THE MULTICORE CONUNDRUM: OVERCOMING BARRIERS IN THE ADOPTION OF MULTICORE PLATFORMS IN VTOL AVIONICS SYSTEMS

Claire Aventini, Airbus Helicopters, Marignane, France
Yannick Degot, Airbus Helicopters, Marignane, France
Frédéric Faubladier, Airbus Helicopters, Marignane, France
Louis Fabre, Airbus Helicopters, Marignane, France

Abstract

The demand for smaller, lighter avionics in vertical take-off and landing aircraft requires advanced processing capabilities, pushing beyond single-core processor limitations. While multi-core processors offer significant computational gains, their integration into safety-critical avionics presents substantial certification and engineering challenges. The demonstration of the software determinism on multi-core processors is of high complexity for current commercial-off-the-shelf platform solutions. Their unpredictable architectures and undocumented features lead to contentions and hard to compute worst-case execution time when shared resources are accessed. This paper focuses on Airbus Helicopters' strategy to overcome those barriers by identifying, characterizing, and mitigating the sources of non-determinism. It highlights that a traditional top-down development approach from system requirements is not sufficient on its own to cope with the challenge of integrating multi-core for critical software. The approach described in the paper integrates an adaptive, iterative system engineering process that complements the top-down approach with a bottom-up approach based on the outcomes of the Hardware and Software design.

1. INTRODUCTION

The avionics system for helicopters and Vertical Take-Off and Landing (VTOL) aircrafts is required to be smaller and lighter than for commercial airplanes, due to weight and space constraints. However, airborne systems continue to evolve and more advanced avionics processing capabilities are required to ensure new operations are safe and efficient.

Single core processors were traditionally used by the aeronautical industry to host safety-critical software applications, as they offer a good predictability in execution time and memory access, as well as a reduced complexity of the software architecture. On the other hand, the limitations induced by their processing capacities are a major obstacle to implementing new computational intensive functionalities. The consumer market has succeeded, for several years, to introduce software ever more efficiently thanks to the integration of multi-core processors providing more computing power while maintaining the bill of material.

This paper will first make a status of the research projects on multi-core and highlight the challenge of determinism. It will recall the existing Regulation framework when certifying an aircraft and present the Standard applicable when using a Multi Core Platform (MCP). Furthermore, this paper will outline Airbus Helicopters strategy for identifying, characterizing, and mitigating the sources of non-determinism, through an adaptive, iterative system engineering process that combines traditional top-down approach with bottom-up constraints coming from hardware and software design processes.

2. RESEARCH STATUS

For the past twenty years, many research projects have been conducted by the aeronautical industry and research bodies, jointly with Airworthiness Au-

thorities, to study the use of multi-core in avionics equipment with regard to safety.

They primarily focused on identifying the inherent challenges of predictability in multi-core systems. For example, the MULCORS project (Ref. 1) carried out for the European Aviation Safety Agency (EASA) aimed to provide recommendations for a safe integration of multi-core processors into embedded aircraft systems. Its objectives include surveying the MCP market, defining assessment criteria, investigating representative processors, identifying mitigation strategies, and recommending changes to the current Airworthiness guidance.

The report “Assurance of Multicore Processors in Airborne Systems” (Ref. 2) sponsored by the Federal Aviation Administration (FAA) also discusses this topic, but highlights the necessity to complement the analysis of the sources of non-determinism in the hardware platform with a usage domain definition (coming from functions definition and software architecture choices) that will help limiting the scope of the demonstration.

The key outcome of most of those projects is that for multi-core systems based on Commercial-Off-The-Self (COTS) MCP, the traditional definition of determinism (as a predictable outcome based on the preceding operations and data, occurring in a specific period of time with repeatability) has to be refined. While processing determinism in the same way as single-core systems is often impossible due to the inherent complexity of the COTS MCP hardware design, the focus should be made on predictable and bounded behaviors in a safety oriented approach. A special care should be made on the source of non-determinism: the interference channels. They are platform properties that may cause interference between SoftWare (SW) applications or tasks. They are of different types, and can either lengthen the execution time of a SW partition or corrupts its data resulting in an unexpected or unsafe behavior of the MCP platform.

