White Paper on Issues Associated with Interference Applied to Multicore Processors

NOTICE

Technical Report Documentation Page

1. Report No. DOT/FAA/AR-xx/xx	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle White Paper on Issues Associated with Interference Applied to Multicore Processors		5. Report Date (month and year printed)
		6. Performing Organization Code
7. Author(s) Guy Berthon, Marc Fumey, Xavier Jean, Helene Misson, Laurence Mutuel, Didier Regis.		8. Performing Organization Report No.
9. Performing Organization Name and Address Thales Avionics, Inc., 2733 South Crystal Drive, Suite 1200, Arlington, VA 22202		10. Work Unit No. (TRAIS)
		11. Contract or Grant No. DTFACT-13-D-00008
12. Sponsoring Agency Name and Address Federal Aviation Administration Aircraft Certification Service - Technical Program & Cont. Airworthiness Branch 950 L'Enfant Plaza North, S.W. Washington, DC 20024		13. Type of Report and Period Covered Interim Report
		14. Sponsoring Agency Code AIR-134

15. Supplementary Notes
The Federal Aviation Administration Aviation Research Division COR was Srini Mandalapu.

16. Abstract
Interferences in multicore processors are undoubtedly an undesirable behavior for usage in avionics equipment, regardless of the number of cores. These interferences make the processor's performance assessment complex to achieve and therefore raise safety issues. The main risk is to trigger timing failures on the whole equipment, even if data failure modes might also be discovered during the interference analysis.

In this White Paper, the authors propose a safety process to address the question of interference. The process is inspired by partitioning analyses performed on IMA systems and described in DO-297 [8]. Because of the wide variety of equipment, processors being used, level of criticality, and needs in terms of real-time requirements, no off-the-shelf generic solution seems to be practicable. Therefore, such a process ought to be instantiated on a case-by-case basis, and presented in the certification plan as soon as possible. Its definition should contain acceptability and termination criteria that will be applied during the safety assessment phases.

Several points need be noticed. First, no limitation on the number of core seems relevant, even if a processor that embeds more cores makes the interference analysis more complex. Second, exhaustiveness of analyses is not likely to be reachable on COTS processors containing either highly complex and/or black-box components. Hence, confidence should be obtained by combining several analysis methods, e.g., semi-formalized analyses coupled with information shared with the manufacturer and expert feedbacks on the processor, or similar ones in avionics. Safety experts must drive these methods. Similarly, the interference analysis should be performed with final applications, if possible. When it is not the case, representative applications fitting with the usage domain can be used, even if trustworthiness may be more complex to achieve.

Finally, interference mitigations, especially interference elimination techniques, do not dispense with an interference analysis, which brings evidence of interference control.

17. Key Words Multi-Core Processors, DO-297, DO-254, Interferences analysis, Timing failures, Worst case execution time (WCET), CAST Position Paper #32, Deterministic Execution Models.	18. Distribution Statement This document is available to the U.S. public through the National Technical Information Service (NTIS), Springfield, Virginia 22161.		
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages	22. Price

White Paper 3, Revision B

Contributors to the paper: Guy Berthon, Marc Fumey, Xavier Jean, Helene Misson, Laurence Mutuel, Didier Regis

Contract #: DTFACT-13-D-00008
Task Number: DO-0005

COTR: Srini Mandalapu, Software and Electronics Section (ANG-E271), WJH Technical Center, Atlantic City Intl. Airport, NJ 08405.
Srini.Mandalapu@faa.gov

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AEH	Airborne Electronic Hardware
AMP	Asymmetrical Multi-Processing
CEH	Complex Electronic Hardware
CRI	Certification Review Item
COTS	Commercial Off-The-Shelf
CPU	Core Processing Unit
DAL	Development/Design Assurance Level
DDR	Double Data Rate
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
FAA	Federal Aviation Administration
IDAL	Item Development Assurance Level
IMA	Integrated Modular Avionics
IO	Input/output
IOMMU	Input/output Memory Management Unit
IP	Intellectual Property
LRU	Least Recently Used
MCP	Multicore Processor
MMU	Memory Management Unit
OS	Operating System
PCIe	Peripheral Component Interconnect Express
PLRU	Pseudo-Least Recently Used
RTCA	Radio Technical Commission for Aeronautics
RTOS	Real-Time Operating System
SMP	Symmetrical Multi-Processing
SRAM	Static Random Access Memory
UD	Usage Domain
WCET	Worst Case Execution Time

1. INTRODUCTION

1.1. BACKGROUND

For a multitude of domains, safety-related or not, the user demand for significantly increased performance in reinforced size, weight and power constrained environments, results in an increased share of multicore processors (MCP) in the current market of highly complex semiconductor devices. On the other hand, legacy embedded avionics systems are based on single core processors and may suffer from an obsolescence of these components.

Aircraft software applications themselves expand in capabilities that require increased computational performance (e.g., advanced signal processing, manipulation of large quantities of stored information) while achieving gains in size and power (e.g., integrated modular avionics). MCPs address this need, so that their usage is foreseen to steadily increase. MCPs are currently used in airborne electronic hardware, although the technology is conservatively targeting MCP with no more than two cores.

The capabilities of multicore processors stretch the current assurance processes for both software and hardware. The supporting design and verification tools may not be adapted either. Since these processors were not initially design with aircraft applications in mind, a preemptive investigation of the potential safety concerns is warranted [1]. If the MCP technology driven by the overall market shows a faster growth than the aerospace domain is able to address assurance concerns, the potential for a long-standing 'catch-up' situation is possible. The investigation therefore needs to be not only preemptive but also prognosticative.

One specific safety-related concern is associated with the demonstration of deterministic behavior. The need for determinism is requested at aircraft level and depends on the aircraft function considered. These functions may develop several types of dysfunctional behaviors potentially linked to non-determinism issues.

1.2. PURPOSE

This White Paper is an intermediate research document focusing on issues related to interferences within multicore processors. The very design of MCPs could cause interferences between software applications hosted on the separate cores and lead to non-deterministic behavior that could be unacceptable in safety-critical systems [2].

The purpose of this White Paper is to explore new methods and tools of MCP verification in order to address certification issues related to the use of MCP in hard real time avionics systems. It focuses on:

- The introduction of interference in MCP and the associated safety process;
- The provision of examples of interference analysis techniques;
- The provision of examples of internal mitigation techniques within a MCP: reduction techniques, bounded interferences solutions and some interference free solutions.

The structure of this White Paper follows the above topics. Section 2 describes interference in MCPs and how they could be taken into account from a safety point of view. Section 3 introduces interference analysis technique for black-box components. Section 4 explains possible internal mitigation techniques. Finally, section 5 highlights some points that must be taken into account when dealing with interferences in MCP.

1.3. CONTEXT

This White Paper focuses on interferences that bring jitters in the transactions' processing time. Interferences linked to hardware or software failures are not addressed.

Most safety-critical avionics systems are defined as "hard real-time," i.e. they must perform their function within pre-defined deadlines. Missing a single deadline at system level is considered a failure condition (see reference [1] §1309) that may be classified as catastrophic.

The main concern related to the use of commercial off-the-shelf (COTS) multicore processors, in the safety-critical domain, is their lack of predictability. This issue is mostly due to the difficulty in foreseeing and managing inter-core interferences stemming from the sharing of common resources (memory, system-level caches, interconnect and input/output) among multiple cores. Various applications are executed on different cores within the same time slot, which means that they must compete to access shared hardware resources. Accesses are managed by hardware arbitration mechanisms that allow one of the tasks to access the resource while delaying the others, resulting in a contention as shown in figure 1 (excerpted from reference [3]).

From the delayed tasks' point of view, these additional delays, introduced because of the other concurrent tasks' unpredictable behavior, are interferences that break the time isolation principle required by avionics standards. From the component manufacturer's standpoint, this behavior is not dysfunctional, but it is rather a performance bottleneck. For the avionics equipment provider, however, the behavior is considered dysfunctional.

Multicore processors have dedicated hardware for spatial partitioning [2], including memory management unit (MMU) and input/output memory management unit (IOMMU), so that enforcing spatial isolation is considered today as a resolved issue. That is not the case for the timing aspects. To enable the usage of multicore processors on safety critical systems, interferences need to be controlled and techniques need to be developed to exploit multicore performance benefits.

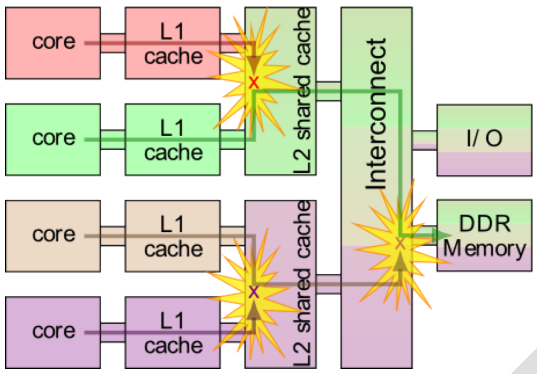

Figure 1. Interferences Due to Concurrent Access on Shared Resources

To achieve this objective, new concepts have to be introduced, potentially relying on hardware specificities, as long as they are supported by COTS processor manufacturers. That may impact processor selection processes.

1.4.DEFINITIONS

The following definitions are applicable to this White Paper, in addition to the glossary of terms that was updated in the White Paper #2 appendix A:

Initiator	A component of the processor capable to initiate proactively operations within shared resources. Examples of initiators are core processing unit (CPU), direct memory access (DMA) engines, master input/output (I/O).
Interference	An interference is an alteration of the processor's behavior seen by software running on one core, due to activities ordered by software running on other cores. A statement developed in this White Paper considers that the impact of interferences is firstly a problem of performance assessment and timing determinism enforcement. Interferences are undesired phenomena that are considered by the manufacturer as belonging to the functional domain of the processor. Interferences are however seen as dysfunctional behaviors by the avionic equipment provider, as explained by Bieth and Brindejonc in reference [4].
Interference analysis	An interference analysis is a process that considers interference paths and their usage according to the processor's usage domain. The analysis determines which paths are acceptable (from a performance's and safety's points of view) and which are not. The latter are called "interference channels" (see definition below). The objectives of an interference analysis are detailed in section 2.5.

Interference channel	An interference channel is an interference path for which interferences have actually been observed and do not cope with the equipment's functional domain.
Interference path	An interference path is a configuration wherein a given set of initiators (e.g., cores, DMA engines, master I/O) are allowed to communicate with a given set of targets (e.g., main memory, shared caches, slave I/O) without restriction. Having an interference path does not necessarily mean interferences will actually occur. It only represents a configuration for which there is a risk of interference. Hence, interference paths can be seen as test cases wherein given initiators are allowed to request given targets (e.g., core #0 targets the main memory while core #1 targets the controller of the peripheral component interconnect express or PCIe). Furthermore, the processor's behavior can be stated under each of these test cases. Interference paths aspects are developed in section 2.4. In other words: having an interference path does not necessarily mean interferences will actually occur, while having an interference channel means undesirable interference occurs.
Interference source	An interference source is a component on the processor whose simultaneous use by several cores or other initiators may entail interferences. Examples of interference sources are shared caches, peripherals, and interconnects.
Target	A component of the processor that can be requested by initiators, and either absorbs this activity (e.g., for write operations), or emits its own activity as an answer (e.g., for read operations) to the initiator's requests. Examples of targets are memories and slave I/O.

1.5. EXAMPLE

To illustrate the effect of interferences on software, consider a quad-core P5040 processor from Freescale. That processor embeds four PowerPC e5500 cores and belongs to the QorIQ™ series, which is considered by several equipment providers and organizations such as Multicore For Avionics.

