Fault Management Impacts on the Networking Systems Hardware Design

Carlo Vitucci
Technology Management
Ericsson AB
Stockholm, Sweden
carlo.vitucci@ericsson.com

Daniel Sundmark
Computer Science and Software Enigineering
Mälardalen University
Västerås, Sweden
daniel.sundmark@mdu.se

Marcus Jägemar
Sys Compute Dimensioning
Ericsson AB
Stockholm, Sweden
marcus.jagemar@ericsson.com

Jakob Danielsson
Sys Architecture
Ericsson AB
Stockholm, Sweden
jakob.danielsson@ericsson.com

Alf Larsson
Senior Specialist Observability
Ericsson AB
Stockholm, Sweden
alf.larsson@ericsson.com

Thomas Nolte
Networked and Embedded Systems
Mälardalen University
Västerås, Sweden
thomas.nolte@mdu.se

***Abstract*—Processing capacity distribution has become widespread in the fog computing era. End-user services have multiplied, from consumer products to Industry 5.0. In this scenario, the services must have a very high-reliability level. But in a system with such displacement of hardware, the reliability of the service necessarily passes through the hardware design. Devices shall have a high quality, but they shall also efficiently support fault management. Hardware design must take into account all fault management functions and participate in creating a fault management policy to ensure that the ultimate goal of fault management is fulfilled, namely to increase a system's reliability. Efficiently and sustainably, both in the system's performance and the product's cost. This paper analyzes the hardware design techniques that efficiently contribute to the realization of fault management and, consequently, guarantee a high level of reliability and availability for the services offered to the end customer. We describe hardware requirements and how they affect the choice of devices in the hardware design of networking systems.**

***Index Terms*—requirements, fault management, hardware design, networking system, reliability, availability, and serviceability;**

I. Introduction

Industry uses embedded systems in several fields: automotive, smart TV, washing machines, smartphones, and wearable devices, to mention only a very small part of the embedded market. Also networking uses embedded systems and, due to specific hardware characteristics, it is called networking systems. The evolution of virtualization technology [1], [2] has changed the way of seeing and managing this enormous processing capacity [3], introducing the "softwarized resources" paradigm [4], [5], and the networking development has moved this paradigm to the world of connectivity, where everything is interconnected to everything, everywhere, and all the time [6], [7]. Notions such as "fog computing" [8] and "Internet of Things" [9] have become increasingly popular and have moved the concept of "smart" [10] into previously unthinkable service types, opening the door for the Vertical Service [11], Industry 5.0 [12] and Society 5.0 [13]. The intrinsic quality of the hardware components has undoubtedly increased over time, however the failure of a hardware component have a significantly higher cost due to the increased complexity of the system [14]. Complete board replacements are, however, often the only viable strategy to cope with failure state for a hardware device, which comes at great cost: fault analysis on-site, board removal, packaging, expedition, board analysis, and test to confirm the failure condition diagnosis for the component, and, when possible, faulty hardware replacement. In telecommunication networks, multi-chip packages, robotics, automotive, and, more generally speaking, in an increasingly widespread distributed system, the hardware devices must work and inter-work properly, react to external disturbances promptly, and remain resilient as long as possible. Therefore, maintaining a low value for the system's hardware failure rate (and high reliability) is no longer just a problem of component quality but of the ability to detect, manage and correct a fault before it becomes a failure condition. Since the ultimate purpose of fault management is to locate, isolate and recover a fault condition before it results in a system failure condition, investing in fault management during hardware design will make the working state phase of a product stable and lasting.

In this paper, we propose the following:

- mapping of Reliability, Availability, and Serviceability (RAS [15]) requirements towards fault management areas, which simplifies the understanding of the requirements for the hardware design;
- a set of fault management hardware requirements for a networking system that can improve the product lifetime;
- a new power-on-board self-test procedure based on hierarchical test control system solution.

Paper Structure

Section II recalls helpful definitions and sets the reference hardware architecture model. Section III mentions other interesting work in hardware fault management requirements definition. Section IV maps RAS requirements in five areas of fault management. Section V describes the RAS requirements for the subsystems of the reference architecture. It also describes possible existing solutions as implementation examples. Section VI reports the conclusion and possible future work.

Research Context

This paper focuses exclusively on the characterization of hardware components. Readers should consider the following definitions, architecture, and requirements in the hardware domain only. This paper does not mention or refer to issues related to their configuration, use, and, more generally, the platform software.

II. Definitions and System Model

1) Definitions Summary: This subsection recalls helpful definitions for fault management characterization. Many standards specifies definitions for the reference field, like ARP 4761 [16] (avionic), ISO 26262 [17] (automotive), IEC 61226 [18] (nuclear), or IEC 61508 [19] (electrical/electronic control systems). We use a general definition that maps onto the aforementioned standards.

Definition *1:* A *failure* is the inability of a device to meet the specified functionality [20].

Definition *2:* We define λ as the *failure rate* of the system. During the lifetime phase, it tends to remain constant [21].

