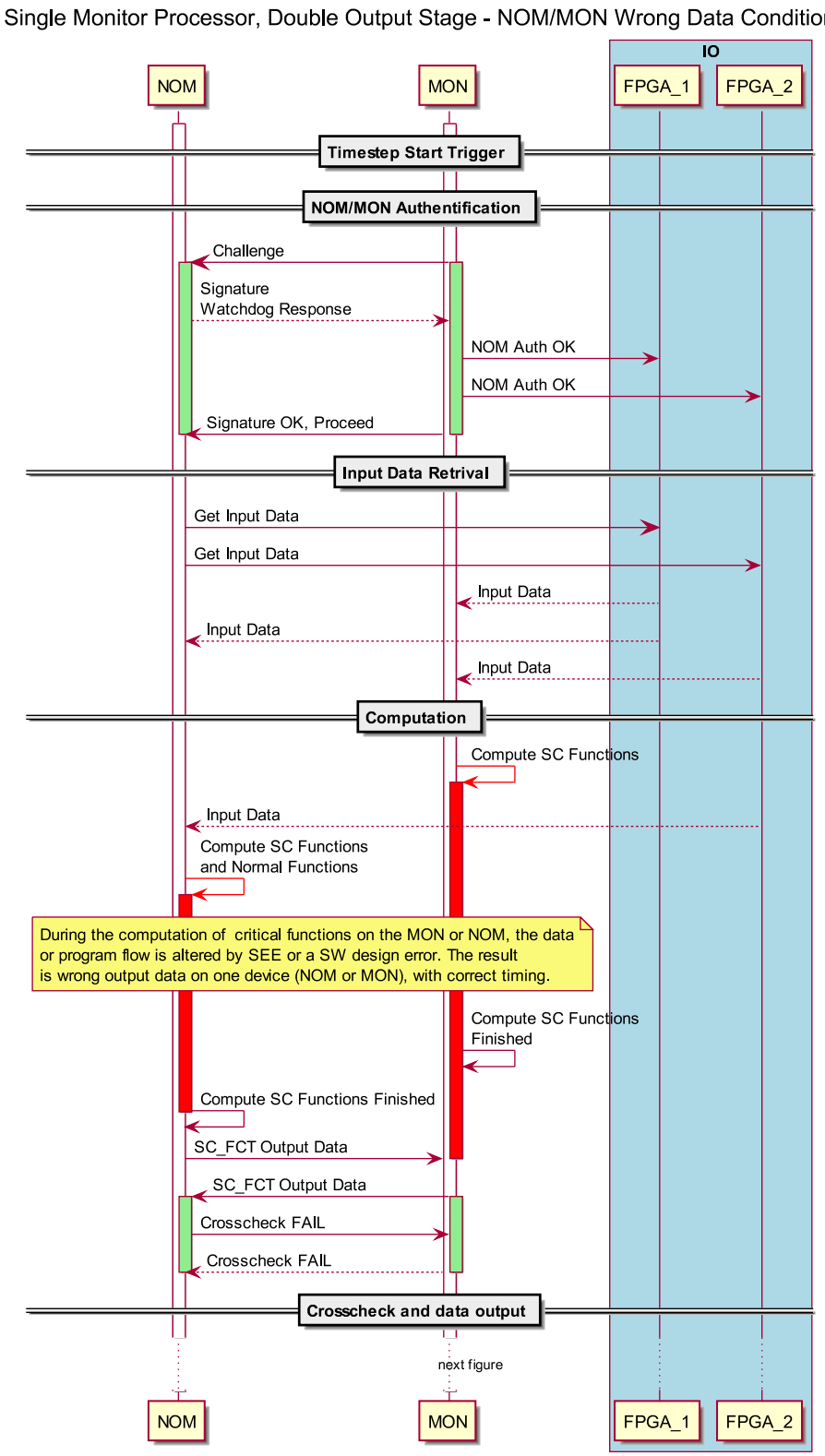

Figure A.8: *Example for fault unmasking and reaction in a SMON cluster LRU - figure is split for readability, part 1 of 2*

