Profiling and controlling I/O-related memory contention in COTS heterogeneous platforms

Matteo Zini¹ | Giorgiomaria Cicero¹ | Daniel Casini¹,² | Alessandro Biondi¹,²

¹TeCIP Institute, Scuola Superiore Sant’Anna, Pisa, Italy
²Department of Excellence in Robotics & AI, Scuola Superiore Sant’Anna, Pisa, Italy

Correspondence
Matteo Zini, TeCIP Institute, Scuola Superiore Sant’Anna, Pisa, Italy.
Email: matteo.zini@santannapisa.it

Funding information
Huawei Technologies; Italian Ministry of University and Research (MIUR) / SPHERE project PRIN 2017, Grant/Award Number: 93008800505

Abstract
Motivated by the increasing number of embedded applications that make use of traffic-intensive I/O devices, this work studies the memory contention generated by I/O devices and investigates on the regulation of the bus traffic they generate by means of COTS regulators, namely the QoS-400 by Arm. To this purpose, the behavior of the QoS-400 regulators is analytically characterized and then, taking the Xilinx Ultrascale+ as a reference modern heterogeneous platform, a software infrastructure to control such regulators from Linux is proposed. As an experience report, this article presents the results of an extensive experimental evaluation, based on both benchmarks and microbenchmarks, aimed at validating the effectiveness of QoS-400 regulators in predictably controlling I/O-related memory traffic, as well as assessing the impact of the regulation on software applications and I/O devices themselves.

KEYWORDS
I/O regulators, memory contention, memory regulators

1 | INTRODUCTION

The move from single-processor to multiprocessor and even heterogeneous platforms in response to the always increasing computational requirements of modern applications raised the attention toward the design of mechanisms to control the delays due to contention in accessing shared architectural resources, such as caches, interconnects, and Double Data Rate Synchronous Dynamic Random Access Memory (DDR SDRAM) memories. A recent FAA¹ (Federal Aviation Administration, an operating mode of the U.S. Department of Transportation) report witnesses how memory contention is unquestionably deemed a key factor harming the predictability of real-time software systems. Often, this gives rise to an inevitable pessimism when deriving analytical bounds on the worst-case timing performance of the system.

Many solutions have been proposed over the years to mitigate this problem under disparate settings, ranging from memory bandwidth regulation strategies,²-⁴ cache partitioning techniques,⁵-⁸ memory bank partitioning techniques,⁹ up to the adoption of task execution models specifically designed to predictably control memory contention.¹⁰-¹²

Despite all the attention devoted to the memory contention induced by computational activities running on processing cores or hardware accelerators, limited consideration has been reserved to the memory contention generated by I/O devices.¹³ Indeed, I/O devices typically include a direct memory access (DMA) module to autonomously store and retrieve data from shared memories, hence contributing to the overall memory contention experienced by software tasks and hardware accelerators, and potentially producing non-negligible delays during their execution.
I/O-related memory contention is therefore of utmost importance in many emerging applications that make intensive use of I/O such as autonomous driving, where tasks tightly interact with I/O devices to perceive the surrounding environment. To mention a concrete example, the reference vehicle of the Apollo framework for autonomous driving\textsuperscript{14} makes use of two high-resolution cameras, four Lidars, a Radar, an IMU, a 4G LTE router, a GPS device, and a CAN device.

In such a scenario, memory contention due to I/O devices competes or even overtakes the one due to software tasks running on processor cores. As a result, the timing behavior of safety-critical real-time tasks can be hopelessly compromised, causing, at best, drastic reductions in the employable processing capacity due to the analysis pessimism, or even catastrophic consequences when the delays induced by I/O devices are ignored or not properly considered.

Chip manufacturers are fortunately starting to integrate specific hardware components to regulate the bus traffic produced by the I/O subsystem. This is the case of the Zynq UltraScale\textsuperscript{+} MPSoC by Xilinx, which embeds the ARM CoreLink QoS-400 regulators\textsuperscript{15} to supervise several devices within the chip.

**Contribution.** Taking the Xilinx Ultrascale\textsuperscript{+} MPSoC as a reference platform, this article studies the memory contention generated by I/O devices and investigates on the regulation of the bus traffic generated by such devices by means of QoS-400 regulators. To this purpose, the behavior of QoS-400 regulators is first reviewed and characterized from an analytical perspective. Then, a software infrastructure is proposed to allow controlling QoS-400 regulators from Linux, both at the application and the kernel level. Finally, the results of an extensive experimental evaluation are reported as an experience report with the use of QoS-400 regulators aimed at

- validating their effectiveness in predictably controlling I/O-related memory traffic; and
- assessing the impact of the regulation on the timing performance of software applications and on I/O devices themselves.