In this example, the focus is on a small piece of code that performs sequential loads of 4K contiguous memory on a dynamic random access memory (DRAM) while the backward cache has been disabled, and on a piece of software within an application performing data fusion within the cockpit's software. This application is single-threaded and therefore will not benefit from intra-partition parallelism, such as symmetrical multi-processing (SMP) execution. For both cases, we collect memory access time and application's execution time while one core executes the benchmark and others execute a stressing benchmark over the same DRAM controller. At the lower level represented in figure 2, the impact of interferences between a single-core and a quad-core configuration rises to a factor of three.

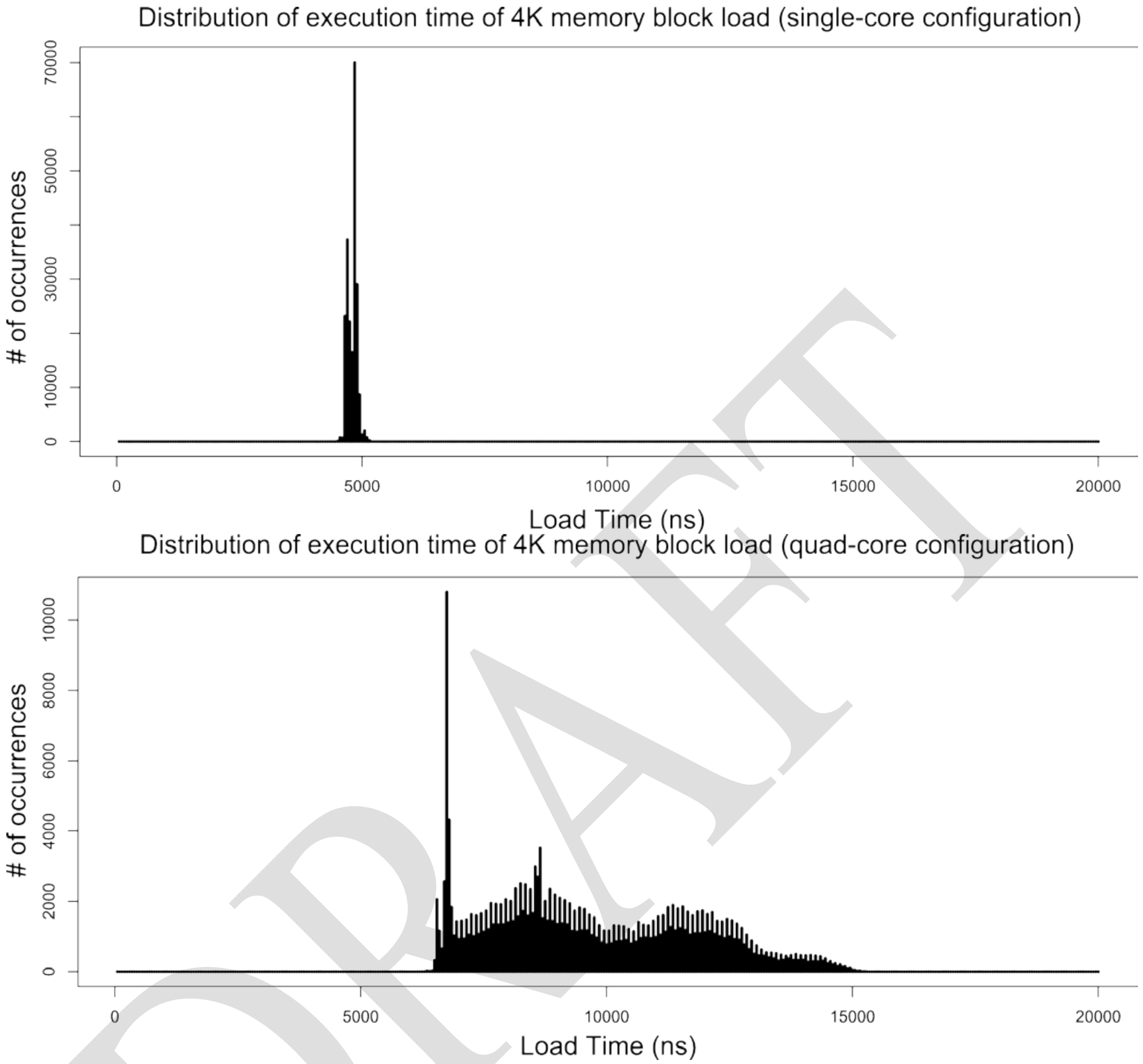

Figure 2. Example of Impact of Interferences on 4K Memory Load Operations (1 vs. 4 cores)

As represented in figure 3, within the same time, interferences impact the execution of the application with a factor close to four, i.e. the number of cores. In the literature, worse situations have been even observed. For instance, Nowotsch et al. [5] have observed a slow-down of the processor's activity with a factor of 20 on a P4080, which is an octo-core processor in the same series as the P5040.

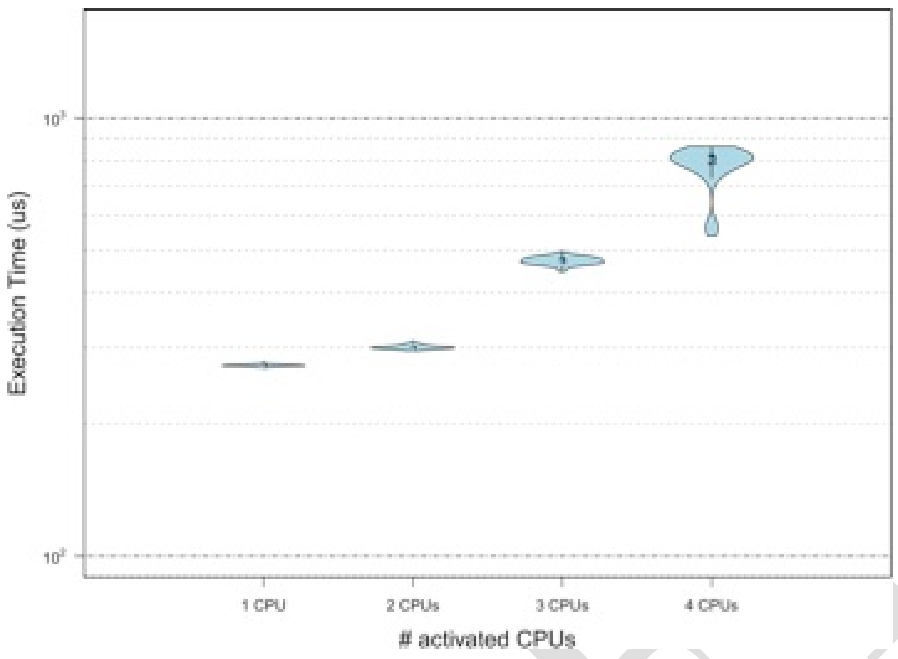

Figure 3. Impact of Interferences on a Piece of Application vs. Number of Active Cores

These experiments show that interferences are phenomena that significantly impact the embedded software's performances, and make the whole equipment prone to timing failures. Therefore, their impact must be assessed with trusted methods, and when this impact is too significant or unbounded, the appropriate mitigation has to be implemented in close relationship with the safety analyses performed at a higher level.

1.6. APPROACH

The use of MCP must be integrated in a system safety approach. This statement justifies the use of a top-down method as the primary approach, completed by a bottom-up approach. Both methodologies are presented in this White Paper.

The top-down approach (see White Paper #2) first allocates safety objectives to the MCP in accordance with the top level safety requirements' criticality. Functional failure paths are then identified. A bottom-up interference analysis on the MCP allows for identifying sources of interference not linked to microprocessor failures, and their effect on the system. Mitigations can then be implemented and justified in accordance with the identified failure effects.

The safety analysis is then completed to determine if the level of exhaustiveness and the specified sanctions are appropriate to ensure the compliance with the safety requirements. The tolerance to interferences depends on the criticality of the implemented functions.

Note that sources of interferences linked to processor failure were not considered in the scope of this White Paper. These interferences are addressed through the performance of existing failure mode and effect analysis as they are consequences of known failure modes.

2. INTERFERENCES IN MULTICORE PROCESSORS

2.1. STATE OF GUIDANCE WITH RESPECT TO INTERFERENCES

The objective of this section is to illustrate the point of view of existing guidance on the question of interferences in MCP. In some cases, the guidance has been interpreted to cover the case of interferences, while it did not intend to.

2.1.1 Multicore Certification Review Item

The multicore Certification Review Item (CRI) [5] or CAST Position Paper #32 [6] has been released in 2012 to tackle multicore-related issues. This document defines EASA's requirements regarding the use of MCP that host safety critical applications. The CRI is currently being revised by certification authorities. While the CAST Position Paper #32 scope is limited to dual core, the future CRI/IP goes beyond two active cores.

The main requirements in this document can be summarized as follows:

- Identify shared resources used in MCP;
- Identify configuration settings and define Usage Domain of MCP;
- Identify interference channels and associated mitigations;
- Define relevant integration scheme depending on interference channel analysis;
- Define relevant monitoring of MCP.

Compliance to these requirements must be demonstrated.

2.1.2 Certification Memorandum SWCEH-001

This currently published EASA Certification Memorandum (CM) [7] provides non-binding guidance material on COTS in general and on highly-complex COTS, which, per such guidance, includes MCPs. In addition to recommending activities to be performed and data to be collected depending on COTS complexity and criticality while taking into account relevant product service experience, the CM calls for quite a few additional expectations related more specifically to MCPs:

- The complexity of MCPs and the proprietary nature of their design call for more in-depth design data. Furthermore, it is assumed that these data are not readily available to users without specific agreements with the COTS supplier;
- The validation of the MCP usage domain is emphasized due to numerous built-in functionalities featured by MCPs that are potentially affecting the intended functions ultimately performed by such COTS;
- Partitioning analysis is an expected practice whenever a COTS is involved in mechanisms implemented to ensure robust partitioning between independent software portions, which is the case for MCPs hosting multiple software functions.

2.1.3 RTCA DO-297

DO-297's [8] objective is to provide guidance for integrating hosted applications on an integrated modular avionics (IMA) platform. As hosted application in an IMA context may request independence for system-level safety reason (e.g., hypothesis taken in the functional hazard assessment), it is necessary to provide an adequate partitioning solution on an IMA platform. Robust partitioning is the answer. The overall approach is based on the identification of interference channels and relevant means to mitigate their effects with regards to the safety objectives.

MCP-related challenges are not necessarily relative to IMA, but there is a similarity (considering the CRI's objectives) since interference channels' identification and mitigation is also the approach to obtain assurance for MCP usage.

2.2.SAFETY PROCESS OBJECTIVES

The increase in integration level of safety-critical avionics functions induces a more widespread use of multicore processors. As a consequence, avionics computers can host more and more safety-critical functions and are subjected to real-time constraints according to the criticality of these embedded functions. In addition to classical safety analysis on mono-core processor, special attention is required when MCPs are used, as some of the MCP additional features could cause specific failure modes.

As depicted in figure 4, the Functional Hazard Assessment (FHA) identifies and classifies the failure conditions associated with the basic functions of the aircraft. A criticality level is then assigned to every function and a Development Assurance Level (DAL) is allocated depending on the severity of failure conditions. Several functions with different DALs can then be implemented on a multicore processor that can execute several software applications at the same time.