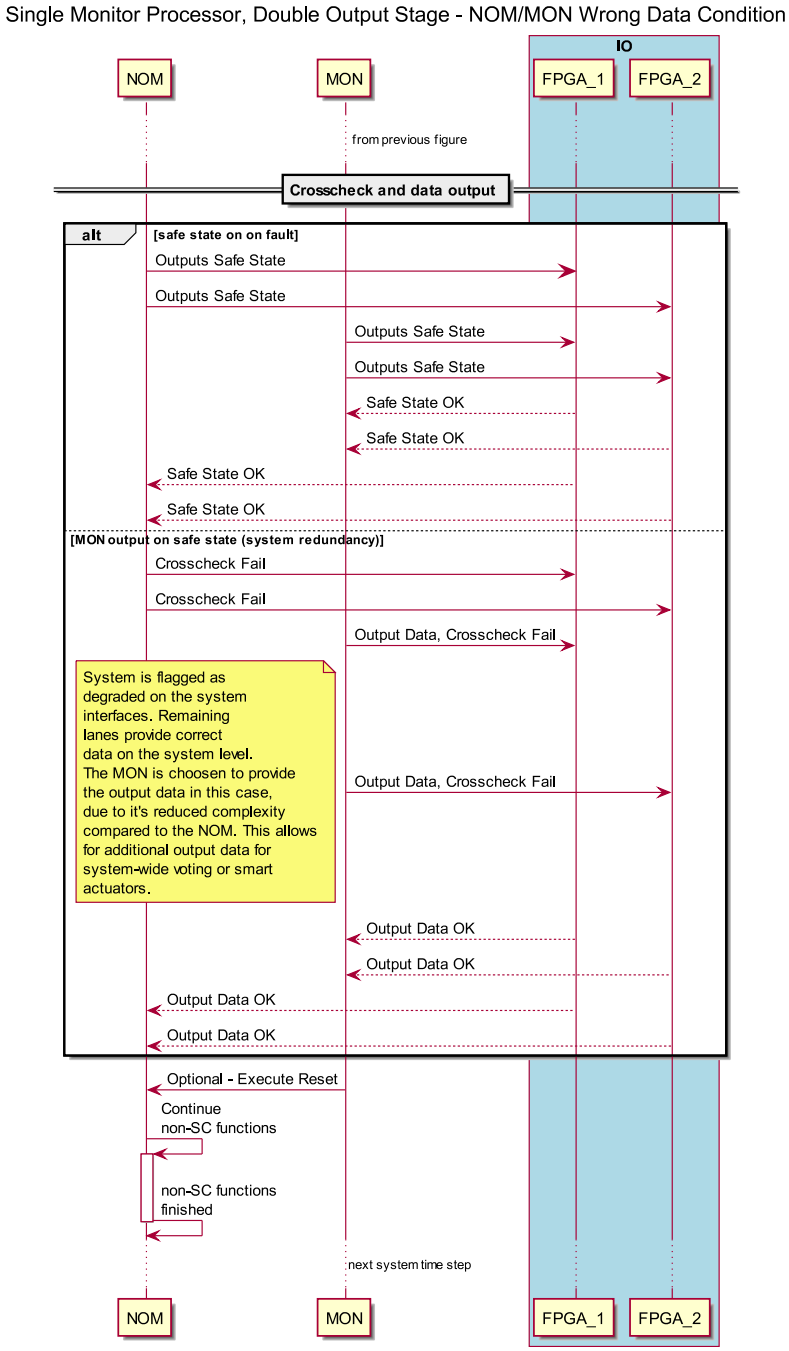

Figure A.9: *Example for fault unmasking and reaction in a SMON cluster LRU - part 2 of 2*

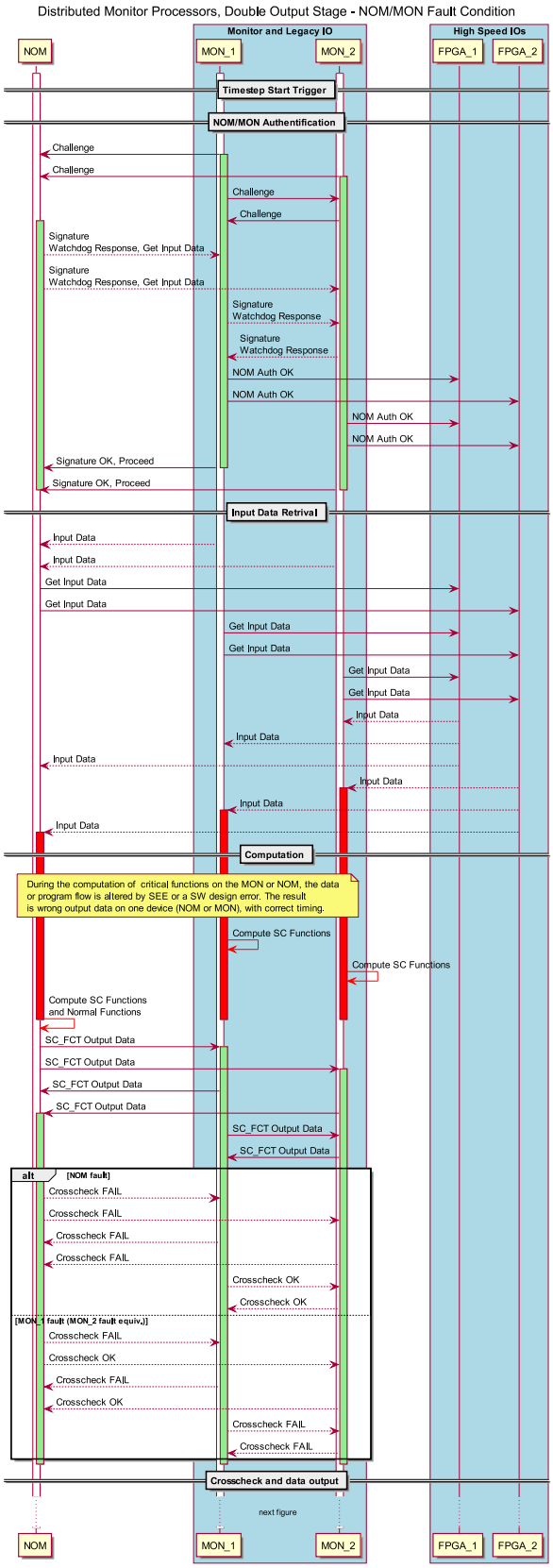

Figure A.10: *Example for fault unmasking and reaction in a DMON cluster LRU - figure is split for readability, part 1 of 2*

