On the Tailoring of CAST-32A Certification Guidance to Real COTS Multicore Architectures

Irune Agirre*†, Jaume Abella‡, Mikel Azkarate-askasua* and Francisco J. Cazorla‡§
* IK4-Ikerlan Technology Research Centre, Mondragón, Spain
† Universitat Politècnica de Catalunya, Barcelona, Spain
‡ Barcelona Supercomputing Center, Barcelona, Spain
§ Spanish National Research Council (IIIA-CSIC)

***Abstract*—The use of Commercial Off-The-Shelf (COTS) multicores in real-time industry is on the rise due to multicores' potential performance increase and energy reduction. Yet, the unpredictable impact on timing of contention in shared hardware resources challenges certification. Furthermore, most safety certification standards target single-core architectures and do not provide explicit guidance for multicore processors. Recently, however, CAST-32A has been presented providing guidance for software planning, development and verification in multicores. In this paper, from a theoretical level, we provide a detailed review of CAST-32A objectives and the difficulty of reaching them under current COTS multicore design trends; at experimental level, we assess the difficulties of the application of CAST-32A to a real multicore processor, the NXP P4080.**

Keywords*—*Timing analysis, COTS multicore, certification.

I. Introduction

COTS multicore processors are the preferred industry choice to deal with the increasing integration demands in the embedded domain. Their outstanding benefits in performance and power efficiency are very attractive to consolidate multiple applications into a single silicon die, reducing the overall costs, size, weight and power overhead. In addition, the usage of readily available and tested COTS components considerably shorten the development time and associated costs.

Many embedded applications have safety and real-time requirements and must follow strict certification processes. Embedded applications often present functions of different levels of safety-criticality, widely referred to as mixed-criticality systems. The safety certification of such systems requires providing guarantees of the correct temporal behaviour of the applications and to demonstrate that the applications are isolated among them. In this line, although COTS multicore processors present great opportunities for mixed-criticality integration, their timing unpredictability and temporal inter-core (contention) interferences in the access to shared resources, seriously limit providing guarantees on timing, which remains as an open research issue. Single-core solutions do not scale well for multicores and safety standards do not provide satisfactory enough guidance yet for multicore architectures.

On the road to addressing this limitation, recently, certification authorities in the Airbone domain have published a position paper for multicore processors (CAST-32A) [1]. CAST-32A lists a set of objectives to help addressing multicore certification challenges. However, COTS multicores are, in-general, non amenable for timing analysis and applying those measures can still be very challenging. As a consequence, several limitations arise for providing sufficient evidence to confirm that timing requirements are met in the access to hardware shared resources hindering certification. In the scope of CAST-32A, this paper makes two contributions.

① We provide a deep analysis of CAST-32A objectives and discuss the feasibility of achieving them with current COTS multicore designs. We show potential limitations that the embedded system designers may face in achieving those goals. ② We tailor some of the generic principles in CAST-32A to a specific COTS multicore processor, the P4080 [2]. In particular, we show the main stumbling blocks we have found to reach CAST-32A goals in this architecture.

We show that, despite CAST-32A is a step forward for software planning, development and verification on multicore systems, the application of its objectives is not straightforward and requires in-depth analysis of the architecture and appropriate hardware support to deal with interference channels. Further, we highlight the importance that measurements may have for an embedded system designer (not only time measurements but also architectural events through mechanisms such as Performance Monitoring Counters (PMC)). Despite the participation of experts in real embedded systems development and in multicore hardware design in this work, our analysis shows that, for the P4080, the conformance of some features with CAST-32A can only be assessed by the chip vendor.

The rest of this paper is structured as follows. Section II provides some background. Section III presents our analysis of CAST-32A. Section IV introduces the P4080 and the challenges of achieving CAST-32A objectives on such processor, together with some quantitative results that support our claims. Some related work is provided in Section V. Finally, Section VI presents the main conclusions of this work.

II. Background and Problem statement

The main challenges that hinder the adoption of multicores in safety-critical domains stem from their inherent complexity, not proven temporal predictability, interferences coming from the access to shared resources, lack of previous experience and weak guidance on current safety standards. These challenges are absorbing considerable research efforts in both real-time industry and academic community [3], [4].

A. Exploratory Research

A vast amount of research works target to enable the use of multicores. In this paper we provide illustrative references of works in the main research lines, while a detailed summary of existing works can be found in [3], [4].

Several hardware designs are proposed to favour time predictability [5], [6]. However, COTS multicore architectures very slowly incorporate time-predictable features, requiring software solutions to handle contention.

Scheduling techniques usually build on simplifying assumptions about the contention suffered by the tasks in a

multicore. Others assume that the tasks are derived a Worst-Case Execution Time (WCET) estimate before the scheduling is carried out [7]. However, the latter only holds when all resources can be partitioned so that the load that co-runner tasks put on a resource does not increase a task's execution time (i.e. there are not interference channels in CAST-32A terms). Other works prevent read/write phases of tasks to simultaneously use shared resources (e.g. interconnect) to prevent contention [8]. Intuitively, as the core counts increase, this type of solutions will find scalability issues.

Approaches relying on static analyses to derive contention bounds entail many limitations in COTS architectures due to the complexity of modelling high-performance features and the lack of hardware details to do so. As an illustrative example, the authors in [9] propose a novel approach for estimating an upper bound of multicore interferences and enforce it at runtime for guaranteeing isolation. Even if they base their approach in static timing analysis techniques, the evaluation of their approach on a real processor (the P4080) exposes the need for measurements to quantify the access latencies in which their interference model is based on [10].