Definition *3:* We define *reliability*, $R(t)$ (1) (where t is the time), as the probability of a system to operate correctly, without a failure condition, during a time interval $[0, t]$, assuming that the system is performing correctly at the time 0 [22].

$$R(t) = e^{(-\lambda t)} \tag{1}$$

Definition *4:* MTTF, *Mean Time to Failure* (2), is the expected time a system works properly before the first failure is observed [22].
MTTF is strictly connected to reliability by definition.

$$MTTF = 1/\lambda \tag{2}$$

Definition *5:* MTBF, *Mean Time Between Failure* (3) is the average time between two consecutive failures of the system [23].

$$MTBF = \frac{\#operationhours}{\#failure_{sametime}} \tag{3}$$

Definition *6:* RR, *Return Rate* (4) is the percentage of installed products that are returned to the hardware repair center because of a suspected failure condition.

RR is a fault avoidance requirement type because it is the most obvious way to reduce costs: high-quality hardware components are achieved through careful device selection in the design test and device screening in the production test. The return rate is also a function of time [23].

$$RR_{annual} = \frac{1}{MTBF_{annual}} = \frac{24 * 365,25}{MTBF_{hourly}}$$

$$RR_{annual} = \frac{8766}{MTBF_{hourly}} \tag{4}$$

Definition *7:* MTTR, *Mean Time to Repair* is the average time required to isolate, repair, and test a fault condition for the system [21].

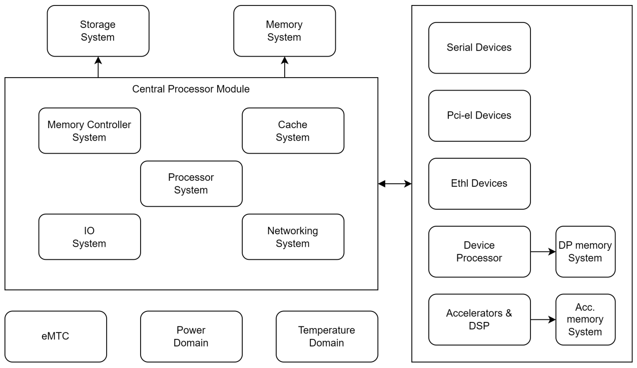

Fig. 1. Hardware reference system.

2) Reference Architecture: In this subsection, we describe our reference architecture. The reference architecture consists of devices and connections that are most common in a networking system platform, presented in Fig. 1).

The reference architecture utilizes a Central Processor Module (CPM) as a main controller for operations. The processor system within CPM is a complex multi-core system. It contains an internal memory subsystem hierarchy with different levels of cache memory and TLB's that connects to the system memory. The CPM has access to non-volatile memory (Storage System in Fig. 1) of a size in a range comparable to or greater than the system memory. The CPM is responsible for controlling board devices and also maintaining connections between these devices. The board devices can be classified by type of connection (PCI-e, ethernet, or serial bus) and by processing (device processor/FPGA or DSP/hardware accelerator). Devices can also have dedicated memory (not necessarily accessible by the CPM). Master Test Controller *eMTC* is an IEEE 1149.1 [24] compliant test generator that controls JTAG test sequences for both internal logic and external devices. The eMTC is commonly a part of the CPM system, but we suggest using an external chipset for this function which will extend the overall test coverage to also span over the CPM. A power domain and a series of temperature sensors complete the reference system.

3) Fault Management Techniques: There are four distinct fault causes: specification errors, implementation errors, external disturbances, and random component failures [22], which can lead to software faults, hardware faults, or both. There are different techniques for maintaining the system in a working state, and we group them into three main barriers; fault avoidance, fault masking, and fault tolerance (see Fig. 2).

***Definition** 8: Fault avoidance* is the set of actions executed to minimize the failure rate due to Specification and implementation errors, external disturbances, and defected components [22].

Accurate HW/SW design, extensive design review, component screening, and system verification are typical fault avoidance techniques.

***Definition** 9: Fault masking* is the set of functions and features that allows the system to work without impacting its functionality in case of detected faults [22].

Information redundancy and majority vote algorithms are fault-masking techniques. An example allows us to understand better what information redundancy means: the error-detection codes, like CRC, ECC, or parity bit, add a few bits (in this sense, more information) to manage the integrity handling of information. The algorithm adds extra information to the data used by the system, which enables verification of the data integrity.

***Definition** 10: Fault tolerance* is the five main components of fault management: detection, location, isolation, prediction, and recovery of the fault to avoid system malfunction [20].

We also list two other important definitions for fault management:

***Definition** 11: Fault coverage* is the probability that fault management detects and handles any system fault.

***Definition** 12: Fault reporting* is the capability of fault management to identify, collect and propagate fault information into the system.

Fault reporting is essential to understand how to cope with, and recover from a fault condition.