The evaluation considers different types of workloads, ranging from synthetic microbenchmarks to state-of-the-art benchmark suites such as San Diego Vision Benchmark Suite (SD-VBS)\textsuperscript{16} and IsolBench\textsuperscript{17}.

**Paper structure.** The remainder of this article is organized as follows. Section 2 reviews the related work. Section 3 discusses the problem of I/O-related memory contention and characterizes the QoS-400 regulators from an analytical perspective. Section 4 proposes a software infrastructure to control QoS-400 regulators from Linux on the Ultrascale\textsuperscript{+} platform. Section 5 presents our extensive experimental evaluation. Section 6 concludes the article.

## 2 RELATED WORK

The papers related with this work are mainly divided into two broad categories: I/O management in embedded systems and mechanisms to control memory contention.

**I/O management in embedded systems.** Some authors targeted the problem of handling I/O operations by proposing and implementing hardware-based custom mechanisms. For example, Pellizzoni and Caccamo\textsuperscript{18,19} introduced the concept of hardware server, that is, a hardware device designed to preserve isolation at the bus level and control the unpredictable behavior of commercial components. Later, hardware servers have been implemented in FPGA\textsuperscript{20} Bak et al.\textsuperscript{21} presented a framework to limit the traffic induced by high-bandwidth peripherals in the I/O bus of a COTS embedded platform. To this end, the authors devised an I/O management system that include real-time bridges and a reservation controller. The system has been prototyped using FPGA technology. Later, Betti et al.\textsuperscript{22} provided an extension of the previous work designing a new bridge with support for device virtualization and timing guarantees.

Jiang and Audsley\textsuperscript{23} proposed GPIOCP (GPIO Command Processor), a timing-accurate I/O controller. The component allows programming I/O operations to occur in the future at precise time instants, and implementing it in FPGA on the Xilinx VC709.\textsuperscript{24} The same authors proposed other mechanisms, still implemented in programmable logic, mainly related with the integration of a predictable I/O with a virtualized system.\textsuperscript{25-27} Zhao et al.\textsuperscript{28} considered the presence of a dedicated I/O co-processing unit and proposed two scheduling methods to achieve predictable and time-accurate I/O operations.

However, all such works are based on custom hardware and none of them considered bus traffic regulators available in COTS platforms, nor explicitly evaluated the effects of I/O-related memory contention in a commercial platform.

**Mechanisms to control memory contention.** Many works have been proposed over the years to predictably control the impact of memory contention on processor cores and hardware accelerators. Notably, Yun et al.\textsuperscript{4} proposed MemGuard,
a memory bandwidth reservation mechanism implemented as a per-core bandwidth regulator. The implementation lever-
ages COTS performance counters to enforce memory budgeting. Floidin et al.\textsuperscript{29} proposed a dynamic budgeting mechanism for upper bounding the delay caused by contention on the memory subsystem, and implemented it in the Fiasco.OC microkernel. Farshchi et al.\textsuperscript{2} presented BRU, a hardware component that performs memory bandwidth regulation for multiple cores collectively. The authors implemented BRU in an FPGA-accelerated full-system simulator and synthesized the design in 7 nm technology. Nowotsch and Paulitsch\textsuperscript{30} proposed mechanisms for enhancing the quality of service when accessing shared resources on a multicore system, implementing them on the Freescale’s P4080. Very recently, Sohal et al.\textsuperscript{31} proposed the Envelope-Aware Predictive model, a framework that allows analyzing the memory demand of applications and making predictions on the timing behavior of workloads running on CPUs and accelerators. Similar mechanisms have been implemented to realize memory access regulation in the context of hardware accelerators.\textsuperscript{3,32-34} Very recently, Serrano-Cases et al.\textsuperscript{35} studied the behavior of the QoS-400 regulators on the Ultrascale+, but without covering I/O traffic and considering a bare-metal use case. Several techniques have also been proposed to improve the predictability and to analyze caches:\textsuperscript{36-38} the survey by Gracioli et al.\textsuperscript{5} presents a good summary of such techniques.

Another interesting survey\textsuperscript{39} summarizes methodsto control memory contention.

Awan et al.\textsuperscript{40,41} proposed methods to analyze mixed-criticality systems under memory bandwidth regulation. Agrawal et al.\textsuperscript{42} analyzed the worst-case execution time and schedulability of real-time tasks under dynamic memory bandwidth regulation.