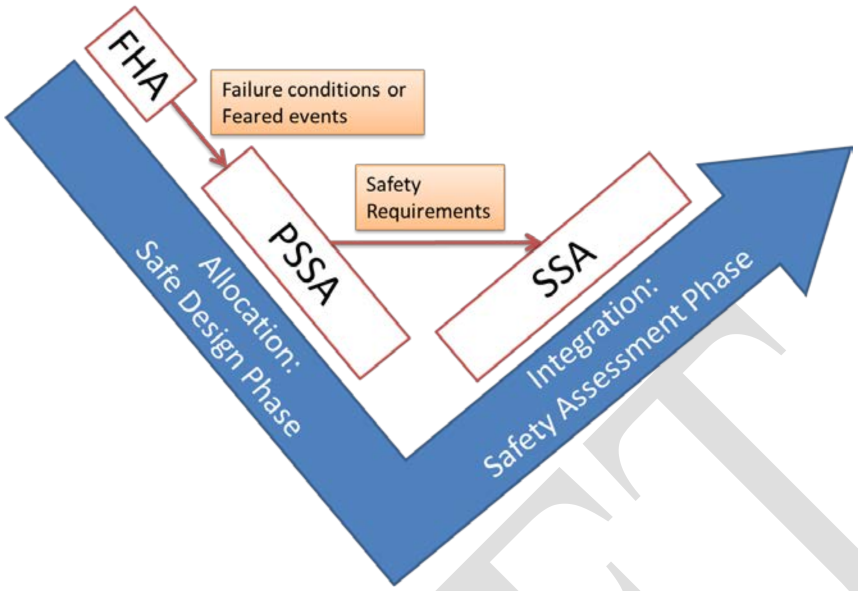
FHA
Failure conditions or Feared events
PSSA
Safety Requirements
SSA
Allocation: Safe Design Phase
Integration: Safety Assessment Phase

Figure 4. Overview of Generic Safety Process

According to ARP4754A/ED-79A [9], DALs are allocated to hardware and software items. Considering software applications, an item DAL (IDAL) is affected to each software application according to the identified criticality level. The implementation must be such that a lower-level DAL application does not jeopardize an application developed with a higher DAL to guarantee the required level of safety in terms of availability and integrity.

Interferences between applications executing simultaneously on separate cores of a MCP may contribute to these feared events. The problem does not come from the interferences themselves but rather from their consequences. The failure modes related to interferences and their effects in term of loss of integrity, loss of availability, or non-deterministic behavior of hosted applications must be identified. Depending on these defects and the safety objectives, mitigations may be required. The coverage of these mechanisms and their sanctions, i.e. corrective actions taken at processor or system level, are then analyzed and validated by safety analysis for compliance with the system's behavior.

The implementation of every appropriate mitigation must be planned, developed, documented, and verified to ensure its efficiency.

2.3. INTERFERENCE-AWARE SAFETY PROCESS

The contribution of interferences to feared events can embody several aspects.

First, interferences are a bottleneck with regards to the performance assessment on a piece of equipment. Intuitively, software running on one core is slowed-down by software running concurrently on other cores because of interferences. It is called an interference penalty. Quantifying an interference penalty from exhaustive testing is impossible in many cases as the number of test scenarios grows exponentially. Hence, a major challenge is to estimate an interference penalty from non-exhaustive test campaigns and to provide a defendable rationale that this estimated penalty is trustworthy.

Second, the situations in which interferences occur are numerous. A chip manufacturer faces the same limits of exhaustiveness in testing as an equipment provider. Therefore, one or more configurations might trigger failure modes on the processor that were not covered by the manufacturer's tests. That case is no longer a matter of performance but it impacts the system's integrity.

This White Paper describes a generic process, represented in figure 5, that covers several analyses performed on the processor (regardless of embedded software) or on the whole platform.

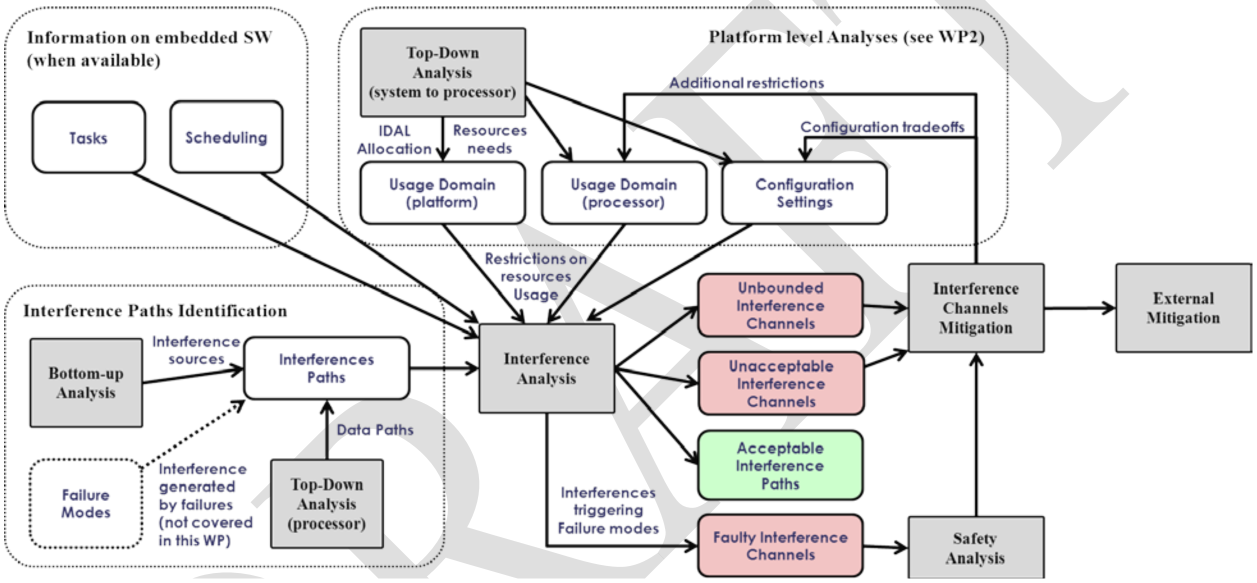
Information on embedded SW (when available)
Tasks
Scheduling
Platform level Analyses (see WP2)
Top-Down Analysis (system to processor)
IDAL Allocation
Resources needs
Additional restrictions
Configuration tradeoffs
Usage Domain (platform)
Usage Domain (processor)
Configuration Settings
Restrictions on resources Usage
Interference Paths Identification
Bottom-up Analysis
Interference sources
Interferences Paths
Data Paths
Failure Modes
Interference generated by failures (not covered in this WP)
Top-Down Analysis (processor)
Interference Analysis
Unbounded Interference Channels
Unacceptable Interference Channels
Acceptable Interference Paths
Interferences triggering Failure modes
Faulty Interference Channels
Interference Channels Mitigation
External Mitigation
Safety Analysis

Figure 5. Overview of the Interference Aware Process

The proposed philosophy is close to partitioning analyses, described in DO-297 [8]. Its process takes the following inputs:

- A top-down analysis (see White Paper #2) that allocated IDALs to all software items, and refined their needs in terms of real-time requirements. Additionally, this analysis allocated software items to execution platforms and defined a usage domain, both for the platform (e.g., scheduling rules, software development restrictions) and the processor (i.e., resources actually being used). All this information allows for having restrictions on the use of shared resources within the processor. Additionally, the top-down analysis will fix acceptability conditions for the different stages of this process, according to safety criteria;
- For federated systems, embedded software can be known with a high-level of details at an early stage of the equipment's development. That is not the case for integrated systems, such as IMA, wherein usage domain and representative applications compensate for the lack of knowledge on the final embedded software.

The proposed interference-aware safety process contains the following steps that are developed in the following sections:

1. Identification of interference paths

Thanks to a bottom-up approach, sources of interferences are listed, with a level of details that depends on the presence of black-box components within the processor. Confidence about the coverage can be obtained by coupling the result of this bottom-up analysis with a top-down analysis that evaluates which components are used when embedded software triggers operations to shared resources.

The output of this step is a set of interference paths (see figure 6), as defined in section 2.4.

2. Interference analysis

This stage takes as input a set of restrictions that is driven by the usage domain over the platform, the processor, and its configuration. These restrictions apply at two levels:

- On the overall set of interference paths. Some of these paths may be unused and do not need to be covered;
- On specific interference paths. Specific configurations within these interference paths may be stated as unreachable (with the appropriate mitigation) and thus do not need to be covered for the interference analysis.

When interferences are actually observed on a given interference path, it is called an interference channel. The interference analysis will tag each interference channel as:

- *Acceptable.* The interference channel has a bounded impact, so that an interference penalty can be assessed. Moreover, this interference penalty copes with the equipment's functional domain;
- *Unacceptable.* The interference channel has a bounded impact. However, the interferences penalty that was assessed is too high;
- *Unbounded*. No interference penalty could be assessed on this interference channel. Its impact on software execution time is not properly known. Mitigation is required to meet performance requirements only;
- *Faulty.* The interference channel has triggered a failure mode within the processor. Further investigation in collaboration with the manufacturer is required, and formalized through a safety analysis. That stage may be followed by a mitigation.

An interference analysis is, at first, a matter of performance assessment. Assuming that, ideally, the processor has no failure mode triggered by interference channels, some interference channels will have a bounded and acceptable impact on software execution time. Others will not be bounded or the bound will be unacceptable. They will be considered as dysfunctional, i.e., out of the functional domain for an equipment provider’s viewpoint but not from a processor manufacturer’s.

Even if this is not the primary objective, an interference analysis may also find failure modes within the processor that were not discovered by the manufacturer. Hence, the equipment provider performing the analysis should be aware of these potential failure modes and, in case one is observed, should retain as much information as possible to allow for further investigations in collaboration with the manufacturer.

As developed in section ☐2.5, a major challenge of conducting the interference analysis is to reach a compromise between the complexity (in time and cost) of the test campaigns, and the need to have a trustworthy coverage of interference situations.

3. Interference mitigation

This step takes as input the set of tagged interference channels. For unbounded, unacceptable and faulty interference channels, mitigation is required. Mitigations can either be internal or external to the processor. As shown in Figure 5, an internal mitigation is a restriction (e.g., configuration setting, usage domain). For example, an internal mitigation can be placed on the way shared resources are used concurrently by requiring that no more than two CPUs can request the main memory at the same time. When mitigations are internal, they may alter the processor’s usage domain and influence configuration trade-offs. In this case, the process described above is incremental. Indeed, the interference analysis should take into account these additional restrictions, and re-evaluate the interference channels that are impacted.
External mitigations mainly deal with the monitoring aspects and sanction strategy, in accordance with the safety objectives. Monitoring may be performed with a low-level granularity, for instance by tracing hardware events on the processor, or at a higher level, for instance by sanctioning deadline misses on software.

Finally, collecting statistics on interference may be used to fill a knowledge base that may be reused for other certified contexts, for instance with a higher level of criticality.

This proposed process should be instantiated on a case-by-case basis, depending on the equipment being considered, its criticality level, the selected processor and the level of detail of its internal components. The level of criticality of the equipment directly impacts the depth of analyses that will be performed.

Acceptability criteria can be proposed as early as possible, for instance during the safe design phase of the V cycle (see section 2.2) and discussed with the processor’s manufacturer. Trustworthiness in the interference analysis is obtained after the safety assessment phase.

2.4. IDENTIFICATION OF INTERFERENCE PATHS

The identification of interference paths is the first step of the proposed process.

As represented in figure 6, an interference path is a configuration wherein several initiators (e.g., CPUs) use various targets (e.g., memories, I/O) at the same time, while the electronic activity they create crosses one or more interference sources. For example, the core in the center of figure 6 uses the memory in the center along the path highlighted in pink through the right-most arbiter. That same arbiter is used by the DMA in the top-right corner to use the slave I/O in the lower right corner along the path highlighted in green. Thus the pink path and the green path may interfere.

This section proposes a classification of interference sources.

As depicted within the overall process in figure 5, the identification of interference paths is composed of:

- A bottom-up analysis that lists the interference sources encountered on the processor;
- A top-down analysis that identifies, for given initiators, which targets are concerned by the operations. For instance, a common side effect of I/O operations is to request a memory to fetch IOMMU configuration that will be cached. Potential sources of interferences affected by this side activity have to be taken into account.

