


Quasi Isolation QoS Setups to Control MPSoC Contention in Integrated Software Architectures

Sergio Garcia-Esteban ✉ 

Polytechnic University of Catalonia, Barcelona, Spain
Barcelona Supercomputing Center (BSC), Spain

Alejandro Serrano-Cases ✉ 

Barcelona Supercomputing Center (BSC), Spain
Rapita Systems S.L., Barcelona, Spain

Jaume Abella ✉ 

Barcelona Supercomputing Center (BSC), Spain

Enrico Mezzetti ✉ 

Barcelona Supercomputing Center (BSC), Spain
Rapita Systems S.L., Barcelona, Spain

Francisco J. Cazorla ✉ 

Barcelona Supercomputing Center (BSC), Spain
Rapita Systems S.L., Barcelona, Spain

Abstract

The use of integrated architectures, such as integrated modular avionics (IMA) in avionics, IMA-SP in space, and AUTOSAR in automotive, running on Multi-Processor System-on-Chip (MPSoC) is on the rise. Timing isolation among the different software partitions or applications thereof in an integrated architecture is key to simplifying software integration and its timing validation by ensuring the performance of each partition has no or very limited impact on others despite they share MPSoC's hardware resources. In this work, we contend that the increasing hardware support for Quality of Service (QoS) guarantees in modern MPSoCs can be leveraged via specific setups to provide strong, albeit not full, isolation among different software partitions. We introduce the concept of Quasi Isolation QoS (QIQoS) setups and instantiate it in the Xilinx Zynq UltraScale+. To that end, out of the millions of setups offered by the different QoS mechanisms, we identify specific QoS configurations that isolate the traffic of time-critical software partitions executing in the core cluster from that generated by contender partitions in the programmable logic. Our results show that the selected isolation setup results in performance variations of the partitions run in the computing cores that are below 6 percentage points, even under scenarios with extremely high traffic coming from the programmable logic.

2012 ACM Subject Classification Computer systems organization → Real-time system architecture

Keywords and phrases Multicore, Interference, QoS

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2023.5

Funding This work has been supported by Grant PID2019-107255GB-C21 funded by MCIN/AEI/10.13039/501100011033 and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 772773).

1 Introduction

MPSoCs are progressively used in safety-critical domains, like avionics and space, to cater for augmented performance requirements. Besides the sheer computational power they provide, MPSoCs increasingly incorporate substantial Quality of Service (QoS) features [60, 44]. QoS



© Sergio Garcia-Esteban, Alejandro Serrano-Cases, Jaume Abella, Enrico Mezzetti, and Francisco J. Cazorla;

licensed under Creative Commons License CC-BY 4.0

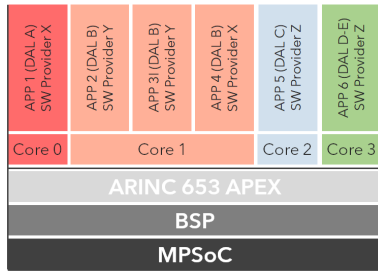
35th Euromicro Conference on Real-Time Systems (ECRTS 2023).

Editor: Alessandro V. Papadopoulos; Article No. 5; pp. 5:1–5:25

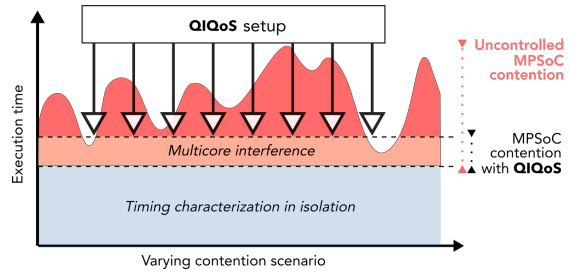


Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** IMA setup with multiple applications and Design Assurance Levels.



■ **Figure 2** Representation of interference mitigation impact under a QIQoS setup.

support was originally designed for performance optimization and load balancing, but is increasingly considered as a means to better control contention effects among co-running tasks and enforce a more deterministic timing behavior [48, 50]. QoS support is provided as software-controlled “knobs” that allow to intentionally bias the execution towards a given task by providing it with privileged access to hardware shared resources, more bandwidth in an interconnect, reserved space in stateful hardware resources, and the like.

When an MPSoC is used to support *exclusively a monolithic application* using several or all underlying cores (e.g. a single ARINC 653 partition), deployed for example by a single avionics original equipment manufacturer or TIER1 provider, the contention each process can suffer can be bounded already in early software development stages: contender information is available as they are part of the same monolithic application. In such scenario, different QoS setups may be explored to find a setup that satisfies the performance requirements of all processes [48].

However, this is seldom the case and instead an *IMA-MPSoC setup* is deployed: multiple applications from different software providers are typically integrated on the same target MPSoC, with a view to reducing integration and validation costs. In this line, integrated architectures¹ like IMA [12] build on time and space partitioning concepts, as defined by the ARINC 653 [4] open standard, to simplify the allocation of computing resources to different applications. This effectively allows integrating several applications onto less computing hardware, as illustrated in Figure 1. While IMA time partitioning ensures each application receives a given amount of CPU time, the actual progress made by an application in the time window also depends on the share of hardware resources the application receives.

MPSoC timing interference arises on contending accesses to shared hardware resources [19, 39, 55], which causes an application’s execution time to depend on the use of resources made by applications in other software partitions. Waiting until late development and integration stages, when all providers make their applications available, to address the timing dimension and assess timing interference has opposing effects. On the positive side, only the system’s intended final configuration [1, 18, 21] is considered, which reduces the risk of overestimating the contention impact. On the negative side, there is a non-negligible risk of detecting software timing violations too late in the validation and verification stages, when reacting to time misconfigurations can result in unaffordable changes to the applications, system design and schedule, and the entailed regression testing.

¹ For instance, Integrated Modular Avionics (IMA) for avionics, IMA for Spacecrafts (IMA-SP) [56] in space, and AUTomotive Open System ARchitecture (AUTOSAR) [13].

In IMA-MPSoC scenarios, QoS support can be leveraged to anticipate late timing issues related to multicore contention by preserving partition-level time budgets against the interference of (unknown or partially known) contender applications. This is, in fact, aligned with CAST-32A [18] advisory circular and A(M)C 20-193 [21] for multicore certification in the avionics domain, establishing that separate determination of the WCET of an application, without any other applications executing, is only valid if the applicant can demonstrate that they build on a multicore Platform with robust partitioning or that time interference from other applications is avoided or mitigated for that application.

Concept. We contend that existing hardware QoS solutions can be used to enforce setups under which timing interference is mitigated and the execution time of an application is less exposed to contention from other co-running applications. We aim at devising a set of QoS setups, which we call Quasi Isolation QoS setups or QIQoS, that guarantee a high-degree of isolation (performance guarantees) to the applications regardless of the contention its co-runners put on MPSoC's hardware shared resources, hence meeting IMA assumptions and requirements. QIQoS setups are meant to reduce the impact on execution time when varying the contender loads on the system shared resources. Figure 2 illustrates the effects of a given QIQoS setup in reducing the sensitivity to contention scenario by enforcing an upper bound to the incurred timing interference. As an immediate effect, the enforcement of a proper QIQoS setup will *increase the representativeness of early time budgets*, typically obtained by running the application against synthetic aggressors, making them much tighter and stable. The gap between the multicore interference empirically observed during the timing verification campaign by deploying synthetic worst-case contention scenarios and that observed in the final system configuration will be sensibly narrowed.

Realization. In this work, we instantiate our QIQoS approach on the Xilinx Zynq UltraScale+ [60, 59] to address contention arising on accesses to the DDR main memory. The DDR memory controller (DDRMC) provides DDR access to the different computing elements on the MPSoC through its six ports, and offers a complex multi-layer QoS mechanism to control the traffic coming through each port. This includes traffic classes, port throttling, and per-traffic class resource allocation. Hence, the QoS of the DDRMC allows millions of configurations that are remarkably challenging to master for the end users [48]. Our first step is then, identifying several specific DDRMC QIQoS configurations that allow, to different degrees, isolating DDR traffic of critical tasks from that generated by other tasks. Subsequently, we expose the specific characteristics of the identified QIQoS from the standpoint of the type and degree of isolation they can assure.

When facing the increasing QoS support in modern MPSoCs, the challenge lies in making effective and consistent use of the available features. One of the proposed QIQoS setups exploits the timeout features at the port level, while the second QIQoS instead mainly builds on starvation prevention features. Both QIQoS setups use traffic classes and transaction throttling. Other common features to both include preventing specific settings for the DDRMC that, if wrongly set, could defy the benefits achieved through the QoS layer. These include limiting the allowed DDR memory commands that can be sent out of order to the DDR device, and preventing critical and non-critical tasks from sharing the same memory port.

Our results show that the proposed DDRMC QIQoS configurations can effectively isolate the execution of the critical tasks that run in the A53 and the R5 cores from the load coming from the programmable logic (PL) in the access to memory, despite the latter produces in our

experiments huge amounts of DDR accesses. The degree of isolation can be configured based on a set of parameters provided for each QIQoS setup, which allows end users to achieve the desired balance between isolation and performance. Under the most aggressive QIQoS setups, the performance variation across substantially different loads that the PL put on the DDRMC is as low as 6 percentage points.

The rest of this paper is organized as follows. Section 2 presents the related works. Section 3 introduces our target platform. Section 4 presents the principles of QIQoS setups and describes their application on the target platform. Section 5 provides the experimental results. Finally, Section 6 presents the main conclusions of this work.

2 Related Works

Domain-specific safety standards and support documents specifically identify the need for controlling and reducing the impact of contention arising on shared resources accesses as a fundamental requirement for certification [1, 18, 21, 34]. Massive research efforts have been devoted to cope with the impact of multicore timing interference on system performance and predictability [46, 48]. Whereas custom hardware designs have been proposed to balance predictability and performance [42, 30, 54, 37], in this work we focus on a COTS MPSoC and analyze the predictability of its memory controller.

Several works pursue analytically bounding the worst-case interference suffered by each application [17, 19, 24, 39, 57]. Regardless of the tightness of the adopted method, contention can be too large (e.g. some works report more than 20x performance degradation [55]), which ultimately leads to severe system under-utilization. While we are still interested in analyzing the timing interference, our main focus is to enforce a-priori control of contention via appropriate QoS configurations.