Still in the context of systems with multiple criticality levels, Hebbache et al.\textsuperscript{43} proposed a dynamic time-division multiplexing approach to reduce contention. Kim et al.\textsuperscript{44} and Yun et al.\textsuperscript{7} proposed bank-aware memory allocation methods to reduce contention in accessing DDR memories.

In the context of virtualization, Wu et al.\textsuperscript{45} proposed a contained lifecycle-aware scheduling algorithm for serverless computing, where each computing request is processed in a container with a specified resource requirement in terms of CPU and memory. Other mechanisms have been proposed by Fan et al.\textsuperscript{46} in the context of cloud computing.

Other authors pursued the goal of achieving a predictable memory-access behavior by proposing memory-aware execution models. Pellizzoni et al.\textsuperscript{10} introduced the PRedictable Execution Model (PREM) where tasks are divided into memory and computational phases. Similar models have been adopted by other authors;\textsuperscript{47,48} for example, Tabish et al.\textsuperscript{49,50} designed and implemented methods to preload a scratchpad memory using a predictable execution model.

Other authors proposed to adopt other contention-free execution paradigms\textsuperscript{51-54} to access memory only at predefined time instants, thus eliminating memory contention due to processing cores.

Despite their high relevance and effectiveness in controlling the memory contention experienced by software tasks, the scheduling strategies devised for these execution models were not designed to take I/O-related memory contention into account.

Finally, another branch of literature focuses on timing analysis in the presence of memory contention. For example, Mancuso et al.\textsuperscript{55,56} presented a method to compute a WCET bound in the presence of percore isolation mechanisms. Dasari et al.\textsuperscript{57} proposed a framework to analyze memory contention for different types of bus arbitration policies, such as fixed priority and time division multiplexing. Other authors considered the contention due to DRAM memory controllers: some relevant contributions are due to Hassan et al.\textsuperscript{58,59} Yun and Pellizzoni,\textsuperscript{60} Kim et al.\textsuperscript{44} and Casini et al.\textsuperscript{61}

Akesson and Goossens\textsuperscript{62} proposed modeling approaches for DDR memory controllers to enable their analysis with techniques used in the field of networking.\textsuperscript{63,64} Kim et al.\textsuperscript{13,65} showed the importance of considering interprocess communication and I/O-related memory contention to perform a sound schedulability analysis and proposed memory allocation techniques to reduce their impact in terms of memory interference. Custom memory controller designs have also been presented.\textsuperscript{66-68}

Overall, to the best of our knowledge, no other work extensively studied the effects of I/O-related memory contention on a COTS platform nor studied the behavior of COTS bus traffic regulators in real-time systems. Indeed, previous work either targeted custom regulators (which may not be available in most commercial platforms) or considered scheduling strategies for task execution models conceived to limit core-related memory contention that neglected I/O devices.

### 3 ADDRESSING I/O-RELATED MEMORY CONTENTION

This section shows how I/O devices may generate contention delays to processing cores and other bus masters, such as hardware accelerators, by taking as a reference a modern and complex heterogeneous platform, namely the Zynq UltraScale+ MPSoC by Xilinx. Subsequently, it studies the QoS-400 regulators by ARM as a mean to mitigate the problem.
### 3.1 I/O-related memory contention on the Zynq UltraScale+ MPSoC

The Zynq UltraScale+ MPSoC by Xilinx is a heterogeneous platform that combines FPGA-based user-programmable logic (PL) with a processing system (PS). The latter includes a quad-core ARM Cortex A53 processor (called APU) and a Cortex R5 dual-core processor (called RPU). The APU is equipped with a two-level cache: the first one is a private level with separate data and instruction memories; the second one is shared among all the cores. The RPU is instead provided with a single private level of cache for each core. The heterogeneity of the MPSoC allows running both software tasks on the processor cores and FPGA-based hardware accelerators on the PL.

The platform has four power domains: the low-power domain (LPD), the full-power domain (FPD), the PL power domain (PLPD), and the battery power domain (BPD). Each power domain can be individually isolated to implement functional isolation and enhance safety and security. Furthermore, it supports different types of I/O devices and peripherals. For example, in our experimental evaluation (Section 5), we considered the DMA devices provided by the low-power and full-power domain (i.e., LPD-DMA and FPD-DMA, respectively).

