Working Paper - Safety Critical Systems based on COTS Multi-Core Processors

Towards new system architectures for highly automated and autonomous systems

Lukas Steinert, M. Sc.
Institute for Flight System Dynamics, TU Munich

Abstract

In upcoming highly automated systems in safety-critical application domains, such as aerospace and industrial machinery, the demand for more computational resources per control unit is constantly rising, accompanied from a shift in the chip industry towards multi-core devices with enormous complexity and high levels of feature integration. Current regulation and standards for certification does not yet consider these devices, or explicitly denying their use as a pure single-core processor replacement. Within this work, we address the issue of multi-core certifiability on the system level, providing two possible board-level architectures. In addition, a safety case is provided to show how a proper system architecture approach can fit multi-core devices in current regulation, raise system reliability and solve the common failure mode problem in the multi-core on the board level. Alongside with the hardware architecture, a high level software architecture is presented to allow mixed-criticality applications to be executed side-by-side on the multi-core with spatial and temporal isolation. We follow a requirements-driven work flow, to first define a set of requirements, to be fulfilled by the secondly proposed architectural configurations, followed by a safety analysis and the presentation of a possible certification argumentation for an industrial and aerospace context.

Introduction

During the recent years, multi core based System-on-a-Chips (multi-core SoCs) rose and found a wide spread application in general purpose, mobile and high performance computing. In the embedded systems / real time domain and especially in the field of safety critical systems, be it in an industrial, aerospace or medical context, multi-core SoCs have not found application in certified applications. Usually, these domains rely on either special build devices (ASICs) or domain specific COTS processors, usually generations behind current technology levels for the most critical applications. Due to the tremendous efforts and development costs associated with multi-core SoCs, the trend for tailor-made or domain specific devices will not be continued broadly with those devices. Today's SoCs are designed for either Networking applications (Routers, Switching, Baseband), general-purpose- or server-computing applications. They generally lack specific architectural features for the safety critical domain, for example strict core separation at silicon level, special monitoring function units or deterministic on-chip networks, which render them practically un-useable in general for a simple single-core processor replacements in current board level architectures. With shrinking technology levels and the continuous demand for more computational power in a single package, multiple cores in a single device are already starting to pop up even in small microcontroller packages and will certainly be the norm, rather than the exception, in future device generations.

A multi-core based solution for safety applications faces a multitude of certification concerns and risks. In the light of the ISO/TR 61508 [1] or the ISO 13849 [2] for example, an applicant has to derive an FEMA on the system and board level, possibly accompanied by fault tree analysis, markov analysis or other methods in order to approximate the remaining possibility for predetermined dangerous hazards during the safety assessment. This also requires in depth, silicon level information, reaching far into the SoC when internal function units guarantee certain safety features, which the semiconductor manufacturer usually does not provide for multi-core devices. Based on this information, one derives the base events for the FMEA, in order to compute the respective safety metrics. In the aerospace domain, we face an even

worse situation. The certification authorities yet lack a common position on this new technology and present their current view in the CAST-32 [3] and CM-SWCEH-001/002 [4]. In order to provide a reliable platform for future generations of safe systems, one has to overcome the inherent problems associated with multi-core devices[1], while designing with existing COTS devices to avoid costly custom ASICs. This in turn will raise the computational power, available for future highly automated or even autonomous systems, for current development trends towards unmanned aerial vehicles, driverless vehicles or new generations of industrial systems. With a steep increase of available computational resources per LRU, many current system architectures, with largely dedicated LRUs per system function, can be reduced and centralized to a single, redundant, high-integrity platform, instead of providing per-function redundancy via dedicated LRU clusters.

Since the margin and volume of embedded applications (despite automotive) is quiet low compared to server or telecommunication applications, the semiconductor manufacturer may not invest in the extremely costly development efforts for domain-specific devices with long ROI periods. Therefore, we focus solely on applying available COTS devices within our proposed architectures.

Within the next chapters, we will first present possible future system architectures for highly automated systems with a resulting set of high-level requirements for the single LRU. Based on these requirements, we derive a board level architecture based on COTS multi-core processors as the main computing device. In addition, we provide a possible certification argumentation to support a safety claim in certain domains and certification levels based on current regulation.

When applying multi-core technology to LRUs, the system builder may benefit financially by removing great portions of currently distributed, function specific LRUs from the system and integrate their functions on a single LRU, given that the multi-core LRU provides enough computational resources and communication interfaces. This not only reduces weight (LRUs, cabling, mounting, etc.), size, cost but also, depending on the processors used, also power dissipation. If we design the architectural concept and board-level architecture of a multi-core based LRU in such a way that the overall system safety is not affected by a centralized platform or the integration of different system functions, the benefits of multi-core based LRUs is evident.

The issues associated with the certification of multi-core SoC are well known in academia and in the industry. We will therefore only present the proposed architectural configurations and certification approach without further detailing the specific pitfalls, since this has been subject to many talks and publications in the past. However, one can obtain a first introduction into the topic by the study available in [5], conducted to provide some guidance for European aviation authorities on the topic, outlining potential hazards and risks.

High Level System Architectures

In current system development processes for safe systems, nominal system functionality is subject to safety analysis for potential hazards, deriving a notion whether a function is safety related or uncritical in the context of the system and its environment. In the civil aerospace domain, a function design assurance level (FDAL, from E to A) describes the criticality of a function in relation to the hazard impact and resulting target occurrence rate for failures of the function. Almost the same methodology is applied in an industrial certification process, where a safety integrity level (from 1 to 4, IEC 61508) or an appropriate category (from B to 4, ISO13849) is derived on a per-function basis. In the military aerospace domain, functions are classified by their "mission criticality". A mission critical function is not directly responsible for a hazard, but compromises the system's ability to fulfill a given task or mission successfully.

While this holds true for classic semi-automated systems, heavily relying on a human to provide higher level automation functions, cognitive functions to master complex situations, or a fallback level in case of failure, the classification and system level assurance

[1] Multi-Master on Shared Bus within SoC, Shared Peripherals, Shared Resources, Separation of Cores and Common Mode Failures on the SoC, just to name a few

with highly or fully automated, unmanned or even autonomous systems becomes increasingly complex.

We will therefor further classify higher level system functions during the course of this working paper in three levels. These classifications are interoperable with current regulation, in fact the currently available criticality levels can and will be used within the context of these meta classes in order to further group system functions. Furthermore, these classifications will be of importance later on when we enter a more detailed LRU board level architecture and safety argumentation:

i. Impact Level 2 (IL2): A critical function in the sense that any kind of failure of this function leads to potential harm.

   Examples include: Aircraft/Vehicle baseline control, Brake control, Motor Control, Low level logic for system modes, Movement or trajectory Generation, etc.

ii. Impact Level 1 (IL1): A function which directly impacts the system behavior, but without direct command authority over actuation elements. These functions might fail without compromising the safety of the system, but disable its ability to complete a given task. In this class, a wrong result (wrong command to a IL2 function) is a hazard, but a complete failure in the sense of unavailability is not.

   Examples include: High Level Path and Mission Planning, Mission or System Behavior Optimization, Context Recognition, Nonlinear System Adaption, Remote Control Data Links, etc.

iii. Impact Level 0 (IL0): A non-critical function. Failures (unavailable or wrong result) will only affect the comfort in using a system or its effectiveness but will not influence the behavior with regards to safe operation or completing a given mission or task.

IL0 and IL2 are very similar to the current understanding by basic regulation, being either potentially hazardous or not. IL2 functions normally execute control algorithms, state machines or other forms of automata, or supervisory tasks of critical elements. IL1 functions exist today in mostly non-critical classifications like in flight management systems, aiding for example the pilot in commercial and small aircrafts to complete their tasks more

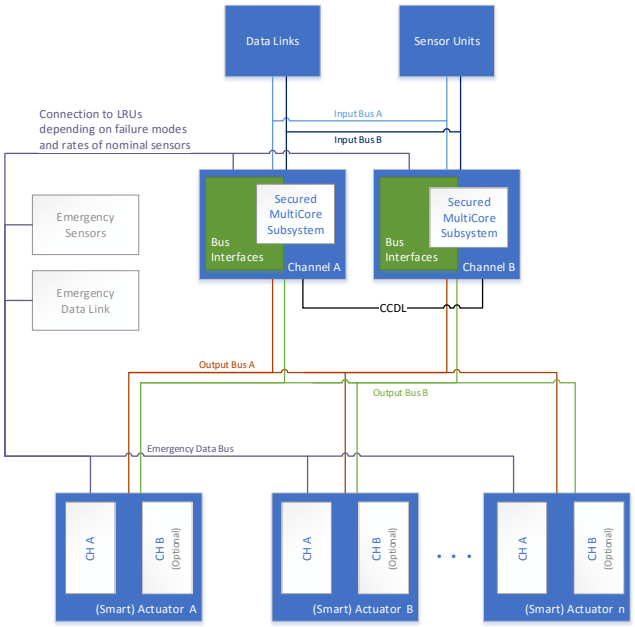


Figure 1: Small target system architecture example (simplified)

effectively, for example by directly communicating with autopilot systems to reduce cockpit crew workload. Due to human oversight, these functions are currently not safety critical. This changes considerably when a human is no longer present or incapable (due to physical limitations like reaction time / precision or very limited situational awareness) to control the system, making these function equally critical in a classical certification classification, or in our case IL1. Note also that functions from the IL1 category are usually more complex functions and therefor require much larger computational power compared to IL2 functions, which are in most cases control algorithms with limited computational requirements in general. We'll address IL1 functions again later in this chapter, when we describe system architectures, voting strategies and degradation patterns. Before entering a more detailed LRU level, applicable system level embedded system architectures will now be defined.

"Small System" Use-Case (UC1)

Often times in industrial systems, cost is a major factor in system development. This also holds for small scale unmanned aerial vehicles (UAVs) in the sub 25kg/50kg MTOW class. Power draw has to be minimized for this class of systems, as well as

electronics weight, size and also power dissipation in order to ease cooling requirements. It is nonetheless necessary to provide a high availability and redundant system, to cope with board, parts and interconnect failures as they appear throughout the lifetime of the system, due to harsh environment conditions (high vibration, high and fast changing temperature load cycles, etc.). Due to these constraints, a high degree of availability has to be achieved with very little LRU-level redundancy in this context, even if not directly required by certification criteria, but from a commercial and usability standpoint.

Figure 1 shows an exemplary architecture, where, depending on the desired failure rate of the compute platform, only a limited number LRUs is used. Instead of relying on a distributed computing architecture with dedicated LRUs per system function (like base control, fallback, higher-level functions), the multi-core based LRU is powerful enough to complete all necessary computations timely. Besides the shown compute platform, only smart sensors and actuators, as well as optional human control elements or wireless data links are present in the system shown. For multiple LRUs, voting is either distributed with a single command send to the actuators or computed on the actuators itself in the case of smart actuation elements. A cross-channel datalink (CCDL) interconnects each LRU. Further reducing the system complexity and cost, one could completely omit redundancy for most industrial systems, as long as the single LRU is compliant to the applicable certification category, which is already the case today in most industrial applications with control systems being certified up to SIL3/KAT3,PL.d in certain applications.

"Large System" Use-Case (UC2)

In contrast with the previous example, in large and complex systems with eased electronics cost constraints, we propose a different architectural approach. Also centered on a centralized compute platform, as shown in Figure 2, smart actuators, sensors and optional control elements are utilized. Zonal Safety can be established by physically distributing parts of the redundant LRU cluster in separate locations in the system, providing resilience against environmental hazards like fire or leaks, just as with today's commercial aviation, or military avionics concepts.

In Figure 2, a dual duplex configuration is shown, which can be extended as needed to higher orders of redundancy, for example in two/three triplex or even two/three quadruplex clusters. Note that the cluster is interconnected by a dedicated cross-channel datalink which allows voting and data exchange over clusters and between LRUs in a cluster. As a side effect, with increasing numbers or LRUs in the system, there are also growing computational resources for IL0 as we'll see in the next chapters, since they can be executed without redundancy in a distributed fashion over the whole cluster.