The combination of these top-down and bottom-up analyses allows for building the set of interference paths.

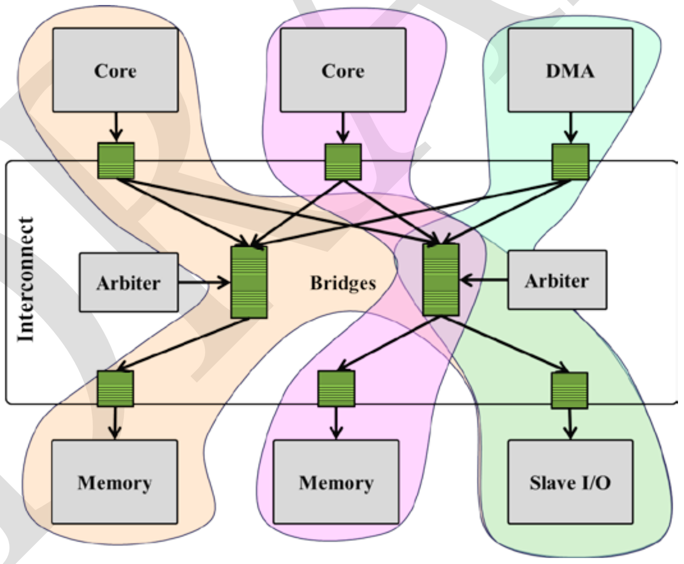

Figure 6. Example of Interference Path

2.4.1. Bottom-up Analysis to Identify Interferences Sources

Sources of interferences can be identified from a bottom-up analysis on the processor's shared resources being actually used and on the embedded operating system, if any. This analysis depends on the restrictions imposed by a usage domain over the shared resources.

Interference sources are tightly coupled with sources of non-determinism investigated in the White Paper #2. Therefore, we propose the following classification, which is close to the one presented in White Paper #2.

1. Physical sources

These sources are linked to hardware components and mechanisms.

- Internal sources: components that may process concurrent activities. Examples of such components include the main memory, the interconnects, and the high-speed busses. In the White Paper #2, we proposed an abstraction of this kind of resources in the cache and buffer paradigms. Hardware interferences are being considered according to these paradigms.

    - Cache interferences. The cache paradigm was defined as the combination of a set of data that could be requested, and a replacement logic that maintained this set of data according to the incoming requests. It is not limited to the usual cache hierarchy that is encountered in almost all processors, but also extends to features such as open pages table in double data rate (DDR) controllers.

      Caches were associated with a source of non-determinism dealing with uncertainty on their content. The same applies to interferences. This source of interferences deals with non-simultaneous operations. Intuitively, an incoming operation will alter the cache's content by triggering the replacement logic. When another operation arrives in the cache, requested data may have been removed by the replacement logic, thus entailing a cache miss. Even if operations occurred at different dates, they interfered.

    - Buffer interferences. The buffer paradigm was defined in the White Paper #2 as a set of requests that is fed by incoming flow, combined with an arbitration logic that selects the requests to be emitted. It refers, for instance, to arbitration mechanisms in the interconnect, but can also apply to request queues located in high-speed busses, and in DDR controllers.

      Buffers generate interference when their arbitration logic depends on the presence of requests from different cores. In other words, it is a source of interferences between simultaneous operations.

- External sources. On a MCP, I/Os are often shared between several applications running on several cores. Some I/O controllers, e.g., PCIe, are capable of managing multiplexing on their own, both for incoming and outgoing flows. That multiplexing can be described with cache and buffer paradigms. However, the processor has no fine view or control on incoming flows received by several remote peripherals. That is why they are refined as a distinct source of interferences.

2. Logical sources

These sources are linked to software mechanisms that are implemented within the platform software (e.g., real-time operating system or RTOS).

The notion of buffer and cache paradigms also applies to logical sources of interferences, even if this kind of source is traditionally addressed in partitioning analyses. Examples of logical sources of interference that do not refer to cache nor buffer paradigms are locks and semaphores. They can be requested at the same time by different tasks on each core, while, at most, one of them is granted the resources (and sometimes all tasks may fail).

2.4.2. Top-Down Analysis for Identification of Interference Paths

The set of interference paths can be built from a top-down analysis performed on initiators. Depending on the operations they can initiate, various components can be attained, even when they are not the main targets of these operations. For instance, a COTS processor often contains internal cache memories, such as IOMMUs that are loaded at runtime from a global configuration within the main memory. Thus, operations on I/Os may also interfere with concurrent operations on the main memory.

These targets and the resulting data paths (the components that are impacted) should be listed to determine what sources of interference they reach. Interference paths can be obtained by combining data paths from several initiators that share some interference sources.

2.5. OBJECTIVES OF THE INTERFERENCE ANALYSIS

As shown in figure 5, an interference analysis takes as input a set of interference paths (see figure 6), filters it according to the usage domain restrictions (platform and processor), identifies interference channels, and tags each of them as:

- Acceptable: interferences occur, but a bound has been found on the interference penalty, and this penalty meets the performance objectives for the equipment;
- Unacceptable: the interference penalty could be determined (i.e., a bound can be found) but it does not meet the performance requirements;
- Unbounded: the interference penalty could not be determined;
- Faulty: tests on the interference path have triggered a failure mode that was not discovered or documented by the manufacturer. Further investigation is required.

There can be up to one interference penalty per interference channel, but a same interference penalty may apply to several interference channels. In the first three cases, the interference channels are not associated with a failure mode, and thus are only considered from the performance standpoint. In the fourth case, the interference channel has triggered a failure mode and must be analyzed more in-depth from a safety standpoint. Even if the failure modes are a concern for the equipment safety, they are addressed by several analyses that apply to complex COTS. Interference analysis can simply be aware of this eventuality and report them, if any are found.

The main objective of the interference analysis is to provide a trustworthy performance assessment on the processor. The main challenges are:

- Dealing with a high number of interference paths. In many cases, it will not be possible to identify all of them. Hence a sufficient subset may be covered, as long as a rationale can be provided to justify that the reduced coverage is sufficient. For instance, it can be shown that the gathered interference penalties apply to uncovered interferences paths;

- Obtaining a trustworthy interference penalty on a given interference path, in spite of:
    - The high complexity of hardware mechanisms;
    - The eventuality of black-box and/or partially documented components within the processor;
    - The limits of testing capacities, e.g., the impossibility of reaching heavy stress levels with given initiators. Defining tests space and clarifying its limits in terms of which situations are reachable and which ones are not may be helpful.

For both challenges, termination criteria can be proposed to limit the complexity of the interference analysis. Meeting these criteria can be a sufficient condition to ensure the trustworthiness of interference penalties.

Various approaches and termination criteria can be proposed; they can be more or less formalized, and more or less computational. For instance, human-in-the-loop processes can be introduced, including experts' reviews, collaboration with the manufacturer, and collected in-service experience. The in-service experience can cover the processor itself or similar ones, such as the processors in the same series or using the same Intellectual Properties (IPs). These approaches may be developed alongside formal analyses performed on the whole processor, or specific components – some of them are detailed in section 3. The worst-case behavior of some components, such as a static random access memories (SRAM), can be found, which is supporting the trustworthiness of interference penalty. However, the equipment provider performing the interference analysis should be aware that this may not always be the case. What is important is to know and document the limitations of the interference analysis, so that the rationale associated with the interference penalty is properly justified.

A specific concern points to processors that contain black-box components. Methods have been proposed to tackle black-box components to find failure modes in reference [10]. From a performance's point of view, the following events can be feared:

- Unbounded behavior within the limits of the test's exploration space: For instance, it can be necessary to consider an infinite test space when its bounds are not clearly defined. Hence, asymptotic behaviors can be considered for test configurations that are not reached, but do not seem unreachable. These asymptotic behaviors may not be bounded, thus the interference channel becomes unbounded;

- Hidden mode changes and/or subcomponents' activation within the black-box component: These events introduce discontinuities in the processor's behavior, i.e. situations wherein adjacent tests entail non-adjacent behavior. Having discontinuities is not a problem in itself, as long as their impact is covered. What is important is to secure the absence of singularities, i.e. finite discontinuities while the interval between them remains uncovered, so that the component's behavior is unknown. One of the results of the interference analysis dealing with black-box components can be the absence of singularities within a given limit, according to metrics defined over the test space.

The interference analysis mainly brings obligations in terms of results, i.e. to have a sufficient coverage of interference paths, to identify interference channels and to provide trustworthy interference penalties for each of them. Exhaustiveness is probably not reachable using any one method, because of the "black-box" situation when using COTS. Nevertheless, trustworthiness should be reached by combining a semi-formalized approach, knowledge of the device from the MCP supplier, test campaigns, or any other method.

Finally, with this approach, the equipment provider is left the opportunity to propose and argue the methods, together with their level of formalization, automation and human intervention. This plan can be proposed and defended in front of certification authorities at an early stage of the certification process, and then be refined during the safe design phase of the V cycle.

As an output, the interference analysis produces the set of tagged interference channels. The unacceptable and unbounded channels can be studied with a mitigation objective in mind. Faulty channels are analyzed from a safety standpoint first, before being considered for mitigation.

2.6.INTERFERENCES MITIGATIONS

Interferences mitigation comes at the end of the process and applies to interference channels. As an extension, we consider that mitigation techniques can also be injected in the initial definition of the usage domain, before performing the interference analysis, with the goal to make the analysis simpler.

As represented on figure 5 the mitigation can be internal, and thus be reported, as additional restrictions within the usage domain and/or specific configuration tradeoffs; or the mitigation can be external, i.e. hosted by a third-party hardware.

The case of an internal mitigation is the richest, as it implies new iterations with the interference analysis. In the first iteration, an 'internal mitigation' will take care of the specificities of the hardware and software architectures to influence the definition of usage domain. Additionally, it may orient the equipment's development toward one of the following paradigms:

- Apply interference reduction techniques, use specific hardware features (if any), but do not try to bound interferences with software;
- Ensure that interferences are bounded within the interference channel by internal means;
- Ensure that interferences are eliminated by construction.

Each of the three paradigms above has their advantages and drawbacks. Selecting one of them has a strong impact on the rest of equipment's lifespan, as it is significantly complex to recover from such a choice.

The case of external mitigation is more straightforward. Indeed, an external mitigation for an interference channel consists of a detection means and a sanction strategy, in accordance with safety objectives. Detection is the bottleneck of an external mitigation strategy and detecting that an interference channel has ‘too much’ interferences is not well defined. A proactive detection strategy can be proposed and simply deployed. For instance, hardware events can be monitored at runtime, and unexpected events, or usage domain violations can entail errors that are handled by health monitors, either internal or external. However, it becomes difficult to give a rationale about the coverage of situations with unintended interferences.

Another approach consists in addressing detection means at a higher level, for instance by monitoring deadline misses. However, it becomes arguable to assimilate this detection as a consequence of interferences.

Finally, a long-term mitigation could also consist in collecting statistics on the processor’s behavior, both at hardware and software levels. These statistics feed a database maintained by the equipment provider, who can reuse such information in other equipment’s projects. Hence, unintended events such as deadline misses could be documented with a snapshot as precise as possible of the hardware and software conditions in which the problem occurred, so that the equipment provider could investigate.

As a conclusion, no external mitigation strategy dealing with interferences seems to fit optimally. Interference mitigation should thus combine several detection means and propose adapted sanctions, in accordance with the safety objectives.