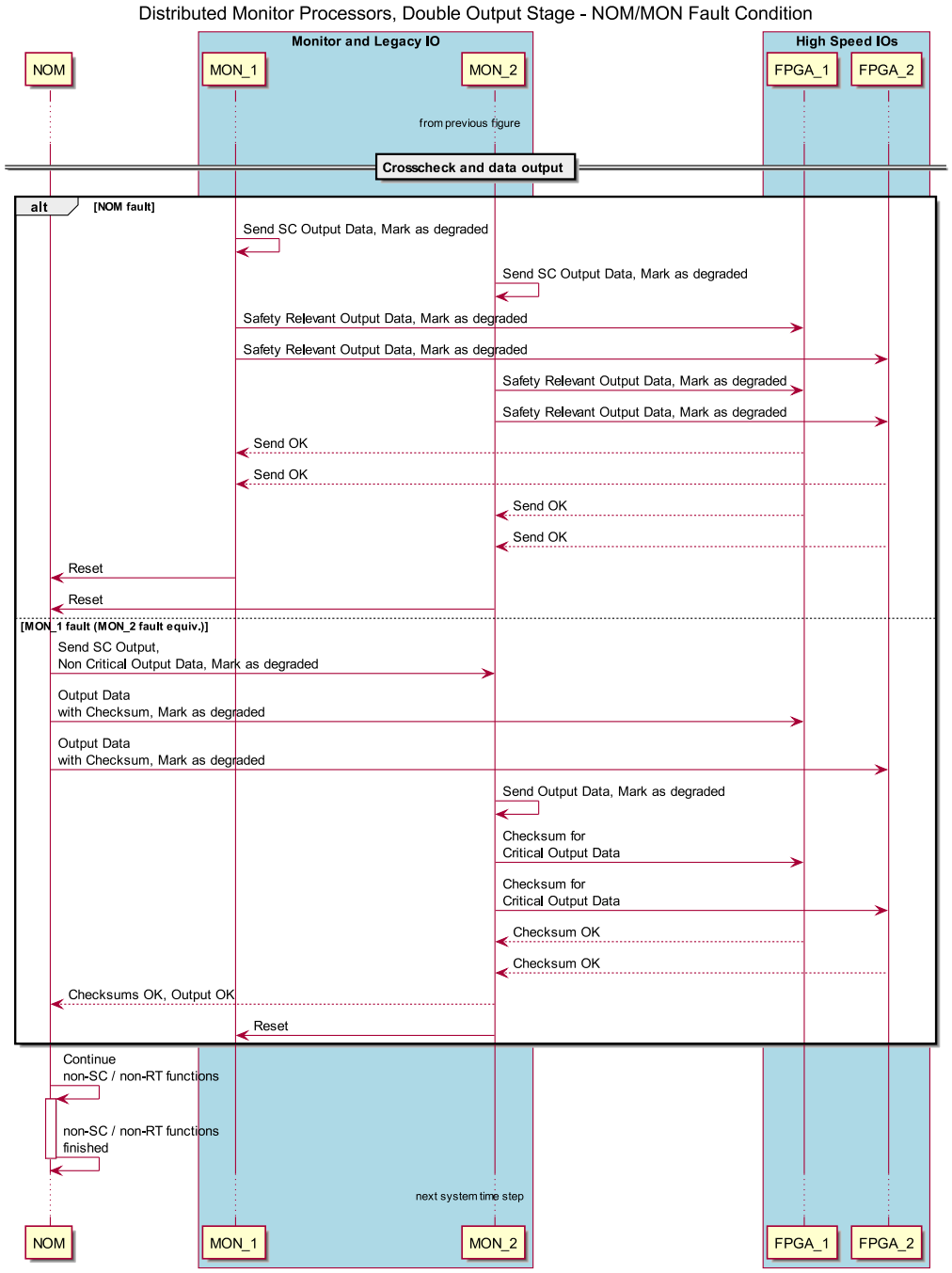

Figure A.11: *Example for fault unmasking and reaction in a DMON cluster LRU - part 2 of 2*

Appendix B

Memory and Cache Throughput Measurements

The following measurements shown in table B.1 were conducted on a NXP Semiconductor T1042, in a custom system with DDR3-1600 (800MHz Transfer Speed, 1600MT/s, 72bit wide bus with ECC, CL11) in a custom system provided by MicroSys Electronics GmbH for reference equipped with the 1200MHz core clock speed grade of the T1042. For minimal overhead, measurements were conducted in OS-9 V6.1 using the tool "memspeed" developed by Kei Thomsen [1] as 32-bit wide reads and writes in variable sized memory regions in order to traverse the different cache levels. A single core instance of OS-9 was used with memory regions setup as write-back cacheable on both platforms. The measurements shown in table B.2 were conducted on a more recent NXP Semiconductor LS1046A with DDR4-1600 (72bit wide bus with ECC, CL11) and 1200MHz core speed. The LS1046 features the ARM prefetch and prediction units and a newer generation memory controller, which leads to significant improvements compared to older platforms.

[1] Contact MicroSys Electronics GmbH, Mühlweg 1, 82054 Sauerlach, Germany for a copy of the source code and further information on OS-9.

	Bytes	Runs	Mbyte/sec.	Comment
Write	256	10485760	3545	L1 Cache
	512	5242880	4044	L1 Cache
	1024	2621440	4259	L2 Cache
	2048	1310720	4383	L2 Cache
	4096	655360	4444	L2 Cache
	8192	327680	4475	L2 Cache
	16384	163840	4491	L2 Cache
	32768	81920	4499	L2 Cache
	65536	40960	3018	L2 Cache
	131072	20480	3018	L2 Cache
	262144	10240	3018	L2 Cache
	524288	5120	1570	DRAM
	1048576	2560	1320	DRAM
	2097152	1280	1320	DRAM
	4194304	640	1320	DRAM
	8388608	320	1320	DRAM
	16777216	160	1320	DRAM
	33554432	80	1320	DRAM
Read	256	10485760	4129	L1 Cache
	512	5242880	4037	L1 Cache
	1024	2621440	4259	L2 Cache
	2048	1310720	4383	L2 Cache
	4096	655360	4444	L2 Cache
	8192	327680	4475	L2 Cache
	16384	163840	4491	L2 Cache
	32768	81920	4444	L2 Cache
	65536	40960	2925	L2 Cache
	131072	20480	2929	L2 Cache
	262144	10240	2919	L2 Cache
	524288	5120	918	DRAM
	1048576	2560	758	DRAM
	2097152	1280	718	DRAM
	4194304	640	710	DRAM
	8388608	320	708	DRAM
	16777216	160	708	DRAM
	33554432	80	708	DRAM

Table B.1: *T1042 (NXP Semiconductor) memory bandwidth measurements with DDR3-1600 at 1200MHz core frequency over different memory levels (cache and DRAM)*