3. CERTIFICATION AND MULTI-CORE

The regulatory landscape has also undergone significant maturation in the past five years to consider the latest industrial standards on System, Hardware and Software. An update with regard to multi-core processor aspects was then necessary.

3.1. Standards and Regulations Framework

In order for a newly developed aircraft model to enter operational service, it must receive a Type Certificate (TC) from the relevant aviation authority. This certificate ensures that the aircraft meets all applicable safety and regulatory standards.

The EASA has published a set of detailed airworthiness requirements and standards in Certification Specifications (CS) documents: CS-27 “Certification Specifications, Acceptable Means of Compliance and Guidance Material for Small Rotorcraft” (Ref. 3) applies for small rotorcrafts with a maximum weight of 3175 kg or less than ten passengers seats, whereas CS-29 “Certification Specifications, Acceptable Means of Compliance and Guidance Material for Large Rotorcraft” (Ref. 4) applies for large rotorcrafts. Specificities of VTOL are handled through EASA “Special Condition for small category VTOL-capable aircraft” SC-VTOL (Ref. 5). Those rules must be met to obtain a TC.

One specific requirement (CS27.1309, CS29.1309 or VTOL.2510) addresses equipment, systems and installations. Its main purpose is to ensure the safety and the reliability of all equipment, systems and installations within the aircraft. It is written as a performance-based rule, by setting the safety objective without prescribing any design solution. Compliance is typically demonstrated through a rigorous safety assessment process, often guided by Acceptable Means of Compliance (AMC) and Guidance Material (GM) that the EASA publishes alongside the Certification Specifications.

Two of the industry standards that have been developed for meeting those objectives are the EUROCAE ED-79B / SAE ARP 4754B “Guidelines for Development of Civil Aircraft and Systems” (Ref. 6) published in 2023, and the EUROCAE ED-135 / SAE ARP 4761A “Guidelines for Conducting the Safety Assessment Process on Civil Aircraft, Systems, and Equipment” (Ref. 7) published in 2023. These documents define the rigorous system development processes, including the use of Functional Hazard Assessment (FHA), to identify potential failure conditions and classify their severity (Catastrophic, Hazardous, Major, Minor, No Safety Effect).

Based on the severity of the failure condition and the system architecture, a corresponding Functional Development Assurance Level (FDAL A, B, C, D or E) and Item Development Assurance Level (IDAL A, B, C or D) are assigned. FDAL focuses on the air-

craft and system functions and the level of rigor of their development, whereas IDAL focuses on the items (hardware and software) and their development assurance.

To comply with a Software IDAL, the production of software needs to comply with the EUROCAE ED-12C / RTCA DO-178C “Software Considerations in Airborne System And Equipment” (Ref. 8) standard which is recognized by the EASA and FAA as an Acceptable Means of Compliance (AMC) in the A(M)C 20-115D (Ref. 9).

Similarly to comply with a hardware IDAL, the production of airborne electronics, needs to comply with the EUROCAE ED-80 / RTCA DO-254 “Design Assurance Guidance for Airborne Electronic Hardware” (Ref. 10) standard which is recognized by the Authorities in A(M)C 20-152A (Ref. 11).

In addition to those recommended practices, a joint effort of the Airworthiness Authorities allowed an evolution from informal position papers such as CAST-32A (Ref. 12), to guidance document for the use of multi-core processors in airborne systems and equipment provided in the A(M)C 20-193 (Ref 13) published in 2022 by the EASA and the FAA. For specific case of European VTOL, an update of Means of Compliance to VTOL.2510 is under consultation incorporating A(M)C 20-193.

As depicted in Figure 1, showing compliance with the CS 27.1309 & 29.1309 airworthiness requirements can be achieved by following the guidance proposed in the AMCs & GMs defined by the Airworthiness Authorities which recognize the standards applicable for system and electronics hardware and software engineering.