Controlling the impact of contention by determining how resources are shared among applications, as opposed to just analyzing it, allows to reduce the impact of contention and is a fundamental enabler for the analyzability of multicore systems [48, 43, 28]. Time and space partitioning of shared resources has been explored building on top of existing hardware and software solutions, typically handled either as part of the static system configuration [4, 28], or as part of more or less complex run-time monitoring of resource usage at RTOS or Hypervisor level [43]. Software solutions, in particular, constraint applications usage of shared resources, like the last level cache and DRAM [29, 16], to predetermined quotas. In this work, we do not focus on specific run-time support, which is in fact dependent on the full software stack, but we only focus on exploiting the existing hardware QoS support. In fact, software-based are deployed on top of existing hardware support for QoS and need to be consistent and compatible with the underlying QoS configuration at the risk of obtaining suboptimal or even counter intuitive results. For example, while it is possible to regulate the memory bandwidth of each core by constraining read/write requests, for example with MemGuard [16], the actual operation of bandwidth regulation and prioritization mechanism can be jeopardized by a QoS configuration assigning low priority in the memory controller to the high bandwidth thread. Nonetheless, QoS features can also be leveraged to support and improve the effectiveness of more complex software-level paradigms.

Hardware providers are increasingly aware of the importance of regulating the impact of contention and are providing hardware level solutions for controlling and apportioning the usage of shared resources such as Intel Resource Director Technology (RDT) [33] and ARM Memory System Resource Partitioning and Monitoring (MPAM) [9]. Initial assessment of those solutions from the standpoint of timing predictability has been conducted in [49, 63]

arising some concerns on the effectiveness of design and implementations of such modules. A custom FPGA implementation to regulate memory accesses in the Zynq UltraScale+ platform has also been proposed in [32]: despite its effectiveness, the approach entails non-negligible performance overheads due to routing core DDR requests via the PL.

More recently, the availability of increasingly-powerful QoS features [6, 7] in COTS platforms has prompted the exploitation of QoS elements, traditionally used for performance balancing and tuning, as an effective means to control multicore timing interference [48]. An adequate degree of software controllability of such QoS features is also a mandatory requirement [22]. An initial study on the QoS features in a representative platform for the avionics domain has been reported [48], providing evidence of how the manifold QoS features provide sufficient malleability to enforce different resource sharing scenarios. While sharing with [48] the focus on the Zynq UltraScale+ memory controller, in this work we focus on defining quasi isolation envelops for critical tasks against non-critical activities generated from the PL rather than exploring different performance trade-offs among software partitions. In fact, while the work in [48] fits the monolithic application scenario, where the goal is finding a setup that satisfies the performance requirements of all processes, it cannot be used in IMA-MPSoC scenarios to increase the representativeness of early time budgets by devising a set of QoS setups that guarantee a high-degree of performance isolation.

Partial explorations of the impact of SMT-related QoS modules on execution time performance have been conducted in [15, 31] for IBM and Intel processors respectively. The QoS support in the UltraScale+ platform is partially addressed in [41] where a specific QoS setup for the memory controller is used to explore possible throughput configurations for the DDR memory module. These works are focusing on preventing performance degradation rather than exploiting contention control for enabling stronger performance guarantees.

Finally, some works attempt to analytically model the effect of configuring QoS features and contention regulation mechanisms in general. The work in [32] builds on profile-based analytical predictive models to enforce a bandwidth regulation policy, including the use of a set of QoS parameters to regulate traffic from the PL logic. An analytical model of the QoS-400 module in the Zynq UltraScale+ is analyzed in [64]. Despite the expected higher accuracy of analytical characterization approaches, they are often building on partial and potentially misleading information on the hardware modules they are meant to model [14], due to increasing hardware complexity and poor documentation. For this reason, they can be useful for deriving early time estimates, they cannot be generally considered a robust and generic alternative to more empirical approaches, as the one proposed in this work.

3 Target System

We focus on an MPSoC IMA setup in which a system integrator is in charge of deploying in the same computing platform multiple applications with different functional safety requirements from different software providers. These systems are typically scheduled following a layered approach where the first level, either provided by an executive layer or a hypervisor, is responsible for scheduling the different software partitions and each partition is in turn responsible for executing single applications or processes. However, no or limited information is available on software components from other providers unless in the very final stage of system development. In particular, we do not consider relatively simpler monolithic system designs where software components are responsibility of a single provider, and hence simultaneously available for timing analysis purposes, including the early characterization of contention impact and the exploration of different system configurations and QoS setups [45, 48].

In an MPSoC IMA scenario, instead, different software modules are incrementally made available and integrated into the final intended configuration. The lack of early and precise information on the different software modules requires the application of compositional approaches for the analysis of multicore timing interference. Hence, timing budgets can be consolidated before the whole system is available and do not need to be re-determined whenever a new component is integrated.

One practical approach for early characterization of timing interference consists in using synthetic 'aggressor' programs [35] during early stages to test representative contention scenarios despite some software modules may not be available. These synthetic applications can be exploited to generate high load on shared resources. Thus, by running each application against aggressors already in early development stages, early figures on the impact of high contention scenarios on the application's execution time can be produced. However, as the amount of shared resources in multicore processors used in embedded domains is constantly increasing, the impact of aggressors on application's execution time is potentially huge (20x and higher) [36, 55], resulting in overly pessimistic execution time budgets.

In this IMA-MPSoC scenario, tighter figures on contention impact of each module can only be provided by limiting the amount of interference possibly suffered by the application or module under analysis, regardless of (or with limited correlation with) the co-runners. In this work, we show how available QoS support in modern MPSoCs can be exploited to enable early consolidation of tight contention-aware time budgets by providing quasi-isolation scenarios where the impact of contention is bounded by specific QoS configurations.

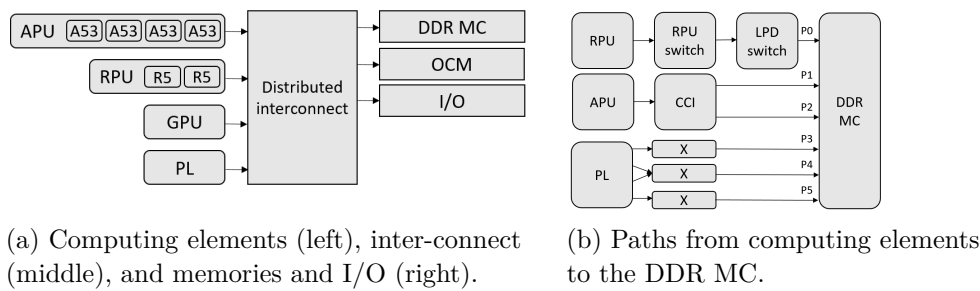
3.1 Introduction to the Zynq UltraScale+

The Zynq UltraScale+ MPSoC [60, 59] comprises four main types of hardware blocks: computing elements (CEs), communication elements, memory, and I/O controllers.

The CEs, see Figure 3(a), include a high-performance CPU cluster (called APU or application processing unit) comprising 4 Arm Cortex A53 cores [8], a real-time CPU cluster (called RPU or real-time processing unit) with 2 Arm Cortex R5 cores [10], an Arm Mali-400 GPU [11], and a programmable logic (PL) block that in real-time systems is usually deployed to synthesize components to support I/O or computing acceleration of some functionalities.

In terms of memories, the MPSoC includes an on-chip memory (OCM), the DDR SDRAM memory controller (DDRMC), and interfaces to access ROM and flash memories, which are generally not used for the normal operation of end-user applications and hence excluded from our discussions in the rest of the paper. The MPSoC also includes a complex I/O system that handles accesses to generic (e.g. USB) and specific controllers (e.g. CAN).

An Arm AXI-based distributed network orchestrates the communications among all elements – usually from (to) CE to (from) memory or I/O. It includes the cache-coherent interconnect (CCI) hardware block that controls aspects related to coherence and distributed memory; top-level switches like the RPU switch; and smaller or secondary switches – highlighted with an 'X' in Figure 3(b). The interconnect is heterogeneous and distributed meaning that the set of switches that each CE has to traverse to reach a given destination varies per CE. For instance, Figure 3(b) shows an abstraction of most relevant connection between the different elements in the system. It shows the interconnect IP blocks each CE has to traverse to reach the DDRMC. This path can be configured, e.g. the APU can use one or two ports to access memory, while the RPU can also send requests to the DDRMC via the CCI using a single DDRMC port.



■ **Figure 3** Different block diagrams of the Zynq UltraScale+ MPSoC.

3.2 Hardware support for QoS

The UltraScale+ offers a variety of hardware QoS mechanisms that help shaping the speed at which the requests are sent conveyed from source to destination. The most relevant ones are: **Static QoS**. Every AXI request in a point-to-point communication is tagged with a QoS value (AXQoS) from 0 to 15 that the target of the communication can use to prioritize it (the higher the AXQoS value, the higher the priority).

Dynamic QoS. Different interconnect elements, when they can receive requests from different sources, can apply mechanisms that dynamically adjust the static QoS value to reach a given target metric like controlling the maximum number of outstanding requests.

QVN. QoS virtual networks (QVN) use tokens to control transaction flows to ensure that a transaction can always be accepted at its destination before it is sent by a source.

DDRMC QoS. The QoS at the memory controller offers a complex multilayered prioritization system that we analyze more in detail in the following section.

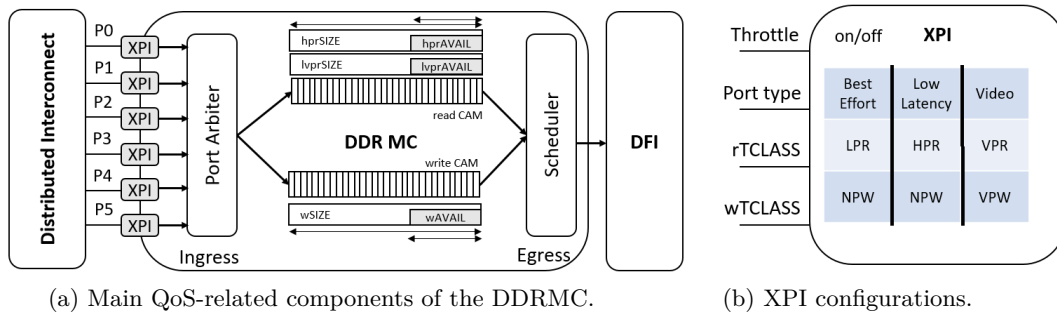
QoS on the interconnect is especially relevant when dealing with request flows from different CEs to different destinations, like DDRMC and I/O [64, 50]. However, when different CEs target the same destination, the speed at which the target processes the requests of each flow is the main factor determining the QoS each flow receives. In this work, we focus on memory contention on the DDR memory as it is one of the major bottlenecks in real-time systems [36, 55]. In fact, it is the last arbitration tier in charge of contain contention when other mechanism failed to provide isolation or timing guarantees. This trend continues [38] as more AI software is used in real-time systems processing huge amounts of data coming from different sensors like video cameras and radars. Hence, in this work, we focus on the QoS in the DDRMC, fix the same static QoS for all requests, and disable all dynamic QoS and QVN mechanisms.