	Bytes	Runs	Mbyte/sec.	Comment
Write	256	524288	4571	L1 Cache
	512	262144	4571	L1 Cache
	1024	131072	4571	L1 Cache
	2048	65536	4571	L1 Cache
	4096	32768	4571	L1 Cache
	8192	16384	4571	L1 Cache
	16384	8192	4571	L1 Cache
	32768	4096	4571	L1 Cache
	65536	2048	4571	L2 Cache
	131072	1024	4571	L2 Cache
	262144	512	4571	L2 Cache
	524288	256	4571	L2 Cache
	1048576	128	4571	L2 Cache
	2097152	64	4571	L2 Cache
	4194304	32	4571	DRAM
	8388608	16	4571	DRAM
	16777216	8	4571	DRAM
	33554432	1	4571	DRAM
Read	256	524288	4571	L1 Cache
	512	262144	4571	L1 Cache
	1024	131072	4571	L1 Cache
	2048	65536	4571	L1 Cache
	4096	32768	4571	L1 Cache
	8192	16384	4571	L1 Cache
	16384	8192	4571	L1 Cache
	32768	4096	4571	L1 Cache
	65536	2048	4413	L2 Cache
	131072	1024	4413	L2 Cache
	262144	512	4413	L2 Cache
	524288	256	4413	L2 Cache
	1048576	128	4413	L2 Cache
	2097152	64	3764	L2 Cache
	4194304	32	4129	DRAM
	8388608	16	4129	DRAM
	16777216	8	4129	DRAM
	33554432	1	4129	DRAM

Table B.2: *LS1046A (NXP Semiconductor) memory bandwidth measurements with DDR4-1600 at 1200MHz core frequency over different memory levels (cache and DRAM)*