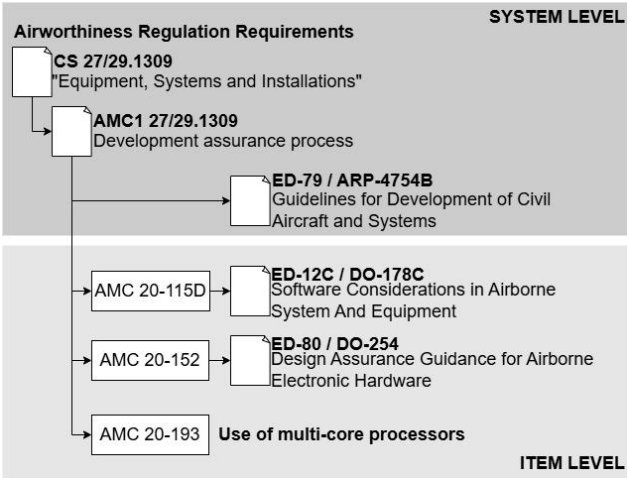

Figure 1: CS27/29.1309 rules compliance tree

3.2. A(M)C 20-193

The A(M)C 20-193 guidance applies to systems and equipment with COTS MCPs that have two or more activated cores, where the IDAL of at least one hosted software application or the hardware item containing the MCP is A, B, or C. Items DAL D or below typically do not require this guidance.

This AMC provides a set of objectives split into few stages that certification applicants have to meet along their development life cycle. These objectives give a rigorous and structured approach to show how the software applications or tasks executing on an MCP will behave in a deterministic and safe manner, and will have sufficient time to complete their execution.

The first stage of this AMC guidance gathers the planning objectives to clarify information regarding the MCP platform hardware and software items and method and tools used to meet the objectives.

The second stage aims to identify and document the critical configuration settings of the MCP that are essential for the hardware and hosted software to meet functional, performance, and timing requirements. A focus is also put on protecting the configuration integrity and detecting or recovering from inadvertent modifications.

The third stage concerns the identification and characterization of all potential interference channels on the platform (e.g., shared caches, memory buses, Inputs/Outputs paths) and characterization of their impacts. This includes verifying any deployed interference mitigation strategies. It is completed with a verification that the allocated resources of the MCP and its interconnect (e.g. bandwidth, memory regions) are not exceeded by the demands of the hosted software applications in their intended final configuration.

The fourth stage continues with a verification that the hosted software applications execute correctly within their allocated resources and make sure that data and control coupling between them has been exercised.

The fifth stage finally takes into account the additional complexity of developing a safety critical MCP with a complement of detection of any misbehavior and handling with a safety net.

4. AIRBUS WITH HELICOPTERS STRATEGY TO COMPLY WITH A(M)C 20-193

The following paragraphs will describe the strategy defined by Airbus Helicopters (AH) to integrate MCP in avionics systems according to the A(M)C 20-193.

4.1. Impact on AH design processes

The classic top-down design approach defined in EUROCAE ED-79B / SAE ARP 4754B, flowing down from System requirements to hardware and software design, is no longer sufficient on its own. It has to be supplemented with the constraints encountered during the hardware and software design processes in a bottom-up approach as early as possible in the schedule, to avoid discovering late in the development incompatibilities between system needs, hardware solution and selected software architecture.

AH has put in place an iterative process as described in Figure 2, between the System development process (top-down approach), and the hardware and software development processes (bottom-up approach) to comply with A(M)C 20-193.