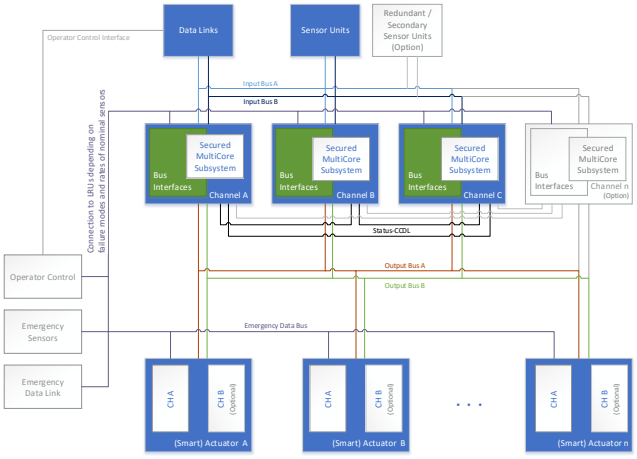


Figure 2: Large target system architecture example (simplified)

Since the detailed architecture is highly dependent on the actual system and not only on the application domain, we found it difficult to derive further detailing steps as these are by nature strongly coupled with the desired nominal and failure mode system behavior.

Requirements for an LRU

Stepping from the higher level system architectures down into the LRU itself for a board-level architecture, one needs a set of derived requirements. Table 2 (see Appendix I) list our set of requirements, alongside with the applicable use-case from which each requirement was derived.

The first requirements (RQ1-RQ5) mark an important point. In essence, we want to achieve up to SIL2/KAT2 for a single LRU, or SIL3/KAT3 classification depending on the surrounding system configuration (e.g. secondary safety shut-off paths) for the industrial context. Starting from a dual-LRU configuration on system level, SIL4/KAT4 shall be

possible, again, depending on the surrounding system or final level of redundancy. In an aerospace context, a single unit should at least reach design assurance level C, with an upgrade path to DAL B or DAL A if adequate levels of redundancy are provided (at least two for DAL B, more units for DAL A). Again, this is heavily depending on the system context and possible fault mitigation or control system degradation paths desired for the final system, depending not only on weight and size, but also on the degree of automation, presence of emergency operation control and the intended operating area of the vehicle in question. With current cost structures and upgrades for legacy systems coming into play, it is vital that a single LRU is able to provide medium certifiability (DAL E-C), just like todays LRU designs, so the designer is only forced to a redundant configuration for high availability and reliability (DAL B-A). RQ6 is derived from in order to facilitate cross LRU communication, further harden the system architectures and permit duplex systems where a LRU may not output wrong results but fail silent to simplify the voting between LRUs and the interaction on system level. In redundant configurations, RQ7 ensures that a unit passivates itself if an error has been detected. A unit may rejoin the cluster, if permitted by system requirements, if the error has been resolved after a LRU reboot (transient errors). RQ8 is required for systems in a non-transportation domain, like reactors or complex machinery which cannot simply stop operation at once but require a series of mode transitions in order to safely stop, or must remain operating in a degraded mode with very basic control carried out to prevent substantial financial losses (like in chemical reactors, furnaces, etc.). Note that this is most likely achieved by system level redundancy, however, for future systems, a single remaining unit (or in case of common mode component failures) shall be able to maintain a safe system state. RQ10-13 deal with requirements commonly arising from ISO 13849 applications with single fault tolerance, translating roughly to the “no single point of failure” credo in higher aerospace DAL levels.

RQ15 marks an in important point. A multi-core based LRU can only be reasonable from a cost and risk perspective, if it replaces multiple legacy LRUs by combining their system functions, which are classified with different criticality and impact levels by the legacy system. It is therefore essential, that those functions can be executed side-by-side on the multi-core without interference, which leads to deadline violations of hard real-time, IL2 functions. Note that strict electrical isolation in the terms of true freedom of interference will most likely not be achievable on COTS devices. It is however sufficient, with appropriate architectural mitigation techniques in place on the board level, to prove (formally or during testing) that under normal operating conditions, all functions can meet their respective deadlines and a misbehavior of a function can be timely detected and mitigated. Also note that this in general implies an AMP configuration on the multi-core from a software perspective, but also leaves the possibility to execute a SMP OS for IL0 or IL1 functions along with an AMP RTOS if only a subset of the available cores is used in the SMP configuration.

Requirements RQ16-18 constrain the available interfaces. “High-Speed”, as well as legacy “low-speed” interfaces shall be provided in order to enable retrofitting and use of proven/qualified sensor and actuation units fitted with domain specific legacy data busses like CAN, ARINC 429, RS485/Profibus, etc. In contrast, recent system architectures already move towards more data bus bandwidth by employing Ethernet-based sub-standards like AFDX, EtherCAT, PowerLink, or the open SpaceWire high speed interface standard, as well as other domain specific standards and protocols. Exemplary minimum bandwidth requirements are also given, in the sense that the LRUs internal architecture should permit concurrent full theoretical throughput on all interfaces. The cross channel datalink should not bottleneck system level data transfers, which entails a higher internal interconnect bandwidth, but again this should be seen as an example only, since the final data bus specifications are depending entirely on the system context, application domain and use-case of the system.

Board Level System Architectures

Starting from the requirements presented in the previous section, we now derive suitable board level architectures for different application scenarios, based on a COTS multi-core device. While we will discuss some safety matters as we present the proposed architectures, a more detailed certification perspective for different application domains will take place in the next chapter. Note that the concepts are designed to allow easy scaling in terms of processors and interfaces, limited by physical board / unit constraints

and the available high speed interfaces on SoC level. Note also that certain high speed interfaces have been specifically avoided where applicable, since they introduce major issues with certification, for example in the aerospace domain. These include for example serial high speed interfaces like PCI-Express (PCIe), where in most IP implementations, microcode is used to implement certain bus functions in hardware. Since microcode is considered software in terms of certification, it is necessary that the microcode development process follows a conformant development process for the desired certification level, forcing the applicant into substantial involvement with the SoC manufacturer with access to design data. Such undertakings have not been successful in the past, resulting in failed project, greatly exceeding cost targets and time schedules. However, depending on the application domain, a black-channel approach can be applied to justify the use of said interfaces where no in-depth insight in these peripherals is required.

Furthermore, the architectures aim for requiring as little detailed SoC manufacturer data as possible to lower the actual project risks with COTS devices with little financial interest by the IC manufacturer due to the low volume of our application domains.

The software architecture and means to provide spatial and temporal isolation between cores or different system functions are discussed after presenting the proposed board level architectures.

Non-Distributed Monitoring Configuration

The first architectural configuration is built upon the simplest command-monitor architecture. On the board level, a multi-core device is used to compute IL2, IL1 and IL0 functions, together with a simpler, single-core device executing the IL2 monitor functions. A monitor function can be algorithmically different to the nominal function, but it is also possible to use the same algorithm, provided some form of software diversity is used to mitigate software related common mode failures. As shown in Figure 3, the monitor only verifies the IL2 functions before their final results are send out via the output stages (FPGA-based, for legacy and high speed interfaces). IL1 functions are either not present in the system due to the application or are being compared on the system level with redundant LRUs over the cross-channel data link. IL0 functions do not feature any kind of monitoring or fault protection.

This configuration permits to detect if either the monitor (MON) or nominal channel (NOM) have failed by cross comparison. Since both devices are able to provide output data to the data busses, a fail-silent behavior is easily achieved by bringing the system into a safe output state, once a cross comparison mismatch has been detected by either the NOM or MON. It is also possible to implement a fallback operating mode, where the system stays operational after detecting a failure, as well as fallback strategies, for example in the output stages with simple state machines taking over, driving the physical system into a safe state. Note that this configuration is intended to be used in either a very cost-sensitive environment with medium availability demands, since there is no hardware fault tolerance provided on the board level enabling a true fault masking, or within a redundant LRU configuration. A second LRU, connected via a cross-channel data link, would then be able to keep the critical functions operative while one LRU is executing an ordered restart or is deactivated

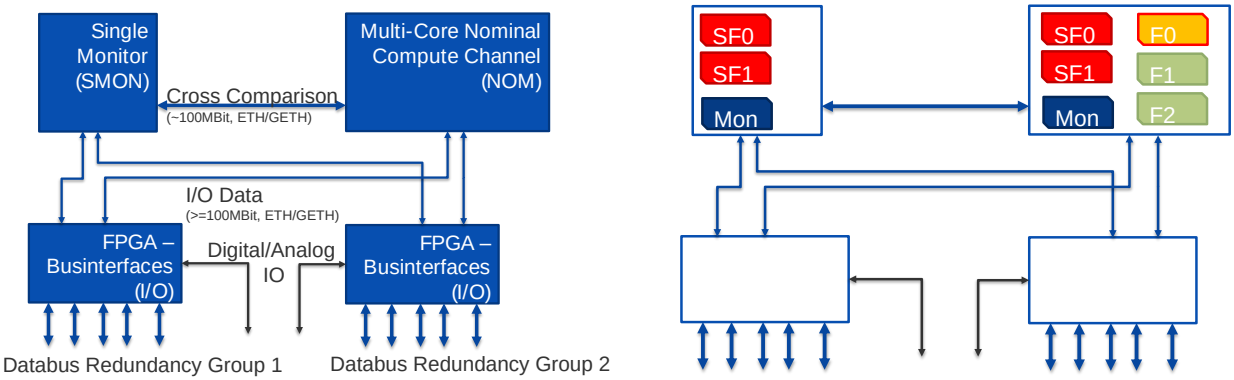


Figure 3: Non-Distributed Monitor Configuration. Left: General Overview; Right: Function Allocation by Impact Level (IL2 red, IL1 orange, IL0 green)

until replacement if the failure is persistent. Note also that the MON and NOM must be diverse devices to mitigate common mode failures in both channels.

In order to provide enough processing resources for all IL2 functions executed on the multi-core NOM, the MON must be a relative high powered single core device, or a multi-core device with no more than two cores, one handling the SoC peripherals, interrupts and SoC supervision, while the other core executes the IL2 monitoring. The clear partitioning omits the needs for complex core-to-core isolation strategies, since the timing behavior can be determined during testing or through an on-target and formal methods approach based on single-core equivalence (see [6]). It is, however, more practical to use a simple single core device as the MON to ease certification around the MON. With future SoCs moving more and more towards multi-core, it might the case that fast single core devices might become unavailable for the next generations of embedded systems, so at least a dual-core device was chosen for this example, with a clean, isolated AMP approach to equal existing single core devices with IO accelerator co-processor cores, which reduce system load by offloading interrupt processing and low level data handling for peripherals from the main CPU core to a separate, smaller, less-capable core. In most cases however, the IL2 functions require very little computational resources and consist mostly of control functions which run presently on low power microcontrollers or microprocessors in the sub-500MHz range. Adequate single core controllers with more than 1GHz, based on recent core architectures will satisfy most computational needs regarding those functions easily while they are still available on the market for at least another decade.

In certain use-cases, where IL2 functions are less computationally intensive, it is possible to extend this configuration to a fail-operating architecture with only one LRU by using a self-checking, safety MCU, based on a lockstep architecture with internal fault detection[2]. In this case, random or persistent failures can be traces in run-time to either the MON or NOM, enabling a shut down or restart of the respective channel whilst either upholding the IL2 functions on the remaining channel, driving the physical system into the safe state or executing an emergency operating mode on the NOM (if MON failure has been detected on the safety-type MCU) or the MON (if cross check failed and the safety-type MCU detects itself as operational). Note that this argumentation most likely requires in-depth failure mode analysis for the MON and on board-level.

On the board level, the NOM, MON and output stages can be easily interconnected via dedicated Ethernet links, since most modern SoCs offer at least one Ethernet interface, capable of 1Gbit/s link speed. Operating on the MAC layer, a black-channel protocol (legacy or custom) should be employed to further secure the data transfer which does not entail the use of higher OSI-level protocols like IP, TCP/IP or PowerLink. If one of the MPUs used does not provide adequate numbers of interfaces, especially in the case that more than two output stages are needed due to special application requirements, one can readily design in switched networks or a shared bus based on COTS MAC and PHY layer networking devices. Note that in a switched network, single point of failures can be introduced by these switch ASICs. This also holds for several available Ethernet-Ring structures based on specialized devices [7], which we do not consider in this work due to the reduction in MTBF by the added, complex device. In a shared bus topology, the determinism, available bandwidth allocation and collision avoidance can be ensured by a fixed time-division protocol (either distributed or with a timing master) or polling the respective nodes. New 802.1 standard additions, such as IEEE 802.1Qbu, IEEE 802.1Qbv, IEEE 802.1CB, and others standards from the time-sensitive networking task group [8] might lead to new and better suited MAC and PHY devices in the future, enhancing real-time and reliability for standard Ethernet by a fair amount. In some applications, the use of other high speed interconnects like PCIe is permitted within certification constrains, which enables on-board diverse communication between NOM, MON and the output stages.