Appendix C

Additional Cluster Fault Trees

C.1 SMON Cluster Fault Trees

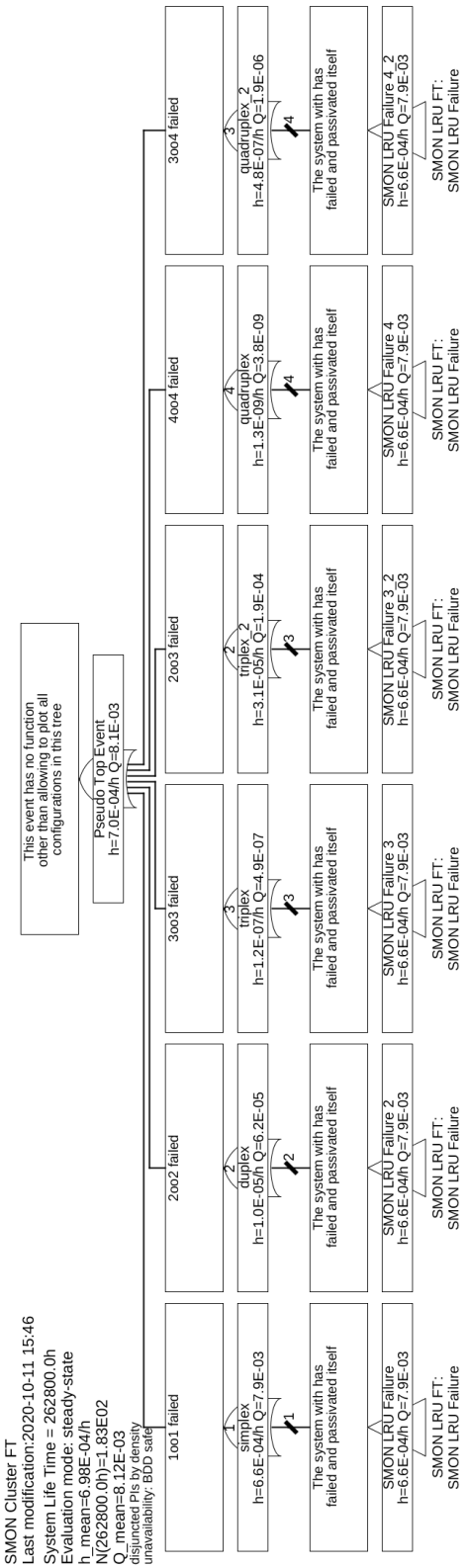

Figure C.1: *Fault tree for small clusters based on the SMON architecture*

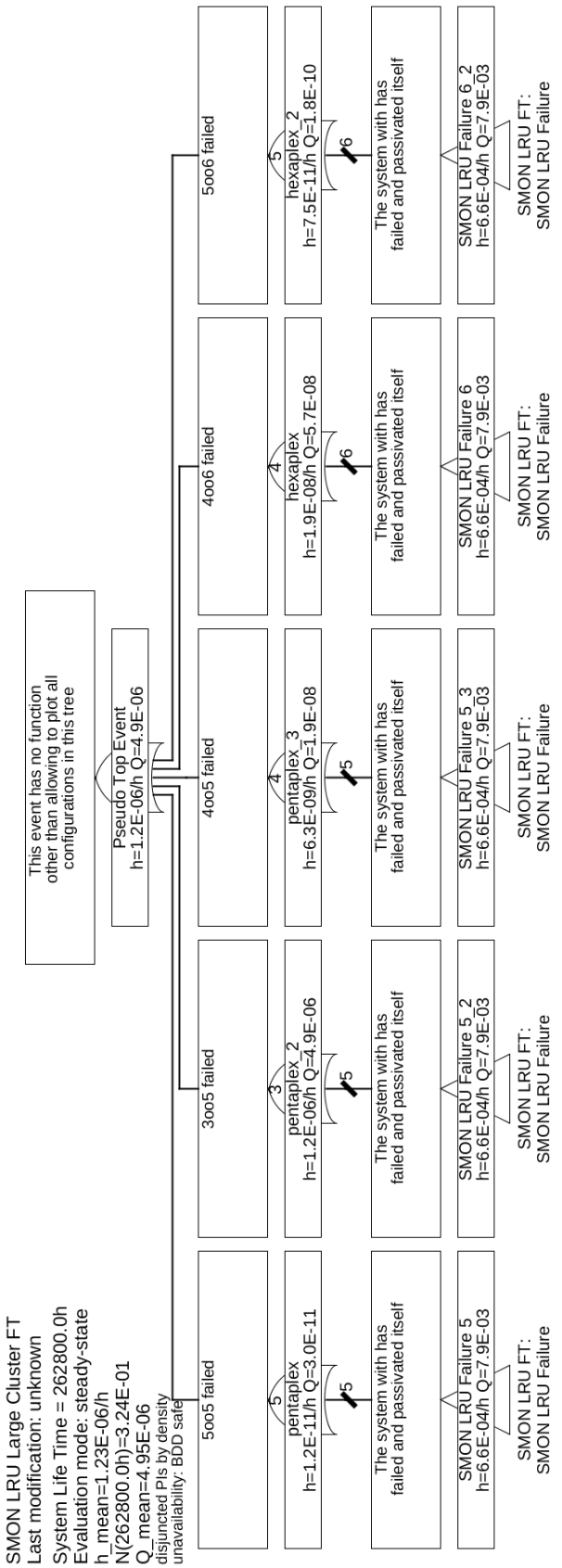

Figure C.2: *Fault tree for large clusters based on the SMON architecture*

C.2 DMON Cluster Fault Trees

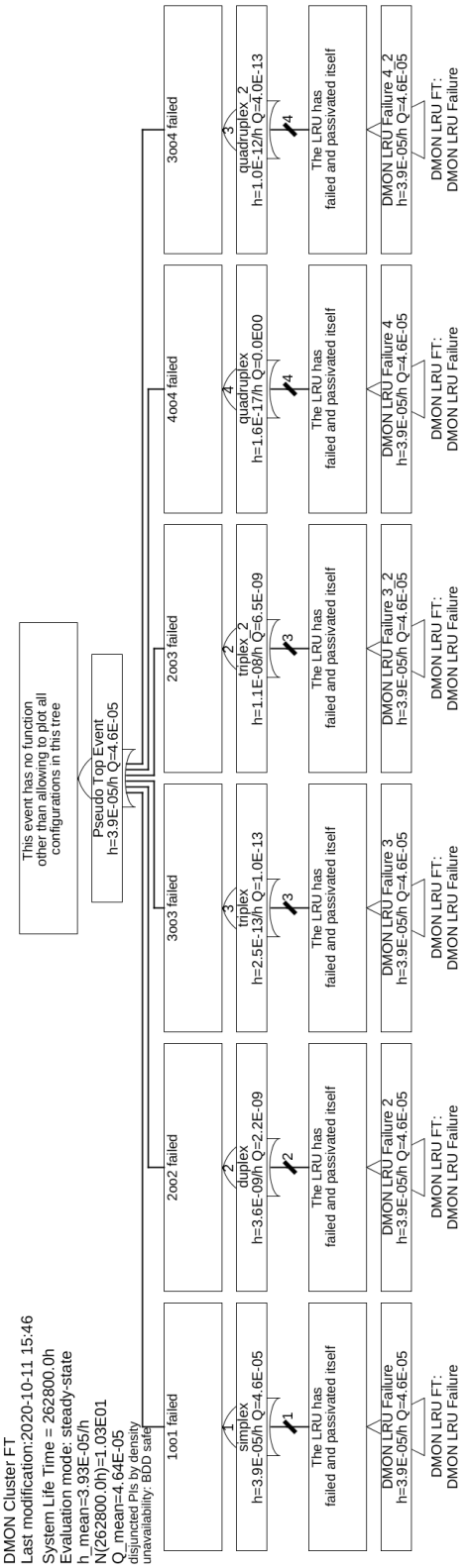

Figure C.3: *Fault tree for small clusters based on the DMON architecture*

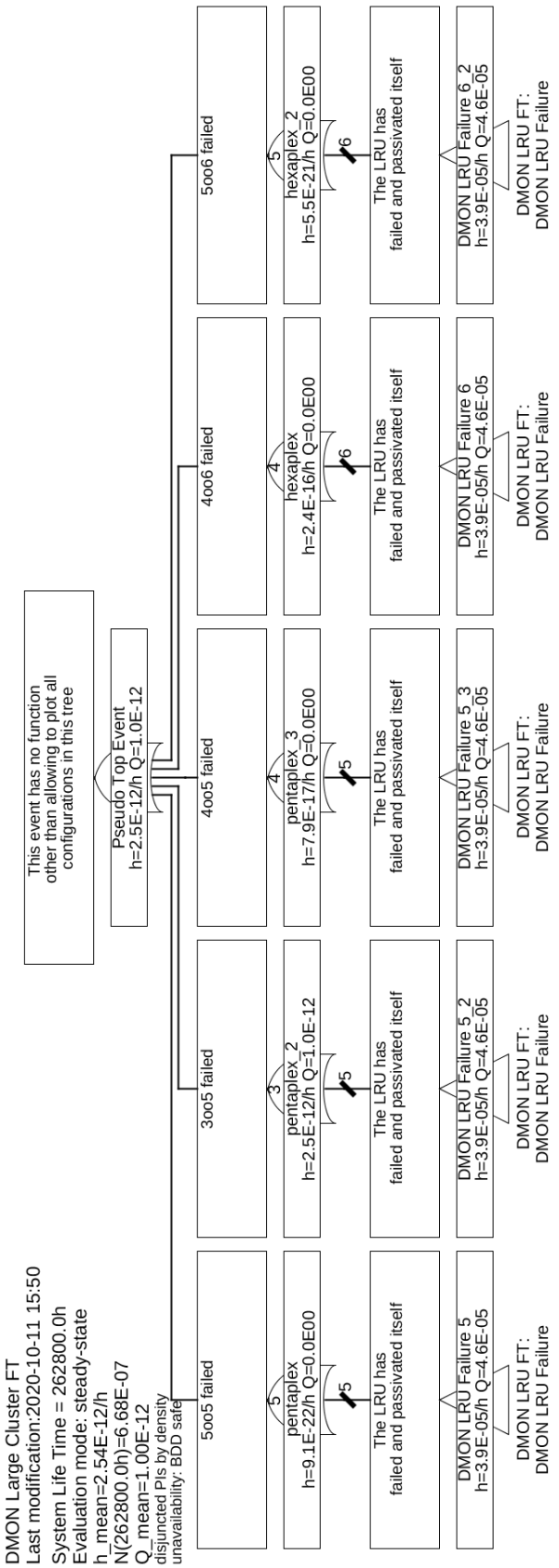

Figure C.4: *Fault tree for large clusters based on the DMON architecture*

Bibliography

[2718] 27, ISO/IEC JTC 1.: ISO/IEC 10118-3:2018, IT Security techniques — Hash-functions — Part 3: Dedicated hash-functions / ISO. 2018. – Standard

[3219] 32, TC 2.: ISO/PAS 21448:2019, Road vehicles — Safety of the intended functionality / ISO. 2019. – Standard

[4421] 44, TC: IEC 62061:2021, Safety of machinery - Functional safety of safety-related control systems / International Electrotechnical Commission. 2021. – Standard

[65A10] 65A, TC 6.: IEC 61508-X:2010, Functional safety of electrical/electronic/programmable electronic safety-related systems (Multi Part Standard) / International Electrotechnical Commission. 2010. – Standard

[ADC96] Aircraft, S-18 ; Dev, Sys ; Committee, Safety A.: ARP4761, GUIDELINES AND METHODS FOR CONDUCTING THE SAFETY ASSESSMENT PROCESS ON CIVIL AIRBORNE SYSTEMS AND EQUIPMENT / SAE International. 1996 (10.4271/ARP4761). – Standard

[ADC10] Aircraft, S-18 ; Dev, Sys ; Committee, Safety A.: ARP4754A, Guidelines for Development of Civil Aircraft and Systems / SAE International. 2010 (10.4271/ARP4754A). – Standard

[AE10] Andersson, Björn ; Easwaran, Arvind: Provably good multiprocessor scheduling with resource sharing. In: *Real-Time Systems* 46 (2010), Nr. 2, S. 153–159

[AEE10] AEEC: ARINC 653P1-3, AVIONICS APPLICATION SOFTWARE STANDARD INTERFACE PART 1 – REQUIRED SERVICES / AERONAUTICAL RADIO, INC. 2010. – Standard

[Arm10] Armoush, Ashraf: *Design patterns for safety-critical embedded systems.*, RWTH Aachen University, Diss., 2010

[Bai07] Bai, Haowei: Analysis of a SAE AS5643 Mil-1394b based high-speed avionics network architecture for space and defense applications. In: *2007 IEEE Aerospace Conference* IEEE, 2007, S. 1–9