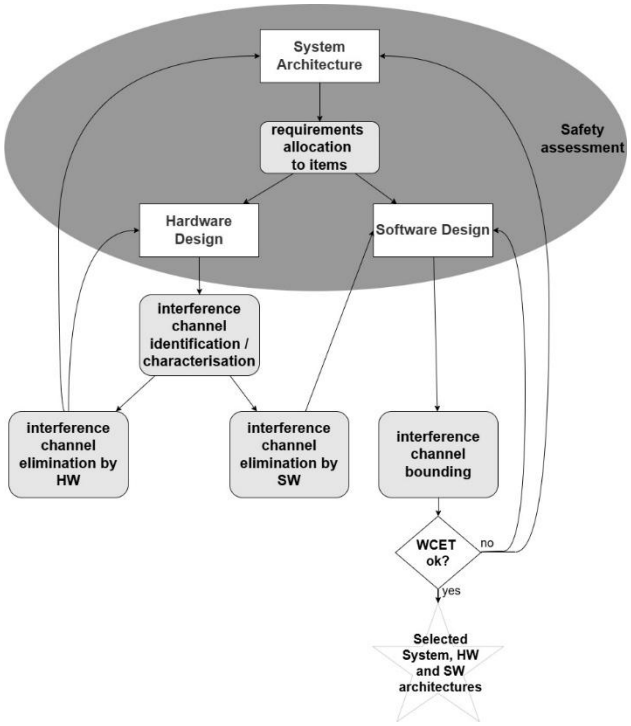

Figure 2: Iterative process between System, HW and SW processes

This specific process is based on four activities, and is inspired from the framework proposed by PHYLOG certification methodology (Ref. 14). From the hardware design process, the first activity is to perform the interference channels identification (see section 4.2) and characterization (see section 4.3). Then activity of interference elimination is done at HW and SW level (see section 4.4), and a bounding of remaining interferences contributes to validate the chosen architecture (see section 4.5).

These activities have to be coordinated between system design architecture, safety engineering, software design and hardware design disciplines, as all processes can be reassessed during the selection of the System architecture.

4.2. Identification of the interference channels

In an MCP device, multiple instruction and data paths can exist, starting from various initiators to target (as defined in Ref. 14, the initiator being a component initiating the transaction, and the target being an end-component where single or multiple instructions or data transactions can take place). A physical path may go through shared hardware resources with other paths. An interference occurs when a shared resource is used by concurrent transactions, and this sharing cannot be supported by the resource without creating a penalty on one transaction time.

Two types of interference channels can be distinguished. The first type is called direct interference, when there is a direct competition for resources between tasks or applications running on different cores, as illustrated in Figure 3. In this example, both the cores 0 and 1 need to access the Double Data Rate (DDR) memory simultaneously. Core 0 faces a Cache L2 miss (the core has to go to slower main memory with a longer access time to fetch the data) and Core 1 writes into the main memory, resulting in contention for read and write access to that shared memory.

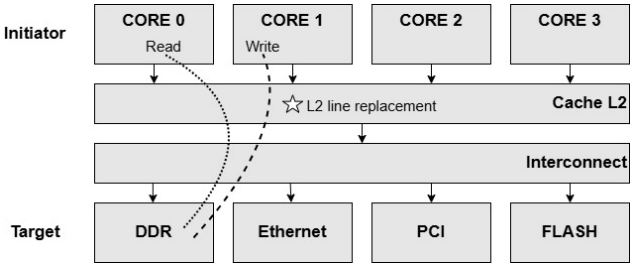

Figure 3: Example of direct interference channel

Most of the time, the direct interferences are related to:

- shared last-level caches L2/L3: when multiple cores access a shared cache, they compete for cache lines and one core might

evict data or instructions that another core needs
- main memory access through a common memory controller or memory bus
- shared bus and Interconnect interferences due to congestion and arbitration of transactions
- shared Inputs/Outputs (I/O) device interferences.

The second type is called indirect interference where an hardware unpredictable behavior is caused by contention to access shared resources. For example hardware can activate new path routing, a change in an arbitration logic, creating a specific contention as described in Figure 4. In this example, the bus's interconnection block changes unexpectedly from a Core-index priority algorithm to Round-Robin. As a result the time penalty prediction for a given core to access the DDR memory is no longer valid and interference shall be re-characterized.