B. Safety Standards

Certification authorities are extremely wary about certifying any multicore based solution, which would require to provide trustworthy guarantees of the correctness of the system's both functional and temporal behaviour. This requires a robust system design and development process that involves high efforts even for single-core processors. As a consequence, safety-related industry has adopted the trend of employing simple, predictable and proven-in-use processors, features that are absent in modern multicore platforms.

In addition, for mixed-criticality integration, standards require to certify all components (including the non safety-related software) for the highest criticality level present in the system unless enough evidence of independence among the applications is demonstrated. Proving independence is thus crucial to considerably reduce development and certification costs and effort. Standards provide some guidance for achieving such independence based on the concept of partitioning [11]. A partition encapsulates the physical resources both spatially and temporally to avoid interferences. In single-core approaches, space partitioning is typically guaranteed by segregating the memory space and protecting it with mechanisms such as Memory Management Units (MMU). For time, partitioning is usually achieved by a static cyclic scheduler where each partition is assigned an exclusive time interval as suggested in different domain standards (IEC-61508-3 Annex F, ISO-26262-6 Annex-D, ARINC-653, DO-175 6.3.3f). However, usually it is not feasible to ensure complete temporal independence in multicores where partitions access to shared resources in parallel causing inter-partition contention delays.

In this respect, current safety standards are still in their infancy to provide guidance on addressing these multicore challenges. For instance, IEC-61508 [12] does not explicitly mention multicore architectures and the references the standard makes to modern integrated circuits (e.g. IEC-61508-2 Annex E) is oversimplified. Modern multicore processors fall into a "highly complex" category according to the European Aviation Safety Agency (EASA) [3],[13]. In 2014, the Federal Aviation Administration (FAA) and the EASA presented the conservative view of the aviation community with respect to the certification of multicore processors in a position paper (CAST-32) [14]. CAST-32 focuses on the certification of multicore processors with only two active cores. Recently, FAA/EASA published an update of this paper, called CAST-32A [1], extending its applicability by removing the "only two active cores" restriction. CAST-32A provides certification guidelines based on approaches accepted by certification authorities in projects using multicore technology in airbone systems.

III. On the Analysis of CAST-32A

Before tailoring CAST-32A objectives to a real processor, we first provide an introduction to its requirements with particular emphasis on multicore timing considerations.

A. General Definitions

CAST-32A analyses multicore processor (MCP) elements potentially impacting the safety, performance and integrity of a software airborne system. It is partially motivated by the fact that current standards in civil avionics cover single-core systems, since those standards are previous to MCPs usage in civil avionics. In providing additional guidance to cover MCP-specific aspects, CAST-32A builds on several definitions that classify MCPs based on the type of support they provide to reach its goals. Emphasis is put on hardware/software elements that create 'coupling' among software running on the MCP, and hence may lead to interference among them. Below we introduce those definitions related to timing.

Robust Resource Partitioning (RRP) requires (i) software partitions not to manipulate the code, I/O or data of other software partitions; (ii) software partitions not to consume more than their assigned portion of shared resources; and (iii) hardware failures unique to a software partition not to affect other software partitions.

Robust Time Partitioning (RTP) requires identifying and mitigating inter-partition interferences such that no software partition exceeds its deadline even in the presence of software executing simultaneously in other cores.

Robust Partitioning encompasses both, Robust Resource Partitioning and Robust Time Partitioning. CAST-32A classifies multicore processors into two categories depending on whether or not they provide Robust Partitioning.

An *Interference Channel* refers to a platform property that may cause interference between independent applications (i.e. with no explicit data or control flow between them).

Configuration Settings of the MCP cover any software-configurable element that affects software timing behaviour, e.g., frequency and cache partitioning. Special care is required to prevent the inadvertent changes of those settings that affect planned application's timing behaviour.

Table I lists some CAST-32A objectives, their identifiers refer to whether the objective relates to planning (PL), resource usage (RU) or software verification (SWV) activities. We only focus on those CAST-32A requirements that relate to interference analysis and WCET determination[1]. These objectives can be grouped into three high level principles:

Determining the Final Configuration. The designer shall determine which the intended final configuration is that will enable to satisfy system requirements (*RU_1*). This configuration is protected against unintended modification at runtime

[1]CAST-32A has three additional objectives that are not considered in this paper: one for inter-partition communication, another for runtime error management and the last one for reporting compliance with CAST-32A. Some definitions mostly related to these objectives (e.g. safety net for error management) are not discussed in this paper.

TABLE I: Summary of CAST-32A Objectives [1]

	ID	Description
Software Planning	PL_1	**Include MCP specific planning details in the SW plan doc.** *Specific processor, number of active cores, software architecture, dynamic software features, whether it hosts an IMA-like system (with applications from different systems) or not, Robust Partitioning supported or not, methods and tools for development and verification.*
Planning and Setting Resources	RU_1	**Determine configuration settings that enable to satisfy the functional, performance and timing requirements.**
	RU_2	**Critical configuration settings shall be static and protected against unintended modifications.**
	PL_2	**Include a high level description of shared resource usage and active dynamic hardware features in the hardware and software planning documents.** *Intended shared resource allocation and verification to prevent resource capabilities from being exceeded.*
Interference Channels and Resource Usage	RU_3	**Identify interference channels and verify the chosen means of mitigation.** *Interferences caused by shared memory, shared cache, interconnect, shared I/O or any other shared resource.*
	RU_4	**Identify available resources in the intended final configuration, allocate them to the applications and verify that the demands do not exceed the available resources (under worst-case scenarios).**
Software Verification	SWV_1	**Verify that all software components function correctly and have sufficient time when all the software is executing in the intended final configuration.** *Depends on the platform classification: 1. Platforms with Robust Partitioning: SW verification and WCET analysis can be done separately for each SW app. 2. All Other Platforms: If interference is mitigated for any software component or set of requirements, the verification of such components can be done separately. Otherwise, verification and WCET analysis shall be done with all software components executing together.*

(*RU_2*). The decisions taken at this phase should be documented as required in objective *PL_1*.