Distributed Monitoring Configuration

As we will discuss in the next chapter, the single-monitor configuration can present some certification pitfalls in certain domains, due to its inherent lack of hardware fault tolerance in a single board

[2] Note that the manufacturers provide extensive FMEDA and failure rate approximation data for certification on those devices

configuration. Therefore, we propose this second configuration, enhancing the previous one with hardware fault tolerance, while providing a fallback operating mode, on-board error location and masking capabilities for very high-reliability applications. As with the previous configuration, a single LRU can most likely not achieve the desired failure rates and availability constraints (due to the component count, printed circuit board failures, etc.), implying the use of multiple LRUs depending on application requirements.

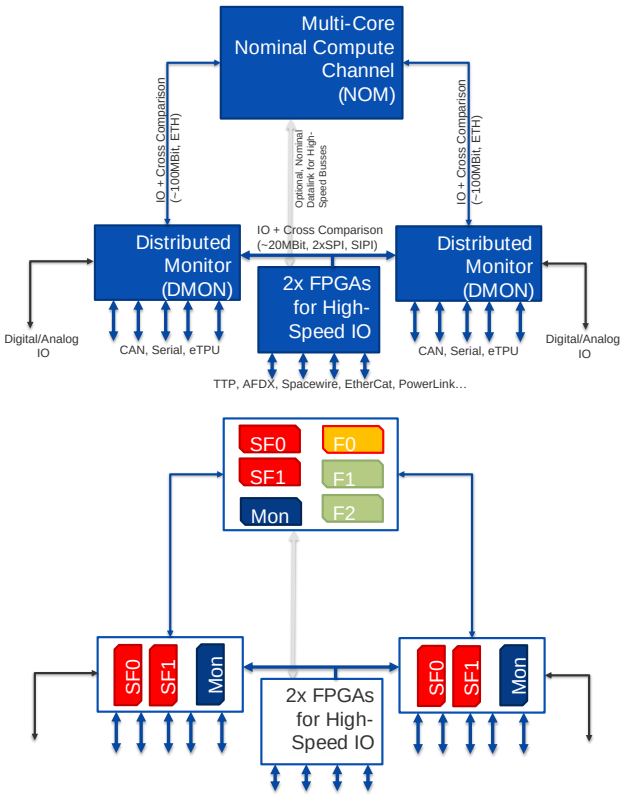


Figure 4: Distributed Monitoring Configuration. Upper: General Overview; Lower: Function Allocation by Impact Level (IL2 red, IL1 orange, IL0 green)

As shown in Figure 4, IL2 functions are executed on all on-board devices and are thus voted in each time frame. Only a valid computational result is allowed to propagate outwards, emitted either on a single output MCU with the remaining stages listening for correct transfer, or on all output MCUs for redundant transfer on the redundant data bus groups[3]. IL1 functions are voted within the redundant LRU cluster via the cross-channel data link (CCDL), but are not part of the on-board voting to ease the performance requirements on the monitor processors. The distributed monitoring MCUs (DMONs) also act as interface devices for legacy data bus and discrete interfaces (CAN, ARINCx, RS2x/RS4x, Digital I/O, etc.), which are grouped to two redundant data bus clusters in Figure 4, to mitigate physical layer failures occurring on the system level. Note that this architectural configuration is not limited to two DMONs, but can be further extended to provide more legacy interfaces and DMON compute power if needed. FPGAs provide high-speed CCDL connectivity to other LRUs in the cluster or to other system units such as sensors (camera, LIDAR, RADAR, IR, etc.) or actuation elements with high bandwidth requirements.

The example shown in Figure 1 originates mainly from small UAV avionics, where a high availability and error mitigation will be necessary for future systems, whilst minimizing weight and size, and therefore board space and number or redundant LRUs in the cluster. In such systems, the baseline control together with some automation functions (for example trajectory control, autopilot, waypoint navigation) do not require vast computational resources [4] and fit inside small safety MCUs, which allows for internal fault monitoring if the DMONs itself. Each DMON therefor execute extensive internal error detection and mitigation without compromising on computational resources. In general, the choice of DMON MCUs is depending on the IL2 function workload, since all functions must be run at least at three different nodes on the board to enable proper voting algorithms in a triplex configuration and cannot be executed on a single DMON and the NOM. In addition, the need for legacy interfaces, or the lack thereof if they are implemented within the FPGAs, limits the range of available MCUs or MPUs by a fair amount (CAN usually only found in great numbers on automotive MCUs). If, however, more than two DMONs are

[3] Note: This is already the case in today's systems, especially where the risk of structural damage is present. In such systems, busses are often grouped and routed independently on different paths in the system.

[4] Estimation on current implementations at the institute, subject to industry projects, exact functions and requirements cannot be disclosed publicly

employed, the issue might be resolved by further dividing interface groups and allocating them to the added devices, also providing addition common mode failure isolation.

The interconnects between NOM, DMONs, and the FPGAs must be able to cope with single, or even multi device failures in order to enable a fault tolerant system. As shown in Figure 4, all devices are thus interconnected by dedicated high-speed links from the NOM to the DMONs and the FPGAs, and separate links from DMON to DMON, via the FPGAs. This arrangement even allows for a double failure of dissimilar devices (NOM/DMON) in the system, which does not lead to any loss of IL2 functions, since the LRU can perform a fallback to an emergency operating mode with at least one DMON operational. With a single failure, full operating capability for IL2 functions is assured, which also holds for IL1 and IL0 if a DMON is affected and the NOM is still operational. Note that different IL0 function can be processed at each note in a redundant LRU cluster, while IL1 functions are executed depending on their desired degree of fault tolerance on different LRUs in a cluster, for example IL1_F{0..2} on LRUs A and B, IL1_F{3..5} on LRUs C and D for a per-function, single fault tolerant scenario.

Like in the single monitor architecture, NOM and DMONs are interconnected via dedicated high-speed links via Ethernet (100Mbit/s or 1Gbit/s, depending on the capabilities of the DMONs, large multi-core SoCs usually provide enough dedicated interfaces). Note that this can also be substituted by half-duplex links to save on high speed interfaces or compensate for the lack of Ethernet MACs on DMONs for a fast interconnect between the FPGAs and the DMONs. The bidirectional DMON-FPGA interconnect must be present, in order to allow access to high-speed data bus links from the DMONs to uphold IL2 output data in case of NOM failures and CCDL connectivity. Note that the designer is free to choose an appropriate interconnect at this level, considering the available DMON interface features. Usually, fast serial interfaces like Ethernet, SIPI via LFAST (inter-processor interconnect), or standard SPI/UART connections might be adequate if they offer enough bandwidth. However, parallel FIFO-type interfaces are implemented easily at the FPGA and the DMON to compensate for already allocated serial interconnects. Further hardening at the data links between NOM, DMONs and the FPGAs should be considered, in the form of black-channel protocols, in order to mitigate/detect random errors or failed board infrastructure.

Since this architecture already provides on board fault tolerance and error masking, it is applicable to low redundancy system configurations that require high availability with limited cost budget for physical units. In addition, added layers of degradation can be introduced to a redundant system with per-unit degradation paths. For example, a degraded LRU with a failed NOM or DMON might still participate in the inter-LRU voting for IL2 functions, provide output data to smart actuation units or conduct advanced error detection to gracefully recover a degraded LRU by restarting and resyncing specific processors. Again, this is dependent on the targeted certification context, the physical system architecture, other safety precautions, safe state type or reachability, etc. and must be discussed early on with the respective certification authorities, especially when different levels or system function degradation strategies are within the scope of development. The ability to degrade along such paths is not supported by current generations of LRUs in the field, and offer a significant improvement for availability and multi-point fault tolerance for future system architectures.

Note that we do not use the multi-core to achieve multiple, redundant compute channels on a single device, since common mode failure scenarios via the SoC busses, memories and peripherals effectively deny such designs, while heavily relying on an in-depth SoC analysis and test during certification. We also do not advocate redundancy by multiple executions of a function within a single time frame, due to the persistence [5] of transient failures. All functions are executed once per system time frame on the device, allowing maximum device utilization without sacrificing performance to the overhead of redundant executions.

[5] Transient failures can usually only be resolved to a deterministic state when a device reset is issued to resolve SEU-based upsets in internal circuitry. Memory is protected by ECC or spatial redundancy. Since device restarts are non-critical, a clean device state via restart is preferred.

Software Architecture Considerations, Common Circuitry and Interconnects

Since we must provide a software infrastructure on the multi-core which provides the means for mixed criticality functions running in parallel on the device, we propose an AMP architecture on the multi-core, with one dedicated master core, carrying out supervisory tasks as well as IO handling.

As shown in Figure 5, the master core in in control of the device's peripherals, manages their configuration and periodically checks their status (health, status, configuration). The master core (c_0) also manages the internal inter-core communication framework (ICC, via shared memory), slave core software watchdogs, external MON watchdog communication and system supervision, as well as the resource separation between the slave cores for temporal isolation of mixed critically functions. Spatial separation and access right control is ensured by core-local and SoC level functions such as MMUs, privilege levels, System-Memory Protection Units, and a dedicated peripheral controller core, which is c_0. The external watchdog, implemented by the MON or each DMON[6], should be a windowed, challenge-response signature watchdog, with a random challenge computed by the MON. The challenge is issued each system time frame, to check the status of the multi-core device and the MON (bidirectional verification). On master (c_0) and slave cores (c_i), a real-time operating system is instantiated, to control the allocated tasks via a per-core task list, which can also be managed or modified dynamically[7] by the master to further supervise the execution of tasks in degradation scenarios. Within each slave, a watchdog instance embedded in the RTOS ensures the detection of a stuck program execution, managed by c_0, typically executed via a challenge issued at each system time frame via ICC. If the application requires specific control of dedicated hardware features, like video ports or accelerators for image or video processing applications, a slave core may contain a subset of the peripheral driver authority. Sharing of peripherals may be permissible in special cases, but separation on a per-peripheral basis of different functions (e.g. cores they are executed on) may not be achievable with current generations of devices (fine graded permission control or prioritization not supported on the SoC).

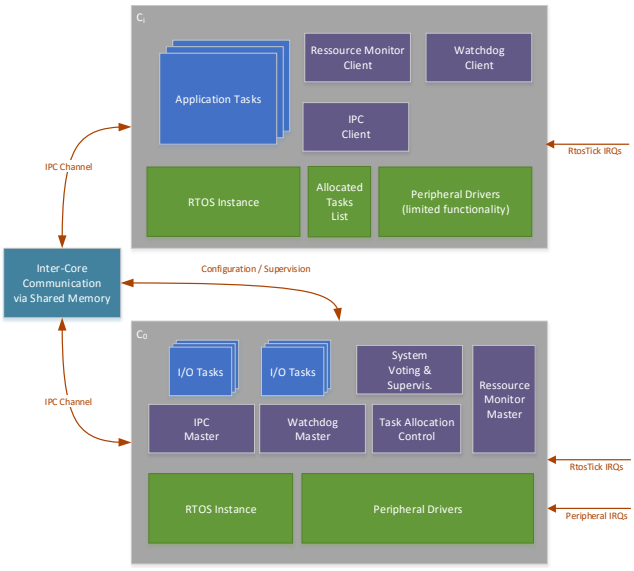


Figure 5: High level software architecture on the multi-core SoC

Note that an ICC client is executed on each slave core, to process/issue ICC request which can also be embedded in the RTOS layer. Static task allocation may be necessary to meet determinism requirements, but in certain circumstances, a SMP operating system spanning multiple cores can be used for IL0 or IL1 functions, if c_0 still has full authority and the SMP OS can be super- or hyper-vised on the respective cores by build-in privilege level. Under all circumstances, core and task level separation must be guaranteed to meet certification requirements. While spatial separation is guaranteed like stated before, and can be analyzed and tested during validation phase and run-time, temporal isolation of cores and tasks is not straight forward and requires special precautions. We propose to use the available core or SoC level performance counters, to isolate the last remaining shared resource in the system, the memory subsystem. Shared cache should be partitioned or allocated to resolve determinism issues, arising from the interference by other cores. In most core architectures, this feature is implemented in the form of cache line/cache way lockdown or totally separated core caches (L1 and L2 per core), but

[6] A true, separate watchdog only reduces availiability and does not contribute to error detection or effectiveness of the watchdog action. We re-use the already present dissimilar device for this purpose.