[BB18] Becker, Jürgen ; Bapp, Falco K.: The ARAMiS Project Initiative. In: *International Symposium on Applied Reconfigurable Computing* Springer, 2018, S. 685–699

[Bru] Brunnengräber, Thomas: *Functional Safety Suite*. `http://www.thomas-brunnengraeber.de/fusasu.shtml`, Abruf: 15.06.2020

[Cla09] Clark, John O.: System of systems engineering and family of systems engineering from a standards, V-model, and dual-V model perspective. In: *2009 3rd Annual IEEE Systems Conference* IEEE, 2009, S. 381–387

[Com14] Committee, Component T.: AEC - Q100 - REV-H, FAILURE MECHANISM BASED STRESS TEST QUALIFICATION FOR INTEGRATED CIRCUITS / Automotive Electronics Council. 2014. – Guideline

[Com15] Committee, LAN/MAN S.: 802.3bw-2015, IEEE Standard for Ethernet Amendment 1: Physical Layer Specifications and Management Parameters for 100 Mb/s Operation over a Single Balanced Twisted Pair Cable (100BASE-T1) / IEEE International Standard. 2015 (IEEE 802.3bw-2015). – Standard

[Com17] Committee, LAN/MAN S.: 8802-3:2017/Amd 4-2017, Elecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Standard for Ethernet Amendment 4: Physical Layer Specifications and Management Parameters for 1 Gb/s Operation over a Single Twisted-Pair Copper Cable / ISO/IEC/IEEE International Standard. 2017 (IEEE 8802-3:2017/Amd 4-2017). – Standard

[Com20] Committee, LAN/MAN S.: 802.3ch-2020, IEEE Standard for Ethernet–Amendment 8:Physical Layer Specifications and Management Parameters for 2.5 Gb/s, 5 Gb/s, and 10 Gb/s Automotive Electrical Ethernet / IEEE International Standard. 2020 (IEEE 802.3ch-2020). – Standard

[Cor05] Corporation, Lockheed M.: JOINT STRIKE FIGHTER AIR VEHICLE C++ CODING STANDARDS FOR THE SYSTEM DEVELOPMENT AND DEMONSTRATION PROGRAM. 2005 (2RDU00001). – Standard

[DBG10] Dasgupta, Dipankar ; Bedi, Harkeerat ; Garrett, Deon: A conceptual model of self-monitoring multi-core systems. In: *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, 2010, S. 1–4

[Deu11] Deubzer, Michael: *Robust scheduling of real-time applications on efficient embedded multicore systems*, Technische Universität München, Diss., 2011