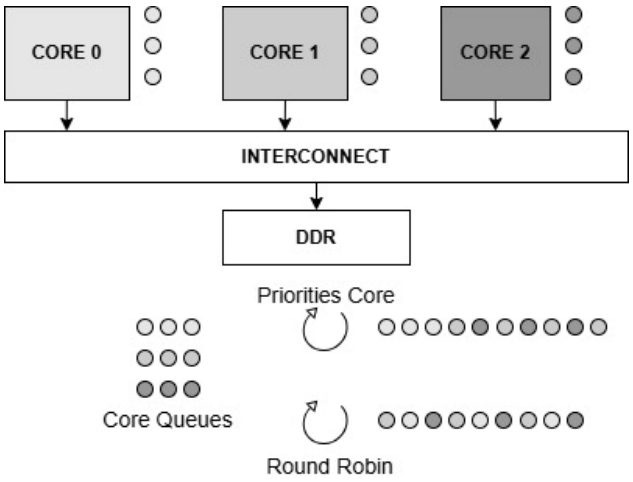

Figure 4: Example of indirect interference channel

Most of the indirect interferences can be identified by having access to COTS MCP manufacturer data, or by performing tests.

4.3. Characterization of the interference channels

Once the interference channels have been identified, their resulting time penalties need to be characterized for all transaction profiles.

A transaction profile details how a specific initiator, called victim (e.g. CORE 0), access a certain target (e.g. main memory), while other(s) initiator(s), called aggressor(s) (e.g. CORE 1), might target the same or different target as illustrated in Figure 3.

Specific test cases will be built for each transaction profile and used to measure time with and without aggressor(s). All results are then examined in front of the intended System application to determine if interference channels are either acceptable, i.e. the interference channel penalty should not prevent compliance with performance requirements set at System level, or unacceptable, i.e. the interference channel results in data corruption, or prevents compliance to performance requirements set at System level.

An example of unacceptable interference channel is given in Figure 5: in this scenario, the victim performs a Load instruction, whereas the aggressor performs a Store on shared L2 cache. When both are running concurrently, the execution time of an atomic Load instruction performed by the victim is ten times longer than when the CORE 0 is running alone.

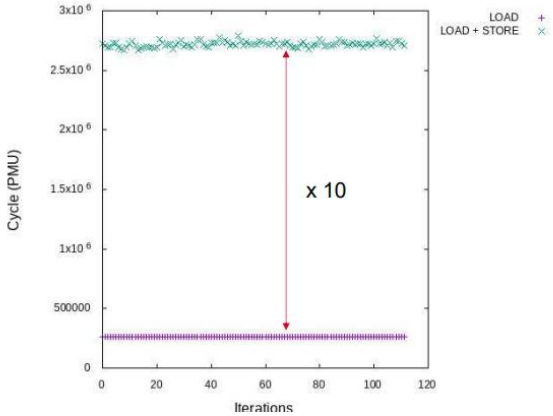

Figure 5: Characterization of interference on Load instruction

All unacceptable interference channels shall be eliminated by design choices at hardware or software level.

In case an interference channel has triggered an unexpected failure mode within the processor, it is characterized as unexplained, and further investigation in collaboration with the manufacturer is required.

4.4. Elimination of interferences by HW and SW

As the number of interference channels is growing with the complexity of the Multi-core Processor architecture, analysis of interference channels may result in extensive and costly engineering work. Any simplification of the MCP usage to reduce the number of possible activated channels has to be consid-

ered. Mitigation by elimination can be managed at Hardware, Software or System level.

At hardware level, an internal elimination of interference can be done for instance by deactivating the use of shared resources (e.g. L3 cache), or by deactivating some mechanisms, such as the cache replacement policy.

At equipment software level, software-based memory management solutions can be developed to prevent one application from corrupting the memory space of another, or from creating concurrent access to a shared memory region. For instance, when multiple cores are operating concurrently and sharing one or more levels of cache (for example L2 or L3 caches), a common and effective technique is the cache coloring technique. Instead of letting all cores compete for the entire cache, the cache is logically or physically divided into distinct regions. Each region (or "partition") is then dedicated to specific applications or cores. This can be achieved by "coloring" memory pages such that only certain pages can be mapped into specific cache ways or sets. This method significantly reduces cache contention between different software applications or cores. By isolating the cache usage of critical tasks, the timing interference is eliminated, as they are less likely to be "evicted" from the cache by other core's activity. On the other hand, it can lead to a non-optimal cache use, as parts of the cache might be empty if the associated application is not using it fully.