[7] If permitted by applicable certification constraints, in general, a static, tested and deterministic scheduling is preferred for IL2 functions. Requirements for IL1 and IL0 functions may vary and permit a dynamic scheduling.

controlling the number of memory transactions per core within a given timeslot requires constant monitoring due to the lack of specific function units working on the performance counter data. Using the performance counters (PMUs), an application is profiled based on its required nominal memory transactions, on a per OS-tick time scale, which is usually available at a high frequency, allowing fine graded failure detection[8]. In the local OS-tick, each slave core collects its performance counters, and transfers their content via ICC to c_0. The master core than processes these counter values, comparing them to a predefined worst-case or a static resource allocation lookup-table built during test and verification phase, based on debug trace data or formal techniques. If an overconsumption is detected for the past tick period, the respective core is signaled to temporarily schedule out the task, to reduce memory transactions until conformance to the predetermined profile is again established[9]. This can also result in a deactivation of the task in question for a system time frame, to allow for deterministic computation of IL2 and IL1 functions. An appropriate action for IL2 function failures must be defined, because a failure is critical in the single monitor configuration. The non-distributed monitor configuration offers additional protection due to the availability of two other redundant output results from the DMONs and possible system level LRU redundancy. A similar approach has been applied in [9], [10] or [11] why we spare the formal background at this point and refer to these references. While the temporal isolation is essential for WCET guarantees, the containment also serves as an excellent mean to control rouge cores with crashed tasks who issue memory transactions in an uncontrolled, faulty manner due to transient, permanent, or programmatic design errors. An in depth investigation is future work on actual hardware. Exemplary sequence diagrams can be found in Appendix II for the single and distributed monitor case together with an exemplary failure mode behavior for each configuration for an overall overview of internal and external communication steps. Note that we specifically do not recommend a specific hypervisor or RTOS architecture in this work, since the overall architectural configurations, especially with distributed monitors, allow for a multitude of possible detailed architectures, which are out of scope due to their system and application domain specific parameters. Further mitigation strategies, like core restarts of detected transient core failures on the multi-core NOM, or deactivation of such cores are also subject to project specific architectures.

Shifting focus, other board level subsystems are also of great importance, especially for the distributed monitoring architecture. In order to allow fault isolation if a processor exhibits a permanent fault in form of a latch up, the possibility exists that the device fails in a “short to ground” manner, driving a power supply circuit into current limit and shut off in worst-case conditions. To isolate these failures on the board level, local power supply for each critical subsystem (NOM, MON, etc.) must be established, with as few common circuitries as possible for maximum separation. In addition, power inputs to the unit must be redundant to remove power input single point of failures from the system (connectors, wiring, fuses, filters, etc.). Equally, bus transceivers should not be re-used on the board level, and need to be laid out fully redundant for each output stage, being FPGAs or DMONs peripheral outputs, in order to avoid common mode failure scenarios, especially in the distributed monitor configuration.

Outside the LRUs, high-speed data bus interconnects are used for the CCDL and high bandwidth sensors. While different options exist, high-speed interconnects are usually domain specific like AFDX, TTP, and FireWire (USAF F-35) in the aerospace or PowerLink, EtherCAT, etc. in the industrial domain. The use of such standards also entails the availability of sensor and actuation units capable of these interconnect standards, which is while we will not focus on a specific design based on those standards in this work. With the CCDL, however, the designer is usually free to choose with a higher degree of freedom since the CCDL is seldom exposed outside the compute platform. The requirements for a CCDL differ based on the bandwidth needs to for the nominal data transfers (voting, supervision), LRU resync speed

[8] Very Important: Most RTOS already offer OS-tick hooks for easy and practical insertion of custom routines, without modifying an already certified kernel, which is no option at all for a live project.

[9] Due to no execution on the respective core, no further memory transactions are generated by the task/function.

in case a LRU requires state data, mostly for IL2 and IL1 algorithms after a fault-triggered reboot, or a time synchronization requirement for synchronous systems. Bandwidth requirements for resynchronization reach from a few states (a few hundreds of bytes), but may easily exceed thousands even for baseline control (Megabyte region) in the case of sophisticated control architectures. New types of CCDLs are subject to ongoing research, but a viable candidate already exists in the form of SpaceWire (SpW, [12]). SpW offers a robust, high-bandwidth physical layer with minimum constraints on the upper protocol implementations, while providing sophisticated time synchronization, on-line fault tolerance when multiple routes to connected units are possible and the ability to operate in ring, switched, or peer-to-peer networks. Figure 6 shows an exemplary LRU cluster, interconnected by a fault tolerant SpW-CCDL. In case of an interconnect failure between two nodes, data packages can be re-routed by using intermediate, redundant physical connections or traversing the ring in another direction. Since SpW is an open standard, open-source implementations are available free of charge, offering a cost-effective solution to be implemented in the FPGAs in form of three or five port bus interfaces. Due to the specific design for harsh space environments and the requirement-based standard, SpW is well suited and will be part of the future demonstrator platform.

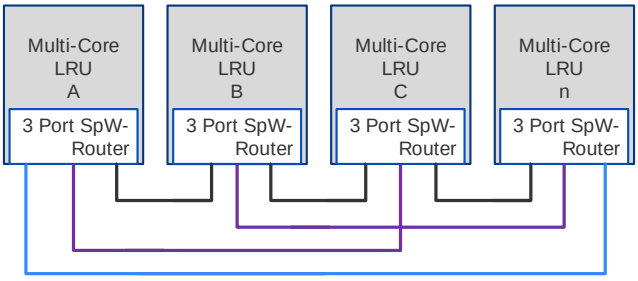


Figure 6: CCDL via SpaceWire. Minimum interconnects (black), ring closure (blue), and optional failover links (purple) via minimalistic 3 port routers (FPGA-based)

In the previous chapter, we derived a set of requirements for the individual LRU (Table 2, Appendix I), which we will now compare against the proposed architectural configurations. RQ1-5 will be discussed in the next chapter within their specific context.

RQ ID	SMA	DMA	Remarks
6	X	X	SMA only detection, DMA with mitigation
7	X	X	Passivates itself if number of allowable errors exceeded (SMA=1, DMA>=2)
8	(x)	X	SMA: Possible with special monitor for error localization
9	X	X	Shutoff possible with power removal
10	(x)	X	See RQ8
11		X	
12	X	X	
13	(x)	X	SMA: Only with special monitor to detect latent monitor faults ahead of time
14	X	X	Possible with both configurations
15	X	X	NOM/MON Watchdog, Inter-Core Watchdog, Spatial and Temporal Isolation
16	X	X	SMA: Via FPGA, DMA: primarily via DMON, also possible via FPGA
17	X	X	Via FPGA
18	(x)	X	NOM/(D)MON are dissimilar by design, DMONs may be dissimilar. LRUs can be constructed with dissimilar parts in a cluster. This is pointless since on-board dissimilarity between NOM and MON is already given. SMA might not be suited as good as DMA, since no true fallback operating mode is provided when a non-safety type MCU is used for the monitor.

Table 1: Requirements fulfilled by presented configurations. X: fulfilled, (x): fulfilled under certain conditions.

Safety and Certification Aspects

In this chapter, we are going to discuss certain aspects of certification concerning the presented architectures. After domain specific considerations, we focus on common issues such as WCET. Note that we will use the terms defined by the respective standards without further definition. The reader is referred to [4], [1], [13], [14], [2] and [15] for the terms and abbreviations used in this section.

Industrial Certification (IEC/TR 61508, EN ISO 13849)

In our requirements, we defined SIL2/KAT2,PL.d as the certification level for the single unit, with the possibility for SIL3/KAT3,PL.d in certain arrangements. This leads to $90\% \leq DC_{avg} \leq 99\%$ in order to comply with SIL3/PL.d levels or $DC_{avg} \geq 99\%$ for PL.e. We followed the common procedure of completing a fault tree analysis (FTA) as well as a markov analysis (MA) for DU-failures (see Appendix III). Both analysis were conducted in FuSaSu v3.3.1 [16].

As the markov analysis shows, the calculated DC is above 99% in both configurations, resulting from the cross-verification which only fails in case of an already present or latent fault in the MON if also the NOM fails, while the fault is still present or not yet detected. Since this already involves two distinct failures in dissimilar hardware devices, which, if different core architectures (ARM, PowerPC, MIPS, etc.) are used, also entails software dissimilarity at the binary level[10], is extremely unlikely as shown in the markov analysis and the fault tree. When distributed monitoring is employed, the system is further hardened, as shown by the slight increase in DC for the distributed monitoring architecture. A HFT of one is given in any case by the distributed monitoring configuration (DMONC), while HTF=2 can be claimed when certain assumptions hold on the system level (multi-core can still access some IO via a bypass in case of complete monitor failure). The non-distributed monitoring configuration (SMONC) can only be considered as HFT=1, if a MCU with integrated error detection to unveil latent and transient faults is used, in order to provide the described degraded/fallback operating mode with reduced operational capabilities.

In IEC 61508-2, several tables are listed to guide the applicant with adequate measures for diagnostic techniques, testing techniques, controlling systematic faults, etc. In Table A.3, “Monitored Redundancy” and “Comparator” apply to both configurations, while the “Majority Voter” technique applies to the DMONC. In Table A.7, for I/O units, “multi-channel output”, “monitored outputs” and “input comparison/voting” apply in both configurations. While Table A.8 is deals with internal device elements, one could argue that transmission, information and complete hardware redundancy (due to the separate interconnects between the I/O elements and the NOM/MON(s)) is already present on the board level, and is therefore accomplished for the function executed on both the NOM and MON. The program execution, either on a NOM or MON is permanently monitored, which leads in conjunction with the cross verification to the “Combination of temporal and logical monitoring of program sequences” in Table A.10 being fulfilled. The program sequence is not only monitored by the NOM/MON combination (watchdog), but also by an internal watchdog on each device itself, either in hardware or by a hardware/software combination when the multi-core is considered (to detect stuck cores). This is required by Table A.15, where also “Failure detection by on-line monitoring”, “Tests by redundant hardware” and “Diverse hardware” is satisfied by our architecture. The nature of the presented on-line tests and techniques leads to a high effectiveness for detecting errors, according to Table A.18. Moving on to Table B.5 for validation techniques, no method is explicitly excluded due to technical reasons in the proposed architectures. “Fault insertion testing” might provide an adequate method to extensively validate the error detection and masking functionality, alongside a multiple condition / decision coverage for the associated software.

In terms of the ISO 13849, we provide single fault tolerance with both configurations to satisfy one of the central KAT3 requirements, while DMONC also provides the means to mitigate certain multi-point faults, which must be considered for a KAT4 approval.

[10] Note that N-version programming is also a viable option when within project budget limits and can lead to common cause software error mitigations

Interestingly enough, SMONC provides the standard's proposed architecture for KAT4 for IL2 functions, since we conduct a true cross verification between NOM and MON. DMONC extends this approach further, in a tipple modular redundancy fashion. Like shown based on the IEC 61508 tables, the measures inherent to the proposed configurations justify a high $DC_{avg} \geq 99\%$.

During a standard certification of single core SoCs, a device-level FMEDA usually provides the necessary failure modes at chip level for higher certification levels. While a device-level FMEDA is obtainable from the chip vendor for certain devices specially targeted at the safety market, no or very limited data is provided for COTS devices, especially multi-core SoCs, due to their origin in other domains like telecommunication or server computing. This data is necessary however, if the device is used explicitly to provide a safety function and must therefore be analyzed accordingly to the desired SIL/KAT. In our proposed configurations, every possible device failure mode, manifesting in either wrong output data, no output data or untimely output data from the multi-core SoC, is detected by the cross comparison with a second device in each system time frame, or by the even more trustworthy voting with more than one DMON. Under these conditions, a true FMEDA can be omitted (saving the effort with/by the chip vendor), if transient and permanent failure rates can be provided, which are nonetheless required to compute the standard specific metrics (MTBF, $MTTF_d$, DC, etc.), in conjunction with a FTA and/or markov analysis to show the dangerous detected and dangerous undetected system states. This resembles more or less a grey box approach[11].

Graceful degradation in case of failures can be argued, if the desired application permits for example the classification of system functions in the presented impact levels, and is vital for the distributed monitoring configuration, where different degradation paths can be designed depending on the final application. This is especially useful to provide conservative mitigation paths, where for example only IL2 functions are upheld when for example an error on the multi-core has been detected, while disabling IL1 and IL0 functions until the system arrives at a safe state.