III. Related Works

The functional requirements and the need for time-to-market products often drive hardware design. Fault management specifications unluckily arrive later in the product development process. Bidokhti [25] describes the consequence of not considering the Reliability, Availability, and Serviceability (RAS) requirements from the first design phase to the product quality. Chalermarrewong [26] proposed a hardware failure prediction model for high-performance computing systems firmly based on asset migration. The work of Siasi [8] also focuses on the dynamic configuration of resources used in a fog network to protect the serviceability of the Service Function Chains (SFC). Kundu [27] analyzes the effects of hardware reliability on Artificial Intelligence (AI) and Machine Learning (ML) systems, emphasizing, above all, the consequences of external disturbances on the reliability of memory circuits. They propose system techniques to increase fault tolerance specific for AI/ML architectures and neuromorphic hardware, such as the ”Fault-aware training”, but not very characteristic of the hardware itself. Scargall [28] in his book describes what it means to have RAS requirements, even if limited to persistent memory components. Yao [29] have carefully analyzed the RAS requirements for the memory of a data center, describing the fault management with a multi-technological approach very close to the multi-layer framework proposed in [30]. Finally, we mention the work of Safari [31] about the fault tolerance techniques for embedded systems of the power, energy, and thermal domains, even if they also focus only on these two domains. Our paper does not consider the fault tree analysis and the evaluation of the probability of faults for the node (a possible reference is the work of Das [32]).

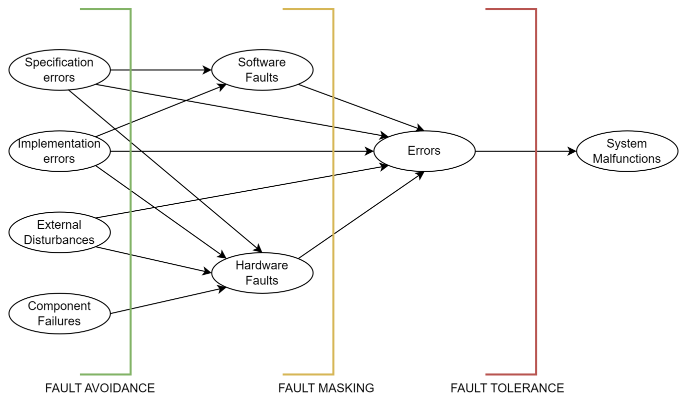

Fig. 2. The three fault barriers [22].

IV. Requirement Tree

Any fault management technique (see Section II-3) implements one of the RAS requirements, no matter to which barrier it belongs, as shown in Fig. 3. The requirement tree recalls and extends the requirement classification done by Bidokhti in [25], and creates a map between the requirements and the five macro fault management domains.

Reliability

- MTTF is a fault avoidance requirement type and is based on the quality of service expectations. The device selection shall be based on the vendor’s MTTF value to guarantee the λ is as low as possible during the product’s lifetime.
- Our target position is that the annual RR shall not exceed a single-digit percentage for networking systems. Since the RR refers directly to the installation, our proposed way to ensure high product quality is a high level of fault coverage during the production’s hardware screening. Finally, after the product installation, the RR can be reduced through fault tolerance: not only with recovery mechanisms from a fault condition but also with the implementation of the ”degraded function condition”: a system that can isolate the faulty component and restore to a working condition, but with reduced performance.

Availability The items listed in the availability group match what we previously described as the main components of a fault management framework in [20] and, by definition, belong to the fault tolerance requirement class. In this Section, we only add comments and concepts that allow a mapping into any other of the five macro fault management domains related

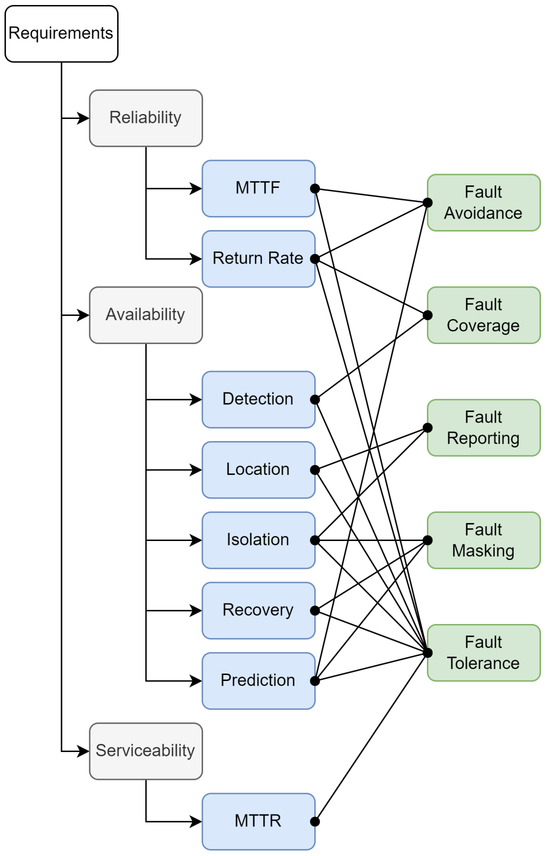

Fig. 3. The requirements tree.

at the beginning of the Section. More intricate descriptions of functions behind the macro fault management framework components are available in [30]. We list the availability components as follows:

- Detection: self-test (both power-on and runtime) and runtime hardware monitoring are typical examples of fault detection functions and belong to the fault coverage requirement class. Increasing the fault coverage is mandatory to avoid the ”bad state”, where the system loses the fault detection capability: a possible fault will remain unannounced, with the unpleasant drawback that the fault propagation will crash the system without any chance to run a root cause analysis.
- Location: Fault information and fault reporting, is just as crucial as fault management's ability to detect a faulty component. In [20], we previously discussed the importance of taxonomy for error indications, where it is also possible to find a description of the topology in the domain of possible recovery actions. Using a standard format for fault reporting (for example, the syslog format [33]) is valuable because the hardware-platform-applications ecosystem can manage it.
- Isolation: The fault location is the system's ability to avoid the effects of a faulty condition of a component in adjacent devices. Fig. 3 displays a connection between ”Location” and ”Fault reporting” to emphasize the need to provide the best information for recovery action to the fault management. In this sense, the isolation's granularity directly impacts the redundancy cost necessary to maintain the system in fully-functional working conditions. For example, a generic error indication for a memory device requires the availability of a spare memory device. In contrast, the ability to identify the error in memory on a single array allows significantly cheaper redundancy solutions. While reporting a fault indication, the more fine-grained is the hardware granularity, the lower will be the redundancy cost.
- Recovery: There are two possible methods: *passive recovery* (fault-masking) and *active recovery* (fault tolerance). Passive recovery features includes Error Correcting Codes (ECC), scrubbing, and data poisoning (information redundancy techniques), while software and hardware redundancy techniques, such as error handler, Post Package Repair (PPR), or spare components, are part of active recovery.
- Prediction: The application of AI/ML allows analysis of system data. AI/ML can recognize patterns in data set as possible risk of a fault condition and act on the system before this happens (fault avoidance) or recognize when a device begins to work in non-optimal states, allowing a replacement with spare elements before this causes a fault condition (fault-masking) [34].

Serviceability

- MTTR. From the definition of serviceability, we derive how it depends on the availability of the mechanisms used by the system for error recovery (on-site or off-site). The MTTR value also depends on available recovery techniques. For example, a system can recover a memory soft-error fault by reinitializing the device, an operation that would have an MTTR of the order of a few minutes. On the other hand, a system can recover a memory hard-error by an on-site replacement with spare elements, thus keeping the MTTR value low but with a higher product cost. Instead, if the system design requests an offline device replacement, the MTTR value could be even more than several weeks, with a higher maintenance cost.

V. Fault Management Hardware Design Characterization

In this section we define the list of hardware features that designers shall consider when designing service-oriented networking system hardware platforms. We list a summary on how RAS requirements impacts the hardware design of an networking system as follows:

Hardware shall be able to detect errors (fault coverage). General examples of hardware detection feature are the connectivity terminations, that shall supervise the link termination according to the physical link standard specification [35], and the data flow supervision.

Hardware shall be able to correct errors (fault masking and fault recovery). A typical example is Single Error

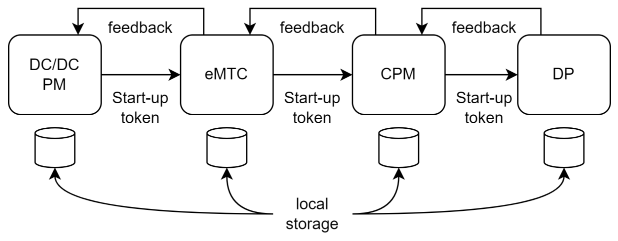

Fig. 4. The hierarchical start-up chain.

Correction / Double Error Detection (SECDED) algorithm for memories device.

Hardware shall be able to report errors (fault reporting). Any device shall report detected corrected or uncorrected errors to the software.

Hardware shall be able to isolate the errors (fault isolation and fault recovery). Hardware shall provide features to avoid fault propagation in the system. A typical example is the data poisoning.

Hardware shall be able to provide self-test validation (fault coverage and fault avoidance). Any device shall be able to run a sanity check by command or automatically in case of power-on. Hardware shall provide a fault supervision verification mechanism like, for example, the error injection.

A. eMTC

This section introduces an innovating power-on-board self-test procedure based on the usage of an external master test controller (eMTC) engine.

The eMTC's primary function is the verification of the board devices integrity during the power-on-board self-test. It shall have persistent storage to load the test configuration file. Multiple test configuration files allow eMTC to execute specific tests on demand. eMTC decides to execute different configuration file based on latest event before the processor reset. For example, an indications set by the CPM in a persistent memory before the processor reset, like the ID of a component in a faulty state. The flexible test coverage option in the eMTC allows for the execution of a single component instead of testing the entire board, reducing the MTTR. On power-on, the board test could run in a hierarchical model (see Fig. 4). As a first action, the Power Management unit (PM) validates and stores the status of the Master Test Controller (eMTC) before letting eMTC continue the power-on test. eMTC validates and stores the central processor's status and inform PM to continue the power-on cycle. Any processing device on board could participate in this hierarchical power-on-test cycle: the device's status could be validated and stored before letting PM continue the power-on cycle and allow device processor (DP) to run next verification phase. The overall process requires non-volatile storage per any object involved in the power-on test chain. The persistent memory of an eMTC can work as a ROM hardware inventory table that lists all devices on board, their unique ID, and the corresponding memory map. The ROM table allows protection from malicious and unauthorized device replacement and supports fault management to manage the status of the devices.