Managing Interference Channels. It is required to identify interference channels in the intended final configuration and to define the means to either avoid interference by design or upper-bound it so that timing deadlines are not exceeded (*RU_3*). Upper-bounding interference involves analysing the use of shared resources and designing the means to control multicore contention. This should be documented in the appropriate certification deliverables (*PL_2*).

Verifying the use of Shared Resources. Resource usage shall be verified by guaranteeing that in the chosen final configuration the software does not exceed the use of available resources even in worst-case scenarios (*RU_4*).

In this paper we focus on these three principles and, following CAST-32A, assume that software configuration is static (e.g. dynamic scheduling and dynamic partition to core allocation are not allowed) and hardware configuration is protected by the OS/hypervisor at runtime (*RU_2*).

B. Interpretation and Achievability of the Objectives

The safety argumentation of CAST-32A is based along the partitioning line of reasoning: guaranteeing Robust Partitioning is crucial at the time of meeting the objectives of CAST-32A, specially for allowing incremental verification of different software components integrated in the MCP system as stated in objective *SWV_1*.

B.1. Robust Resource Partitioning (RRP)

We interpret RRP as i) spatial partitioning, ii) resource quota monitoring and enforcement, and iii) fault containment at (software) partition level, which may also (indirectly) refer to resource partitions as discussed next.

Spatial partitioning can be achieved with MMU/MPU support in most current COTS multicore architectures.

Quota monitoring and enforcement, however, involves many more difficulties. For each shared resource it should be identified which events provide more accurate information about resource usage.

In terms of quota monitoring, intuitively, while access counts can provide a good approximation, different type of accesses use the shared resource for different time. This does not just require deriving this information from available manuals, but also checking whether the target platform provides support for monitoring architectural events (e.g. by Performance Monitoring Counters, PMCs) that allow tracking the desired type of accesses. For instance, in the LEON4 architecture, the last-level (L2) cache is write-back. L2 cache misses not evicting dirty data take shorter than those evicting dirty data. However, the L2 miss count PMC only captures non-dirty misses [15]. While authors [15] manage to upper-bound the number of dirty misses with existing counters, for other scenarios PMCs cannot provide enough support for access tracking, hindering a tight quota monitoring. Furthermore, some shared resources, like AMBA – one of the most used interfaces for communication – buses, allow a single request to hold the bus for an unbounded duration [16]. This poses new challenges, since additional means are required to determine whether hardware masters (i.e. resources allowed to start transactions on the bus) use this unbounded-duration feature for requests. This, however, requires deep hardware understanding, which might not be possessed by end users. Moreover, PMCs counting access types might not suffice. Instead, PMCs counting the time each core uses the shared resource (hence potentially delaying the other cores) may also be required.

The difficulty of implementing quota enforcement depends on the target resource and the hardware support provided.

First, for *space resources*, like the data array in a cache, some processors provide partitioning so that one task cannot evict the data of another. However, despite cache space partitioning, some tasks can still impact others. This may happen if partitioning is not implemented at the cache bank level. That is, tasks share the same bank though the cache ways in that bank are split among tasks. This may make cache accesses of one task delay the accesses of another task [5]. Further, caches may have queues to hold cache miss requests, which, if shared, can create huge contention among tasks despite cache ways are split [17]. This extremely low-level type of information about hardware requires the involvement of a hardware expert with proper access to manuals and also making experiments to empirically determine whether this situation can happen (analysed in the experimental section).

And second, *bandwidth resources*, like buses, can be split in time and space. For instance, a task can be given 50% of the bus bandwidth over a period of 10,000 cycles. This can be implemented with alternative full-access/no-access periods of 1,000 cycles, 2,500 cycles, 5,000 cycles, etc. Each of these ways to implement partitioning affects tasks' execution time. Further, given task τ_a and τ_b, where task τ_b makes a given number of accesses, how those overlap with τ_a's

requests has an impact on τ_a's execution time. Moreover, request overlap can change drastically from run to run with different or even the same inputs. In practice, determining how tasks' requests overlap during operation is unaffordable (in the final configuration), so existing approaches build on the assumption that tasks overlap their requests in the worst possible manner [18], [9]. Worst overlapping occurs when each request of the task under analysis has the lowest arbitration priority and it becomes ready when all other tasks in other cores have pending requests.

Fault containment at the resource partition. For physical faults, any MCP shall be considered to form a single fault containment region, that is, it is not possible to guarantee that the immediate effects of any possible fault are limited to a single software partition in a MCP unless it is specifically addressed in the design [19]. This occurs because, as a consequence of sharing the same silicon die, a number of shared elements become a single point of failure for all partitions simultaneously, such as the power supply or clock source. Thus, it cannot be claimed that different software partitions will fail *always* independently. CAST-32A addresses this issue by providing the concept of safety net. The *external* safety net aims at ensuring continuous safe operation of the system by detecting MCP failures and handling them preventing propagation to other components (e.g. physical fault containment).