3. EXAMPLES OF INTERFERENCE ANALYSIS TECHNIQUES

Section 2.5 outlined the objectives to be fulfilled by an interference analysis. As mentioned earlier, the interference analysis is mainly focused on assessing the performance of the platform to leverage potential timing failures.

Many different situations can be encountered with respect to the equipment’s needs, the depth of knowledge on the processor’s architecture, and the openness of the system (e.g., with an IMA). No generic method seems relevant; instead, a combination of several methods, more or less formalized, can be proposed with a case-by-case evaluation for each equipment.

Some methods may be formalized or semi-formalized (i.e. relying on hand-written abstractions). Others may rely on engineering sense and involve experts’ experience with multicore processors, for instance on lower-criticality equipment or with processors from the same series. In all cases, the support from the MCP manufacturer is beneficial to the quality of the interference analysis.

This section details examples of methods that can be used within an interference analysis. These methods are formalized or semi-formalized.

3.1. ANALYSIS OF PROCESSORS CONTAINING BLACK-BOX COMPONENTS

Many COTS processors embed black-box components, especially when these components bring competitiveness advantages. This is usually the case for interconnects and IPs for I/O control.

Sometimes, these components embed micro-controllers and execute micro-code that is also proprietary. Hence, their behavior is highly complex, and exhaustiveness of testing cannot be reached but only approximated. Termination of the analysis should then follow predefined termination criteria.

In this section we propose examples of such termination criteria on a method that targets processors embedding black-box components.

3.1.1. Enumeration of Interference Paths

As mentioned in section 2.5, the first challenge of an interference analysis is to reduce the set of interference paths down to one that is analyzable with an affordable complexity.

The initiator-target model [10] provides (and demonstrates) a formula to enumerate *test classes* on a processor containing black-box components. The processor is composed of initiators (e.g., CPU, DMA engines, master I/O) and targets (e.g., memories, slave I/O). The notion of test class, as defined in reference [10], copes with the definition of interferences channels. Hence, it is possible to apply the enumeration rules developed in the referenced paper.

The authors distinguish smart initiators (containing a memory, and thus capable to perform master-to-slave communications) from non-smart initiators (with no built-in memory, thus capable to initiate slave-to-slave communication). For example, DMA engines are non-smart initiators. Considering these classes of initiators and the number of test classes, the number of interferences channels is given by:

$$\sum_{k_{si}=0}^{n_{si}} \binom{n_{si}}{k_{si}} n_t^{k_{si}} \cdot \sum_{k_{nsi}=0}^{n_{nsi}} \binom{n_{nsi}}{k_{nsi}} (n_t^2)^{k_{nsi}} - 1 = (1+n_t)^{n_{si}} \cdot (1+n_t^2)^{n_{nsi}} - 1$$

With these notations, n_{si} refers to the number of smart initiators, n_{nsi} refers to the number of non-smart initiators, and n_t refers to the number of targets. Table 1 below illustrates the exponential growth of the number of interference channels for given processors configurations.

Table 1. Example of Interference Channel Numbering

Smart Initiators		Non-smart Initiators	Targets		Number of Interference Paths
Number of Cores	Master I/O	Number of DMA	Slave I/O	Number of Memories (DDR, flash)	
2	0	1	2 PCIe	2 DDR	424
2	0	1	2 PCIe + 1 CAN	2 DDR	935
2	1 PCIe	1	2 PCIe + 1 CAN	2 DDR	5615
4	0	1	2 PCIe	2 DDR	10624
4	0	1	2 PCIe + 1 CAN	2 DDR	33695
4	1 PCIe	2	2 PCIe + 1 CAN	2 DDR	5256575
8	0	1	2 PCIe	2 DDR	6640624
8	0	1	4 PCIe + 1 CAN	2 DDR	838860799
8	1 PCIe	2	4 PCIe + 1 CAN	2 DDR	335544319999
12	0	1	2 PCIe	3 DDR	56596340735
12	0	1	4 PCIe + 1 CAN	3 DDR	18357919871264
12	1 PCIe	2	4 PCIe + 1 CAN	3 DDR	10739383124690000

Except for very simple architectures and usage domains, the sheer number of interference paths makes an exhaustive coverage impossible to reach (remember that a second analysis has to be performed on each interference path).

We can first note that the usage domain usually already restricts the allocation of shared resources to initiators. For instance, the platform can be configured to balance the workload over the shared resources. Hence, the values given in table 1 represent upper bounds; nonetheless, the set of interference paths still grows exponentially.

The following rationale can be employed to justify that an interference path is not tested:

- Partial ordering: The interference path is already covered by a larger interference path. This rationale works as long as no mode change occurs between the smaller and the larger interference paths, e.g., dynamic voltage and frequency scaling;
- Equivalence class: From the interference sources' point of view, the path is equivalent to another interference path. To make such an argument, architectural patterns can be exploited such as the symmetry of the processor's architecture, and more generally any information stating that an initiator is treated by the hardware in the same way as another.

The last type of rationale may be used to build classes of interference paths, for which the interference analysis would perform a more in-depth exploration of an item in each class; in other words, the interference analysis would investigate a representative of each equivalence class at each level within the pre-defined order.

3.1.2. Analysis of an Interference Path

The second challenge of the interference analysis is to get a correct coverage of the possible interference situations within a given interference channel.

The first point of note is that the state space is significantly large. Examples of parameters that can be explored include:

- Types of operations available for each initiator (e.g., read/write with different width);
- Length of transactional sequences, and stress level that can be reached by the embedded software;
- Memory area targeted (in case a transaction targets a memory).

These parameters allow for defining metrics on test scenarios, i.e. *by how much* several scenarios differ from each other. We consider that the whole state space cannot be covered, so that some situations will remain unexplored. As long as the processor's behavior is continuous, exploring close configurations can be sufficient. However, one issue related to processors that embed black-box components is the difficulty to predict mode changes, i.e. discontinuities in the processor's behavior due to specific inputs.

The intuition behind the evaluation of an interference channel is to observe the presence or absence of singularities, for example, the explosion of transactions' propagation time outside of acceptable ranges. When a singularity is discovered, the interference channel is tagged as unbounded. Otherwise, the interference channel can be tagged as bounded.

This analysis makes the following assumptions on the processor:

- The processor’s behavior is piecewise-continuous, i.e. adjacent tests lead to close interferences impact, except when a mode change leads to discontinuity;
- A limit exists on test metrics under which, for any pair of tests beyond this limit, at most one discontinuity is possible on the processor.

This method guarantees that no singularity is found while the test space is covered through a mesh which granularity is lower than a given limit. In other words, the equipment provider defines metrics and mesh for the test space (i.e., the parameters that can be varied), wherein the mesh’s granularity is predicated on the selected metrics. The test results would then state that no singularity has been observed over a mesh of granularity ‘X’. The equipment provider proposes such a limit. The limit proposed by the equipment provider can vary per the criticality of the equipment. This limit sizes the number of tests to be performed, and thus the complexity and cost of the interference analysis. The assumption that the microprocessor’s interference level is piecewise continuous can be used to state that such evaluation of interference over a specified mesh provides evidence; however, the demonstration is not complete. This approach allows for providing an answer tied to a non-exhaustive test campaign that is identified as such.

Finding singularities, i.e. determining whether the interference channel is bounded or not, is one objective of this step. Another goal is to propose a trustworthy interference penalty derived from measurements. One of the problems is to deal with the imperfection of measurement methods, especially those using instrumented code. First, these measurement methods are biased by the intrinsic timing variability of the processor’s behavior. For instance, several clock domains that are not synchronized can be encountered and introduce a random synchronization delay. Second, measurements are in many cases end-to-end. For read operations, this is a go-and-back-again operation. For write operations, the end-to-end measurement is more straightforward, but may be harder to collect. For instance, it is hard to get the arrival date of a write transaction within a memory. Finally, measurements by code instrumentation may limit the range of parameters that can be explored. For instance, non-instrumented code may reach a higher level of stress on an interference channel than instrumented code.

As a conclusion, a trustworthy analysis of interference paths on processors containing black-box components may be practicable using a systematic enumeration method such as initiator-target when:

- The usage domain is sufficiently restrictive on use-cases so that the number of interference paths remains manageable. Eventually, classes of interference paths may be built so that the analysis is performed on the whole classes;
- Granularity of evaluation has been established and is sufficiently high with regard to the safety objectives.

3.2. GLOBAL INTERFERENCE PENALTIES

Approaches dealing with global interference penalties aim at jointly considering the task-set and the platform during the interference analysis. These techniques can be applied on the equipment for which one stakeholder has visibility on both the platform and the embedded software. They rely on an in-depth analysis of each task to rebuild a detailed view of its use of shared resources. An interference penalty applied onto each task can then be computed from that view.

The following methods have been proposed:

- IsWCET (Interference Sensitive WCET) is proposed by Nowotsch et al. [11]. The authors use aiT, a static analyzer developed by AbsInt. They consider a given number of time windows, and for each one, they explore all execution paths of each task, and compute a bound on shared resources usage during that time window. The global interference penalty is obtained by adding interference penalties computed for each time window;

- Application's signature is proposed by Bin et al.[12] [13]. The authors associate a signature to each component of the processor, i.e. an evaluation of the component's response or traversal time depending on its workload. The signature of a task is defined as the workload generated by this task on each component (e.g., interconnect, caches, memory). Hence it is possible to combine the signature of several tasks, and compare them with the signature of components to compute a global interference penalty;

- A computational method is proposed by Pellizzoni et al. [14]. The method uses network calculus theory. The authors compute arrival curves that correspond to the workload sent by each task to shared resources. They define a service model for the processor and a set of equations that determine delays in arrival curves. By adding these delays they obtain an interference penalty.

Global interference penalties have the advantage to perform an end-to-end analysis of interferences' impact. Their use in closed equipment, i.e. wherein the whole software is known, would be profitable to the quality of the interference analysis. Their main drawback is the difficulty in applying them early in the safe design process.

3.3. LOCAL ANALYSES

A couple of techniques have been developed to perform fine-grain analysis on specific components that are usual sources of interferences. These techniques mainly focus on interconnects and shared caches. Since they are applied locally, they must be combined with other methods to constitute the interference analysis.

3.3.1. Shared Cache Analysis

Cache analyses techniques for shared caches extend existing methods developed for private caches, and rely on static analyzers.

Historical methods consider the Control Flow Graph (CFG) of a software task, and simulate the cache content during the execution, for all possible executions [15]. A reference (instruction or data) may be classified as "always miss", "always hit", "first miss", or "unknown", depending on its presence in the cache with regards to the past executions of the sole task. Cache analyses have the following limitations:

- Data caches are hard to analyze because data references are often determined at runtime, and vary among software execution. Approaches based on value analysis using abstract interpretation techniques have been proposed [16], and WCET evaluation tools such as aiT have a good support for data caches analyses;
- Cache analyses only support well-defined policies, such as least recently used (LRU) or pseudo-LRU (PLRU) [17]. Caches embedded on COTS processors have sometimes-proprietary replacement policies, for example, streaming PLRU for Freescale processors.

For shared caches, the difficulty is to take into account conflicts between concurrent tasks without being too pessimistic. Yan and Zhang [18] represent such conflicts by annotating "always hit" and "always miss" states with a tag "except one", meaning that a concurrent task will alter this reference only once during its execution. This applies to structures such as loops that may conflict with sequential code. However, this approach still remains pessimistic. To improve on it, Hardy et al. [19] propose a concept named "bypass". Intuitively, the authors state that many conflicts in shared cache are due to evictions from lower-level caches. A significant part of these evictions deals with cache lines that are used only once, and thus have become useless. Hardy's approach consists in identifying such lines, also called "static single-usage blocks", and forces the hardware to bypass the shared cache. Hence, conflicts within the shared cache are significantly reduced.