One reason, why hardware redundancy (cross verification) is preferred over simple on-line monitoring, is the occurrence of latent faults in devices or systems. The potential presence of latent faults also denies a degraded, single device operating mode with an SMONC. Latent faults can be detected either by special on-line tests, consisting of software and/or integrated hardware (special hardware included in the SoC, like with safety MCUs). If adequate hardware support is present, as with safety MCUs, on-line tests for latent faults can be triggered periodically, which may be destructive (in order to trigger all software and also hardware elements involved in the error reaction path) in the MON. This potentially involves a restart of the device and therefore unavailability for the on-line cross verification during the MON downtime. DMONC can be tested while always providing redundancy for IL2 functions when the NOM is restarted, or one of the DMONs is tested, while IL1 and IL0 functions become unavailable on the LRU when the NOM is shut-down or rebooted. This enables non-destructive run-time tests on the DMONs, resulting in a great benefit of this architecture, since unavailability is reduced, even in the case that only a single LRU is present in the system. Furthermore, if safety MCU type DMONs have been selected with special internal hardware units to test internal units or detect errors at run-time, a processor or SoC-wide BIST can be executed periodically, resulting in a SoC restart and loss of internal state (destructive test). These tests are usually executed at startup in current applications, mainly in the automotive domain, but can be used in our configuration in a non-destructive fashion during runtime, greatly enhancing the detection of latent faults in the DMONs.

Common cause failure scenarios are mitigated effectively by hardware dissimilarity between the NOM and the MON/DMON devices, as required in the highest certification levels. Note that the potential for CCFs exist in the DMONC, if both DMONs consist of the same devices, which could in theory lead to a double failure and an associated DU board level failure. Proper isolation of the DMONs is therefore

[11] Note: To define the application context in current standards, device internal function units used must not only be described in the aerospace domain, but also for industrial certification. If they are used to detect/mitigate errors/failures, they must be analyzed and tested accordingly.

critical, in terms of power supply, physical board location, cooling, etc. to reduce common effects on those devices if possible. To eliminate this concern entirely, dissimilar devices should be used for each DMON, but this leads to additional development effort and project complexity and should be discussed very early in the design process with certification authorities.

Aerospace Certification (ARP 4761/4754, DO-254/178)

Compared to the industrial standards, the respective aerospace guidance for hardware (DO-254) and software (DO-178) are rather vague on COTS items. As a result, with more and more requests from the industry, two documents have been issued by the EASA and the FAA in order to provide additional guidance on COTS parts and especially on complex COTS devices. These documents, the CAST-32 by FAA and the CM-SWCEH-001 (hardware aspects) and CM-SWCEH-002 (software aspects) by EASA, either present their current position on on multi-core processors (CAST-32 is limited to two cores, more than two cores not recommended) or provide additional guidance on highly complex COTS parts in general. We will therefore examine the architecture in the light of the CM-SWCEH-001 (SWCEH from here on). The certification memorandum defines 16 activities which are applicable depending on the DAL and the product service experience (PSE). Since our goal is to justify the use of a single board up to DAL C, a LRU cluster of two units up to DAL B and a LRU cluster of more than two units up to DAL A, resulting from potential FDAL A functions being executed on the multi-core, all SWCEH-activities must be executed and documented, since low/none PSE can be provided for any multi-core SoC and highly-complex classification (see CM-SWCEH-001, Section 9.3.13, device classification acc. to activity 1). Relevant from a technical point of view are activities 1, 4, 5, 12, 13, 15 and 16. We will ignore process and documentation activities within this work.

Besides the hardware and software specific guidance, the ARP4754 and ARP4761 must be followed for most certification efforts to achieve compliance to the specific certification specification. Extending the ARP4754 FDAL/IDAL classification, we added the impact level definition, which also allows the use of the standard FDAL and IDAL definitions within each category to remain compliant to existing regulation. SMONC and DMONC do not contain a single point of failure, as shown by the FTA and markov analysis in Appendix III. Again, neither the multi-core SoC, nor the MON or a single DMON is on a critical failure path. Nevertheless, harsh environment conditions entail at least two LRUs from DAL B onwards, due to the possibility for mechanical printed circuit board or component failures under vibrations, shock or thermal stress. The derived board level FTA and markov graph can further be used to extend the system/function fault trees. While it is possible to show with these analysis, that any failure of the multi-core is non-critical in general, a black-box approach to avoid certain activities for the multi-core is only permitted when field-based evidence[12] on failure rates and the failure modes[13] is available, or can be estimated in form of an FMEA of the device. The device level FMEA should provide transient and permanent failure rates, for each function unit of the SoC, in order to calculate a resulting total event probability in the application environment, based on the used functions[14]. In this context, a very pessimistic assumption is required to justify activity 12, in which we assume that every internal failure in the multi-core will lead to a wrong result, no result, or untimely result condition of this complex device. This is necessary, because sufficient independence of internal function units within the multi-core cannot be claimed under any circumstances, due to the extreme complexity[15] of these SoCs by the semiconductor manufacturer. Likewise, deactivated units must be checked periodically, to ensure that they remain disabled, by disabling power, clock or both and restrict bus access for those units[16] when possible and applicable.

Extensive architectural mitigation is performed on the board level, as suggested by activity 15, removing the multi-core device from any direct failure path in the

[12] SWCEH activity 13, sufficient PSE for DAL A after $>10^5$ hours in aircraft use

[13] Base events in the FTA and markov analysis

[14] Identification and justification of these units required by SWCEH activity 4

[15] Multi-layer routing on the die; no isolation substrates between function units, cores, busses due to timing constraints; etc.

[16] Internal, complex function unit can access memory as a bus-master, for example complex communication peripherals or graphic processor units

case of single faults. Multi-point failures can be mitigated by DMONC, if both DMONs do not fail simultaneously, as shown by the FTA and markov analysis. Note that it is possible to mitigate common mode failure scenarios by dissimilar multi-core or monitoring devices within each LRU in a redundant cluster. We will not explore this path any further in this work.

Activity 5 results in a great impact on documentation and analysis, in order to comply with the described actions. The first part, out of three, of activity 5 is accomplished by not relying on any internal function units to ensure safe operation of the multi-core device. Note that internal units will be used to ensure determinism and partitioning on the multi-core, but not on the board level, effectively canceling their effect on the safety of the LRU. The second part recommends that extensive validation should be performed, which is possible in the proposed architecture with fault injection, run-time trace (offered by most modern multi-core SoCs) and other test strategies. The third and fourth paragraph of activity 5 require a substantial development effort. The exact use case must be documented with great depth, accompanied with an in-depth device documentation and analysis. However, this can be achieved, given our proposed architecture and architectural mitigation strategy, with very little impact on overall system safety and certification outcome, since the multi-core is no longer in a critical failure path and only degrades the overall availability of the system in a realistic worst-case scenario. The exact nature of the assessment of activity 5, will most likely be subject to extensive discussions with the authorities in a live project, but could be reduced to a documentation task with limited certification risk given the architectural precautions and mitigations.

WCET and Mixed-Criticality Partitioning on the Multi-Core SoC

Common to both the industrial and aerospace domain is the need for strong determinism and partitioning on the multi-core device, especially with applications with mixed criticality levels running concurrently on the same device[17].

In order to overcome the WCET and partitioning issue in the certification context, we suggest the method proposed in [6], [9] and [10] , which uses a single core equivalence approach to determine WCET in the contention scenario on the shared bus, and temporal isolation via software-driven, per-core memory bandwidth management. [6] and [9] provide a good overview of related work and techniques explored. Partitioning is especially needed in the mixed criticality scenario, where less well tested applications with lower SIL/DAL levels execute alongside highly critical functions, due to the failure mode of non-critical functions contending the shared memory or shared interconnect of other cores for critical functions[18]. In terms of internal function units, the proposed method only relies on the core local performance counters, which become safety relevant if used for this purpose. Therefor these counters have to be verified for correct operation during design, with the associated documentation provided. Likewise, to detect latent faults, a dynamic run-time validation must be conducted in regular intervals (depending on the application domain). This check is rather simple and non-destructive, by logging the current event counter value, conducting a defined number of events to be logged (e.g. memory transactions, cache misses by accessing non-cached regions, etc.) and comparing the resulting counter value with the expected result. When augmenting the proposed method in [10], with a per-function execution profile like we proposed, tighter bounds can be placed on the detection time, much like with [17], where this approach is used to detect malicious instructions for a security reason, where we propose this method to also provide unspecified program flow detection for safety reasons.

In [6], the properly isolated cores are then used to derive a WCET analysis, conducted on the single core under the assumption that a pre-allocated memory bandwidth is available per core which bounds the WCET in the contented SoC. In previous generations of safe systems, WCET analysis was either conducted statically by formal methods and verified during test (tool support available) while triggering the identified worst case memory consumption or timing path on the real system within a measurement/trace setup or hardware-in-the-loop arrangement. Likewise, WCET

[17] SWCEH activity 5 and activity 16, IEC 61508-3 Annex A and F

[18] Not only due to SEU, but also related to undiscovered design errors on less well tested, non-critical functions.

analysis can be conducted using the single core equivalence approach in the same manner, while worst-case profiles on each core are generated during the validation phase. These profiles ensure determinism in the field, even in the case of SEU introduced rogue cores or undiscovered software design errors in less stringent certified software on other cores, by allowing the RTOS instance to suspend the execution of non-critical tasks on a core or prematurely detecting abnormal program execution symptoms for critical functions. Note that, current generations of multi-core SoC already feature extensive time and data trace capabilities via dedicated interfaces (Nexus AURORA, ARM ETM/ITM, etc.) which can be exploited to conduct precise on-target measurements with an unmodified binary image to provide further confidence during validation of the design.

Conclusion

Although no guidance is currently provided on the certification of high-complex, COTS multi-core SoCs, a suitable architecture can be derived, which fits into current regulation and builds on the simplest concept possible. If one cannot be sure, that a part, device or system can function without any failures under any conditions, another system, or group of such, must provide the means necessary to detect the failure and mitigate its effects. The failed complex system itself cannot under any circumstances be used to detect his own failure, since it has already failed[19]. Note that this approach also denies all possible pathways to use different cores of the COTS multi-core as redundant compute channels to mitigate errors outside another core's local elements, due to the common failure modes.

During the course of this paper, we presented two architectural configurations to accomplish compliance with current regulatory frameworks for certification of COTS multi-core SoCs. The first approach utilizes a simple centralized command-monitor pattern, with redundant output paths, to detect failures via cross comparison and is especially useful when the application already requires more than one LRU, or even higher numbers of units distributed throughout the physical system. The second approach distributes the monitoring and incorporates the low-speed interfaces into these units, providing on-board voting, error masking and a hardware fault tolerance build in on the LRU level. It is designed for physical systems with stringent size, weight and power requirements, where a large number of LRUs is either not feasible or cannot be justified due to commercial constraints.

System functions have been categorized into separate impact levels, to reflect their impact on the system behavior in case of wrong computation results and complete loss of functions. This classification is used to distribute functions according to their level to the on-board cross comparison or voting for on-board error detection, localization and masking, or to inter-LRU voting in a cluster of redundant units.

The proposed high-level software architecture provides fast error detection within the multi-core device and to separates functions of different criticality levels executing on different cores, with little computational overhead and complexity. Associated software and measurements are future work on actual hardware.

With a FTA and markov analysis, we showed that no LRU failure mode could directly be associated with an error inside the multi-core device, resulting from the board level architecture, while addressing certain requirements by domain specific standards and guidance. As a result, we propose that certification of systems based on COTS multi-core processors is possible, even with current regulatory frameworks and the inherent hesitation of certification authorities.

Related Work

Academic contributions to the topic mainly focus on either custom function unit integration into multi-core ASICs or entirely purpose build devices, like [18], [19] or [17], in order to either enforce, monitor or enhance determinism and core decoupling. While they provide guidance for future generations of multi-core SoC design, these works do not explicitly address certification strategies and shortcomings of their designs, and are also of scope for almost any application, due to high cost of custom ASICs. Other academic works focus on WCET calculus ([20], [6], [10], [11], [9]), while also not explicitly considering design constraints due to certification, in terms of possible common failure scenarios, where the correct

[19] Special safety MCUs excluded

function of any core in the SoC can no longer be guaranteed by a line of code, and the certification aspects when internal function units are used to guarantee certain safety claims.