Another choice of cache management, when used sequentially by multiple cores, is to force each application software to trigger the flushing or the invalidation of the cache. That ensures that fresh data will be fetched from main memory for the next core. In this situation, at first fetches or reads, the software is always executed in the worst condition of cache access timing. It introduces a significant latency but guaranties determinism of the execution by removing the variability in the cache content.

The use of an Operating System (OS) or an hypervisor can also help managing the access to shared resources and providing control mechanisms to enforce the isolation of the software applications.

At application software level, elimination of the interferences will rely on the Software architecture definition from System or Functions requirements allocated to the software detailing expected behavior and performance.

As discussed earlier at equipment software level, while the primary goal is isolation, applications on different cores or within different partitions often need to communicate and share data. This can be done through controlled shared memory regions, that are specific memory areas that are explicitly mapped and accessible by multiple partitions, with careful synchronization mechanisms (e.g., semaphores or mutexes) to manage concurrent access, or by using OS services.

Robust synchronization protocols have to be implemented to prevent data corruption due to simultaneous read and write operations, and to avoid deadlocks when several tasks are waiting for the resource held by others.

Several software applications of different IDALs may have to run on the same MCP platform. To avoid jeopardizing the execution of a safety critical partition by one of lower IDAL, one software architecture choice may be to define groups of partitions based on IDALs that are allowed to run concurrently on the platform. In addition, real-time constraints derived from System requirements have to be taken into account in the analysis.

The Software architecture activity has to take into account the need expressed at System level within the required performance, and the interference channel analysis performed at hardware level.

Some software architecture patterns may therefore not be suitable. As shown in Figure 6, a pattern as Case a. where software applications run totally isolated from the others will not permit to take advantage of the increased processing capabilities offered by the multi-core configuration compared to the single-core one.

Parallel pattern illustrated in Case b. will exploit the full capacity offered by the cores, but it will induce interferences between software applications running concurrently that can be difficult to eliminate.

In the silo pattern described in Case c., potential interferences will be limited to the context of a single parallelized software applications, but are not eliminated.

Any combination of those patterns may be envisaged, and there is no “typical architecture”, as one optimal for a dedicated function may likely not be optimal in another functional context. Analysis will have to be reperformed each time.

Software architecture will therefore aim at eliminating as much interference between software applications as possible, without compromising the timing performance. When interferences cannot be eliminated by design, they will need to be quantified.