Using eMTC has a positive impact on RR reduction. The cost analysis can indicate the weight of the eMTC usage. Figure 5 shows the total costs impacts, i.e., production and maintenance, for the single product installed. If C is the cost of eMTC, its use allows the elimination of some hardware components (e.g., resistors and capacitors) with a weight on the production cost of the Printed Boardd Assembly (PBA) equal to -0.2C. The maintenance cost due to a 1% reduction in RR has an impact equal to -2C, which means that the total cost per single product improves by 1.2C for 1% RR reduction.

B. Central Processor Module

The CPM shall support fault management multi-technology approach as we proposed in [30]. This approach is possible only if the hardware supports a "firmware-by-first" framework: firmware always manages any fault information from any device before propagating the fault to the operating system. Hardware shall pass information about fault type and its location. Firmware manages faults using configurable control and a hardware filtering mechanism. The CPM fault detection framework is called "Machine Check Architecture" (MCA), which provides a fine granularity for the fault location and error counters for device status or statistics collection. MCA can detect and report any CPM firmware bug as a software error, and CPM firmware shall be uploaded remotely. The CPM shall also support device verification (self-test and error injection).

1) Processor System: The processor system of the central processor module is a multicore subsystem and core faults shall be available via MCA. In detail, the hardware shall be able to isolate a single faulty core, and plug-in/out mechanisms shall allow for removal of the faulty core from the group of

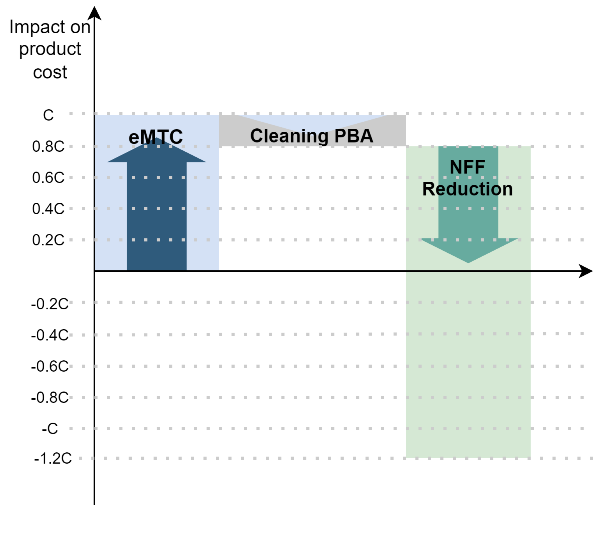

Fig. 5. The eMTC effect on the product cost for 1% RR reduction

available non-faulty cores. The processor system shall support virtualization technology via hardware accelerators for virtual CPU, virtual memory, and virtual IO access. Virtualization of hardware resources can simplify relocation mechanism.

2) Cache System: The most common hardware platform has three cache levels: L1, L2, and L3 [36]. The system shall be able to detect data corruption and invalidate the entry, avoiding fault propagation. Most L1 cache accesses are load and store requests for program data and instruction fetches, and it is the fastest and smallest cache in the system. Therefore, a parity bit mechanism provides a sufficient level of protection for the L1 cache. Higher levels of cache requires more complex algorithms. A proper requirement is that L2 and L3 caches fault detection and recovery shall be at least ECC SECDED algorithm.

3) Memory Controller System: ECC memory devices need a memory controller that supports ECC. Modern memories, such as DDR5, contain built-in ECC. The previous memory device generation counts on the memory controller to implement features like the Sideband ECC and the Inline ECC [37]. The Sideband supervises physical termination of the memory controller and the inline physical connection to the memory device. Both Sideband and inline ECC are mandatory features of a memory controller. The memory controller shall also support some active fault tolerance techniques:

- Error check and scrub (patrol scrubbing): automatic periodic memory verification to find and correct a single error.
- Data poisoning: detect and mark as corrupted memory containing corrupted bits. The software will manage a poisoned memory avoiding fault propagation into the application.

The memory controller provides error counters per corrected, correctable, and uncorrectable memory errors. It is helpful to rely on a rapid recovering mechanism, avoiding the board restart to reduce significantly the MTTR in case of uncorrectable memory errors.

4) Networking System: All networking system faults shall be available using the MCA.

5) IO System: All IO system faults shall be available using the MCA. Most I/O devices interface with their controller in CPM, which is responsible to detect and report I/O device fault indication through MCA.

C. System Storage Module

There are two types of non-volatile storage device in an embedded system: flash and solid disk. The flash is a bootable device with a limited size, commonly not more spacious than 32GB, while the solid disk can be optional, more spacious, and can contain application files. The typical distinguishing feature is that SSDs are typically read in blocks whilst code flash in words. The secure boot signing system [38] authenticates and validates the signature on system boot to protect the flash contents. The authentication phase includes making a hash-key of the images and comparison to the encrypted hash-key contained in the signature. The primary purpose of the signature procedure is to avoid malicious or unwanted write commands into the flash, but it can also detect a data retention issue. In case of corrupted data, the processor can swap from one start address to another address that points to a backup solution. This way, there is a chance to start the board and recover the corrupted data via an authorized flash update procedure. Embedded systems can use SSD disk instead of HDD for their intrinsic fault tolerance [39]. The MTTF SSD is higher than HDD [40]. HDD tends to have non-predictable catastrophic crashes, while SSD fails predictably because the wear-leveling algorithm spreads block erases and gradually introduces spare blocks. Most vendors offer disk state tools, a significant benefit for fault prediction. For the above reasons, the networking systems use SSD, even if they are more expensive (cost per GB) than HDD.