Future Work

In our future work, we will assemble a testbed for the DMONC and conduct measurements on the error detection and mitigation methods described, as well as the watchdog implementation. A certifiable, industry standard RTOS microkernel will be used on the multi-core in multiple instances to implement the proposed the software architecture. The institute of flight dynamics will provide a certifiable flight control application suite with baseline control, autopilot software, on-line flight path planning and automatic landing based on augmented visual data to serve as a benchmark with a certifiable, model-based developed application collection. The final timing and validation verification example will be integrated in one of the existing hardware-in-the-loop aircraft system simulations existing at the institute.

Acknowledgement & Disclaimer

This work was funded by a doctoral research grand, funded by MicroSys Elecronics GmbH. The author thanks the FEA and QorIQ team from Freescale Semiconductor (future NXP Semiconductor) for the insight into current and future devices, as well as Mr. Ramold and his group at TüV Süd Rail GmbH for the health discussions and critical reviews on the architectural configurations and certification aspects. Any opinions, findings and conclusion, especially in the terms of certifiability, do not necessarily reflect the views of TüV Süd Rail GmbH.

References

[1] "Functional safety of electrical/electronic/programmable electronic safety-related systems (IEC 61508-x:2010)," International Electrotechnical Commission, 2011.

[2] ISO - International Organization for Standardization, „ISO 13849-x:2008 Safety of machinery - Safety-related parts of control systems," ISO - International Organization for Standardization, 2008.

[3] C. A. S. Team, "Position Paper CAST-32 Multi-core Processors," May 2014. [Online]. Available: https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/media/cast-32.pdf. [Accessed 10 November 2015].

[4] EASA, „CERTIFICATION MEMORANDUM - SWCEH - 001 Issue No.: 01," EASA, 2011.

[5] Thales Avionics, „EASA 2011.C31 MULCORS Project. The Use of MULticore proCessORS in Airborne Systems, CCC/12/006898-Rev.07," EASA, 2011.

[6] R. Mancuso, R. Pellizzoni, M. Caccamo, L. Sha and H. Yun, "WCET(m) Estimation in Multi-Core Systems using Single Core Equivalence," 27th Euromicro Conference on Real-Time Systems (ECRTS), 2015.

[7] Micrel, "EtherREL™ Network Fault Recovery," Micrel, [Online]. Available: http://www.micrel.com/index.php/etherrel.html. [Accessed 13 November 2015].

[8] TSN Task Group, "Time-Sensitive Networking Task Group," IEEE 802.1, [Online]. Available: http://www.ieee802.org/1/pages/tsn.html. [Accessed 13 November 2015].

[9] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo and L. Sha, "Memory Bandwidth Management for Efficient Performance Isolation in Multi-core Platforms," IEEE Transactions on Computers, 2015.

[10] "MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-core Platforms," Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013.

[11] J. Nowotsch, M. Paulitsch, A. Henrichseny, W. Pongratzy und A. Schachty, „Monitoring and WCET Analysis in COTS Multi-core-SoC-based Mixed-Criticality Systems," Proceedings of the Conference on Design, Automation & Test in Europe, 2014.

[12] ECSS / ESA-ESTEC, „ECSS-E-ST-50-12C - SpaceWire - Links, nodes, routers and networks," ECSS, 2008.

[13] SAE Aerospace, „ARP4754-A GUIDELINES AND METHODS FOR CONDUCTING THE SAFETY ASSESSMENT," SAE Aerospace, 2010.

[14] SAE International, „ARP4761 Guidelines for Development of Civil Aircrat and Systems," SAE International, 1996.

[15] SC-180, „DO-254 Design Assurance Guidance For Airborne Electronic Hardware,“ RTCA, 2000.

[16] T. Brunnengräber, „Functional Safety Suite,“ Thomas Brunnengräber, [Online]. Available: http://www.thomas-brunnengraeber.de/fusasu.html. [Zugriff am 13 November 2015].

[17] M.-K. Yoon, S. Mohan, J. Choi, J.-E. Kim und L. Sha, „SecureCore: A Multicore-based Intrusion Detection Architecture for Real-Time Embedded Systems,“ Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013.

[18] J. Perez, D. Gonzalez, C. F. Nicolas, T. Trapman und J. M. Garate, „A safety certification strategy for IEC-61508 compliant industrial mixed-criticality systems based on multicore partitioning,“ 17th Euromicro Conference on Digital System Design, 2014.

[19] B. Motruk, J. Diemer, R. Buchty, R. Ernst und M. Berekovic, „IDAMC: A Many-Core Platform with Run-Time Monitoring for Mixed-Criticality,“ IEEE 14th International Symposium on High-Assurance Systems Engineering (HASE), 2012.

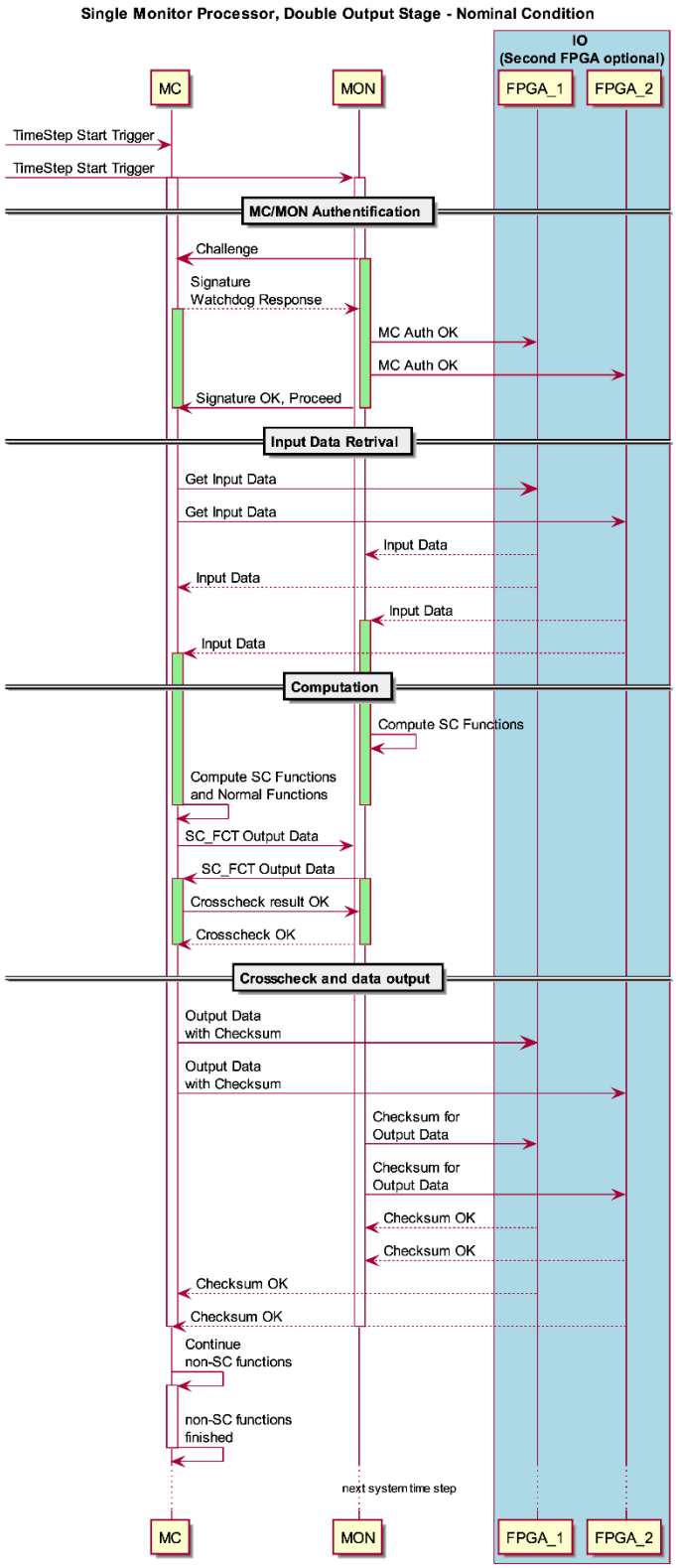
[20] J. Bin, S. Girbal, D. Gracia Pérez, A. Grasset and A. Merigot, "Studying co-running avionic real-time applications on multi-core COTS architectures," International Conference on Information and Communication Technology for Embedded Systems (ICITES'14), 2014.