Furthermore, for achieving RRP, CAST-32A requires that failures of hardware unique to a software partition do not propagate to other software partitions. While partitioning techniques (e.g. hypervisor) have been commonly used to avoid the propagation of design faults (e.g. deadline misses) among software partitions (i.e. *design fault containment*), they do not usually account for hardware faults. Obviously, a failure in a time-shared (e.g. bandwidth) resource extends to all software partitions using it. For space resources, hardware partitioning techniques provide some degree of isolation but (physical) fault containment at the resource-partition level is challenging, which is better illustrated taking a way-partitioned shared cache as example. While it is possible to prevent a fault in a cache bitcell to propagate to partitions not using that bitcell, some hardware resources, such as sense amplifiers and data output buffers (used to read/write the data in cache), may be independent or silently shared, thus making a permanent or intermittent fault affect all software partitions in the latter case. Therefore, it is hard to draw the line between hardware unique to a software partition and the features shared on the underlying MPC platform. As a consequence, in general, end users *do not have the means* to sustain or validate this claim. Hence, the only way it can be deployed is that it comes provided by the hardware vendor.

B.2. Robust Time Partitioning (RTP)

From the previous discussion, it follows that MCP will usually have some form of interference channels due to the difficulties of implementing RRP. Note that we consider that interference channels arise when RRP cannot be achieved and hence, tasks have interactions in the time domain.

In this scenario, the RTP goal of "*no software partition exceeds its deadline even in the presence of software executing simultaneously in other cores*", translates into deriving contention bounds for a task that hold for any load that the contender tasks can put on the shared resources. That is, the increment in time a task is assumed to experience (in the WCET estimate) due to contention through that interference channel, upper-bounds the actual interference it can suffer in the presence of any task. This type of bounds, which are referred to as fully-time composable [18], can be very pessimistic. For instance, let assume that τ_a generates n_a accesses to a round-robin bus of a given type t_1 that uses the bus a single cycle. Further, assume that there are another type of accesses t_2 that holds the bus for 10 cycles. For τ_a access, the worst situation is that a second contender τ_b generates accesses of type t_2 and that arrive at the same time of τ_a requests but are served first. Under a fully-time composable scenario τ_a is assumed to suffer an execution time increase of $10 \times n_a \times T$ cycles. That is, each request is assumed to suffer a 10x increase in its access time due to contention.

To reduce this overhead, a *partially-time composable* approach [18] can be followed. This approach accounts for the worst overlap of the *actual* requests of the contenders. This requires to determine i) the number and type of requests to the shared resources of all tasks [9]; and ii) the time each request type holds the shared resource [18]. The WCET estimate for a task is derived under a given *template* that describes the number and type of accesses performed by its contenders. That WCET estimate holds under any workload for which the actual number and type of requests performed by the contenders is smaller or equal to that assumed in the template. Coming back to the previous example, τ_a can be derived a WCET bound for a template that assumes K_1 access of type t_1 and K_2 of type t_2. It is valid for any contender τ_b that fulfils $(n_b^{t1} \times 1) + (n_b^{t2} \times 10) \leq (K_1 \times 1) + (K_2 \times 10)$.

B.3. Individual Objectives

CAST-32A introduces new objectives in the different phases of the lifecycle to analyse and mitigate interferences, and control shared resource usage (i.e. limit MCP contention) in a given platform configuration. To that end, CAST-32A resource usage objectives (*RU_1*, *RU_2*, *RU_3* and *RU_4*) impose new requirements in the design.

RU_1 is hard to achieve a priori in processors without RRP. This occurs because the impact of interference channels on software timing can be significant. Further, simply assuming the worst-contention (i.e. fully-time composable bounds) is in general impractical. In this situation, the configuration achieving software's timing requirements cannot be determined, until the final configuration is consolidated. With the partially-time composable approach [18], WCET estimates are derived under different templates and hence the work at integration reduces to check the template that upper-bounds contenders' request count (see example above).

RU_2 in general can be achieved with proper hypervisor and/or Operating System support.

RU_3. Some interference channels can be determined from the processor manuals. Others, however, are not properly documented and hence, require an expert performance analyst that can create a set of micro-benchmarks [20] to determine whether further interference channels can exist. Micro-benchmarks are a set of specialized programs that put high load on some shared resources and help (among others) determining the existence of some interference channels and derive bounds to the impact of interference.

RU_4. This generic objective encompasses the successful achieving of those above, with their associated challenges.

Further, in terms of the general approach, it must be understood that objectives cannot be achieved in a sequential

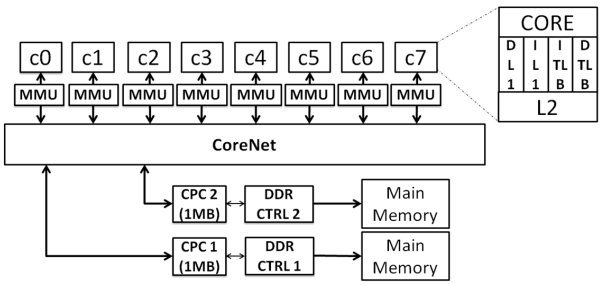


Fig. 1: Block diagram of the P4080 blocks we analyse

manner, as a first read could suggest. Instead, strong dependences exist among them, that require considering all (or some) of them simultaneously.

IV. CAST-32A ON THE P4080

In this section we evaluate CAST-32A principles on a real COTS multicore, both in qualitative and quantitative terms. We select the P4080 [2] multicore processor as a representative of the complexity in modern multicore architectures relevant for real-time industry [9],[21].

A. Platform Description: P4080

The P4080 [2], depicted in Figure 1, embeds eight Power-Architecture e500mc processor cores interconnected through the 'CoreNet Coherency Fabric'. Each core has private first level instruction (IL1) and data (DL1) caches (32 KB each) and a backside L2 cache (128 KB). The interconnect is able to perform several concurrent transactions in parallel and manages cache coherency. In addition to the core-local caches, the platform includes a shared L3 on-chip cache (1 MB) between the CoreNet and each memory controller, called CoreNet Platform Cache (CPC). The P4080 has two independent DDR3 memory controllers with interleaving support. As there are two memory controllers, there are also two CPC (a total of 2 MB) shared among all processors. Additionally, the platform includes multiple peripheral interfaces.