3.3 The DDRMC in the Zynq UltraScale+

The DDR memory subsystem encompasses the DDR memory controller (DDRMC) – which comprises a DDR QoS module – and the DDR Physical Interface (DFI), see Figure 4. The former receives the AXI requests from the distributed interconnect via six different ports and converts them into DDR commands. The latter translates the DDR commands into signals to the external DDR3/4 compliant device.

3.3.1 DDRMC

The DDRMC dynamic scheduling optimizes bandwidth and latency using a programmable QoS controller. Traffic (i.e. flows of AXI requests) arrives to the DDRMC via the six AXI ports (XPI), referred to as P_i in Figure 3(b).



■ **Figure 4** Block diagram of the DDR Memory subsystem in the Zynq UltraScale+.

Traffic Classes. Reads are classified into low, high, or video traffic classes (*LPR*, *HPR*, and *VPR*). Meanwhile writes into low or normal (*LPW=NPW*) and video (*VPW*).

Expired commands. *VPR/VPW* commands behave as low priority when they have not *expired* (i.e. there is not a transaction timeout). Once expired, they are promoted to a priority higher than that of the *HPR/NPW* commands. The timeout period for *VPR* and *VPW* transactions (*RTOUT* and *WTOUT*, respectively) is configured via control registers.

CAMs. DDR commands (or simply commands) translated from incoming AXI requests are stored into the content addressable memories (*CAMs*) both *shared by all ports*.

- A 64-entry write CAM or *wCAM* that is shared by all traffic classes (and all ports).
- A 64-entry read CAM or *rCAM* split into two partitions based on traffic type.
 - The first partition (*hpr rCAM*) is used for *HPR* traffic classes.
 - The second partition (*lvpr rCAM*) is used for *LPR/VPR* traffic classes.

There is a single *rCAM* so that all ports with *HPR* traffic share the *hpr rCAM* and all ports with *LPR/VPR* traffic share the *lvpr rCAM*. The size of the *hpr rCAM* and *lvpr rCAM* is controlled by configuration registers: *hprSIZE* and *lvprSIZE* respectively. Each partition can be configured to have from 1 to 64 entries, with the constraint that their addition must be equal to the *rCAM* size, 64. Hence, for instance, if *hprSIZE* = 24 then *lvprSIZE* = 40.

CAM allocation is performed by the Port Arbiter (*PA*) that selects from all DDR ports the command to issue to the CAMs based on several levels of arbitration.

1. Reads are prioritized while there are *VPR* expired, or there are reads and no expired *VPW*. Writes are served when there are no reads, and if there are expired *VPW* and no expired *VPR*. The expiration period can be configured via setting timeouts for *VPR/VPW*.
2. *HPR* traffic has higher priority than *LPR/VPR* on the read channel and *NPW/VPW* has the same priority on the write channel, with *VPR/VPW* prioritized if they time out.
3. Priorities are given on per-command based on their static QoS (*AXQOS* signals).
4. Conflicts are resolved using round-robin arbitration.

Port throttling changes this behavior by throttling ports that has their throttle-enable control register set, when certain occupancy-related conditions are met:

- When the available entries in the *hpr rCAM* partition are below an availability threshold, set via a control register (*hprAVAIL*), those ports with read traffic mapped to the *HPR* class can be throttled, if their port *hpr* throttling is enabled.

- Likewise, when the number of available entries in the *lvpr rCAM* is below an availability threshold (*lvprAVAIL*), those ports with read traffic mapped to the LPR class can also be throttled, if their port *lpr* throttling is enabled.
- When the count of *wCAM* available entries is below *wAVAIL*, those ports with write traffic mapped to the LPR=VPR class can also be throttled, if their port write throttling is enabled.

Note that ports with traffic mapped to the VPR/VPW class cannot be throttled. Naturally, and beyond the port throttling control, all ports are stalled when any CAM or any other internal resource of the DDRMC is exhausted.

Other. There are other DDRMC features that we have not used because their interaction with the ones we actually use is hard to control, as described in Section 4. These features are port aging (that moves a port to the highest priority when an outstanding request is not served after the established time); urgent transactions (indicating that there is a read/write urgent transaction); and regions, defined at port level to help mapping AXI static priorities and traffic classes.

3.3.2 DFI

When issuing commands from the CAMs to the DFI, command reordering is allowed to favor page hits, potentially causing out-of-order execution of the commands. A regulator limits the issue to up to 4 out-of-order commands. When it is disabled, no restriction is applied, resulting in no control over the number of out-of-order commands executed.

CAM deallocation. On the egress side of the CAMs, depicted as scheduler in Figure 4(a), the SoC allows setting a maximum starve period that a CAM can be without issuing a command to the DFI. There is one period per CAM (*hpr rCAM*, *lvpr rCAM*, and *wCAM*) before the queue goes into a “critical” state and it gets priority to send commands to the DFI. Note that reduced starving periods increase the switching among queues.

3.4 Software Partition Setup

The most natural and efficient way to use the Zynq UltraScale+ in IMA-MPSoC real-time systems is by consolidating different *software partitions* (SWP), ensuring that the needed mechanisms to simplify integration and testing are in place. All main real-time operating systems and hypervisors build on the concept of separation kernels that enable different software partitions to achieve the required safety and security goals. Examples include Lynx Secure [40] and DDC-I Deos [20], which are compliant with the highest-criticality levels in Avionics, i.e. DAL A in DO-178C [47]. SWPs are usually executed in a disjoint set of the available CEs in the underlying platform to reduce timing interactions among them, see Figure 1. In the Zynq UltraScale+, a SWP can span from using a single R5 or A53 core to use a subset of the R5 cores, A53 cores and integrate some acceleration in the PL.

To perform a reasoned exploration of configuration setups, we define the application deployment scenarios (ADS) shown in Table 1. We focus on two classes of applications, depending on whether they comprise critical tasks (CT) or not (NCT). In each ADS, we assume up to two SWPs with CT applications being deployed in the CPU clusters and NCT ones in the PL. Applications are assumed to be independent or share data via predictable communication channels [4]. The goal is to isolate the performance of the applications of the critical SWPs from the traffic coming from the PL, so that they can be analyzed in isolation.

■ **Table 1** SWP active in each core under each ADS.

	R5	R5	A53	A53
ADS1	SWP1	–	–	–
ADS2	–	–	SWP1	–
ADS3	SWP1	–	SWP1	–
ADS4	–	–	SWP1	SWP2
ADS5	SWP1	SWP2	SWP1	SWP2

Other deployment scenarios are possible in which the PL is not used and the NCT runs, for instance, in the A53 cores. In that case, the NCT running in the A53 can be mapped to one port (e.g., P1), while the CT in the R5 can use a different port (e.g., P2), see Figure 3(b). This port selection is a configurable option [48]. In that scenario, the very same principles we describe in this work apply. In fact, the scenarios we address, with the NCT running in the PL, are more challenging since both, the R5 and A53, limit the number of requests in-flight, which further limits the pressure on memory. In particular, the R5 cores allow one in-flight load/store per core and the A53 allows a maximum of 3 in-flight loads per core. Instead, the PL can exploit more ports to memory, and we are able to instantiate several AXI Traffic Generators per port, allowing many more independent requests, hence resulting in higher pressure on memory (more details on the setup are provided in Section 5.1).

4 Quasi Isolation QoS Setups

4.1 Context and Approach

Meeting safety standards requirements against the complexity of current and upcoming MPSoCs is a challenging endeavor. Evidence must be provided that the contention tasks generate on each other is anticipated and controlled. Unfortunately, such evidence cannot be derived by acquiring full information about hardware behavior to build a comprehensive and accurate model on how resources are shared among co-running tasks. While reasonable, such approach is deeply invalidated in practice.

On the one hand, it is extremely improbable, if at all possible, that IP providers give full access to the currently-confidential technical documentation as required to derive detailed contention models. The number of examples in this direction are endless: from the very limited information on NoCs (e.g. the functional behavior of the Arm NIC-400 [5] is limited to few pages and, similarly, the description provided by NXP of its CoreNet Coherence Fabric in the T2080 TRM [26] is minimal and includes no information about its internal behavior in terms of buffering or prioritization), to the almost non-existent information about GPU timing behavior [11]. Such trend is not expected to change in the near future, with recent architectures like the NXP LX2160 [44] and the Xilinx Versal [62] equipping increasingly complex components with increasingly limited descriptions of their functional and timing behavior. The Zynq UltraScale+, target of this work, is not an exception: even just the memory controller exhibits several levels of prioritization (see Section 3.3.1) and there is not available information to derive the exact way the scheduler sends requests to the DFI, how exactly the drain of the CAMs occurs when a CAM goes critical, and many other details.

On the other hand, even if enough information was available, modern MPSoCs include a score of dynamic features that make modeling extremely hard without resorting to overly-pessimistic (conservative) assumptions. The Zynq UltraScale+ is a clear example of such scenario as it includes a relevant set of dynamic features that are triggered based on the

(dynamic) behavior of the traffic. Dynamic features include, among others the read and write timeout feature, which depends on how long a command is waiting until being served; the port throttling mechanism, which is triggered based on the availability of the partitions of the rCAM; the wCAM, which in turn, depends on the (dynamic) traffic coming via the different port to the DDRMC; and the prioritization mechanism in the PA, which builds on the status of expired/not expired commands.

Overall, while several reverse engineering and characterization of specific features have been carried out (from SMT processors, cache, memory, and GPUs), the complexity of the current MPSoCs, the limited information, and the number of different IP components intervening in the computation and communication, prevent the definition of a precise contention-aware functional and timing model in practice.