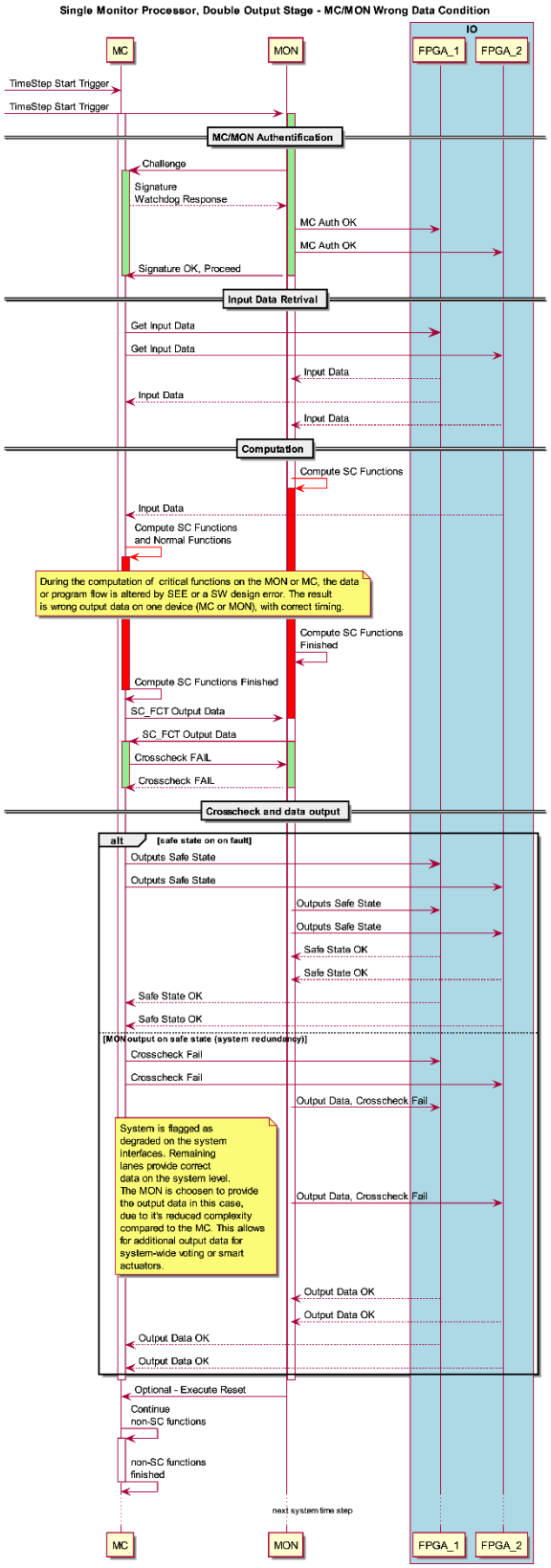
Appendix I

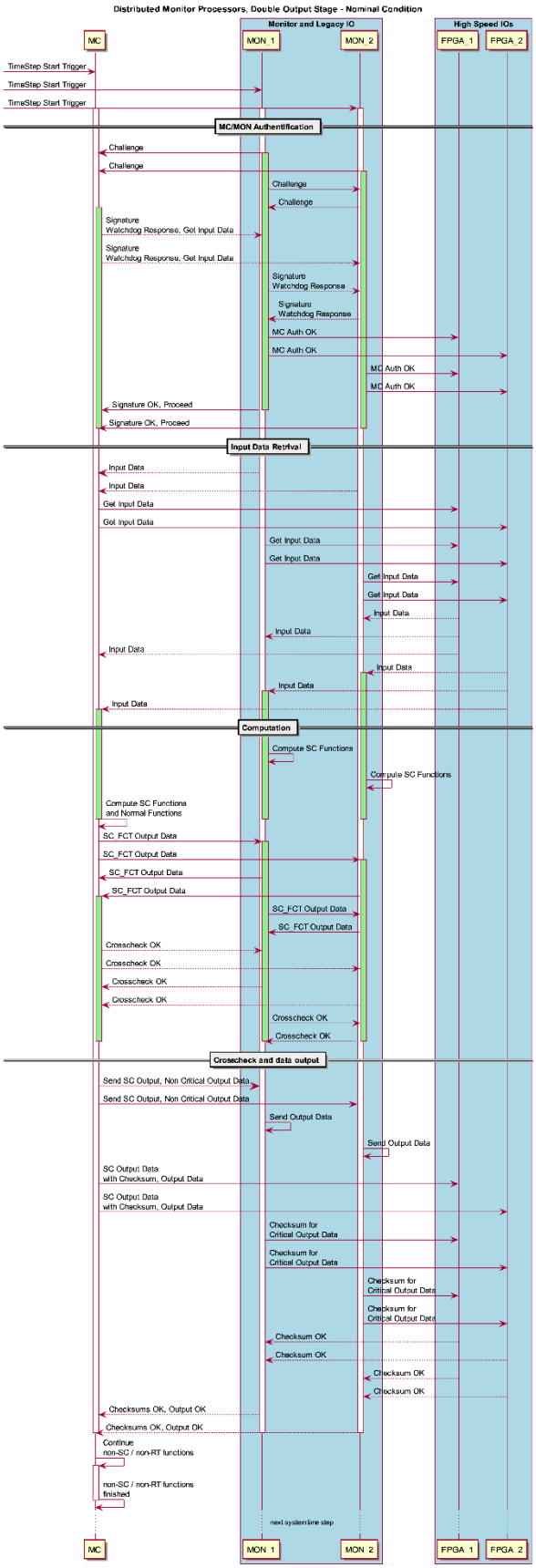
ID	UC1	UC2	Description
RQ1	x		The compute platform shall satisfy a safety claim of up to SIL2 / KAT3 (Pl.a-d) with a single LRU, Safety Claim of SIL3-4/KAT4 (Pl.d-e) depending on system architecture and other failure mitigation means on system level
RQ2	x	x	The compute platform shall satisfy a safety claim of SIL4/KAT.4 (Pl.e) with two LRUs or more
RQ3	x		The compute platform shall be certifiable in accordance with DAL C using one single LRU
RQ4	x		The compute platform shall be certifiable in accordance with DAL B with two LRUs maximum
RQ5		x	The compute platform shall be certifiable in accordance with DAL A with at least three or four LRUs (1oo3/1oo4/2oo5/2oo6, etc.)
RQ6	x	x	The LRU shall provide on-board fault detection / diagnosis / functional monitoring to detect wrong results of critical functions (IL2). A faulty computation result of any IL2 function shall not be distributed by the LRU.
RQ7		x	The LRU shall provide a fail-passive / fail-silent safe state in a redundant configuration.
RQ8	x		The LRU shall provide means to reach a safe state via separate system mode transitions requiring complex actions.
RQ9	x	x	The LRU shall be able to maintain the safe state.
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.
RQ11	x		The LRU shall satisfy ISO13849 KAT3 single fault tolerance.
RQ12	x	x	The LRU shall detect latent faults in its nominal compute channels by online monitoring or cross comparison.
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).
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.
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 in a safe way. The partitioning of different functions must be guaranteed at all times.
RQ16	x	x	The LRU shall provide legacy interfaces such as CAN, RS232/485, ARINC429 or similar databus communication interfaces to support existing infrastructure, sensors or actors.
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.
RQ18	x	x	The LRU shall be designed to mitigate common mode failure, with regards to common complex COTS parts used within the LRU.

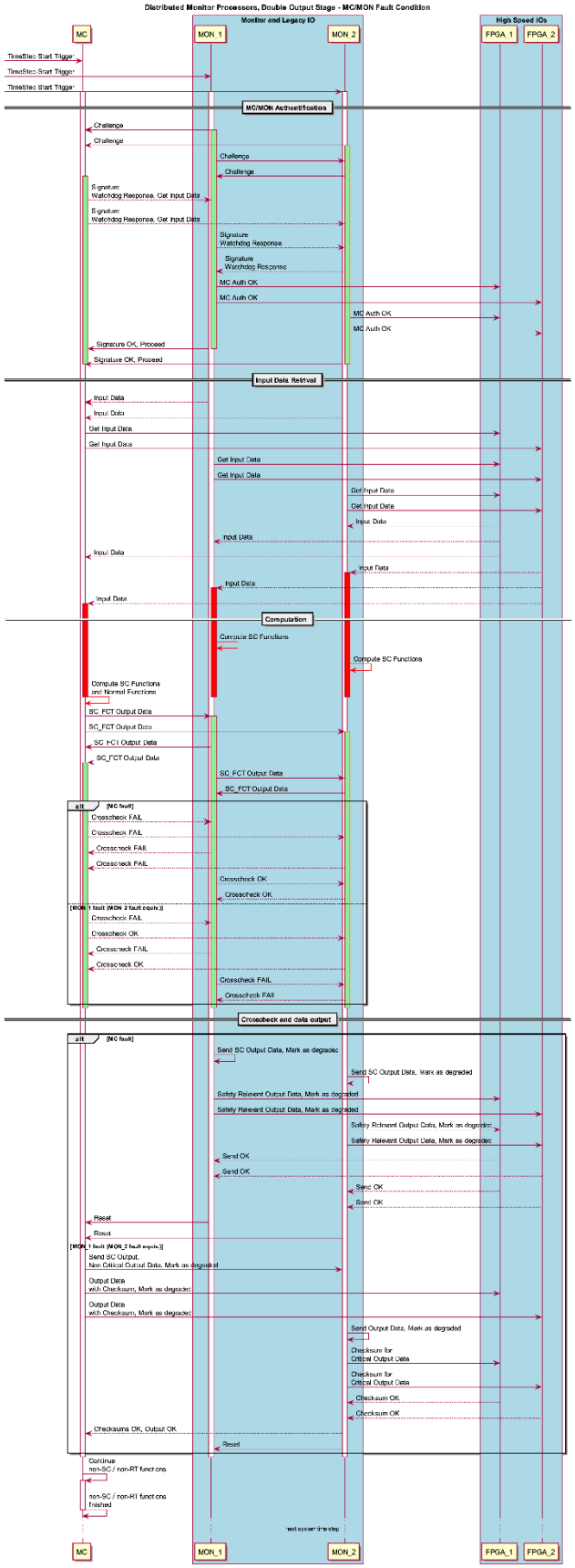
Table 2: Requirements for a LRU for the presented use-cases. Columns UC1 and UC2 signify if the requirement originates from the respective use-case

Appendix II


Single Monitor Processor, Double Output Stage - Nominal Condition
IO
(Second FPGA optional)
MC
MON
FPGA_1
FPGA_2
TimeStep Start Trigger
TimeStep Start Trigger
MC/MON Authentification
Challenge
Signature
Watchdog Response
MC Auth OK
MC Auth OK
Signature OK, Proceed
Input Data Retrival
Get Input Data
Get Input Data
Input Data
Input Data
Input Data
Input Data
Computation
Compute SC Functions
Compute SC Functions
and Normal Functions
SC_FCT Output Data
SC_FCT Output Data
Crosscheck result OK
Crosscheck OK
Crosscheck and data output
Output Data
with Checksum
Output Data
with Checksum
Checksum for
Output Data
Checksum for
Output Data
Checksum OK
Checksum OK
Checksum OK
Checksum OK
Continue
non-SC functions
non-SC functions
finished
next system time step
MC
MON
FPGA_1
FPGA_2


Single Monitor Processor, Double Output Stage - MC/MON Wrong Data Condition
IO
MC
MON
FPGA_1
FPGA_2
TimeStep Start Trigger
TimeStep Start Trigger
MC/MON Authentification
Challenge
Signature
Watchdog Response
MC Auth OK
MC Auth OK
Signature OK, Proceed
Input Data Retrival
Get Input Data
Get Input Data
Input Data
Input Data
Input Data
Computation
Compute SC Functions
Input Data
Compute SC Functions
and Normal Functions
During the computation of critical functions on the MON or MC, the data or program flow is altered by SEE or a SW design error. The result is wrong output data on one device (MC or MON), with correct timing.
Compute SC Functions
Finished
Compute SC Functions Finished
SC_FCT Output Data
SC_FCT Output Data
Crosscheck FAIL
Crosscheck FAIL
Crosscheck and data output
alt
[safe state on on fault]
Outputs Safe State
Outputs Safe State
Outputs Safe State
Outputs Safe State
Safe State OK
Safe State OK
Safe State OK
Safe State OK
[MONoutput on safe state (system redundancy)]
Crosscheck Fail
Crosscheck Fail
Output Data, Crosscheck Fail
System is flagged as degraded on the system interfaces. Remaining lanes provide correct data on the system level. The MON is choosen to provide the output data in this case, due to it's reduced complexity compared to the MC. This allows for additional output data for system-wide voting or smart actuators.
Output Data, Crosscheck Fail
Output Data OK
Output Data OK
Output Data OK
Output Data OK
Optional - Execute Reset
Continue
non-SC functions
non-SC functions
finished
next system time step
MC
MON
FPGA_1
FPGA_2


Distributed Monitor Processors, Double Output Stage - Nominal Condition
Monitor and Legacy IO
High Speed IOs
MC
MON_1
MON_2
FPGA_1
FPGA_2
TimeStep Start Trigger
MC/MON Authentification
Challenge
Signature
Watchdog Response, Get Input Data
Signature
Watchdog Response
MC Auth OK
Signature OK, Proceed
Input Data Retrival
Input Data
Get Input Data
Computation
Compute SC Functions
Compute SC Functions and Normal Functions
SC_FCT Output Data
Crosscheck OK
Crosscheck and data output
Send SC Output, Non Critical Output Data
Send Output Data
SC Output Data with Checksum, Output Data
Checksum for Critical Output Data
Checksum OK
Checksums OK, Output OK
Continue non-SC / non-RT functions
non-SC / non-RT functions finished
next system time step

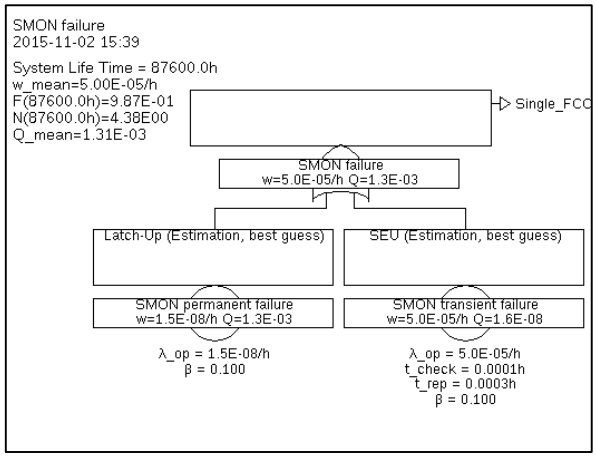

Distributed Monitor Processors, Double Output Stage - MC/MON Fault Condition
Monitor and Legacy IO
High Speed IOs
MC
MON_1
MON_2
FPGA_1
FPGA_2
TimeStep Start Trigger
MC/MON Authentification
Challenge
Signature
Watchdog Response, Get Input Data
Signature
Watchdog Response
MC Auth OK
Signature OK, Proceed
Input Data Retrival
Input Data
Get Input Data
Computation
Compute SC Functions
Compute SC Functions and Normal Functions
SC_FCT Output Data
alt
[MC fault]
Crosscheck FAIL
Crosscheck OK
[MON_1 fault (MON_2 fault equals)]
Crosscheck and data output
Send SC Output Data, Mark as degraded
Send SC Output Data, Mark as degraded
Safety Relevant Output Data, Mark as degraded
Send OK
Reset
[MON_1 fault (MON_2 fault equals)]
Send SC Output, Non Critical Output Data, Mark as degraded
Output Data with Checksum, Mark as degraded
Checksum for Critical Output Data
Checksum OK
Checksums OK, Output OK
Continue non-SC / non-RT functions
non-SC / non-RT functions finished
next system time step