B. Identifying Configuration Options

Some hardware resources of the P4080 platform can be adjusted to applications' requirements by several configuration settings. Below we list the most relevant customizable features that may influence performance, or partitioning, identified with a qualitative analysis of processor specifications [2], [22].

B.1. Private Resources Settings

The eight cores are logically independent, each core has its own boot and reset control. The user can select which specific cores to activate while others remain dormant. Similarly, each core can be set to a different operation frequency. First level caches can be enabled or disabled. The second level backside cache can be either disabled or configured as data only, instruction only or unified (data and instruction) cache.

B.2. Shared Resources Settings

CoreNet Interconnect: The interconnect can be configured in up to 32 address windows (Local Access Windows, LAWs) to route transactions. Each LAW serves to define the internal connections by mapping different address regions to specific target devices (such as DDR controllers or peripheral interfaces). In addition, the configuration of each LAW includes a coherency subdomain identifier that allows to define which cores and caching elements should be maintained coherent.

CoreNet Platform Cache (CPC): It supports flexible configurations on a per-way granularity. Each CPC has 32 ways (32KB per way) that can be configured as L3 cache or as static SRAM. For ways configured as L3 cache, cache partitioning is supported to assign each way to a logical partition. The ways configured as SRAM are mapped to a configurable physical address range. With the appropriate LAW and Memory Management Unit (MMU) configuration, the SRAM address space can also be segregated into different logical partitions.

Main Memory: The two memory controllers have interleaving support. When enabled, interleaving can be configured to switch between memory controllers for every cache line transfer, every page line transfer or at every bank transfer. If disabled, the two memory controllers operate independently. Like for the SRAM, main memory can be divided into different MMU-restricted physical regions for each of the cores.

Peripheral devices: I/O transactions can be controlled with the Peripheral Access Management Unit (PAMU) that acts as a MMU for I/O devices.

Interrupts: They can be routed to any of the eight cores separately or to more than one core simultaneously.

C. Defining Hardware Configuration Settings

We evaluate the first CAST-32A principle, *RU_1*, of selecting a suitable platform configuration by exploring different possible hardware setups (HWS) in the P4080 platform. Based on the configuration options listed above, we define several different HWS, see Figure 2. To that end, we establish a baseline configuration, common across all HWS, for the core local resources and we rather focus on the different possible setups for shared resources. To evaluate isolation, we narrow down to only one (safety critical) application that needs to be strongly independent from the other seven and assume that each core hosts at most one software application. Taking this into account, in Figure 2, the resources used by the critical application (executed in $c0$) are grey shadowed.

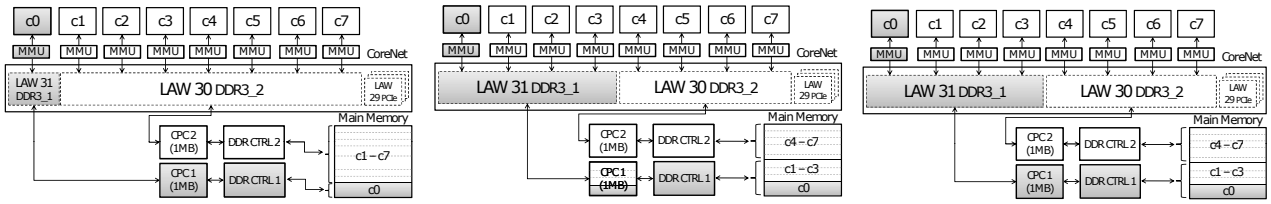
Baseline configuration: All cores are activated with highest supported frequency and local caches enabled (DL1, IL1 and unified L2) for increasing performance. We focus on the memory hierarchy as shared resource and therefore we assume that the same peripheral is not shared among multiple cores.

C.1. HWS 1: Maximum HW Isolation

HWS 1 (Figure 2a) seeks to achieve the highest possible level of resource privatization (maximum partitioning) with the following configuration: 1) Memory controller interleaving is disabled and $DDR3\_1$ is exclusive for $c0$. $DDR3\_2$ is shared among the remaining seven cores ($c1 - c7$); 2) as each CPC corresponds to a memory controller, $CPC1$ is also restricted to $c0$ and can be configured either as (private) SRAM, L3 cache or both. $CPC2$ may or may not be partitioned across the non-critical partitions. In particular, we use both CPCs in cache mode; and 3) main memory is segregated in at least two MMU-protected memory regions, one for $c0$ and the rest for cores $c1 - c7$. If required, the latter can be further partitioned to have a separate address range for each core.

C.2. HWS 2: Limited HW Isolation (1)

This setup, shown in Figure 2b, provides better balance between resource privatization and sharing (and hence, ef-



(a) HWS 1: Maximum HW Isolation. $c0$ uses a private CPC/mem controller.

(b) HWS 2: Limited HW Isolation. $CPC1$ shared and $c0$ receives some private ways.

(c) HWS 3: Limited HW Isolation. $CPC1$ shared among $c0$ and $c1 - c3$.

Fig. 2: Examples of P4080 Hardware setups. HWS4, without any control of contention, is not shown for space constraints.