The problem of shared cache analysis has solutions in specific cases, for instance, considering only data or instruction shared caches [20]. In the general case, especially with unified (data and instructions) caches, it remains an active research field.

3.3.2. Memory Analysis

The problem of memory's response time can be raised both for SRAM and DRAM technologies, both of them being supported by COTS processors (SRAM technologies are used in most caches). Nevertheless, on SRAM, the problem is considered as solved except for highly complex SRAM controllers [21].

An overview of a DRAM system (controller and memory) is done by Drepper [22]. DRAM analysis has been studied by Paolieri et al. [23]. In this paper, the authors model an entire synchronous DRAM (SDRAM), including the controller and the memory system. They provide static upper bounds for read and write operations. This approach yields promising results, but its limitation lays in its applicability to COTS processors that often have proprietary interleaving protocols.

3.3.3. Interconnects Analysis

Interconnects analysis have tackled the following problems:

- Worst-case behaviors of interconnects arbitration policies;
- Worst-case situations regarding the interconnect protocol.

Regarding the first problem, Bourgade et al. proposed a study of several policies [24]. The authors highlighted the case of some protocols not giving bounds on the arbitration time. That is the case for instance for fixed-priority, and some classes of First-In-First-Out (FIFO) protocols. Like other families of local analyses, the limitations of this study is the lack of knowledge of COTS interconnects' arbitration protocols. Nevertheless, some interconnects IP such as Corelink™ from ARM [25] and TeraNet™ from Texas Instruments [26] have their protocol and arbitration policy documented.

Regarding the second problem, corner cases with interconnect protocols have been studied by Shah et al. on the AMBA™ protocol, which is widespread in ARM processors [27]. They have shown that bus-locking phenomena may slow down successive transactions during arbitration phases. That result also covers interference situations.

4. EXAMPLES OF INTERNAL MITIGATIONS

In this section, a couple of state-of-the-art solutions are developed that can be used to provide internal mitigation for interferences.

Internal mitigations consisting in restrictions on the usage domain are enticing. For instance, it can be stated that some targets cannot be used by a more than a given number of initiators, or cannot be used beyond a given rate; the challenge is then on the way to enforce such a restricted usage domain. Several techniques detailed in this section define mechanisms to implement such restrictions.

Additionally, several surveys, referenced in table 2, have been proposed in the literature.

Table 2. Surveys of Interference Mitigation Techniques

Ref.	**Mitigations Covered**	**Comment**
[28]	Interference reduction Interference bounding Interference elimination	This survey is more general than mitigation of interferences, as it also covers global methods for interference analysis.
[29]	Interference bounding Interference elimination	This survey only covers software implementations on COTS multicore processors. The solutions presented in the survey are grouped under the name of "*deterministic platform software*", i.e. software components that do not bring new functionalities to payload software (e.g., applications) but provide interference mitigation.
[30]	Interference reduction Interference bounding Interference elimination	

The following sections respectively detail:

- Techniques for interference reduction that do not remove nor bound interferences by themselves, but enhance the bounds on interferences, when they exist;
- Techniques for interference bounding. These techniques are implemented in regulation software solutions. The main idea consists in granting budgets that are periodically refilled (e.g., execution time, use of shared resources) to embedded software, and let them execute until they have consumed their budget;
- Techniques for interference elimination. These techniques implement a control paradigm, which means that the use of shared resources is controlled by an external software, according to a global policy.

Table 3 describes the classes of interference channels that can be targeted by the different interference mitigation techniques.

Table 3. Classes of Interference Mitigation Techniques and Targeted Interference Channels

Interference Mitigation Techniques	**Targeted Interference Channels**
Interference reduction	Unacceptable (but bounded)
Interference bounding	Unacceptable, Unbounded
Interference elimination	Unacceptable, Unbounded, Faulty

4.1. INTERFERENCE REDUCTION TECHNIQUES

Interference reduction techniques discussed in the literature apply to sources of interferences. They highlight specific mechanisms, at hardware or software level.

More specifically, solutions that target the following sources of interference are detailed:

- Shared caches. They have been identified for a long time as bottlenecks for performance assessment in multicore processors. Hence, several solutions are proposed to reduce interferences within shared caches;
- Memory banks. The problem with memory banks is close to that for shared caches, even if their impact on performance is lower;
- Kernel locks. An issue in legacy RTOS migration to multicore processors is to make independent operations, especially system calls, as little depending on same lock as possible. That may require deep modifications to the OS design.

4.1.1. Cache Coloring

A cache management concept called ‘Colored Lockdown’ has been proposed to reduce cache conflicts by coloring techniques [31] [32]. Coloring technique aims at making a shared cache behave like a private one. Interferences are then addressed by the cache paradigm as described in the White Paper #2.

Set-associative caches are often represented as arrays (see figure 7) which cells contain blocks, usually 64-bytes wide. A data cannot be located anywhere in the cache. As illustrated in figure 8, its address is split into three segments. The 'set' segment determines in which row a data is located. The 'way', or column, is chosen by the cache replacement policy. When a cache block is selected to be evicted so that a new block can be loaded, it is within the same set.

Cache coloring aims at reducing interferences in caches by ensuring that data and instructions used by software running on different cores will be stored in different sets. Allocation and/or mapping of these data is complex and may require specific allocators. In reference [31], the page management system of Linux has been modified to do so. When a cache is properly colored, interferences are almost removed: contents are independent, but access is not. Moreover, the equivalent private caches are efficient as they keep a high associativity level.

In another approach, Ward et al. have assigned colors to tasks with an underlying support for cache coloration, and alter the scheduler to minimize cache conflicts while keeping some dynamism within their system [33]. Hence, several independent tasks may be maintained within the system while their conflicts are reduced over their execution.

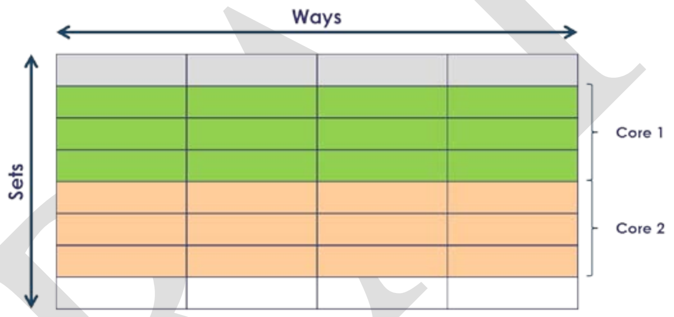

Figure 7. Example of Colored Cache Shared Between Two Cores

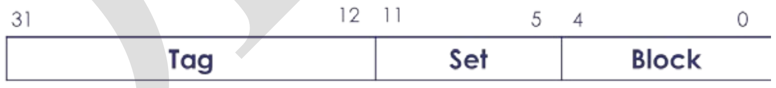

Figure 8. Address Sections from the Cache's Point of View

4.1.2. Cache Partitioning

Cache partitioning is a more straightforward way, than with cache coloring, to simulate private caches from a shared cache. As represented in figure 9, cache partitioning consists in restricting some ways to given address ranges, usually used by software running on specific cores. This method is supported by hardware on several processors, for instance on the Freescale QorIQ™ series. It is simple to deploy because it is a matter of static configuration.

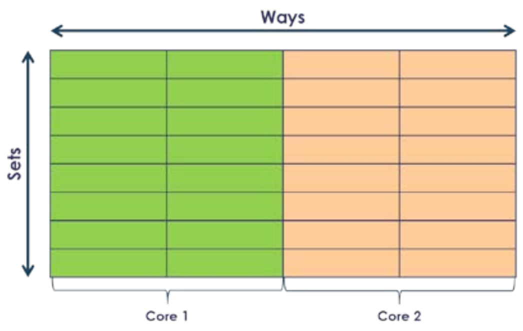

Figure 9. Illustration of Cache Partitioning

The drawback of cache partitioning is the lower efficiency of the equivalent private caches, since they have fewer ways.

4.1.3. Bank-Aware Memory Allocation

A solution called PALLOC has been developed within the "single-core equivalent virtual machines" framework to reduce interferences in open bank tables that are maintained by DRAM controllers [34]. This table was already classified within the cache paradigm presented in the White Paper #2; interferences linked to memory bank conflicts may have a significant impact [35].

Like Colored Lockdown, PALLOC is a dynamic memory allocator that is integrated within Linux. It is called when the kernel allocates memory, for instance to copy process binaries to the main memory from persistent storage.

4.1.4. Kernel Locks Refinement

Many RTOS supporting multicore processors have inherited from the legacy support of single-core processors. During the migration to multicore, special care must be taken to limit the number of conflicts on kernel locks due to simultaneous calls of kernel services by different cores. This problem is also called 'lock trashing', and is described for Linux by Kleen [36].

Solutions have been proposed to limit the effects of lock trashing. For instance, Cui et al. proposed an implementation of a lockless scheduler for multicore processors [37]. A common practice, called lock refinement, consists in multiplying locks for services that are independent. Two difficulties arise. First, it is hard to say on a legacy OS which services are actually independent from each other. Second, the number of locks is likely to significantly increase, making the system harder to analyze. Solutions have been proposed to manage locks, for instance by Zhang [38], but it remains an active research field for RTOS manufacturers.

4.2. BOUNDED INTERFERENCE SOLUTIONS

Several solutions have been introduced to ensure that interferences have a bounded impact on software execution time. Their main idea is to monitor the execution of embedded software and perform regulation, i.e. reconfigure the scheduler to alleviate the workload when some tasks are consuming too much of shared resources.

In practice, each task will be allocated a budget, usually a number of accesses, that grants it the right to freely access shared resources. That budget is periodically refilled and sized to fit with the task's needs. If a task exceeds its budget, a regulation mechanism detects the overflow and delays it until its budget is renewed.

From these restrictions, an interference penalty can be evaluated based on the allocated budget, and applied to each software tasks.

4.2.1. Memguard

The regulation paradigm has been implemented within Memguard [39], which is part of the single-core equivalent virtual machines framework [32]. As represented in figure 10, each core hosts a bandwidth regulator that shares the total bandwidth guaranteed at DRAM controller level (1.2GB/s in the original paper [39]) among each core. Regulation is performed with hardware performance counters hosted on each core. These counters can be configured to count each access to shared resources, and trigger an exception when that number exceeds the allocated budget.

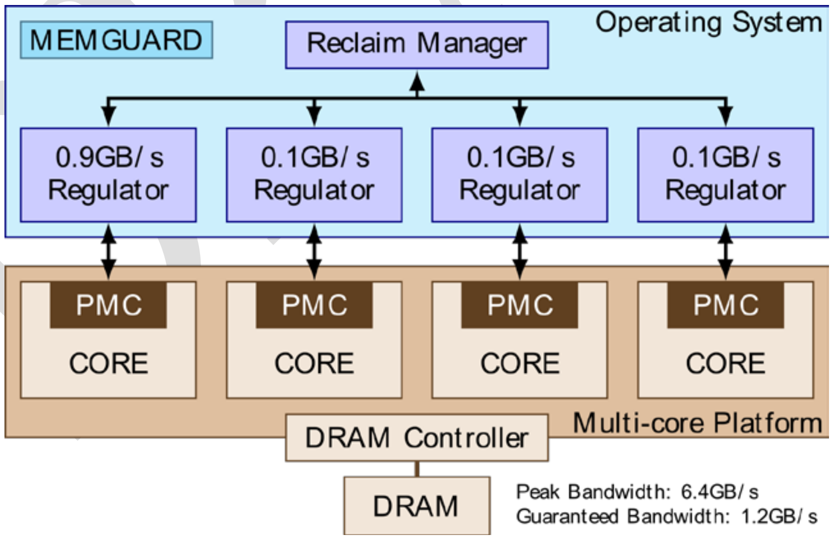

Figure 10. Overview of Memguard Mechanism

The authors have extended the Memguard mechanism to improve the overall performance by allowing software to 'reclaim' bandwidth that is unused by concurrent tasks. Even if this mechanism is not recommended for highly critical applications, it provides a significant performance improvement for equipment with lower criticality level.