Hence, the challenge for the real-time research community and original equipment manufacturers and TIER1/2 companies in critical domains is to make the system as safe as possible building on the (limited) available information supported by empirical evidence. The remaining uncertainty (risks) are covered by specific mechanisms defined in safety standards like *safety nets* that can assume control of the system if the main MPSoC fails either in terms of hardware reliability or in terms of execution time violations [21].

4.2 Concepts and Benefits

Building on the considerations above, in this work, we do not attempt to develop a model that describes how the different QoS mechanisms work and make predictions for other applications, or how the current application would behave under a different QoS setup. Instead, in our target IMA-MPSoC setup (see Section 3.3.1) we aim at enforcing a high-degree of performance isolation. A Quasi Isolation QoS setup, QIQoS for short, is a particular configuration (setup) of the QoS mechanisms in the underlying platform that provides performance guarantees to a particular set of tasks running in different CEs. That is, QIQoS helps containing the impact that MPSoC contention generated by the NCT can have on the CT, and hence makes early time budgets more representative under a set of QoS setups.

We assess the quality of a specific QIQoS along three main axes: first, for performance guarantees, we look at the slowdown of the CT (the lower the better); second, for timing predictability, we evaluate CT's performance variability when run against different NCT (the lower the better), for instance different aggressor benchmarks that put variable high load on the shared resources; finally, for overall performance and fairness, we consider the average performance of the NCT (the higher the better). This last dimension can be used as a tie breaker among comparable QIQoS setups. We will formalize relevant metrics for evaluating QIQoS setups in Section 5.2.1.

When deployed, a QIQoS setup simplifies incremental integration by reducing the contention impact that co-runners (NCT) can have on the analysis tasks (CT). QIQoS allows deriving timing budgets that are robust against contention scenarios and do not build on any specific run-time support on commercial MPSoCs. Both aspects together are fundamental enablers for the integration and reuse of software modules from different vendors in mixed-criticality systems. The achieved quasi-isolation guarantees the applications' timing behavior, partially consolidated in the early development stages, will be confirmed at integration, reducing the risk of unexpected timing misbehavior and commercially disruptive rollbacks. The proposed QIQoS approach contrasts with previous works that assuming that both CT and NCT are known and focus on exploring different QoS setup that satisfies performance requirements [48]. That is while QIQoS focus on IMA-MPSoC setups, previous works [48] target a monolithic application as presented in Section 1.

4.3 Application to the DDRMC

We instantiate the QIQoS approach on the DDRMC by developing two specific and well-justified QoS isolation setups, each one exploiting a different set of QoS mechanisms at the DDRMC level. QIQoS1 exploits the timeouts defined at port level; QIQoS2 leverages CAM draining features (starving) in the DDRMC. A comprehensive overview of QIQoS parameters for each ADS is provided in Table 2.

4.3.1 Common elements to QIQoS1 and QIQoS2

We first develop on the main common features to both QIQoS. These arose from a series of empirical observations on how to configure some QoS features, since other configurations would prevent enforcing performance guarantees.

- **Use of private ports.** Each CT uses a private port to memory that is not shared with any other CT or NCT. We do so because some QoS features, like traffic class and port type, are set at the port level, so sharing the same port can produce uncontrolled CT's performance drop. Taking Figure 3(b) as a reference, ports P0-P2 are reserved for the CT that run in the R5 and/or A53 cores, and P3-P5 for the PL.
- **Reduced command reordering.** In all QIQoS we set the maximum number of out-of-order commands that can be sent from the CAMs to the DFI to the minimum value allowed (4). Without this limitation, the DDRMC is allowed to prioritize many memory commands over an older memory command if they hit in an open page. This is done for performance-improvement reasons obtained by enabling the DDRMC to increase page hits. However, if the memory command that are bypassed by more recent commands belongs to the CTs, out-of-order commands may have disruptive effects on CTs predictability [36, 55].
- **Avoidance of incoherent QoS setups.** We prevent the incoherent QoS setups [48] by configuring the port type in accordance with the traffic class. Best Effort (BE) ports type are mapped to LPR and LPW traffic classes (respectively for reads and writes), low latency (LL) ports to HPR and LPW traffic classes, and video priority (VP) ports to VPR and VPW traffic classes. This setup is summarized in Figure 4(b). In Table 2, port type and traffic assignments follow these rules for every port (P_i) and under any ADS and QIQoS.
- **Maximization of resource usage.** In the same line, we force the number of entries assigned to HPR and LPR/VPR traffic in the rCAM to match the total number of entries. In particular, we assign 32 of the 64 entries to each partition ($hprSIZE = lvprSIZE = 32$). In Table 2 we see that this criterion holds for all ADS.

Another commonality of all QIQoS setups is that some additional QoS features of the DDRMC are not considered, namely: port aging, urgent transactions, and regions defined at port level. While these features provide additional capabilities to control the ingress of requests to and the egress of memory commands from the DDRMC, they are hard to master in conjunction with the other QoS mechanisms in use. On the one hand, the number of possible QoS configurations increases exponentially. On the other hand, their use can easily produce non-linear effects or even jeopardize the effect of other QoS features, ultimately precluding any chance to achieve the required isolation [48].

4.3.2 QIQoS1

The first QIQoS setup leverages on controlling the ingress of requests to the CAMs to provide isolation for the CT. To that end, QIQoS1 builds on the timeout feature at port level, while the egress prioritization (i.e. the starving feature) is not used. The rows ($ADS_i, QIQoS1$) in Table 2 summarize the parameters enforced under QIQoS1 for each ADS.

■ **Table 2** Parameters of each QIQoS for the DDRMC of the Zynq UltraScale+ for each ADS.

		Port															CAM				Other
		P0				P1				P2				P3,P4,P5			Ingres			Egress	
		TOUT	Type	Traff	Throt	TOUT	Type	Traff	Throt	TOUT	Type	Traff	Throt	TOUT	Type	Traff	Throt	HPR rCAM	LPR rCAM	wCAM	
ADS1	QIQoS1	RT=1 WT=X	VP	VPR VPW	Off	-	-	-	-	-	-	-	-	LL	HPR LPW	On	hprSIZE=32 hprAVAIL=Y	hprSIZE=32	wSIZE=64 wAVAIL=Y	ShprCAM=Off SlprCAM=Off SwCAM=Off	X = (32, 128, 324, 936, 1024, 2048) Y = (1,2,4,8,12) Z = (1,16,32,64)
	QIQoS2	-	LL	HPR LPW	Off	-	-	-	-	-	-	-	-	BE	LPR LPW	On	hprSIZE=32	hprAVAIL=Y	wSIZE=64 wAVAIL=Y	ShprCAM=1 SlprCAM=40 SwCAM=Z	
ADS2	QIQoS1	-	-	-	-	RT=1 WT=X	VP	VPR VPW	Off	-	-	-	-	LL	HPR LPW	On	hprSIZE=32 hprAVAIL=Y	hprSIZE=32	wSIZE=64 wAVAIL=Y	ShprCAM=Off SlprCAM=Off SwCAM=Off	X = (32, 128, 324, 936, 1024, 2048) Y = (1,2,4,8,12) Z = (1,16,32,64)
	QIQoS2	-	-	-	-	-	LL	HPR LPW	Off	-	-	-	-	BE	LPR LPW	On	hprSIZE=32	hprAVAIL=Y	wSIZE=64 wAVAIL=Y	ShprCAM=1 SlprCAM=40 SwCAM=Z	
ADS3	QIQoS1	RT=1 WT=X	VP	VPR VPW	Off	RT=1 WT=X	VP	VPR VPW	Off	-	-	-	-	LL	HPR LPW	On	hprSIZE=32 hprAVAIL=Y	hprSIZE=32	wSIZE=64 wAVAIL=Y	ShprCAM=Off SlprCAM=Off SwCAM=Off	X = (32, 128, 324, 936, 1024, 2048) Y = (1,2,4,8,12) Z = (1,16,32,64)
	QIQoS2	-	LL	HPR LPW	Off	-	LL	HPR LPW	Off	-	-	-	-	BE	LPR LPW	On	hprSIZE=32	hprAVAIL=Y	wSIZE=64 wAVAIL=Y	ShprCAM=1 SlprCAM=40 SwCAM=Z	
ADS4	QIQoS1	-	-	-	-	RT=1 WT=X	VP	VPR VPW	Off	RT=1 WT=X	VP	VPR VPW	Off	LL	HPR LPW	On	hprSIZE=32 hprAVAIL=Y	hprSIZE=32	wSIZE=64 wAVAIL=Y	ShprCAM=Off SlprCAM=Off SwCAM=Off	X = (32, 128, 324, 936, 1024, 2048) Y = (1,2,4,8,12) Z = (1,16,32,64)
	QIQoS2	-	-	-	-	-	LL	HPR LPW	Off	-	LL	HPR LPW	Off	BE	LPR LPW	On	hprSIZE=32	hprAVAIL=Y	wSIZE=64 wAVAIL=Y	ShprCAM=1 SlprCAM=40 SwCAM=Z	
ADS5	QIQoS1	RT=1 WT=X	VP	VPR VPW	Off	RT=1 WT=X	VP	VPR VPW	Off	RT=1 WT=X	VP	VPR VPW	Off	LL	HPR LPW	On	hprSIZE=32 hprAVAIL=Y	hprSIZE=32	wSIZE=64 wAVAIL=Y	ShprCAM=Off SlprCAM=Off SwCAM=Off	X = (32, 128, 324, 936, 1024, 2048) Y = (1,2,4,8,12) Z = (1,16,32,64)
	QIQoS2	-	LL	HPR LPW	Off	-	LL	HPR LPW	Off	-	LL	HPR LPW	Off	BE	LPR LPW	On	hprSIZE=32	hprAVAIL=Y	wSIZE=64 wAVAIL=Y	ShprCAM=1 SlprCAM=40 SwCAM=Z	

CT. Read requests from the CT are assigned the VPR traffic class. QIQoS1 sets the timeout values for reads coming from the port(s) used by the CT to one $RT = RTOUT = 1$ as read operations are usually blocking and any delay they can suffer directly affects performance.

Write operations (e.g. stores) instead are assigned the VPW traffic class. They can be processed “off-line” by the A53 as they are not blocking – to a certain extent. For this reason, we set variably high $WT = WTOUT$ values, symbolized with **X** in QIQoS1 entries in Table 2. The lower the value of WTOUT, the more frequently requests of the CT – mapped to the VPR/VPW traffic classes – will transition into expired mode and, therefore, will be prioritized over the other requests. Hence, by varying WTOUT we aim to achieve different balance points between CT isolation and NCT performance.