TABLE II: Identified P4080 Interference Channels

Interference Channel	Effect	Mitigation Setup	Mitigation Measure
CoreNet	Contention (parallel requests)	HWS 1-4	None
CPC	L3 cache space: cache line evictions	HWS 1	Exclusive CPC for c0
		HWS 2	L3 Partitioning
		HWS 3-4	None
	SRAM space: data modification	HWS 1	Exclusive CPC for c0
		HWS 2-3	MMU
		HWS 4	n/a
	L3/SRAM: Contention (parallel requests)	HWS 1	Exclusive CPC for c0
		HWS 2-3	Limited to 4 cores
		HWS 4	None
Memory controller	Contention (parallel requests)	HWS 1	Exclusive CPC for c0
		HWS 2	Limited to 4 cores
		HWS 3-4	None
Main memory	Memory space: data modification	HWS 1-4	MMU
	Contention (parallel requests)	HWS 1-4	None

ficiency): 1) memory controller interleaving is disabled and applications are distributed across the two memory controllers (e.g. $DDR3\_1$ for $c0 - c3$ and $DDR3\_2$ for $c4 - c7$); 2) $CPC1$ is partitioned by assigning one or more 32 KB ways to $c0$. As in HWS 1, these ways may be setup as SRAM, L3 cache or both. The remaining ways of $CPC1$ and $CPC2$ may or may not be partitioned across the non-critical partitions. In particular, in CPC_1 we give 8 ways to $c0$ and 24 ways (shared) to $c1 - c3$; and 3) main memory is segregated as in HWS 1.

C.3. HWS 3: Limited HW Isolation (2)

This setup matches HWS 2, but $CPC1$ is not way partitioned but shared by $c0 - c3$, see Figure 2c.

C.4. HWS 4: Minimum HW Isolation

Most resources are shared across all cores for evaluating the impact of shared resources in the worst-case (omitted in Figure 2 due to space constraints): 1) memory controller interleaving is enabled so both memory controllers are interchangeably accessed by all cores; 2) $CPC1$ and $CPC2$ are configured as L3 cache and shared across all cores; and 3) main memory is segregated as in HWS 1.

D. Identifying Interference Channels (Qualitative Analysis)

CAST-32A requires identifying interferences in the intended final configuration. However, selecting a configuration involves determining a hardware partition setup, which in turn determines the interference channels. In Table II we list some of the potential interference channels we identified from our analysis of the P4080 and we describe how they are addressed by each HWS. Note that other interference channels may exist. In fact, the challenge for the end user is to combine information in the manuals and previous experience to derive potential interference channels.

CoreNet. The information about CoreNet is scarce, which prevents us from concluding whether it is an interference channel since it may contain buffers or queues to hold pending requests. This would make it a stateful resource and hence a source of interference.

CPC and memory. In the CPC, we can have contention for space, i.e., software partitions can evict each other's data. Further, depending on how they are implemented, the CPC can have access contention, in which – despite tasks use different cache parts – they impact on each other access time to the cache. The memory controller and main memory are other well-known sources of contention. Finally, the coherence method in place can also impact tasks execution time.

PMC support. The high complexity of the architecture and the presence of features that lead to unexpected behaviour, as exposed later, render it very difficult to characterize the architecture and find interference channels by relying only on time measurements. Therefore, the PMCs present in many modern architectures play a key role. However, the PMC infrastructure poses many limitations for end users. Often, there are many different configurable events but a limited number of counters (i.e. 4 counters per core in the P4080). Thus, several runs might be needed to extract the required information. Moreover, typically, not all interesting events are accessible through counters and each event is not often described in detail.

Fault Containment at the RRP. Some interference channels may exist only under the presence of faults. Thus, assessing whether they exist is simply impossible until faults occur, which may be too late to take any action. For instance, some logic may exist to drive memory accesses to either $CPC1$ or $CPC2$. Further, such logic, despite shared, may not have any impact in performance if the CoreNet serializes accesses. However, a fault in such logic might affect any subset of the cores in non-obvious ways.

E. Quantitative Analysis

Next we provide quantitative evidence on some interference channels and undocumented features. Note that the goal of this section is not to provide a characterization of the timing

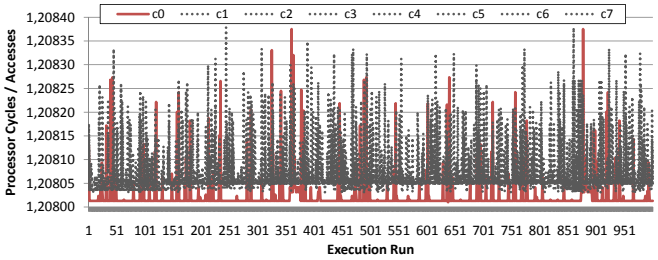
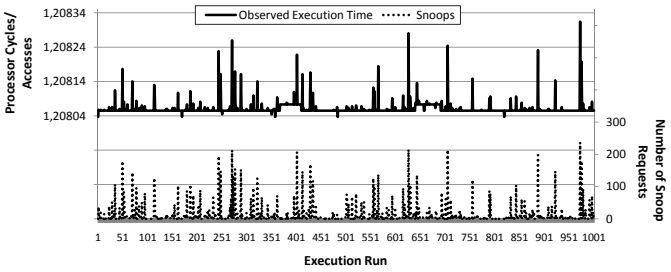
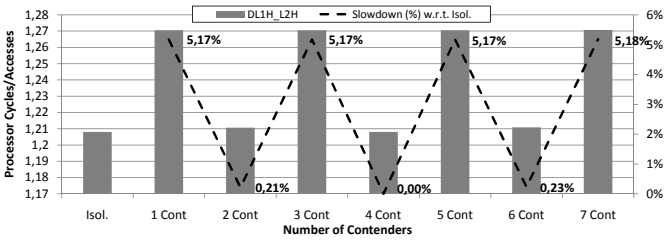


Fig. 3: DL1 Hit Latency variation across cores and runs.