D. System Memory Module

As a complementary technique of what sections V-B2 and V-B3 describe for memory fault management, in this section, we address a beneficial redundancy technique: the Post Package Repair (PPR). It results in recovering a faulty memory location in the system by disabling it and using a spare memory row instead. The number of spare memory rows is a characteristic of the memory device. The system memory device shall also support error injection allowing a fault verification mechanism.

E. Devices

The general requirements are:

- *Fault counters*: All interfaces between circuit or module that uses symbol coding shall have saturated counters of (at least) 32 bits size.
- *Fault detectors triggering*: All HW faults feature control register to enable/disable interrupt to the main processor, read/reset fault occurrence, set/reset threshold, and alarm filtering. All the hardware fault registers shall be accessible in run-time.
- *Interface fault injection*: All interfaces between circuit or module that uses symbol coding should have fault injection possibilities in random symbol positions.

1) Serial Bus Devices: The Platform Control Hub (PCH) is often the name of the CPM IO system and connects most I/O devices inside the CPM. The devices connected to the PCH can also be PCIe devices, and, in that case, Section V-E2 describes the helpful features to support fault tolerance. In all the other instances, protocol handling manages the bus supervision, and devices shall have a system to report error conditions to the central processor. Serial bus fault indication ends up in the MCA registers. MCA registers provide info to populate the fault log and report with Bus, Device, and Function information associated with the I/O error. MCA architecture shall allow for the report of fatal and uncorrectable errors only. The corrected error information shall be available on-demand through counters. MCA architecture shall allow a flexible configuration for the hardware-corrected error reporting, for example, defining fault reporting based on thresholds and associating a fault indication to a degraded function condition. We need device loop capability for all non-PCIe devices to test the component. JTAG chain for I/O devices shall be possible for a complete boundary-scan execution.

2) PCIe bus devices: PCIe architecture is a hierarchical device connectivity structure fully defined in the PCI-express Base Specification. The PCIe Base Specification also describes PCIe error and error reporting technology (including link-level reliability). Note that system software shall configure all PCIe components and devices to enable Advanced Error Reporting (AER) capability and generate AERs in case of detected errors. There are two main components in hierarchical architecture:

- PCIe root port. The PCIe root ports are part of the CPM and firmware typically configures them. PCIe technology allows the delivery of the faulty event using AER for root ports.
- PCIe devices. All PCIe devices shall be identifiable and addressable in the system. The error events are delivered using AER for link supervision and device error indication. AER error reporting allows fine-grained error reporting and response.

Any non-correctable internal device error not covered by the AER error description shall be propagated to the CPM using MSI/MSI-X interrupt vectors. A correctable error shall not automatically generate MSI/MSI-X interrupt vectors, but statistics reporting based on configurable thresholds, fault mask, and interrupt control.

3) Ethernet bus device: This section covers the fault supervision of ethernet links connecting a device to the CPM. The ethernet physical termination shall provide registers to report two conditions as suspected faults for the physical ending: excessive BER and CDR unlock. The physical terminations shall also provide the following:

- Inbound and outbound loop (external and internal loop features for the fault coverage).
- Statistics of the physical termination of the channel to support fault reporting and to support qualified data for a possible evaluation of the health status of the physical termination (fault prediction).

Ethernet connection is a Peer-to-Peer (P2P) type: there is no hierarchical link control (for example, PCIe), and the supervision of the link shall be available in both the end peers of the connection. Status of the link, quality of the link error (FEC/BER/CRC), statistics, and error injection shall be available. Twamp (Two-Way Active Measurement Protocol) [41] is a helpful software feature for evaluating transmission latency.

4) Device Processor: Device processors are specialized computation circuits that often run software bare-metal with real-time requirements. Device processors present a less complex architecture than CPM (such as fewer levels in the cache hierarchy) and, consequently, are cheaper but support fewer attributes and functions. The features supporting fault management are also less than the number available in CPM. We propose to consider a fault management agent for the DP subsystem, gathering errors and statistics under the coordination of the CPM control and supervision that will access DP subsystem fault information on demand or event-driven. The device process shall store fault information in a memory that can survive a warm restart. Local memory can suffer from bit-flip issues, and info redundancy like SECDED shall be available for L2 cache and above device processor memories. For L1 cache, memory access will be in the range of data word size, and, therefore, a parity bit is suitable to prevent data corruption propagation.