NCT. Read traffic of the NCT is set as HPR. This allows to segregate requests from the CT and NCT in the rCAM with a view to favor isolation: the HPR traffic from the NCT will use the HPR part of the rCAM, while the VPR traffic from the CT will exploit the LPR part of the rCAM. It should be noted that, under normal operation, the NCT HPR traffic do have a higher priority than CT VPR traffic. However, the use of sufficiently small values for RTOUT and WTOUT ensures that the CT VPR traffic is steadily expired and hence its priority is promoted to surpass that of the NCT.

By varying the number of entries that must be available in the hpr rCAM partition (hprAVAIL), symbolized with **Y** in QIQoS1 entries in Table 2), it is possible to control the contention generated by the NCT. With high availability thresholds, the port assigned to the NCT will be throttled down more frequently (and vice versa), hence stalling NCT’s generated traffic.

Write traffic of the NCT is set as LPW=NPW traffic classes that are not subject to timeout (only VPR/VPW classes are). For the wCAM availability (wAVAIL) we apply the same value (**Y**) set for hprAVAIL to control the write traffic of the NCT by stalling write traffic from the ports of the NCT (P3-P5) more frequently for low wAVAIL values. Also note that as the CT port (P0-P2) type is *video priority*, CT traffic will not be stalled due to low availability values (it will only when the CAM gets fully occupied).

4.3.3 QIQoS2

Unlike QIQoS1, that relies on the timeout feature to isolate the requests from the CT, QIQoS2 builds on CAM egress control.

Traffic classes. In particular, the read/write requests of the CT are respectively set to HPR/LPW traffic class and the read/write requests of the NCT to the LPR/LPW traffic class (respectively). Hence, since there is no VPR/VPW traffic class, the timeout feature is not used. The main rationale behind QIQoS2 is that no request is upgraded due to a timeout and hence under the default traffic class arbitration policy HPR traffic class is prioritized over LPR and VPR traffic classes.

Starving control. QIQoS2 combines traffic class control as presented above with the CAM egress control feature, or starvation control, to ensure that the time commands of CT are in the CAMs is bounded. In particular:

- We set the starving attribute of the rCAM's hpr partition to 1 ($ShprrCAM = 1$) so as to make reads of the CT to be served as soon as possible.
- The read traffic of the NCT is mapped to the lvpr rCAM partition for which we set a maximum starving period of 40 cycles ($SlvprCAM = 40$).
- Finally, we explore different values for the starving period of the wCAM, which is shared by writes from all ports, to assess its impact on predictability and performance. This is symbolized as **Z** in QIQoS2 entries in Table 2.

Like QIQoS1, QIQoS2 controls CAM availability (**Y** in QIQoS2 entries in Table 2): with high availability thresholds causing the port assigned to the NCT to be throttled down more frequently, hence stalling NCT traffic. Also note that, under QIQoS1, NCT reads and CT reads are respectively mapped to the hpr and lpr rCAM partitions, as opposed to QIQoS2, where the hpr rCAM partition holds the CT reads and the lpr rCAM partition holds NCT reads.

4.3.4 Generalization

The concept of QIQoS, i.e. exploiting specific QoS setups to guarantee a high-degree isolation of some applications regardless of the contention co-runners may generate on MPSoC's shared resources, can be extended to other QoS mechanisms and resources. However, the concrete instantiation requires some adaptations with respect to what has been presented in this work.

In this work we focused on the realization of QIQoS on the Zynq UltraScale+. This decision is motivated not only by the complexity of the QoS mechanisms in its DDRMC, but also because of the industrial relevance of the Zynq UltraScale+, which is already considered for avionics certification when running different functionalities (subsystems) [59]. In terms of potential reuse, the Zynq UltraScale+ instantiates the Synopsis Universal DDRMC(uMCTL2) [51] which is quite configurable, allowing designers to tailor it for optimizing latency, bandwidth, and area. Any MPSoC implementing the same DDRMC can directly benefit from the results of this work.

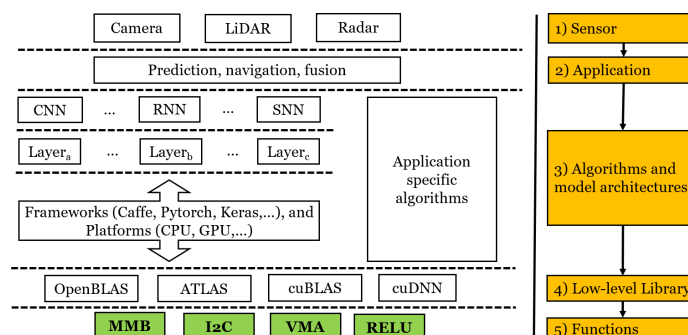
5 Experimental Setup and Results

In this work we focus on a ZCU102 Evaluation Board that comes as part of the Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit [58]. The board is equipped with a Xilinx Zynq UltraScale+ MPSoC [60]. We run on bare-metal (no operating system) and the external

code reduces to the First Stage Boot Loader (FSBL) provided by Xilinx toolchain (Vitis-2019.2). This contributes reducing non-hardware interference (noise) in the experiments. We developed low-overhead software to configure the board by writing to specific control registers and read execution cycles. Under ADS4 and ADS5, we force each SWP to access a different subset of the L2 cache sets, preventing them from evicting each other's data in the LLC. The NCTs were synthesized as accelerators on the PL, which has a direct connection to the three uppermost DDRc ports using the high-performance non-coherent ports (*hp0_lpd*, *hp1_fpd*, *hp2_fpd*, *hp3_fpd*) in its default configuration. The board has different clock domains, and each of them was configured to its maximum allowed frequencies, i.e., PL 250MHz, APU 1200MHz, LPD-interconnect/RPU 500MHz, and FPD-interconnect/DDRc 533.500MHz.

5.1 Experimental Setup

Kernels. We have reviewed different performance-demanding applications relevant for existing and forthcoming safety-critical systems, such as those providing object detection and navigation capabilities. While some of those applications can be run in accelerators, many of them are run on the CPU, either because accelerators are too busy, or because their working set is not overly large and the overheads to issue kernels and transfer data do not pay off [48]. For instance, radar-based object detection uses small matrices, and LiDAR-based object detection may find accelerators busy running heavier camera-based object detection. Hence, both are examples of data-intensive workloads often run in the CPU. The schematic of the hierarchy of their components is shown in Figure 5. A key element in many of those applications is the use of Convolutional Neural Networks (CNNs) for camera and LiDAR-based object detection [52, 2, 3]. In those CNNs, a central element is *matrix multiplication* (MMB) [52], which has been shown to account for most of the execution time (between 67% and 98.5% across deployments [23, 25]). Along with MMB, some other compute-intensive CNN layers rely on *image-to-columns* (I2C) used for tensor lowering to enable matrix convolutions needed by neural networks, and the *rectifier* (RELU) function in neural networks defined as the positive value of its argument [3]. Libraries for CNNs also include other matrix-based operations such as the pervasive *vector-multiply-add* (VMA) and *matrix transpose* (MT) [52]. Those kernels are also present in other key applications such as, for instance, commercial automotive radar applications [27, 53], which build upon MMB to compute the (self) covariance of the input radar data. Overall, as CT applications, we deploy the following benchmarks: MMB, I2C, RELU, VMA, and MT.



■ **Figure 5** Hierarchy of applications and their components with mapping to specific kernels.

NCT. In the PL, as NCT (i.e. aggressor benchmarks), we instantiate one or several AXI Traffic Generator [61] (ATG) modules to generate variable traffic to stress the DDR. The ATG generates read and write traffic with a burst size of 16 bytes. Operations are strided so they access all DDR banks to maximize the chance of generating page misses on the kernels (CT). Note that it is generally not possible to know the exact memory access patterns of the kernels. Likewise, it is not feasible to interleave the request of the kernels and the NCT at will. For instance, if the kernel accesses banks (B0, B0, B0, B1, B1, B2) it is not possible to force the NCT to access the same bank as the kernel but few cycles before to ensure that the kernel suffers a page miss on every memory access. The degree of *controllability* required is not achievable in real hardware platforms, not only because of the noise incurred by the RTOS on the NCT but also due to the inherent hardware jitter arising from execution in out-of-order execution processor pipelines and multi-level cache systems.

We define four configurations where NCT (i.e. ATGs) produces an increasing load: Low, Medium, High, and Very high. Under each of these ATG setups, we instantiate an increasing number of ATG units per port (P3, P4, and P5), see Figure 3(b). In particular we instantiate 1 (L), 3 (M), 5 (H), 9 (V) ATGs per port, each one constantly accessing all banks. Hence, the pressure they put in the memory controller is huge and much higher than expected by a regular accelerator, which combines memory accesses phases with computing phases.

When run in isolation, the PL achieves the following bandwidth results (in brackets the relative percentage of the peak bandwidth measured in giga transfers per second): 6.1 GT/s (14%) for L, 17.5 GT/s (40%) for M, 27.8 GT/s (64%) for H, and 43.2 GT/s (100%) for V.

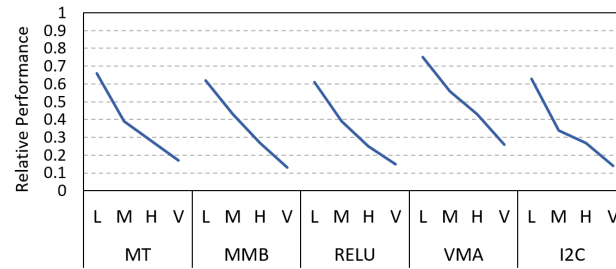
5.2 Experimental Results

5.2.1 Evaluation metrics

QIQoS aim at providing guarantees on the performance of the CT under different loads generated by the NCT. We use three main metrics to assess the effectiveness of a QIQoS setup.

- M1.** Minimize the slowdown, i.e. maximize the relative performance (rperf), of the CT. A rperf of X means a slowdown of $1/X$ (e.g., rperf=50% means $2x$ slowdown).
- M2.** Reducing CT's rperf variability across different loads that the NCT (the PL in our case) can put on the DDRMC. Note that M1 and M2 contribute to the primary goal of finding a QoS setup that satisfies the performance requirements of all processes and increase the representativeness of early time budgets by achieving high-degree of performance isolation.
- M3.** A secondary goal is maximizing NCT rperf, in particular preventing that the NCT receive no service, as long as the target minimum thresholds set for M1 and M2 are achieved.