(a) Impact of coherence on execution time in isolation.



(b) DL1 Hit Latency with parallel execution.

Fig. 4: Impact of coherence

behaviour of the P4080, which would be an extensive technical report. Instead, we provide the result of specific experiments providing key insights about achieving CAST-32A objectives. It is worth noting that authors in [9] focus on a setup with expected interferences: all caches disabled, branch prediction switched off and with all cores sharing the same memory controller. In this paper, instead, we show that even the most conservative partitioning setups can be subject to the impact of interference channels.

E.1. Heterogeneous behaviour

We start by reporting some undocumented features impacting time. In a first experiment, we execute a simple micro-benchmark in each of the cores (ci) of the P4080 in isolation (*benchDL1*)[2]. *BenchDL1* traverses a data array of 24KB with a sequence of load instructions. Such array fits in DL1. We execute *BenchDL1* 1000 times and use the PMCs available in each core of the P4080 to monitor processor cycles and memory related events (number of accesses, DL1 misses, L2 accesses, etc.) after traversing the array once, so that it is present in DL1. From this information we derive the average DL1 hit latency on each core ci as depicted in Figure 3 across runs. The result of this simple experiment reveals that there are small *systematic* deviations from core to core. Our analysis of the data sample of 1000 runs on each core reveals that the central quantiles (Q1-Q3) of $c0$ (thus discarding outliers) are below the minimum value across *all* other cores.

Even if the order of magnitude of this difference makes the variation negligible, its systematic nature reveals some heterogeneity in the physical design of the cores of the P4080 that is not reported in the (public) documentation. Thus, it is unknown whether the source of this difference may have noticeable (relative or absolute) impact in some programs.

E.2. Coherence

The coherence protocol is a potential interference channel. Apart from of its impact in execution time for concurrent memory accesses (evaluated by Nowotsch et al. in [10]), our evaluation results show execution time variations even when one core is running *benchDL1* in isolation (with the rest of the cores dormant), so only accessing local caches. We conclude that those variations are related to snoop overheads: as depicted in Figure 4a, the core under analysis receives unexpected snoop requests, that correlate with execution time variation. As for core-to-core differences, execution time impact is negligible, but uncontrolled. Thus, an interference channel exists and its potential magnitude is unknown.

In a second experiment, we run *benchDL1* in $c0$ against an increasing number of contenders in the other cores (also running *benchDL1*). Thus, all cores only exercise core local resources (i.e. DL1). Therefore, one would expect no impact of the other cores. However, as shown in Figure 4b, DL1 hit latency increases noticeably with 1, 3, 5 and 7 contenders, but negligibly with 2, 4 and 6 contenders. While we suspect that this behaviour is produced by the coherence protocol, the snoop counter does not reveal any specific trend, so all we can state is that an unknown interference channel exists with non-obvious impact across cores.

E.3. Contention in Shared Resources

Due to space constraints we analyse few configurations and interference channels. In particular, Figure 5 compares the execution time of a micro-benchmark performing sustained L3 (CPC) hits (*benchL3H*) by using 256KB of the 1MB $CPC1$ with 1, 2 and 3 contenders that perform all of them either sustained CPC hits (*benchL3H*) or CPC misses (*benchL3mem*). This comparison is performed in HWS 2 and HWS 3, hence, with and without CPC partitioning.

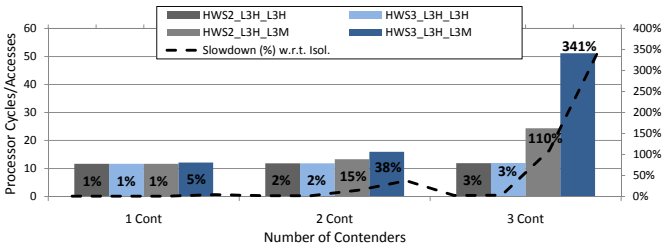


Fig. 5: L3 Contention under HWS 2 and HWS 3

As shown, whenever all cores perform CPC hits (HWS2_L3H_L3H and HWS3_L3H_L3H) cache partitioning makes no difference and CoreNet and CPC port contention have very low impact (1% slowdown per additional contender).

When contenders perform CPC misses, then *benchL3H* experiences a significant slowdown in HWS 2 with 2 and

[2]We execute the benchmarks on top of a real-time hypervisor.

3 contenders (15% and 110% respectively) despite its cache behaviour is not affected. Although it cannot be proven based only on measurements, we suspect that the CPC has some internal buffers that get filled with CPC miss requests, which delay new requests. In any case, our experiment reveals the existence of severe interference channels despite CPC partitioning. If CPC space is shared (HWS 3), the same trend is observed but at a larger scale since contenders start evicting data from the task under analysis, whose performance degrades due to both, internal CPC buffers and additional CPC misses.

V. Related Work

Literature on multicore certification is mainly concerned with ensuring timing guarantees without incurring undue resource over provisioning [3], [14], [23], [24]. The design of time deterministic processors and hardware-based separation is considered in [6], as a means to simplify the collection of evidence required for a certification process by providing predictable timing guarantees by design. However, these approaches require new hardware architectures that often render difficult their deployment. As a result, the literature on timing analysis of COTS multicores is abundant [3], [4], [9], [25]. The authors in [9] propose an interference delay analysis technique on top of regular single-core WCET analysis. The approach requires computing shared resource usage bounds, and monitoring and enforcing them at runtime. The authors rely on static analysis techniques for the computation of both, timing and resource usage, bounds. Nevertheless, their evaluation on the P4080 platform reveals the limitations of static timing analysis on COTS multicores due to the unavailability of a detailed architecture model. A common alternative that is absorbing considerable research efforts is measurement-based timing analysis [26], [27], [28]. In this line, synthetic benchmarks are often used to stress hardware features and characterize the timing of the architecture [10], [20] or to upper-bound the interference that applications may suffer, and ensure that the maximum interference is captured during measurements [18], [21], [29]. To the best of our knowledge, this paper is the first work evaluating current COTS multicore designs and practices with respect to recent certification guidelines for addressing multicore certification challenges such as CAST-32A [1].