Figure 1 shows a selection of relevant components of the Ultrascale+ and their interconnections provided in the PS through switches (e.g., the LPD switch in the figure) and using an ARM Advanced Microcontroller Bus Architecture Advanced eXtensible Interface (AMBA AXI) interface. A cache-coherent interconnect (CCI) is also provided to implement coherent cache interconnections. Note that also other components, not shown in Figure 1 for the sake of clarity, are interconnected in the PS, such as the Xilinx Memory Protection Unit (XMPU) and the Xilinx Peripheral Protection Unit (XPPU). Requests from I/O devices can be either routed to memory directly or using the CCI. Multiple AXI Performance Monitors (APMs) are located at different points in the PS interconnect to collect metrics about AXI transactions (yellow boxes in the figure).

Cores, accelerators, and I/O devices can store and retrieve data by accessing a globally-shared off-chip DDR memory. Most importantly, note that several I/O devices include direct memory access (DMA) modules to autonomously access memory, that is, without the intervention of processors during the accesses. This feature is widely used by devices that can generate intensive I/O traffic, such as the Gigabit Ethernet (GEM) module. Accesses to DDR memory are notably known to be one of the most prominent sources of contention on this kind of platforms, especially in the presence of memory-intensive activities that may potentially exacerbate contention delays.

Figure 1 highlights how I/O devices are connected to memory in the UltraScale+ and shows how they can access the same DDR memory used by both processor cores and hardware accelerators implemented in the PL. This evidences that I/O-related memory contention is a potential threat that can affect both performance and timing predictability in the Ultrascale+ MPSoC.
By analyzing in details the internal architecture of the Ultrascale+ we identified an interesting opportunity to study and mitigate this problem: the usage of QoS regulators (pink boxes in the figure) installed in between of I/O devices and the rest of the bus as a mean to predictably control the amount of traffic generated by I/O devices, and the possibility of monitoring this traffic with APMs. Next, we present more details about the ARM CoreLink QoS-400 Regulators provided by the Ultrascale+. Before proceeding, it is however essential to provide some background about the AXI standard.

AMBA AXI. AMBA AXI is a standard for bi-directional simultaneous communication. Data flows in AXI interfaces (or AXI ports), each of them consisting of five channels, four for reading and writing addresses and data (address read, address write, data read, and data write), and a write response channel. Read and write AXI transactions leverage these five channels. In particular, an AXI read transaction toward memory starts with an address request and completes when the DDR controller sends back the data, which becomes available in the data read channel. A write transaction begins with an address request, follows with a data write, and terminates with a write response. An AXI transaction is said to be issued when its AXI address request is issued. AXI provides two different methods to exchange data: single transaction and transaction burst, where in the latter case the requesting device issues a single address request to transfer multiple data items. Each data item is typically referred to as a beat and the number of beats in a burst transaction is typically called burst size.

3.2 ARM coreLink QoS-400 regulators

The ARM coreLink QoS-400 regulator implements functionalities to limit the bus traffic generated by the device(s) connected to it. As shown in Figure 1, multiple regulators are present in the system, allowing to specify different regulations for different devices. In most cases each device is provided with a dedicated regulator. Each regulator can operate in three modes: Transaction rate regulation, Outstanding transaction regulation, and Transaction latency regulation. The first mode is the one of our interest in this work. Its behavior is based on a variant of the Traffic SPECification (TSPEC, RFC 2215), which was originally proposed in the context of networking. In the first operating mode, each QoS-400 regulator \( q_i \) can be configured with three control parameters:

- \( r_i \) (average rate): average allowed transactions per clock cycle;
- \( b_i \) (burstiness allowance): supplementary transactions budget; and
- \( p_i \) (peak rate): maximum allowed transactions per clock cycle, with \( p_i > r_i \).

The QoS-400 regulator also allows specifying two independent values for each of the three parameters \( r_i, b_i, \) and \( p_i \): one for write and one for read address requests, respectively. To avoid complicating the presentation, in the following we consider just one type of transactions (the same considerations apply for the other).

The behavior of the QoS-400 regulator is unfortunately not properly specified in the official documentation provided by ARM. With reference to AXI transactions issued by the device connected to the regulator, in the following we provide a formalization of the behavior of QoS-400 regulators to the best of our understanding of the documentation and based on observing their behavior with experimental tests:

**R1** At the system initialization \((t = 0)\), a status variable denoted with \( b_i(t) \) is set to \( b_i(0) = b_i \).