4.2.2. <u>Mixed Critical Systems</u>

Mixed-critical paradigm consists in co-running critical and non-critical software, while the critical one must meet real-time criteria, while the non-critical does not and runs in a best effort way [40]. The objective related to interference is to ensure that low-criticality tasks do not interfere 'too much' with highly critical ones.

A common view of mixed-critical systems consists in scheduling the whole task set in a nominal mode, wherein both critical and non-critical tasks coexist. When critical tasks have no choice but to use the entire processing time to meet their WCET, non-critical tasks are suspended. This kind of paradigm can be defended, but the problem of availability for non-critical tasks remains.

A less constrained framework has been proposed by Kiritikakou under the name of distributed run-time WCET controller [41]. Within this framework, tasks may be instrumented to monitor their own execution by inserting intermediate checkpoints. When a checkpoint is reached too late because of interferences (or other effects causing a slow-down), the WCET controller is notified and suspends the execution of non-critical tasks, as shown in figure 11. A critical task may also take the initiative to suspend non-critical tasks.

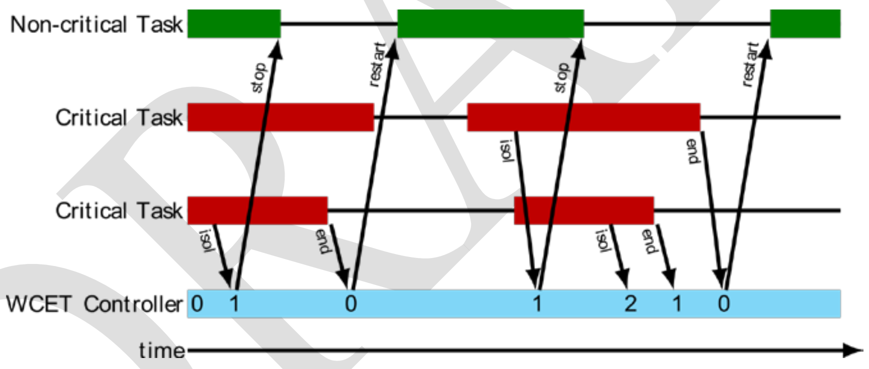

Figure 11. Overview of Distributed WCET Controller

Bounded interference solutions offer the advantage of being as little intrusive as possible. Therefore their impact on software performance is minimized. They introduce guards on the use of shared resources by embedded software. However, they do not help analyzing the low-level behavior of the processor, which means the number of interferences will be bounded, but their impact remains mandatory to assess.

4.3. <u>INTERFERENCE FREE SOLUTIONS</u>

Interference 'free' solutions have been designed for equipment that has hard real-time constraints. They focus on the major sources of interferences, so that most interferences are actually eliminated. However, these solutions do not dispense with a proper interference analysis. For instance, slight jitters may remain relative to minor sources of interference; they must nevertheless be assessed.

Removal of interference on a buffer source is usually performed through timing isolation principles. These principles may be implemented within the scheduler (e.g., critical tasks have dedicated slots), or enforced between concurrent accesses from cores to shared resources. Historically, the second approach has been developed within dedicated hardware solutions [42] [43] [44]. However, these solutions have not - so far - been embraced by COTS processors manufacturers. Hence the solutions presented in this section are only supported by software.

Interference free solutions deployed on COTS processors are grouped in the control software paradigm [29].

4.3.1. Interference Free Scheduling

Interference free schedulers apply to systems that embed applications with heterogeneous levels of criticality. They run low-criticality applications in parallel while highly critical applications are executed alone on the processor, as represented on figure 12.

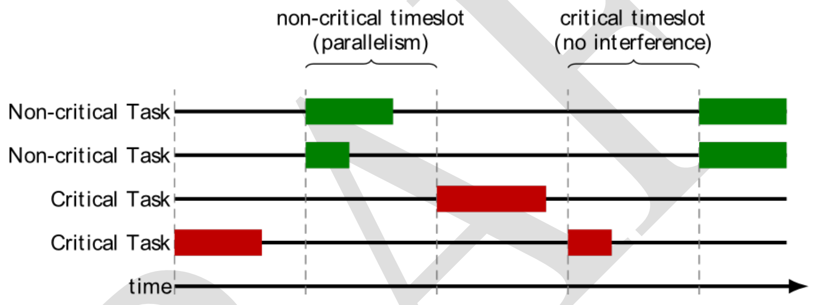
non-critical timeslot (parallelism)
critical timeslot (no interference)
Non-critical Task
Non-critical Task
Critical Task
Critical Task
time

Figure 12. Example of Interference-free Schedule

This method has been published in references [45] [46]. Fundamentally, it does not solve the question of interferences on multicore processors; rather, it proposes a workaround that fits with systems embedding little critical code.

4.3.2. Deterministic Execution Models

Deterministic execution models propose execution patterns that isolate phases where software initiates communications to shared resources, from phases where software executes from internal memories (e.g., private caches). Embedded software must be implemented according to these patterns.

When a piece of software performs an execution phase, it makes no use of shared resources and therefore cannot interfere with concurrent software. As illustrated in figure 13, communication and execution phases are scheduled, and communication phases are isolated according to time division multiple access (TDMA) principles.

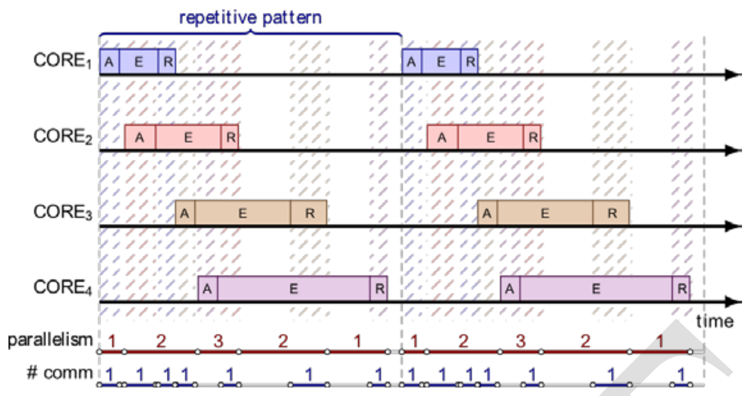

Figure 13. Example of Deterministic Execution Model Schedule [47]

The following execution models have been published in the literature:

- Predictable Execution Model or PREM is proposed by Pellizzoni et al. [48]. The authors introduce the notion of predictable intervals, which correspond to internal execution phases (i.e. without communication). This model scales to multicore processors, and takes into account interferences relative to the master I/O. The authors use an external device that interfaces with the I/O, and can be programmed to cope with a TDMA policy and not interfere with the CPU;

- Acquisition-Execution-Restitution or AER pattern has been used to decouple memory and I/O communications [47] [49]. The acquisition phase for a task consists in fetching data and instructions required for the execution phase, and also fetching data from I/O. During the restitution phase, the main concern is to update the I/O. An approach has been developed to position and optimally schedule execution phases and communication phases [3]. This approach consists in translating the set of tasks' parameters in a constrained linear programming (CLP), and letting a third-party solver find a schedule while maximizing the global response time.

- Flush-Fetch-Execute or FFE is an execution model that has been developed to ease the deployment of code generated from synchronous execution models [50].

Timing analysis and ability to schedule aspects of execution models have been developed by Schranzhofer et al. [51] [49]. They show that a deterministic execution model so far does not impact performances, and brings predictability to a multicore processor. Its main drawback is the difficulty to support legacy software.

4.3.3. Application Unaware Control Software

One drawback of deterministic execution models is the lack of support of legacy software, which introduces an additional effort for redesign. An approach has been proposed to make that control transparent with regard to embedded software [52] [53]. This enables the reuse of legacy software, including RTOS, with minor porting effort. As represented in figure 14, the mechanism relies on a smart configuration of the CPU internal resources (e.g., the MMU and caches) to make fetching operations transparent from the main memory.

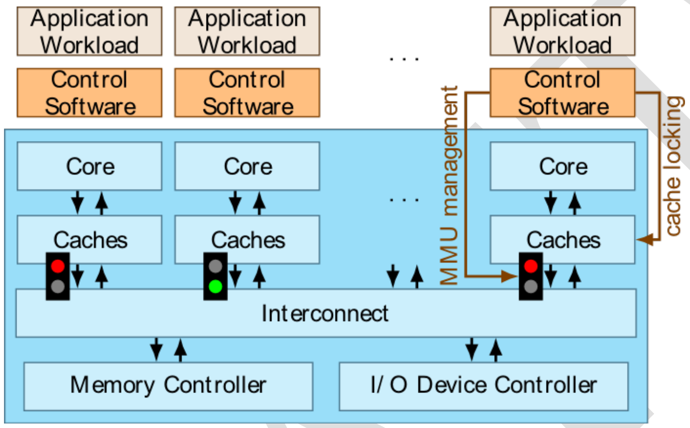

Figure 14. Overview of Marthy Control Software

This approach has been successfully deployed on a Freescale P5040. The prototype, called Marthy, has been included within the kernel of Topaz, which is a lightweight hypervisor developed by Freescale. However, Marthy’s impact on software performance is variable. As described in reference [53], the slow-down is limited for some applications, e.g., a 30% overhead for a quad-core processor. For other applications, it may be a factor 10 or even more. Even if optimizations are possible, the classes of applications for which this solution can be relevant are not properly known.

5. CONCLUSION

Interferences in multicore processors are undoubtedly an undesirable behavior for usage in avionics equipment, regardless of the number of cores. They make the processor's performance assessment complex to achieve and therefore raise safety issues. The main risk is to trigger timing failures on the whole equipment, even if data failure modes might also be discovered during the interference analysis.

In this White Paper, the authors propose a safety process to address the question of interference. The process is inspired by partitioning analyses performed on IMA systems and described in DO-297 [8]. Because of the wide variety of equipment, processors being used, level of criticality, and needs in terms of real-time requirements, no off-the-shelf generic solution seems to be practicable. Therefore, such a process ought to be instantiated on a case-by-case basis, and presented in the certification plan as soon as possible. Its definition should contain acceptability and termination criteria that will be applied during the safety assessment phases.

Several points need be noticed. First, no limitation on the number of core seems relevant, even if a processor that embeds more cores makes the interference analysis more complex. Second, exhaustiveness of analyses is not likely to be reachable on COTS processors containing either highly complex and/or black-box components. Hence, confidence should be obtained by combining several analysis methods, e.g., semi-formalized analyses coupled with information shared with the manufacturer and expert feedbacks on the processor, or similar ones in avionics. Safety experts must drive these methods. Similarly, the interference analysis should be performed with final applications, if possible. When it is not the case, representative applications fitting with the usage domain can be used, even if trustworthiness may be more complex to achieve.

Finally, interference mitigations, especially interference elimination techniques, do not dispense with an interference analysis, which brings evidence of interference control.

6. REFERENCES

[1] *CS-25, Certification Specifications for Large Aeroplanes, Amendment 6,* 2009.

[2] E. Bost, “Hardware Support for Robust Partitioning in Freescale QorIQ Multicore SoCs (P4080 and derivatives),” 2013.

[3] S. Girbal, D. G. Pérez, J. L. Rhun, M. Faugère, C. Pagetti and G. Durrieu, “A complete toolchain for an interference-free deployment of avionic applications on multi-core systems,” in *Proceedings of the 34th Digital Avionics Systems Conference*, 2015.