VI. Conclusion

CAST-32A provides a step forward in terms of guidance for multicore software verification. CAST-32A guideline, as many safety-related standards, is abstract enough to have a broad application. However, a practical application of CAST-32A is challenging. In this paper, we analyse the difficulties to apply CAST-32A to real processor designs qualitatively. Then, through a particular case study, the P4080 processor, we practically and quantitatively assess those difficulties.

Our work shows that appropriate hardware configurations and smart experimentation can reduce the degree of uncertainty. However, for the studied processor, the uncertainty can be neither removed nor deemed as irrelevant due to the non-obvious interactions of tasks at hardware level. It turns out that even the most conservative partitioning measures can be subject to execution time variations in complex COTS architectures due to shared resources and hidden undocumented features. As a consequence, this requires an in-depth analysis of the architecture which, in many COTS platforms, due to the limited documentation, can only be provided by the chip vendor. We also show that some knowledge can be gained by executing experiments on the real platform, but quantitative evidence can only mitigate uncertainty to some extent, thus leaving several open questions to fully adhere to CAST-32A.

Acknowledgements

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness (MINECO) under grant TIN2015-65316-P and the HiPEAC Network of Excellence. Jaume Abella has been partially supported by the MINECO under Ramon y Cajal grant RYC-2013-14717.

References

[1] Certification Authorities Software Team, "Multi-core Processors - Position Paper," CAST-32A, Tech. Rep., November 2016.
[2] NXP Semiconductors, "P4 series P4080 multicore processor."
[3] M. Paulitsch et al., "Mixed-Criticality Embedded Systems - A Balance Ensuring Partitioning and Performance," in *DSD*, 2015.
[4] G. Fernández et al., "Contention in multicore hardware shared resources: Understanding of the state of the art," in *WCET*, 2014.
[5] M. Paolieri et al., "Hardware support for WCET analysis of hard real-time multicore systems," in *ISCA*, 2009.
[6] C. El Salloum et al., "The ACROSS MPSoC – a new generation of multi-core processors designed for safety-critical embedded systems," in *DSD*, 2012.
[7] C. Rochange, "Parallel real-time tasks, as viewed by WCET analysis and task scheduling approaches," in *WCET Workshop*, 2016.
[8] M. Becker et al., "Contention-free execution of automotive applications on a clustered many-core platform," in *ECRTS*, 2016.
[9] J. Nowotsch et al., "Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement," in *ECRTS*, 2014.
[10] J. Nowotsch and M. Paulitsch, "Leveraging multi-core computing architectures in avionics," in *EDCC*, 2012.
[11] R. John, "Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance," NASA Langley, Tech. Rep., 1999.
[12] "IEC-61508: Functional safety of electrical/electronic/programmable electronic safety-related systems," 2010.
[13] EASA, "Certification memorandum - development assurance of airborne electronic hardware," Tech. Rep., 09th of March 2012.
[14] Certification Authorities Software Team, "Multi-core Processors - Position Paper," CAST-32, Tech. Rep., May 2014.
[15] J. Jalle et al., "Bounding Resource Contention Interference in the Next-Generation Microprocessor (NGMP)," in *ERTS*^{2}, 2016.
[16] ——, "AHRB: A high-performance time-composable AMBA AHB bus," in *RTAS*, 2014.
[17] P. Valsan et al., "Taming non-blocking caches to improve isolation in multicore real-time systems," in *RTAS*, 2016.
[18] G. Fernandez et al., "Resource usage templates and signatures for COTS multicore processors," in *DAC*, 2015.
[19] H. Kopetz, *On the Fault Hypothesis for a Safety-Critical Real-Time System*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, vol. 4147, book section 3, pp. 31–42.
[20] M. Fernández et al., "Assessing the Suitability of the NGMP Multi-core Processor in the Space Domain," in *EMSOFT*, 2012.
[21] J. Bin et al., "Studying co-running avionic real-time applications on multi-core COTS architectures," in *ERTS*^{2}, 2014.
[22] E. Bost, "Hardware Support for Robust Partitioning in Freescale QorIQ Multicore SoCs (P4080 and derivatives), White Paper ," 2013.
[23] L. M. Kinnan, "Use of multicore processors in avionics and its potential impact on implementation and certification," *SAE Tech. Rep.*, 2009.
[24] P. Huyck, "ARINC 653 and multi-core microprocessors-considerations and potential impacts," in *DASC*, 2013.
[25] D. Dasari and V. Nelis, "An Analysis of the Impact of Bus Contention on the WCET in Multicores," in *HPCC-ICESS*, 2012.
[26] I. Wenzel et al., "Measurement-based timing analysis," in *ISOLA*, 2008.
[27] ——, "Measurement-based worst-case execution time analysis," in *SEUS Workshop*, 2005.
[28] R. Wilhelm et al., "The worst-case execution-time problem-overview of methods and survey of tools," *Transactions on Embedded Computing Systems*, vol. 7, no. 3, 2008.
[29] S. Girbal et al., "Using monitors to predict co-running safety-critical hard real-time benchmark behavior," in *ICITES*, 2014.