[DNA11] Dasari, Dakshina ; Nelis, Vincent ; Andersson, Björn: WCET analysis considering contention on memory bus in COTS-based multicores. In: *ETFA2011* IEEE, 2011, S. 1–4

[EAS17] EASA: CS-23 Certification Specifications for Normal-Category Aeroplanes, Amendment 5 / European Aviation Safety Agency. 2017. – Standard

[EAS18a] EASA: CM–SWCEH-001 Issue 01 Revision 02, Development Assurance of Airborne Electronic Hardware / European Aviation Safety Agency. 2018. – Certification Memorandum

[EAS18b] EASA: Regular update of AMC-20: AMC 20-152 on Airborne Electronic Hardware and AMC 20-189 on Management of Open Problem Reports / European Aviation Safety Agency. 2018. – Notice of Proposed Amendment

[FOK] FOKUS, Fraunhofer: *Multicore-Architektur zur Sensor-basierten Positionsverfolgung im Weltraum (MUSE), 1402*

[FS13] Freescale Semiconductor, Inc.: e5500 Core Reference Manual. 2013. – Reference Manual

[GHG+12] Grürmann, Kai ; Herrmann, Martin ; Gliem, Fritz ; Schmidt, Hagen ; Leibeling, Gilbert ; Kettunen, Heikki ; Ferlet-Cavrois, Véronique: Heavy ion sensitivity of 16/32-Gbit NAND-flash and 4-Gbit DDR3 SDRAM. In: *2012 IEEE Radiation Effects Data Workshop* IEEE, 2012, S. 1–6

[Gho14] Ghosh, Sukumar: *Distributed systems: an algorithmic approach*. CRC press, 2014

[Gmb] GmbH, Rockwell Collins D.: *FMC-4500 flight mission computer family data sheet*

[Gou11] Goupil, Philippe: AIRBUS state of the art and practices on FDI and FTC in flight control system. In: *Control Engineering Practice* 19 (2011), Nr. 6, S. 524–539

[HGG+11] Herrmann, Martin ; Grürmann, Kai ; Gliem, Fritz ; Kettunen, Heikki ; Ferlet-Cavrois, Véronique: Heavy ion SEE test of 2 Gbit DDR3 SDRAM. In: *2011 12th European Conference on Radiation and Its Effects on Components and Systems* IEEE, 2011, S. 934–937

[HM15] Hancock, William R. ; Miller, Larry J.: *System and method of cache partitioning for processors with limited cached memory pools*. Juli 23 2015. – US Patent App. 14/159,180

[Hol19] Holdings, ARM: ARM Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile. 2019. – Reference Manual

[HP10] Huffman, W C. ; Pless, Vera: *Fundamentals of error-correcting codes*. Cambridge university press, 2010

[Inc20] Inc., Xilinx: Device Reliability Report Second Half 2019. 2020. – Report

[ISO12] ISO: ISO 13849-2:2012, Safety of machinery — Safety-related parts of control systems — Part 2: Validation / SAE International. 2012. – Standard

[ISO15] ISO: ISO 13849-1:2015, Safety of machinery — Safety-related parts of control systems — Part 1: General principles for design / SAE International. 2015. – Standard

[ITC19] ITC, SAE: ARINC 664P7-2, Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet Network / SAE ITC, ARINC Industry Activities. 2019. – Standard

[JG11] Jain, Madhu ; Gupta, Ritu: RELIABILITY ANALYSIS OF M-OUT-OF-N: G SYSTEM WITH MULTIPLE TYPES OF WARM STANDBY. In: *Advances in Modeling, Optimization and Computing* (2011), S. 433

[JWN10] Jacob, Bruce ; Wang, David ; Ng, Spencer: *Memory systems: cache, DRAM, disk*. Morgan Kaufmann, 2010

[JXM12] JEAN Xavier, GATTI Guy BERTHON M. ; Marc, FUMEY: Research Project EASA.2011/6 - MULCORS - Use of Multicore Processors in airborne systems. (2012)

[KGB12] Koga, R ; George, J ; Bielat, S: Single event effects sensitivity of DDR3 SDRAMs to protons and heavy ions. In: *2012 IEEE Radiation Effects Data Workshop* IEEE, 2012, S. 1–8

[Lee08] Lee, Edward A.: Cyber physical systems: Design challenges. In: *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)* IEEE, 2008, S. 363–369

[LPO15] Larrucea, Asier ; Perez, Jon ; Obermaisser, Roman: A modular safety case for an IEC 61508 compliant generic COTS processor. In: *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing* IEEE, 2015, S. 1788–1795

[LRM11] Larson, Aaron ; Roffelsen, Ryan ; Miller, Larry J.: *Cache pooling for computing systems*. November 29 2011. – US Patent 8,069,308

[Man17] Mancuso, Renato: *Next-generation safety-critical systems on multi-core platforms*, University of Illinois at Urbana-Champaign, Diss., 2017

[MDB+12] Motruk, Boris ; Diemer, Jonas ; Buchty, Rainer ; Ernst, Rolf ; Berekovic, Mladen: IDAMC: A many-core platform with run-time monitoring for mixed-criticality. In: *2012 IEEE 14th International Symposium on High-Assurance Systems Engineering* IEEE, 2012, S. 24–31

[MDB+13] Mancuso, Renato ; Dudko, Roman ; Betti, Emiliano ; Cesati, Marco ; Caccamo, Marco ; Pellizzoni, Rodolfo: Real-time cache management framework for multi-core architectures. In: *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)* IEEE, 2013, S. 45–54

[MHPS96] Majzik, István ; Hohl, Wolfgang ; Pataricza, András ; Sieh, Volker: Multiprocessor checking using watchdog processors. In: *Computer Systems Science and Engineering* 11 (1996), Nr. 5, S. 301–310