In terms of M1 and M3, when a SWP encompasses several tasks either as CT or NCT, as it is the case in ADS3 and ADS5, we report the average rperf of all CT tasks and the average of all NCT tasks, respectively. For instance, under ADS3 we report as CT rperf the average of the rperf of the R5 and A53 tasks in each SWP (i.e., SWP1 and SWP2). Likewise, as rperf of the NCT the average of the rperf of all ATGs. Our results show that in all scenarios our results show that the variability in the rperf of the tasks is less than 7 percentage points.



■ **Figure 6** rperf of the CT (kernels) without any QIQoS setup.

5.2.2 Uncontrolled contention

When running the different kernels under the default QoS setup (i.e. no QIQoS) in the Zynq UltraScale+ increasing the traffic load sent from the PL to the DDRMC (L, M, H, and V), we observe that the impact of DDRMC contention on the performance of the benchmarks is huge, see Figure 6. The reported results have been collected while running the benchmarks in one core of the A53 (ADS2), but exactly the same trend is also observed when the benchmarks are run in an R5 core. Even under the smallest PL, i.e. ATG, load (L), rperf drops down to the range 0.60 – 0.75 for the different benchmarks. As we increase the load, the DDRMC saturates reducing the rperf of the CT down to the range of 0.10 – 0.25 for V.

5.2.3 Detailed Analysis

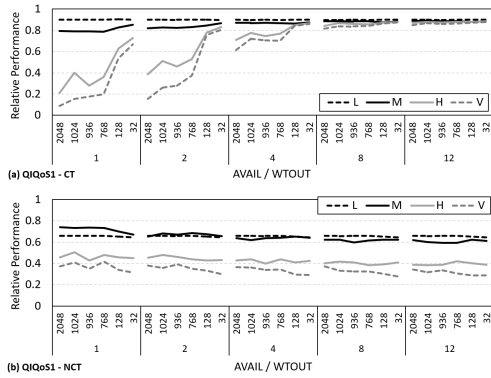
In this section, we analyze the challenging ADS4 setup that comprises two SWPs whose tasks are CT, and the ATGs in PL set as NCT, so all traffic coming from ports P3, P4 and P5 belongs to the NCT. For the sake of conciseness, we also focus on the kernel VMA, for which we provide detailed explanations. The results obtained for ADS1, ADS2, ADS3, and ADS5 and the wider set of kernels are explained in Section 5.2.4. Under ADS4 the baseline performance is that of a single copy of VMA running in isolation. This allows us to discriminate between the performance slowdown caused by deploying a second instance of VMA in another A53 core and the slowdown due to the traffic coming from the PL.

5.2.3.1 QIQoS1

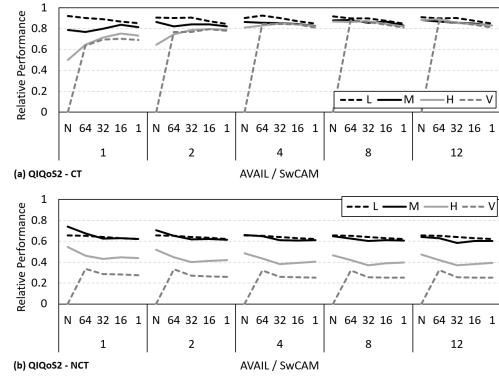
The rperf variation observed for both CT and NCT in ADS4 scenario under QIQoS1 is reported in Figure 7. QIQoS1 parameters values are mapped to the x-axis, reporting different CAM availability thresholds ($\mathbf{Y} = hprAVAIL = wAVAIL$) within 1 and 12, and different $\mathbf{X} = WTOUT$ values from 2048 down to 32 (as defined in Table 2 for ADS4-QIQoS1). Note that for *QIQoS1* we refer to both *hprAVAIL* and *wAVAIL* as *AVAIL* for simplicity.

CT performance. Figure 7(a) focuses on VMA rperf under QIQoS1. Note that the rperf results are the same for *both* VMA copies, each running in a different A53 core. The different series represent an increase load of the PL: L, M, H, and V. We observe the following:

1. Under the PL load *L* (dashed black line), the impact of the NCT traffic on the CT is notably reduced: the slowdown is quite small and stable with rperf around 90% under all *WTOUT* and *AVAIL* setups. This slowdown, that is even observed under the highest isolation configurations (i.e. high *AVAIL* values and low *WTOUT* values) is caused by the contention the two VMA copies generate on each other.



■ **Figure 7** VMA (CT) and NCT rperf for ADS4 under QIQoS1.



■ **Figure 8** VMA (CT) and NCT rperf for ADS4 under QIQoS2.

2. Under the PL load M (solid black line) results are also quite stable, though in this case CT’s rperf sits between 80% (specially for small $AVAIL$ values) and 90%. Under both loads, L and M , the variability across setups is limited because the DDRMC can handle requests of both the PL and both VMA copies running in two A53.
3. Under the loads H and V , the impact of the PL traffic increases with higher variability across $AVAIL$ and $WTOUIT$ setups. In particular, as we increase $AVAIL$, the ports used by the PL (NCT) are throttled more frequently, reducing their impact on the CT. When $AVAIL$ is set to 8 – 12 entries, the slowdown NCT cause on the CT is considerably reduced and the variability across loads also heavily reduces. Also, for any $AVAIL$ value, as we decrease the value of the $WTOUIT$, the CT gets less contention from the NCTs as CT requests are more frequently under expired mode and hence are prioritized over NCTs’ requests.

NCT performance. Figure 7(b) shows the rperf of the PLs. We see that the impact of varying $WTOUIT$ and $AVAIL$ is reduced, with the variability mainly arising because of different PL loads. Under L the PL suffers limited slowdown (i.e. its rperf is high) since the memory controller can comfortably provide the performance required. As we increase the load of the PL to M , H , and V the rperf decreases: for L the rperf stays over 60% while for V it stays below 40%. This is so because heavier loads saturate the DDRMC so they are more affected by the memory activity of the A53. Instead, lighter loads left some bandwidth unused allowing the A53 to inject their traffic with limited impact. As it can be observed, under M and L loads NCT performance is quite similar, even with higher performance under M than under L . Our results seem to suggest that this is due to arbitration among ports that cause higher ID ports to receive comparatively less service under lighter loads.

5.2.3.2 QIQoS2

Figure 8 shows the rperf variation for the CT and NCT in ADS4 scenario under QIQoS2. QIQoS2 relevant parameters are mapped to the x-axis, reporting different CAM availability thresholds ($Y = lvprAVAIL = wAVAIL$) within 1 and 12, and different wCAM starving ($Z = SwCAM$) values, from 64 down to 1 and no starving (as defined in Table 2 for ADS4-QIQoS2). Note that for QIQoS2 we refer to both $lvprAVAIL$ and $wAVAIL$ as $AVAIL$ for simplicity.

■ **Table 3** Analysis of the rperf of the CT and NCT under various benchmarks on ADS4 ($AVAIL=12$ and $WTOUT=32$ for QIQoS1; and $AVAIL=12$ and $SwCAM=1$ for QIQoS2).

		VMA			
		L	M	H	V
QIQoS1	CT	0.90	0.88	0.88	0.87
	NCT	0.64	0.59	0.38	0.27
QIQoS2	CT	0.85	0.82	0.83	0.81
	NCT	0.62	0.60	0.39	0.25

(a)

		MT				MMB			
		L	M	H	V	L	M	H	V
QIQoS1	CT	0.99	0.97	0.96	0.95	0.91	0.88	0.88	0.88
	NCT	0.51	0.27	0.15	0.11	0.62	0.36	0.24	0.19
QIQoS2	CT	0.88	0.85	0.84	0.83	0.82	0.76	0.76	0.76
	NCT	0.59	0.47	0.29	0.19	0.60	0.38	0.23	0.17

		RELU				I2C			
		L	M	H	V	L	M	H	V
QIQoS1	CT	0.96	0.97	0.92	0.97	0.94	0.91	0.89	0.89
	NCT	0.46	0.37	0.12	0.14	0.51	0.29	0.18	0.12
QIQoS2	CT	0.81	0.81	0.80	0.79	0.81	0.77	0.76	0.75
	NCT	0.58	0.42	0.27	0.19	0.59	0.30	0.19	0.14

(b)

CT performance. Regarding the rperf of the CT (VMA), see Figure 8(a), we extract two main conclusions. On the one hand, for the “no starving” case, we see that the impact of the PL is higher. It is particularly relevant the V load for which the experiment, after executing more than 100x times its duration in isolation, did not finish. Hence, we concluded that starving must be enabled and do not further discuss “no starving” results. It is noted that while starving prevention is enabled by default, it is one of the parameters whose impact we wanted to explore and hence decided to observe the impact of disabling it. On the other hand, as we increase $AVAIL$ we see how the rperf of the CT slightly increases. CT’s rperf also increases as we decrease the starvation threshold as CT requests are kept shorter in the CAMs when $AVAIL$ is 1 or 2 (this effect reduces and even disappears for M and L, arguably because the load of the NCT on the DDRMC decreases). When $AVAIL$ goes beyond 2 (i.e. 4, 8, 12), it cancels out the impact of starvation prevention. Anyways, the impact on rperf is relatively small, so QIQoS keeps high quality results (high values for rperf and low variability of rperf across load) under all explored variations.

NCT performance. In Figure 8(b) shows the rperf of the NCT. Other than for the “no starving” setup that, as pointed out before, we exclude from the discussion, the variability is very small across the explored $AVAIL$ and $SwCAM$ values, with a slight decrease with higher values of both parameters. This is so because lower loads of the PL do not saturate the DDRMC, thus leaving some bandwidth for the A53 and R5 to execute with less impact on the PL. The major difference appears across PL loads, with rperf drop values around 30% for L , 40% for M , 60% for H , and 70% for V .

5.2.3.3 Metrics M1, M2, and M3

We assess the quality of QIQoS1 and QIQoS2 using metrics M1, M2, and M3 as defined in Section 5.2.1. For both QIQoS we choose the aggressive isolation setups: $AVAIL = 12$ and $WTOUT = 32$ for QIQoS1 and $AVAIL = 12$ and $SwCAM = 1$ for QIQoS2.