5) DSP Processor: DSP processor subsystems mean the union of a cluster of DSP or hardware accelerators devoted to implementing functions with stringent processing requirements. DSPs are commonly built as a System on Chip (SoC) solutions and need a significant amount of volatile memory for processing data continuously. The SoC shall have independent fault management, which means that it shall be able to detect, locate, isolate and recover faulty states and that it shall have an independent core capable of hosting the error handler for the implementation of the necessary software redundancy. The subsystem shall have a JTAG connection to check the status of components and the connection with the CPM. The subsystem shall be able to isolate any of the cores detected in a failure state, which means that the power control shall have single-core granularity. At the hardware design level, adding a certain number of spare cores is strongly recommended: fault recovery can use them as a physical redundancy solution to cope with a permanent failure condition in a faulty core. The subsystem of the DSP processor shall be able to rely on information redundancy for the integrity of the data of its memories: SECDED is the at-least required capability. Since CPM shall be considered the primary subsystem, the DSP processor subsystem shall provide a self-test and verification mechanism (e.g., error injection) to be performed both in the initialization phase as a recovery option and on-demand by the CPM.

F. Power Domain

The networking system shall use two threshold levels for power control: warning and fault level. The board power module shall detect a permanent warning or a faulty condition for the voltage in time. The board power module shall be able to report power status to the main board controller, both faulty and recovery event. Hardware redundancy for the power domain is mandatory for recovering into a safe condition through hardware-depending actions, like secondary voltage powered off or secondary and internal domains cycle verification. Any subsystem shall use an internal power distribution solution for fine granular power control (fault containment purpose).

G. Temperature Domain

The board's test includes a temperature check to stop execution if the temperature is outside the operating range. If the temperature reaches levels threatening to destroy the hardware, the board control system may autonomously throttle hot devices. Periodical restart or test can then check if the temperature condition returns to a safe mode range. The temperature sensor supervision detects a faulty board temperature sensor using a "reasonable check": a value is not trustable if it is not in the sensor's operating range. For "not trustable", the algorithm considers a sensor in a faulty state if it reports a value too far from the median temperature distribution.

VI. Conclusions and Future Work

This paper presented a comprehensive list of fault management hardware requirements for the networking system's early-stage design phase. With these specific requirements in mind, we aim to improve the fault tolerance in complex RAN boards and produce resilient, long-lasting products with improved RR, MTTF, and MTTR. Our proposed requirements may also reduce the OPEX and maintenance costs and provide increased system serviceability that can likely pay back the fee of the proposed fault management requirements. Two main areas need more attention: integrating the hardware feature with higher system layers (firmware and software) and designing a predictive maintenance algorithm based on the data and fault information from the new set of fault management hardware features. Both investigations will allow for a formal quantification of the expected improvement.

References

[1] N. Goel, A. Gupta, and S. N. Singh, "A study report on virtualization technique." Institute of Electrical and Electronics Engineers Inc., 1 2017, pp. 1250–1255.

[2] M. H. Quraishi, E. B. Tavakoli, and F. Ren, "A survey of system architectures and techniques for fpga virtualization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, pp. 2216–2230, 9 2021.

[3] Ericsson, "Ericsson mobility report june 2022," 2022.

[4] D. E. Sarmiento, A. Lebre, L. Nussbaum, and A. Chari, "Decentralized sdn control plane for a distributed cloud-edge infrastructure: A survey," *IEEE Communications Surveys and Tutorials*, 2021.

[5] M. He, A. M. Alba, A. Basta, A. Blenk, and W. Kellerer, "Flexibility in softwarized networks: Classifications and research challenges," *IEEE Communications Surveys and Tutorials*, vol. 21, pp. 2600–2636, 7 2019.

[6] L. S. Otu, "The internet of economic things: The socio-economic transformation value of the internet of things (iot)." Institute of Electrical and Electronics Engineers Inc., 5 2019, pp. 109–113.

[7] M. Sirma, A. Kavak, and B. Inner, "Cloud based ioe connectivity engines for the next generation networks: Challenges and architectural overview." IEEE, 11 2019, pp. 1–6.

[8] N. Siasi, M. A. Jasim, A. Yayimli, and N. Ghani, "Service function chain survivability provisioning in fog networks," *IEEE Transactions on Network and Service Management*, vol. 19, pp. 1117–1128, 2022.

[9] K. Kaur, "A survey on internet of things - architecture, applications, and future trends." Institute of Electrical and Electronics Engineers Inc., 2018.

[10] M. Ericson, M. Condoluci, P. Rugeland, S. Wänstedt, M. S. H. Abad, O. Haliloglu, M. Saimler, and L. Feltrin, "6g architectural trends and enablers," 2021, pp. 406–411.

[11] J. Santos, T. Wauters, B. Volckaert, and F. D. Turck, "Towards low-latency service delivery in a continuum of virtual resources: State-of-the-art and research directions," *IEEE Communications Surveys and Tutorials*, vol. 23, pp. 2557–2589, 2021.

[12] J. Cotta, M. Breque, L. D. Nul, and A. Petridis, "Industry 5.0 towards a sustainable, human-centric and resilient european industry," *European Commission Research and Innovation (R&I) Series Policy Brief*, 2021.

[13] M. Maier, A. Ebrahimzadeh, A. Beniiche, and S. Rostami, "The art of 6g (tao 6g): How to wire society 5.0 [invited]," *Journal of Optical Communications and Networking*, vol. 14, pp. A101–A112, 2 2022.