[4] P. Bieth and V. Brindejonc, “COTS-AEH - Use of complex COTS (Components-Off-The-Shelf) in Airborne Electronic Hardware - Failure Mode and Mitigation,” 2013.

[5] J. Nowotsch and M. Paulitsch, “Leveraging Multicore Computing Architectures in Avionics,” *European Dependable Computing Conference,* vol. 0, pp. 132-143, 2012.

[6] Certification Authority Software Team (CAST), “CAST-32: “Multi-core Processors” Rev 0,” FAA-CAST position paper, Atlantic City, 2015.

[7] EASA, "EASA CM - SWCEH – 001 Iss. 1 Rev. 1: Development Assurance of Airborne Electronic Hardware," 9th Mar. 2012.

[8] RTCA, DO-297, “Integrated Modular Avionics (IMA) development, guidance and certification consideration,” 2005.

[9] EUROCAE & SAE, ED-79A/ARP-4754A, “Guidelines for Development of Civil Aircraft and Systems,” 2010.

[10] V. Brindejonc and A. Roger, “Avoidance of Dysfunctional Behavior of Complex COTS Used in an Aeronautical Context,” in *Proceedings of Lambda-Mu RAMS Conference*, Dijon (France), 2014.

[11] J. Nowotsch, M. Paulitsch, D. Buhler, H. Theiling, S. Wegener and M. Schmidt, “Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement,” in *26th Euromicro Conference on Real-Time Systems (ECRTS),* 2014.

[12] J. Bin, “Controlling execution time variability using COTS for Safety-critical systems,” 2014.

[13] J. Bin, S. Girbal, G. P. Daniel, A. Grasset and A. Merigot, “Studying co-running avionic real-time applications on multi-core COTS architectures,” in *7th European Congress On Embedded Real Time Software And Systems (ERTS)*, 2014.

[14] R. Pellizzoni, A. Schranzhofer, J.-J. Chen, M. Caccamo and L. Thiele, “Worst case delay analysis for memory interference in multicore systems,” in *Design, Automation Test in Europe Conference Exhibition,* 2010.

[15] L. Mingsong, G. Nan, Y. Wang, R. Jan and W. Reinhard, “A survey on cache analysis for real-time systems,” 2015.

[16] B. K. Huynh, L. Ju and A. Roychoudhury, “Scope-Aware Data Cache Analysis for WCET Estimation,” in *17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2011.

[17] D. Grund and J. Reineke, “Toward Precise {PLRU} Cache Analysis,” in *10th International Workshop on Worst-Case Execution Time Analysis (WCET),* Brussels, Belgium, 2010.

[18] J. Yan and W. Zhang, “WCET Analysis for Multicore Processors with Shared L2 Instruction Caches,” in *Proceedings of the 2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, Washington, DC, USA, 2008.

[19] D. Hardy, T. Piquet and I. Puaut, “Using Bypass to Tighten WCET Estimates for Multicore Processors with Shared Instruction Caches,” in *30th IEEE Real-Time Systems Symposium (RTSS),* 2009.

[20] W. Zhang and Y. Jun, “Static timing analysis of shared caches for multicore processors,” *Journal of Computing Science and Engineering,* vol. 6, no. 4, pp. 267-278, 2012.

[21] R. Aitken and S. Idgunji, “Worst-Case Design and Margin for Embedded SRAM,” in *Design, Automation Test in Europe Conference Exhibition,* 2007.

[22] U. Drepper, “What Every Programmer Should Know About Memory,” 2007.

[23] M. Paolieri, E. Quiones and F. J. Cazorla, “Timing Effects of DDR Memory Systems in Hard Real-time Multicore Architectures: Issues and Solutions,” *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 1s, pp. 64:1--64:26, 2013.

[24] R. Bourgade, C. Rochange and P. Sainrat, “Predictable bus arbitration schemes for heterogeneous time-critical workloads running on multicore processors,” in *IEEE 16th Conference on Emerging Technologies Factory Automation (ETFA),* 2011.

[25] ARM, “CoreLink CCI-400 Cache Coherent Interconnect Technical Reference Manual, Rev G,” 2012.

[26] Texas-Instruments, “TMS320C6678 - Multicore Fixed and Floating-Point Digital Signal Processor,” 2012.

[27] H. Shah, A. Raabe and A. Knoll, “Challenges of WCET analysis in COTS multicore due to different levels of abstraction,” in *Workshop on High-performance and Real-time Embedded Systems (HiRES),* 2013.

[28] A. Vasudevan, “Techniques for Achieving Time-Predictability in Multicores - A Survey,” 2015.

[29] S. Girbal, X. Jean, J. le Rhun, D. G. Pérez and M. Gatti, “Deterministic Platform Software for Hard Real-Time systems using multicore COTS,” in *Proceedings of the 34th Digital Avionics Systems Conference*, 2015.

[30] S. Chattopadhyay, A. Roychoudhury, J. Rosen, P. Eles and Z. Peng, “Time-Predictable Embedded Software on Multicore Platforms: Analysis and Optimization,” *Found. Trends Electron. Des. Autom.*, vol. 8, no. 3-4, pp. 199-356, 2014.

[31] R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo and R. Pellizzoni, “Real-time cache management framework for multicore architectures,” in *IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2013.

[32] L. Sha, M. Caccamo, R. Mancuso, J.-E. Kim, M.-K. Yoon, R. Pellizzoni, H. Yun, R. Kegley, D. Perlman, G. Arundale and others, “Single Core Equivalent Virtual Machines for Hard Real-Time Computing on Multicore Processors,” 2014.

[33] B. C. Ward, J. L. Herman, C. J. Kenna and J. H. Anderson, “Making Shared Caches More Predictable on Multicore Platforms,” in *25th Euromicro Conference on Real-Time Systems (ECRTS)*, 2013.

[34] H. Yun, R. Mancuso, Z.-P. Wu and R. Pellizzoni, “PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms,” in *IEEE 20th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014.

[35] T. Moscibroda and O. Mutlu, “Memory performance attacks: denial of memory service in multicore systems,” in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, Berkeley, CA, USA, 2007.

[36] A. Kleen, “Linux multicore scalability,” in *Proceedings of Linux Kongress*, 2009.

[37] Y. Cui, W. Zhang, Y. Chen and Y. Shi, “A Scheduling Method for Avoiding Kernel Lock Thrashing on Multicores,” in *IEEE 16th International Conference on Parallel and Distributed Systems (ICPADS)*, 2010.

[38] W. Zhang, “Method for managing a processor, lock contention management apparatus, and computer system,” February 2014.

[39] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo and L. Sha, “Memguard: Memory bandwidth reservation system for efficient performance isolation in multicore platforms,” in *IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2013.

[40] A. Burns and R. Davis, “Mixed criticality systems-a review,” *Department of Computer Science, University of York, Tech. Rep*, 2013.

[41] A. Kritikakou, C. Pagetti, C. Rochange, M. Roy, M. Faugere, S. Girbal and D. Gracia Perez, “Distributed run-time WCET controller for concurrent critical tasks in mixed-critical systems,” in *Proceedings of the 22th International Conference on Real-Time and Network Systems (RTNS'14)*, 2014.

[42] B. Lickly, I. Liu, S. Kim, H. D. Patel, S. A. Edwards and E. A. Lee, “Predictable programming on a precision timed architecture,” in *Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems*, New York, NY, USA, 2008.

[43] T. Ungerer, F. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quinones, M. Gerdes, M. Paolieri, J. Wolf, H. Casse, S. Uhrig, I. Guliashvili, M. Houston, F. Kluge, S. Metzlaff and J. Mische, “Merasa: Multicore Execution of Hard Real-Time Applications Supporting Analyzability,” *IEEE Micro*, vol. 30, pp. 66-75, 2010.

[44] A. Hansson, K. Goossens, M. Bekooij and J. Huisken, “CoMPSoC: A Template for Composable and Predictable Multi-processor System on Chips,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 1, pp. 2:1--2:24, 2009.

[45] S. Fisher, “Certifying Applications in a Multicore Environment: The World's First Multicore Certification to SIL 4,” 2013.

[46] R. Vestal, “Safe Partition Scheduling on Multicore Processors,” Patent US8316368 B2, August 2010.

[47] G. Durrieu, M. Faugère, S. Girbal, D. Gracia Pérez, C. Pagetti and W. Puffitsch, “Predictable Flight Management System Implementation on a Multicore Processor,” in *Embedded Real Time Software and Systems*, 2014.

[48] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo and R. Kegley, “A Predictable Execution Model for COTS-Based Embedded Systems,” in *17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2011.

[49] A. Schranzhofer, R. Pellizzoni, J.-J. Chen, L. Thiele and M. Caccamo, “Worst-case response time analysis of resource access models in multicore systems,” in *Proceedings of the 47th Design Automation Conference*, New York, NY, USA, 2010.

[50] “Method and device for loading and executing instructions with deterministic cycles in a multicore avionics system having a bus, the access time of which is unpredictable,” US Patent 08694747, April 2012.

[51] A. Schranzhofer, J.-J. Chen and L. Thiele, “Timing Analysis for TDMA Arbitration in Resource Sharing Systems,” in *16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2010.

[52] X. Jean, D. Faura, M. Gatti, L. Pautet and T. Robert, “A Software Approach for Managing Shared Resources in Multicore Processors for IMA Systems,” in *IEEE/AIAA 32nd Digital Avionics Systems Conference (DASC)*, 2013.

[53] X. Jean, “Hypervisor control of COTS multicores processors in order to enforce determinism for future avionics equipment,” 2015.

APPENDIX A. ACTIVITIES AND DATA RECOMMENDED TO BE PRODUCED FOR COTS ASSURANCE

The following is based on EASA CM SWCEH-001 [7] to tentatively recollect all activities and results expected from highly complex COTS. Activities are gathered under the main objectives that are generally established as part of development assurance in support to certification.

- With respect to COTS AEH description of functional purpose and architecture content:

E.g.: Criticality (e.g., allocated DAL), novelty (e.g., new technology, service history), functionalities (various modes of operation), internal architecture (including internal interfaces, safety mechanisms and level of integration), structure (e.g., hierarchal breakdown and interconnections), I/O, and hardware-software interfaces (HSI);

- With respect to the intended function(s) (per 2X.1301) provided by COTS AEH:

E.g.: Usage domain (UD) definition (used/unused functions, activated/deactivated, configuration settings, including default configuration settings), UD validation (versus system, safety & software requirements), UD verification (reviews, analyses and tests), errata capture & analysis (impacts on intended function and safety);

- With respect to the technical suitability (per DO-254 11.2.1.6) of COTS AEH to application:

E.g.: characteristics (electrical and logical), timing performance (WCET), deterministic access to resources (execution, I/O, data, etc.), specific usages (e.g., robust partitioning, resources management, communication protocols), and environmental conditions (temperature, range, power consumption, and reliability);

- With respect to proper functioning (per 2X.1309) without anomalous behavior:

E.g.: failure modes and effects summary (FMES) for highly complex COTS AEH for critical applications. Integrate failure rates within upper-level FMEA of the unit of equipment. Perform FFPA at device level if reduction of DAL is sought. Provide directions for common mode & causes analyses at system level;

- With respect to operating performance" (per 2X.1309) under environmental conditions:

E.g.: validation and verification against upper-level requirements and HSI, additional re-verifications in case of changes in the COTS AEH, qualification to environmental conditions at the unit of equipment level, links with system/software architecture context for critical functions;

- With respect to ensuring continued airworthiness of usage conditions:

E.g.: document product service experience gained and lessons learned (errata workarounds, usage limitations, errata discovered), follow-up on stability, maturity, configuration status, and monitor change notices and associated change descriptions, document change impact analysis (CIA), manage COTS AEH obsolescence.