Results are shown in Table 3(a), comparing the QIQoS setups on VMA rperf for variable PL load, yet focusing on the specific scenario ADS4. Regarding M1, for VMA we see that the rperf of the CT is slightly higher with QIQoS1, varying from 0.87 to 0.90, than for QIQoS2 for which CT rperf varies from 0.81 to 0.85. For both QIQoS rperf is quite high. In terms of variability (M2) both are quite similar being 3 percentage points for QIQoS1 and 4 for QIQoS2. The results for M1 and M2 provide evidence that the developed QIQoS are very

competitive, achieving good rperf and isolation figures. This, in fact, allows to define SWPs timing budgets during early stages of the development process of a IMA-SoC product, with high confidence that those bounds are going to hold at operation. In this line, it is worth recalling that under the V load there are 9 ATGs *per port* constantly accessing memory to different banks. This is arguably a much higher load than an accelerator would put on the DDRMC. Finally, the performance of the NCT (M3) is quite similar across both setups.

5.2.4 Wider result set

Figures 7 and 8 provided a detailed set of results and analysis of the proposed QIQoS setups for one specific benchmark (VMA) under a single scenario (ADS4). In this section, we extend the analysis of results to all kernels and all ADS scenarios but, for the sake of conciseness, we restrict our focus on a specific PL load (L) and fixing the parameters $AVAIL = 12$ and $WTOUIT = 32$ for QIQoS1 and $AVAIL = 12$ and $SwCAM = 1$ for QIQoS2. In any case, results for variable loads and kernels under ADS4 are reported in Table 3(b), as will be commented next.

Figure 9 shows the rperf of the CT (left) and the rperf of the NCT (right) for all benchmarks under all ADS scenarios, with PL load set to L . In both charts, results for each kernel under QIQoS1 and QIQoS2 are shown in consecutive bars. For instance, bars 1 and 2 compare the result of QIQoS1 and QIQoS2 for ADS1, bars 3 and 4 for ADS2, bar 5 and 6 for ADS3, and so on so forth. We observe that:

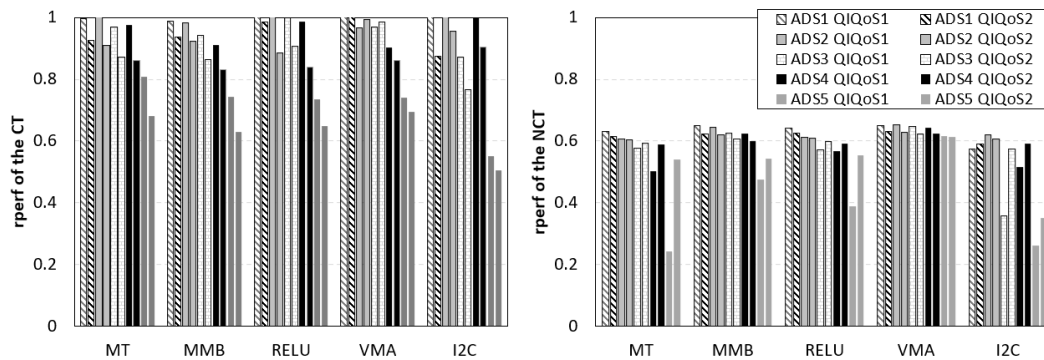
- In terms of the rperf of the CT (left chart), in general QIQoS1 provides slightly better performance than QIQoS2. The rperf reduces for more aggressive ADS staying between 0.5 and 0.8.
- In terms of the rperf of the NCT (right chart), in general results are quite similar across the proposed setups, with QIQoS2 slightly outperforming QIQoS1.

Regarding metrics M1, M2, and M3, under ADS4 the same conclusions derived for VMA under different PL loads generally hold for the other kernels, as reported in Table 3(b). First, QIQoS1 provides higher CT's performance than QIQoS2 (M1). Second, the variability of CT's rperf (M2) is quite similar for both QIQoS: 5 percentage points in the worst case for QIQoS1 and 6 for QIQoS2. And third, in terms of the rperf of the NCT both setups provide similar results with rperf gradually increasing as the load of the PL moves from L to V .

Overall, we conclude the same trends observed for VMA holds for the rest of the kernels and configurations, confirming that the proposed QIQoS achieved the intended goals.

6 Conclusions

In this work we have shown how hardware QoS support can be exploited in modern MPSoCs with the goal of providing a high degree of isolation to selected applications. We instantiate Quasi Isolation QoS setups (QIQoS), introduce and explain two particular QIQoS to achieve isolation in the DDR memory controller of the Zynq UltraScale+. The main lessons learned are that transaction timeout (QIQoS1) and CAM starving control (QIQoS2), both underpinned by traffic classes and port throttling, provide good isolation results. Overall, the proposed QIQoS guarantee that applications' execution time is much less sensitive to co-runners' contention, and hence, the timing estimates obtained when running the application against aggressors in early development stages become much tighter and stable across integrations. Our future work includes exploiting more features of the DDR memory controller to further isolate APU and RPU cores from the PL traffic, and also isolate cores in the APU and the RPU from each other.



■ **Figure 9** Minimum rperf for the CT (left chart) and NCT (right chart), for QIQoS1 and QIQoS2 under L load of the PL ($AVAIL = 12$ and $WTOUT = 32$ for QIQoS1; and $AVAIL = 12$ and $SwCAM = 1$ for QIQoS2).

References

- 1 Irune Agirre, Jaume Abella, Mikel Azkarate-askasua, and Francisco J. Cazorla. On the tailoring of CAST-32A certification guidance to real COTS multicore architectures. In *12th IEEE International Symposium on Industrial Embedded Systems, SIES, Toulouse, France, June 14-16, 2017*, pages 1–8. IEEE, 2017. doi:10.1109/SIES.2017.7993376.
- 2 ApolloAuto. Apollo 3.0 Software Architecture, 2018. URL: https://github.com/ApolloAuto/apollo/blob/master/docs/specs/Apollo_3.0_Software_Architecture.md.
- 3 ApolloAuto. Perception, 2018. URL: https://github.com/ApolloAuto/apollo/blob/r3.0.0/docs/specs/perception_apollo_3.0.md.
- 4 ARINC Inc. *ARINC Specification 653: Avionics Application Software Standard Standard Interface*, June 2012.
- 5 Arm. *ARM CoreLink NIC-400 Network Interconnect Technical Reference Manual*.
- 6 Arm. *ARM CoreLink QoS-400 Network Interconnect Advanced Quality of Service Supplement to ARM CoreLink NIC-400 Network Interconnect Technical Reference Manual*, 2017.
- 7 Arm. *ARM CoreLink QVN-400 Network Interconnect Advanced Quality of Service using Virtual Networks Supplement to ARM CoreLink NIC-400 Network Interconnect Technical Reference Manual*, 2017.
- 8 Arm. *ARM Cortex-A53 MPCore Processor Technical Reference Manual. Version r0p4*, 2022. URL: <https://developer.arm.com/documentation/ddi0500/j/>.
- 9 Arm. *Arm® Architecture Reference Manual Supplement Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A*, 2022.
- 10 Arm. *Cortex-R5 and Cortex-R5F Technical Reference Manual. Version r1p1*, 2022. URL: <https://developer.arm.com/documentation/ddi0460/c/>.
- 11 Arm. *Mali-400*, 2022. URL: <https://developer.arm.com/Processors/Mali-400>.
- 12 ARP4761. *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*, 2001.
- 13 AUTOSAR. *Technical Overview V2.0.1*, 2006.
- 14 Jingyi Bin, Sylvain Girbal, Daniel Gracia Pérez, Arnaud Grasset, and Alain Mérigot. Studying co-running avionic real-time applications on multi-core COTS architectures. In *Embedded Real Time Software and Systems (ERTS2014)*, 2014. URL: <https://hal.science/hal-02271379>.
- 15 Carlos Boneti, Francisco J. Cazorla, Roberto Gioiosa, Alper Buyuktosunoglu, Chen-Yong Cher, and Mateo Valero. Software-Controlled Priority Characterization of POWER5 Processor. In *35th International Symposium on Computer Architecture (ISCA)*, pages 415–426, 2008. doi:10.1109/ISCA.2008.8.

- 16 Marco Caccamo, Rodolfo Pellizzoni, Lui Sha, Gang Yao, and Heechul Yun. MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-Core Platforms. In *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 55–64, USA, April 2013. IEEE Computer Society. doi:10.1109/RTAS.2013.6531079.
- 17 Jordi Cardona, Carles Hernandez, Enrico Mezzetti, Jaume Abella, and Francisco J. Cazorla. NoCo: ILP-based worst-case contention estimation for mesh real-time manycores. In *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, December 2018. doi:10.1109/rtss.2018.00043.
- 18 Certification Authorities Software Team. *CAST-32A Multi-core Processors*, 2016.
- 19 Dakshina Dasari and Vincent Nelis. An analysis of the impact of bus contention on the WCET in multicores. In *HPCC, 2012*. doi:10.1109/HPCC.2012.212.
- 20 DDC-I. Deos, a Time and Space Partitioned, Multi-core Enabled, RTOS Verified to DO-178C ED-12C DAL A, 2022. URL: https://www.ddci.com/products_deos_do_178c_arinc_653/.
- 21 EASA, FAE. *General Acceptable Means of Compliance for Airworthiness of Products, Parts and Appliances (AMC-20). Amendment 23. Annex I to ED Decision 2022/001/R. AMC 20-193 Use of multi-core processors.*, 2022. URL: <https://www.easa.europa.eu/en/document-library/certification-specifications/amc-20-amendment-23>.
- 22 Falk Rehm and Jörg Seitter. Software Mechanisms for Controlling QoS. In *2021 Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Virtual Conference, February 01-05, 2021*, pages 1485–1488, 2016.
- 23 Fernando Fernandes dos Santos, Lucas Draghetti, Lucas Weigel, Luigi Carro, Philippe Navaux, and Paolo Rech. Evaluation and Mitigation of Soft-Errors in Neural Network-Based Object Detection in Three GPU Architectures. In *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 169–176, 2017. doi:10.1109/DSN-W.2017.47.
- 24 Gabriel Fernandez, Javier Jalle, Jaume Abella, Eduardo Quiñones, Tullio Vardanega, and Francisco J. Cazorla. Resource usage templates and signatures for COTS multicore processors. In *Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015*, pages 155:1–155:6. ACM, 2015. doi:10.1145/2744769.2744901.
- 25 Javier Fernández, Jon Perez, Irune Agirre, Imanol Allende, Jaume Abella, and Francisco J. Cazorla. Towards functional safety compliance of matrix–matrix multiplication for machine learning-based autonomous systems. *Journal of Systems Architecture*, 121:102298, 2021. doi:10.1016/j.sysarc.2021.102298.
- 26 Freescale semiconductor. QorIQ T2080 Reference Manual, 2016. Also supports T2081. Doc. No.: T2080RM. Rev. 3, 11/2016.
- 27 Jonah Gamba. Automotive radar applications. In *Radar Signal Processing for Autonomous Driving*, pages 123–142. Springer Singapore, Singapore, 2020. doi:10.1007/978-981-13-9193-4_9.
- 28 Giovanni Gracioli, Ahmed Alhammad, Renato Mancuso, Antônio Augusto Fröhlich, and Rodolfo Pellizzoni. A Survey on Cache Management Mechanisms for Real-Time Embedded Systems. *ACM Computing Surveys*, 48(2):32:1–32:36, November 2015. doi:10.1145/2830555.
- 29 Giovanni Gracioli, Rohan Tabish, Renato Mancuso, Reza Mirosanlou, Rodolfo Pellizzoni, and Marco Caccamo. Designing Mixed Criticality Applications on Modern Heterogeneous MPSoC Platforms. In Sophie Quinton, editor, *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, volume 133, pages 27:1–27:25, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ECRTS.2019.27.
- 30 Danlu Guo, Mohamed Hassan, Rodolfo Pellizzoni, and Hiren Patel. A Comparative Study of Predictable DRAM Controllers. *ACM Trans. Embed. Comput. Syst.*, 17(2), February 2018. doi:10.1145/3158208.