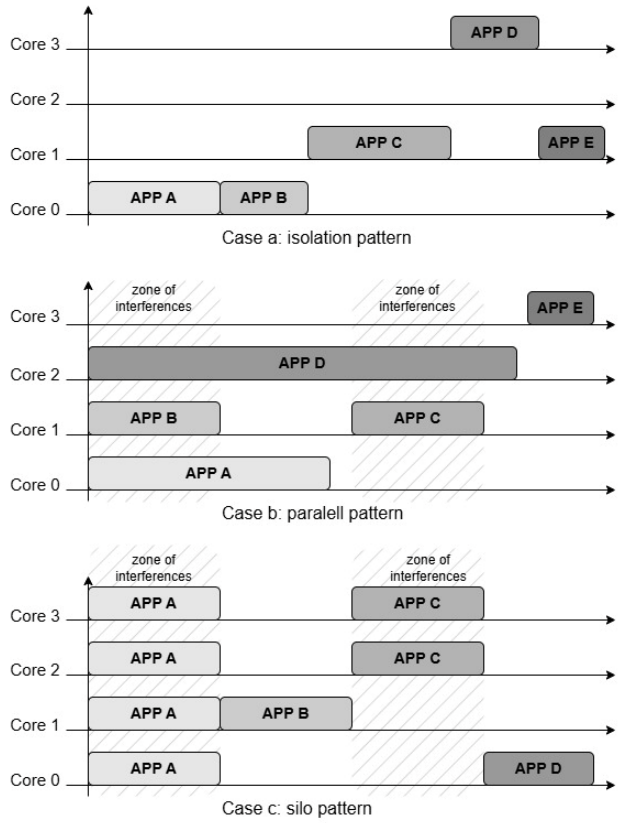

Figure 6: Examples of Software architecture in MCP configuration

4.5. Bounding the interference by measurement

One of the key objectives of the A(M)C 20-193 is to verify that the hosted software applications have sufficient time to execute. To demonstrate this, the goal is to compute per partition a Worst-Case Execution Time (WCET) bound that is safe (as it should never be violated) but also tight (as it should not be too pessimistic).

When interferences cannot be all eliminated, verification activities have to be defined to confirm that the identified interference channels are properly managed and that the WCET of each software application remains bounded even under worst-case interference scenarios.

From the analysis performed during hardware platform characterization and the choices made during hardware design and software architecture processes regarding resource allocation strategy and software architecture pattern, the first step is to characterize the remaining interferences. For instance, specific test applications can be designed to “stress” the software application under test: the goal is to put it in a worst-case state on a specific shared resource. Measures are performed and the maximum accumulated interference overhead due to contention on a shared resource is obtained.

The computation of the WCET value of the software application will have to take into account the sum of the maximum interference delays identified previously. For conservatism, or to cover a potential omission during the hardware characterization, a final global margin can be added to the final value.

If WCET exceeds the time slot allocated to the software application, or if the margins are too small, the process must be iterated. This involves refining the system or software architecture and the resource management, reviewing the allocation of software partitions to a core, or optimizing the source code, and re-evaluating the interference channel analysis. It can also lead to a redesign of the hardware platform.

If the WCET value is in line with the allocated time slot of the software partition, the corresponding objective of A(M)C 20-193 is considered as fulfilled.

4.6. Error Handling and Safety Net

If the major challenge lies in estimating interference penalties from non-exhaustive testing, and providing a rationale for the trustworthiness of these estimates, it will not be enough to consider that the systems can meet the highest level of safety objectives (Catastrophic, Hazardous). In addition to interference channel analysis, elimination and bounding, A(M)C 20-193 requires detection and handling of additional errors and failures stemming from the specific unknown behavior of MCPs.

This unknown behavior can result from unexpected interference which adversely impacts hosted applications. Based on the severity of these impacts, safety net measures may be necessary in addition to the interference elimination mechanism described in the section 4.4.

The safety net approach is not new and has been documented for single core microprocessors in the FAA document “Microprocessor Evaluations for Safety-Critical, Real-Time Application” (Ref. 15).

This approach needs to be extrapolated for multi-core and formally documented.

Several safety net techniques can be used and include, but are not limited to, external monitoring of safety-related behavior, redundancy, external independent watchdogs, architectures that allow run-time correction.

The safety net solution needs to coordinate multiple engineering domains including system architecture design, safety, software and airborne electronic hardware design assurance. These disciplines must be brought together to produce a safe integrated system hosting a less reliable hardware COTS multi-core.

5. CONCLUSION

The integration of COTS multi-core processors into helicopters and VTOL safety-critical avionic systems presents a complex but essential challenge. While these processors offer significant computational advantages over traditional single-core ones, their unpredictable architectures and undocumented features introduce considerable hurdles for engineering and certification, particularly concerning non-determinism.

Thanks to the existence of new methods elaborated by research projects, the aeronautical industry does not prevent itself anymore from using a multi-core processor for software development up to IDAL A. Experience has shown that a pure top-down approach based on System requirements as presented in EUROCAE ED-79B / SAE ARP 4754B is not self-sufficient to cope with the specific challenge of interference channel management. Then, the traditional top-down approach is complemented by a bottom-up approach based on the outcomes of the Hardware and Software design activities.