[14] R. Mercier, C. Killian, A. Kritikakou, Y. Helen, and D. Chillet, "A region-based bit-shuffling approach trading hardware cost and fault mitigation efficiency," in *2021 IEEE DFT*, 2021, pp. 1–4.

[15] M. V. Beigi, S. Gurumurthi, and V. Sridharan, "Reliability, availability, and serviceability challenges for heterogeneous system design," in *2022 IEEE International Reliability Physics Symposium (IRPS)*, 2022.

[16] SAE, "Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment," 1996.

[17] I. 26262, "Road vehicles - functional safety," pp. part 1–10, 2011.

[18] IEC, "Nuclear power plants - instrumentation and control system important to safety - software aspects for computer-based systems performing category a functions," 2009.

[19] ——, "Functional safety of electrical/electronic/programmable electronic safety-related systems," *IEC 61508, Edition 2.0*, 2010.

[20] C. Vitucci, D. Sundmark, M. Jägemar, J. Danielsson, A. Larsson, and T. Nolte, "A reliability-oriented faults taxonomy and a recovery-oriented methodological approach for systems resilience," *Proceeding IEEE 46th COMPSAC*, p. 0, 0 2022.

[21] E. Dubrova, *Fault-Tolerant Design*. Springer New York, 2013.

[22] B. Johnson, *Design and Analysis of Fault-tolerant Digital Systems*, ser. Addison-Wesley series in electrical and computer engineering. Addison-Wesley Publishing Company, 1989.

[23] E. Bauer, *Design for Reliability: Information and Computer-Based Systems*, ieee book ed. Wiley-IEEE Press, 2010.

[24] *IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001): IEEE Standard for Test Access Port and Boundary-Scan Architecture*. IEEE, 2013.

[25] N. Bidokhti, "The impact of reliability requirements on development life cycle," *Proceedings - Annual Reliability and Maintainability Symposium*, pp. 307–311, 2008.

[26] T. Chalermarrewong, T. Achalakul, and S. C. W. See, "The design of a fault management framework for cloud," *9th ECTI-CON*, 2012.

[27] S. Kundu, K. Basu, M. Sadi, T. Titirsha, S. Song, A. Das, and U. Guin, "Special session: Reliability analysis for ai/ml hardware," 2021, pp. 1–10.

[28] S. Scargall, "Reliability, availability, and serviceability (ras)," *Programming Persistent Memory*, pp. 333–346, 2020.

[29] "A memory ras system design and engineering practice in high temperature ambient data center," *InterSociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems, ITHERM*, vol. 2020-July, pp. 1379–1388, 7 2020.

[30] C. Vitucci, D. Sundmark, M. Jägemar, J. Danielsson, A. Larsson, and T. Nolte, "Fault management framework and multi-layer recovery methodology for resilient system," *Proceeding IEEE 6th ICSRS*, 2022.

[31] S. Safari, M. Ansari, H. Khdr, P. Gohari-Nazari, S. Yari-Karin, A. Yeganeh-Khaksar, S. Hessabi, A. Ejlali, and J. Henkel, "A survey of fault-tolerance techniques for embedded systems from the perspective of power, energy, and thermal issues," *IEEE Access*, vol. 10, pp. 12 229–12 251, 2022.

[32] N. Das and W. Taylor, "Quantified fault tree techniques for calculating hardware fault metrics according to iso 26262." Institute of Electrical and Electronics Engineers Inc., 6 2016.

[33] N. W. Group, "The syslog protocol," *RFC Editor, Proposed standard,*, 2009.

[34] M. K. Das and K. Rangarajan, "Performance monitoring and failure prediction of industrial equipments using artificial intelligence and machine learning methods: A survey," *Proceedings of the 4th International Conference on Computing Methodologies and Communication, ICCMC 2020*, pp. 595–602, 3 2020.

[35] "Ieee standard for ethernet – amendment 7: Physical layer and management parameters for 400 gb/s over multimode fiber," *IEEE Std 802.3cm-2020*, pp. 1–72, 2020.

[36] J. G. Anjana and M. Prasanth, "A three level cache structure." Institute of Electrical and Electronics Engineers Inc., 1 2015, pp. 426–430.

[37] V. Sankaranarayanan, "Error correction code (ecc) in ddr memories," *DesignWare Technical Bulletin*, vol. Q420, 2020.

[38] R. Wang and Y. Yan, "A survey of secure boot schemes for embedded devices," in *2022 24th International Conference on Advanced Communication Technology (ICACT)*, 2022, pp. 224–227.

[39] R. Micheloni, A. Marelli, and K. Eshghi, *Inside Solid State Drives (SSDs)*, ser. Springer Series in Advanced Microelectronics. Springer Netherlands, 2012.

[40] A. Klein, "Backblaze drive stats for q1 2021," *BackBlaze*, 2 2022.

[41] J. Babiarz, R. M. Krzanowski, K. Hedayat, K. Yum, and A. Morton, "A two-way active measurement protocol (twamp)," RFC 5357, 2008.