[Mit16] Mittal, Sparsh: A Survey Of Techniques for Architecting and Managing Asymmetric Multicore Processors. In: *ACM Computing Surveys* 48 (2016), 02. http://dx.doi.org/10.1145/2856125. – DOI 10.1145/2856125

[MM10] Mathur, Sonali ; Malik, Shaily: Advancements in the V-Model. In: *International Journal of Computer Applications* 1 (2010), Nr. 12, S. 29–34

[MPC+15] Mancuso, Renato ; Pellizzoni, Rodolfo ; Caccamo, Marco ; Sha, Lui ; Yun, Heechul: WCET (m) estimation in multi-core systems using single core equivalence. In: *2015 27th Euromicro Conference on Real-Time Systems* IEEE, 2015, S. 174–183

[New94] Newport, John R.: *Avionic systems design.* Crc Press, 1994

[NP12] Nowotsch, Jan ; Paulitsch, Michael: Leveraging multi-core computing architectures in avionics. In: *2012 Ninth European Dependable Computing Conference* IEEE, 2012, S. 132–143

[NPB+14] Nowotsch, Jan ; Paulitsch, Michael ; Bühler, Daniel ; Theiling, Henrik ; Wegener, Simon ; Schmidt, Michael: Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement. In: *2014 26th Euromicro Conference on Real-Time Systems* IEEE, 2014, S. 109–118

[NPH+14] Nowotsch, Jan ; Paulitsch, Michael ; Henrichsen, Arne ; Pongratz, Werner ; Schacht, Andreas: Monitoring and WCET analysis in COTS

multi-core-SoC-based mixed-criticality systems. In: *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)* IEEE, 2014, S. 1–5

[PGN+14] Perez, Jon ; Gonzalez, David ; Nicolas, Carlos F. ; Trapman, Ton ; Garate, Jose M.: A safety certification strategy for IEC-61508 compliant industrial mixed-criticality systems based on multicore partitioning. In: *2014 17th Euromicro Conference on Digital System Design* IEEE, 2014, S. 394–400

[PP02] Papoulis, Athanasios ; Pillai, S U.: *Probability, random variables, and stochastic processes*. Tata McGraw-Hill Education, 2002

[PS19] Pinto, Sandro ; Santos, Nuno: Demystifying arm trustzone: A comprehensive survey. In: *ACM Computing Surveys (CSUR)* 51 (2019), Nr. 6, S. 1–36

[PTV+13] Pop, Paul ; Tsiopoulos, Leonidas ; Voss, Sebastian ; Slotosch, Oscar ; Ficek, Christoph ; Nyman, Ulrik ; Lopez, Alejandra R.: Methods and tools for reducing certification costs of mixed-criticality applications on multi-core platforms: the RECOMP approach. In: *Proceedings of the Workshop of Industry-Driven Approaches for Cost-effective Certification of Safety-Critical, Mixed-Criticality Systems* Bd. 156, 2013

[Rot60] Rotenberg, A.: A New Pseudo-Random Number Generator. In: *J. ACM* 7 (1960), Januar, Nr. 1, 75–77. http://dx.doi.org/10.1145/321008.321019. – DOI 10.1145/321008.321019. – ISSN 0004–5411

[SBM+08] Shye, Alex ; Blomstedt, Joseph ; Moseley, Tipp ; Reddi, Vijay J. ; Connors, Daniel A.: PLR: A software approach to transient fault tolerance for multicore architectures. In: *IEEE Transactions on Dependable and Secure Computing* 6 (2008), Nr. 2, S. 135–148

[SC-00] SC-180: DO-254, Design Assurance Guidance for Airborne Electronic Hardware / RTCA Inc. 2000. – Guideline

[SC-11] SC-205: DO-178C, Software Considerations in Airborne Systems and Equipment Certification / RTCA Inc. 2011. – Guideline

[SC-12] SC-135: DO-160G, Environmental Conditions and Test Procedures for Airborne Equipment / RTCA Inc. 2012. – Guideline

[Sem16] Semiconductors, NXP: QorIQ LS1046A Reference Manual. 2016. – Datasheet

[Sem19] Semiconductors, NXP: LS1046A Data Sheet: Technical Data Rev. 3. 2019. – Datasheet

[Ste15] Steinert, Lukas: Safety Critical Systems based on COTS Multi-Core Processors. (2015)

[SVD+02] Stamatelatos, Michael ; Vesely, William ; Dugan, Joanne ; Fragola, Joseph ; Minarick, Joseph ; Railsback, Jan: Fault tree handbook with aerospace applications. (2002)

[Tea18] Team, Certification Authorities S.: CAST-32A, Multi-core Processors / Federal Aviation Administration. 2018. – Postition Paper

[Web17] Weber, Werner: *Embedded multi-core systems for mixed criticality applications in dynamic and changeable real-time environments, D13.26 Final Standardization Report and Outlook*. 2017

[Wil] Wilkinson, Jeff: *Sort-error Testing Ressources - Flux Calculator*. `http://www.seutest.com/FluxCalculator.htm`, Abruf: 15.06.2020

[WJKLMC] Wroble, Mike ; Jack Kreska Lockheed Martin Corporation, Larry Dungan N.: SAE Mil-1394 For Military and Aerospace Vehicle Applications / SAE International Group. – Forschungsbericht

[Yeh96] Yeh, Ying C.: Triple-triple redundant 777 primary flight computer. In: *1996 IEEE Aerospace Applications Conference. Proceedings* Bd. 1 IEEE, 1996, S. 293–307

[YMWP14] Yun, Heechul ; Mancuso, Renato ; Wu, Zheng-Pei ; Pellizzoni, Rodolfo: PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms. In: *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)* IEEE, 2014, S. 155–166

[YYP+13] Yun, Heechul ; Yao, Gang ; Pellizzoni, Rodolfo ; Caccamo, Marco ; Sha, Lui: Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In: *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)* IEEE, 2013, S. 55–64