The engineering strategy developed by AH to overcome the multi-core conundrum is based on a collaborative and iterative effort across system architecture, safety, hardware and software life cycles, supported by a specific MCP framework. There is a strong adherence between the final Software architecture, the chosen HW multi-core solution in the usage domain refined from System definition. In case the HW solution has to be changed, the complete System process needs to be re-entered, leading to extra cost and schedule impact.

One solution to avoid this would be to use custom multi-core processors as studied in MINOTAuR project (Ref. 16). This research project is aiming to eliminate all sources of non-determinism directly in the MCP. It could represent a significant advancement in the design of real-time multi-core processors by demonstrating the feasibility of creating a timing-predictable core specifically designed for safety critical applications. These advanced hardware solutions are not yet mature enough for a widespread adoption, but when they will be market-ready, a tradeoff between development price of the MCP and associated tooling ecosystem, and the cost of the process presented in this paper will have to be performed.

Authors contact:

Claire Aventini claire.aventini@airbus.com

Yannick Degot yannick.degot@airbus.com

Frédéric Faubladier frédéric.faubladier@airbus.com

Louis Fabre louis.fabre@airbus.com

6. REFERENCES

1. Xavier Jean, Marc Gatti, Guy Berthon, and Marc Fumey. 2012. “MULCORS: Use of Multi-core Processors in Airborne Systems” (EASA Project. 2011/6). Technical Report. EASA.
2. Laurence H. Mutuel, Xavier Jean, Vincent Brindejonc, Anthony Roger, Thomas Megel, E. Alepins. “Assurance of Multicore Processors in Airborne Systems” (FAA Report DOT/FAA/TC-16/51). Technical Report. FAA, Jul 2017
3. CS-27 “Certification Specifications, Acceptable Means of Compliance and Guidance Material for Small Rotorcraft”, EASA, 2003
4. CS-29 “Certification Specifications, Acceptable Means of Compliance and Guidance Material for Large Rotorcraft”, EASA, 2003
5. SC-VTOL “Special Condition for small-category VTOL-capable aircraft” issue 2, EASA, June 2024
6. EUROCAE ED-79B / SAE ARP 4754B: “Guidelines for Development of Civil Aircraft and Systems”, 2023
7. EUROCAE ED-135 / SAE ARP-4761A: “Guidelines for Conducting the Safety Assessment Process on Civil Aircraft, Systems, and Equipment”, 2023

8. EUROCAE ED-12C / RTCA DO-178C: Software Consideration in Airborne Systems and Equipment Certification, 2011

9. A(M)C 20-115D: Airborne Software Development Assurance using EUROCAE ED-12 and RTCA DO-178, EASA Oct 2017, FAA Jul 2017

10. EUROCAE ED-80 / RTCA DO-254: Design Assurance Guidance for Airborne Electronic Hardware, 2000

11. A(M)C 20-152A: Development Assurance for Airborne Electronic Hardware (AEH), EASA Mar 2021, FAA Oct 2022

12. Certification Authorities Software Team (CAST) Position Paper CAST-32A “Multi-core Processors”, Nov 2016

13. A(M)C 20-193: Use of multi-core processors, EASA Jan 2022, FAA Jan 2024

14. Frederic Boniol, Youcef Bouchebaba, Julien Brunel, Kevin Delmas, Alfonso Mascarenas Gonzalez, Thomas Loquen, Claire Pagetti, Thomas Polacsek, Nathanael Sensfelder. “PHYLOG certification methodology: a sane way to embed multi-core processors”. 10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)

15. Rabi N. Mahapatra, Jason Lee, Nikhil Gupta, and Bob Manners. “Microprocessor Evaluations for Safety-Critical, Real-Time Applications: Authority for Expenditure No. 43 Phase 4 Report” (FAA Report DOT/FAA/AR-10/21). Technical Report. FAA, Sept 2010

16. Alban Gruin, Thomas Carle, Christine Rochange, Hugues Cassé, Pascal Sainrat. MINOTAuR: a Timing Predictable RISC-V Core Featuring Speculative Execution. IEEE Transactions on Computers, 2023, 72 (1), pp.183-195. ff10.1109/TC.2022.3200000ff. ffhal-03773263f