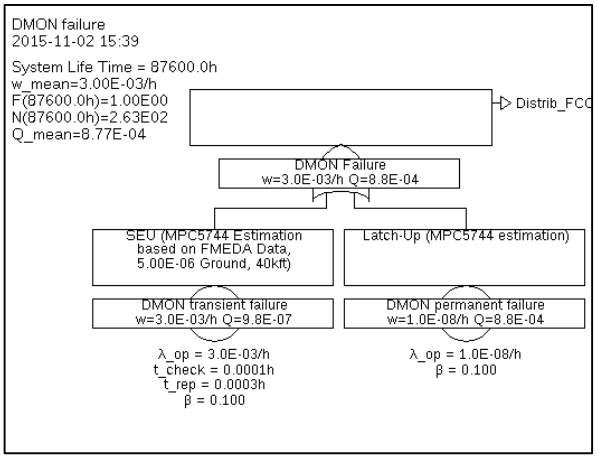
Appendix III

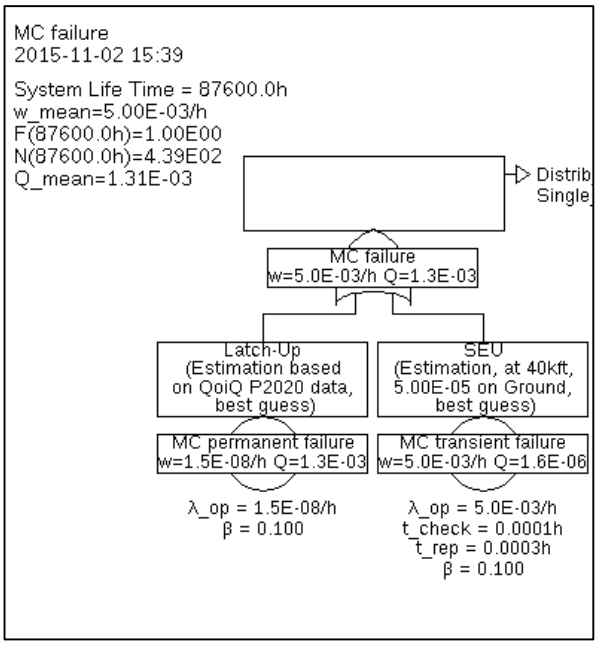
Base Events

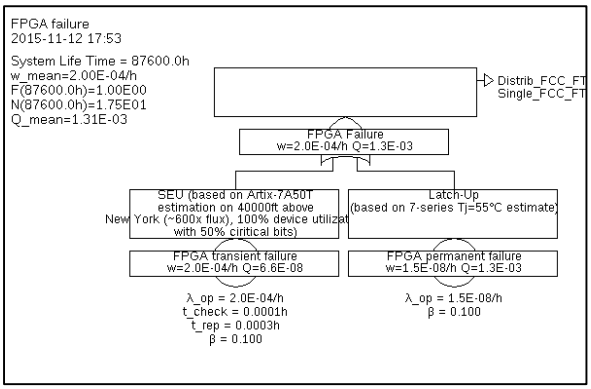
Name	Description	Model	λ_{op}	λ_{sb}	D	WB_k	T_{rep}	T_{chk}	D_t	t_0	β
MC transient failure	SEU (Estimation, at 40kft, 5.00E-05 on Ground, best conservative guess based on NDA data)	Repairable	5,00E-03	0.0	1.0	1.0	0.000278	0.0001	0.0	0.0	0.1
MC permanent failure	Latch-Up (Estimation based on QoiQ P2020 NDA data, best conservative guess)	Non repairable	1,50E-08	1,00E-13	1.0	1.0	0.0	1.0	0.0	0.0	0.1
SMON transient failure	SEU (Estimation, best conservative guess)	Repairable	5,00E-05	0.0	1.0	1.0	0.000278	0.0001	0.0	0.0	0.1
SMON permanent failure	Latch-Up (Estimation, best conservative guess)	Non repairable	1,50E-08	1,00E-13	1.0	1.0	0.0	1.0	0.0	0.0	0.1
DMON permanent failure	Latch-Up (MPC5744 estimation, NDA data)	Non repairable	1,00E-08	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.1
DMON transient failure	SEU (MPC5744 Estimation based on NDA FMEDA Data, 40kft)	Repairable	3,00E-03	0.0	1.0	1.0	0.000278	0.0001	0.0	0.0	0.1
FPGA transient failure	SEU (based on Artix-7A50T estimation on 40000ft above New York (600x flux), 100% device utilization with 50% ciritical bits)	Repairable	2,00E-04	0.0	1.0	1.0	0.000278	0.0001	0.0	0.0	0.1
FPGA permanent failure	Latch-Up (based on 7-series Tj=55°C estimate)	Non repairable	1,50E-08	1,00E-13	1.0	1.0	0.0	1.0	0.0	0.0	0.1

First Aggregate to Device Level

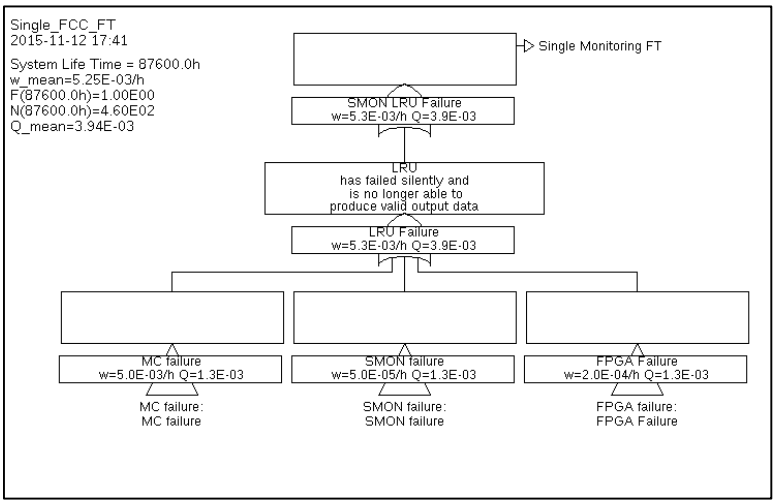


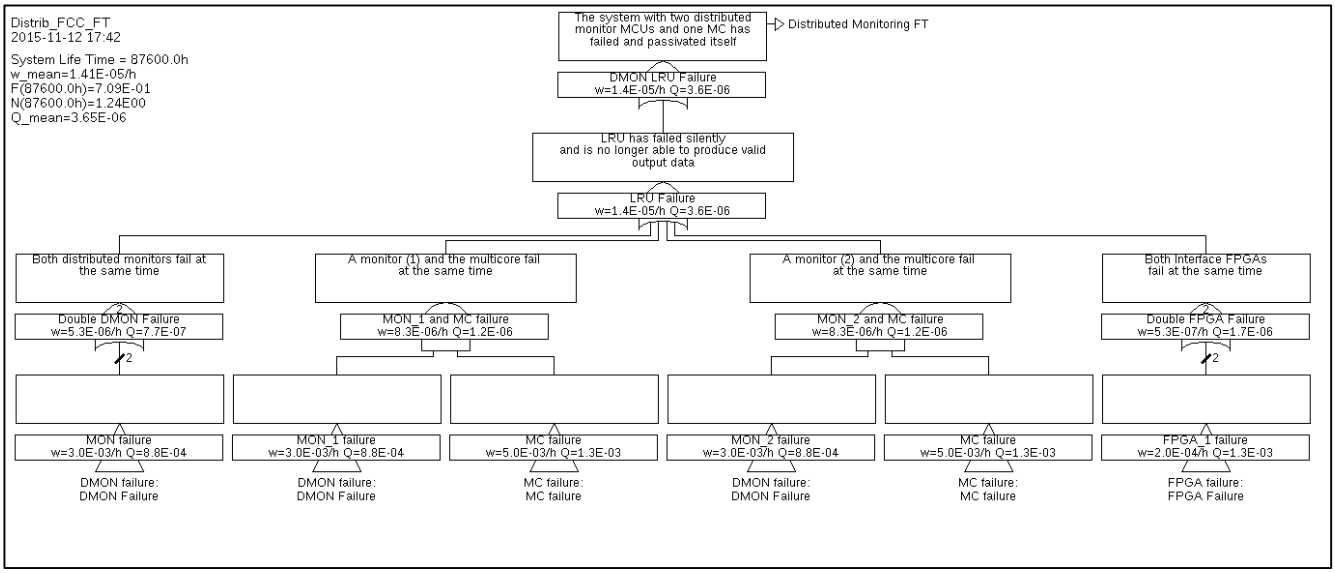




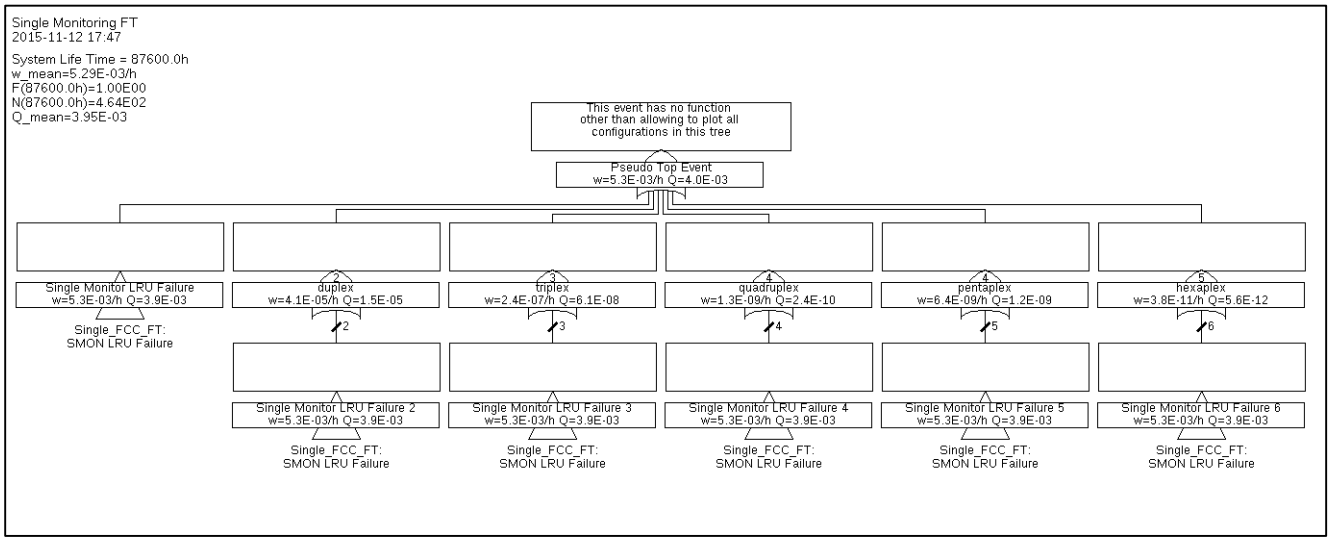


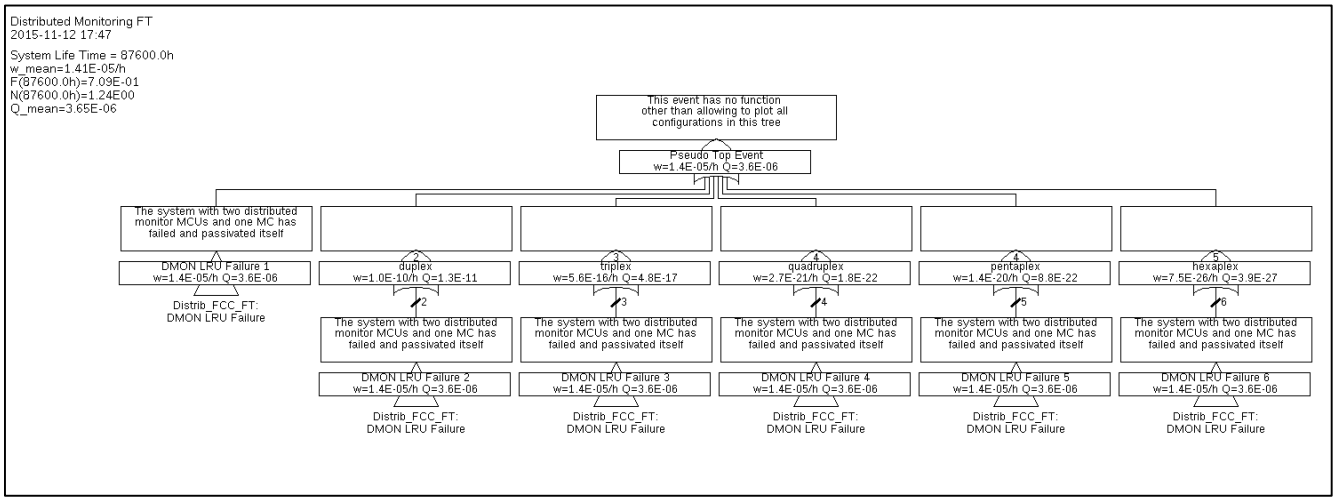
LRU Fault Trees and Markov Diagrams





Resulting System Reliability Calculation with different Redundancy Levels





Markov Models