- 31 Andrew Herdrich, Ramesh Illikkal, Ravi Iyer, Ronak Singhal, Matt Merten, and Martin Dixon. SMT QoS: Hardware Prototyping of Thread-level Performance Differentiation Mechanisms. In *USENIX Workshop on Hot Topics in Parallelism*, Berkeley, CA, 2012. USENIX Association.
- 32 Denis Hoornaert, Shahin Roozkhosh, and Renato Mancuso. A Memory Scheduling Infrastructure for Multi-Core Systems with Re-Programmable Logic. In Björn B. Brandenburg, editor, *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*, volume 196, pages 2:1–2:22, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ECRTS.2021.2.
- 33 Intel. *Intel® Resource Director Technology (Intel® RDT) on 2nd Generation Intel® Xeon® Scalable Processors Reference Manual, Rev. 1.0*, 2019.
- 34 International Organization for Standardization. *ISO/DIS 26262. Road Vehicles – Functional Safety*, 2009.
- 35 Dan Iorga, Tyler Sorensen, John Wickerson, and Alastair F. Donaldson. Slow and Steady: Measuring and Tuning Multicore Interference. In *26th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'20)*, pages 200–212, April 2020. doi:10.1109/RTAS48715.2020.000–6.
- 36 Javier Jalle, Mikel Fernandez, Jaume Abella, Jan Andersson, Mathieu Patte, Luca Fossati, Marco Zulianello, and Francisco J. Cazorla. Bounding Resource-Contention Interference in the Next-Generation Multipurpose Processor (NGMP). In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, 2016. URL: <https://hal.science/hal-01259133>.
- 37 Javier Jalle, Eduardo Quiñones, Jaume Abella, Luca Fossati, Marco Zulianello, and Francisco J. Cazorla. A Dual-Criticality Memory Controller (DCmc): Proposal and Evaluation of a Space Case Study. In *RTSS*, pages 207–217. IEEE Computer Society, 2014. doi:10.1109/RTSS.2014.23.
- 38 Matthias Jung, Sally A. McKee, Chirag Sudarshan, Christoph Dropmann, Christian Weis, and Norbert Wehn. Driving into the memory wall: the role of memory for advanced driver assistance systems and autonomous driving. In Bruce Jacob, editor, *Proceedings of the International Symposium on Memory Systems, MEMSYS*. ACM, 2018. doi:10.1145/3240302.3240322.
- 39 Hyoseung Kim, Dionisio De Niz, Björn Andersson, Mark Klein, Onur Mutlu, and Rangunathan Rajkumar. Bounding and Reducing Memory Interference in COTS-Based Multi-Core Systems. *Real-Time Syst.*, 52(3):356–395, May 2016. doi:10.1007/s11241-016-9248-1.
- 40 LYNX Software technologies. LynxSecure Separation Kernel Hypervisor, 2022. URL: <https://www.lynx.com/products/lynxsecure-separation-kernel-hypervisor>.
- 41 Kristiyan Manev, Anuj Vaishnav, and Dirk Koch. Unexpected Diversity: Quantitative Memory Analysis for Zynq UltraScale+ Systems. In *2019 International Conference on Field-Programmable Technology*, pages 179–187, 2019. doi:10.1109/ICFPT47387.2019.00029.
- 42 Reza Miroslou, Mohamed Hassan, and Rodolfo Pellizzoni. Duetto: Latency Guarantees at Minimal Performance Cost. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1136–1141, 2021. doi:10.23919/DATE51398.2021.9474062.
- 43 Jan Nowotsch, Michael Paulitsch, Daniel Buhler, Henrik Theiling, Simon Wegener, and Michael Schmidt. Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement. In *26th Euromicro Conference on Real-Time Systems, ECRTS*, 2014. doi:10.1109/ECRTS.2014.20.
- 44 NXP Semiconductors. QorIQ LX2160A Reference Manual, 2021. Supports LX2120A and LX2080A.
- 45 Xavier Palomo, Enrico Mezzetti, Jaume Abella, Reinder J. Bril, and Francisco J. Cazorla. Accurate ILP-Based Contention Modeling on Statically Scheduled Multicore Systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 15–28. IEEE, 2019. doi:10.1109/RTAS.2019.00010.

- 46 Jon Pérez-Cerrolaza, Roman Obermaisser, Jaume Abella, Francisco J. Cazorla, Kim Grüttner, Irune Agirre, Hamidreza Ahmadian, and Imanol Allende. Multi-core Devices for Safety-critical Systems: A Survey. *ACM Comput. Surv.*, 53(4), 2020.
- 47 RTCA and EUROCAE. *DO-178C - ED-12C, Software Considerations in Airborne Systems and Equipment Certification*, 2011.
- 48 Alejandro Serrano-Cases, Juan M. Reina, Jaume Abella, Enrico Mezzetti, and Francisco J. Cazorla. Leveraging Hardware QoS to Control Contention in the Xilinx Zynq UltraScale+ MPSoC. In *33rd Euromicro Conference on Real-Time Systems, ECRTS 2021, July 5-9, 2021, Virtual Conference*, volume 196, pages 3:1–3:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ECRTS.2021.3.
- 49 Parul Sohal, Michael Bechtel, Renato Mancuso, Heechul Yun, and Orran Krieger. A Closer Look at Intel Resource Director Technology (RDT). In *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, pages 127–139, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3534879.3534882.
- 50 Parul Sohal, Rohan Tabish, Ulrich Drepper, and Renato Mancuso. Profile-driven memory bandwidth management for accelerators and CPUs in QoS-enabled platforms. *Real Time Syst.*, 58(3):235–274, 2022.
- 51 Synopsys. Synopsys Enhanced Universal DDR Memory Controller (uMCTL2). URL: https://www.synopsys.com/dw/ipdir.php?ds=dwc_ddr_universal_umctl2.
- 52 Hamid Tabani, Roger Pujol, Jaume Abella, and Francisco J. Cazorla. A Cross-Layer Review of Deep Learning Frameworks to Ease Their Optimization and Reuse. In *IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC)*, 2020. doi:10.1109/ISORC49007.2020.00030.
- 53 Lee Teschler. The basics of automotive radar, 2019. URL: <https://www.designworldonline.com/the-basics-of-automotive-radar/>.
- 54 P. Valsan and H. Yun. MEDUSA: A Predictable and High-Performance DRAM Controller for Multicore Based Embedded Systems. In *2015 IEEE 3rd International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA)*, pages 86–93, Los Alamitos, CA, USA, August 2015. IEEE Computer Society. doi:10.1109/CPSNA.2015.24.
- 55 Prathap Kumar Valsan, Heechul Yun, and Farzad Farshchi. Taming Non-Blocking Caches to Improve Isolation in Multicore Real-Time Systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Vienna, Austria, April 11-14, 2016. doi:10.1109/RTAS.2016.7461361.
- 56 James Windsor, Marie-Hélène Deredempt, and Regis De-Ferluc. Integrated modular avionics for spacecraft — User requirements, architecture and role definition. In *IEEE/AIAA 30th Digital Avionics Systems Conference*, 2011. doi:10.1109/DASC.2011.6096141.
- 57 Xi-Yue Xiang, Saugata Ghose, Onur Mutlu, and Nian-Feng Tzeng. A model for Application Slowdown Estimation in on-chip networks and its use for improving system fairness and performance. In *34th IEEE International Conference on Computer Design, ICCD, Scottsdale, AZ, USA*, pages 456–463. IEEE Computer Society, 2016. doi:10.1109/ICCD.2016.7753327.
- 58 XILINX. Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit. URL: <https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html#information>.
- 59 XILINX. *Rockwell Collins Uses Zynq UltraScale+ RFSoc Devices in Revolutionizing How Arrays are Produced and Fielded: Powered by Xilinx*, 2018. URL: <https://www.xilinx.com/video/corporate/rockwell-collins-rfsoc-revolutionizing-how-arrays-are-produced.html>.
- 60 XILINX. *Zynq UltraScale+ Device. Technical Reference Manual. UG1085 (v2.1)*, 2019.
- 61 XILINX. LogiCore AXI Traffic Generator. https://www.xilinx.com/products/intellectual-property/axi_tg.html, 2022.
- 62 XILINX. XILINX VERSAL. AI EDGE. <https://www.xilinx.com/products/silicon-devices/acap/versal.html>, 2022.

- 63 Matteo Zini, Daniel Casini, and Alessandro Biondi. Analyzing Arm's MPAM From the Perspective of Time Predictability. *IEEE Transactions on Computers*, 72(1):168–182, 2023. doi:10.1109/TC.2022.3202720.
- 64 Matteo Zini, Giorgiomaria Cicero, Daniel Casini, and Alessandro Biondi. Profiling and controlling I/O-related memory contention in COTS heterogeneous platforms. *Software: Practice and Experience*, 52(5):1095–1113, 2022. doi:10.1002/spe.3053.