**R2** Every time an AXI transaction is issued, \( b_i(t) \) is decremented by one.

**R3** Every \( 1/r_i \) clock cycles, if \( b_i(t) < b_i \), then \( b_i(t) \) is incremented by one.

**R4** If an AXI transaction is issued at time \( t \) and \( b_i(t) > 0 \), then the regulator allows the transaction and allows the next one to be issued only after at least \( 1/p_i \) clock cycles;

**R5** If \( b_i(t) = 0 \), no transactions are allowed by the regulator.

The official documentation of the QoS-400 regulator reports a qualitative curve that bounds the number of transactions allowed by the regulator over time: that graph is unfortunately not precise. Given the TSPEC of the regulator, the actual maximum amount of transactions allowed by the regulator in any time window of length \( L \) is given by 1

\[
N_i^T(L) = \min\{1 + p_i \cdot L, b_i + r_i \cdot L\}.
\]
Note that this equation follows as long as the regulator respects the TSPEC guarantees, irrespectively from its implementation.\textsuperscript{69}

Function $N_T^i(L)$ is illustrated in Figure 2.

As it can be observed by the above rules, the device connected to the regulator is capable of issuing transactions at the faster rate $p_i$ as long as variable $b_i(t)$ is positive. The maximum number of consecutive transactions with rate $p_i$ allowed by the regulator is bounded by the intersection of the two lines that define $N_T^i(L)$: the intersection occurs for $L^*$ such that $1 + p_i \cdot L^* = b_i + r_i \cdot L^*$. By solving with respect to $L^*$ we get

$$L^* = \frac{b_i - 1}{p_i - r_i}.$$ 

Hence, the regulator does not allow more than $1 + L^* \cdot p_i$ consecutive transactions at the peak rate. Also, since $p_i > r_i$, note that if $b_i = 1$ then the separation in time of transactions mandated by rule R4 is ineffective: as such, transactions will always be regulated by the recharge of $b_i(t)$ only (rule R3), hence being served with a maximum rate $r_i$. In this case Eq. 1 reduces to $N_T^i(L) = 1 + r_i \cdot L$, which works as an upper bound for a service curve of the form $\left\lceil \frac{L}{1/r_i} \right\rceil$, which is typically adopted in real-time analysis techniques. This hence makes the analysis of the bus traffic generated by I/O devices regulated by the QoS-400 regulator compatible with several state-of-the-art techniques.

It is important to note that the logic implemented by the QoS-400 regulator works on AXI address requests. Since AXI burst transactions can be composed of a different number of beats, the burst size\textsuperscript{*} is a crucial parameter to determine the actual amount of contention produced by the device subject to regulation. For example, if one beat corresponds to one 64-bit word, and if the regulator allows an AXI transaction with burst size equal to 16 beats, then traffic for $16 \cdot 64 = 1024$ bits of data is allowed, while if the allowed transaction has a burst size of 2 beats then traffic of just $2 \cdot 64 = 128$ bits is allowed.

We can therefore convert Equation 1 to consider the number of bytes that are allowed to transit on the bus due to regulator $q_i$ by introducing two additional parameters: the burst size $s_i$ and the beat size $w_i$ of the AXI transactions issued by the device connected to $q_i$. Therefore, the number of bytes $N_T^B(L)$ allowed by $q_i$ in any time window of length $L$ can be simply computed as:

$$N_T^B(L) = N_T^i(L) \cdot s_i \cdot w_i.$$ \hfill (2)

Another important aspect to take into account when evaluating the behavior of the QoS-400 regulator is that it is controlled by the clock of the AXI bus. As such, the length $L$ of the time windows mentioned above has to be intended in AXI clock cycles. To evaluate the impact of the regulation on software systems running on the CPU, it could be useful to express the bound $N_T^B(L)$ as a function of CPU clock cycles, which are easier to measure. If $f_{\text{axi}}$ and $f_{\text{cpu}}$ are the clock frequencies of the AXI bus and the CPU, respectively, a time window of length $L_{\text{cpu}}$ expressed in CPU clock cycles can be simply converted into AXI clock cycles as $L_{\text{axi}} = \left( \frac{f_{\text{axi}}}{f_{\text{cpu}}} \right) \cdot L_{\text{cpu}}$. Paying attention at this conversion is also essential to jointly control the regulation of multiple I/O devices as, for instance, in the Ultrascale+ the AXI clock frequency is not fixed and depends on the considered interconnection. For instance, the QoS-400 regulator that supervises the Full-Power

\textsuperscript{*}Note that the burst size is not directly related to the burstiness allowance $b_i$, although the names adopted by the AXI standard and the QoS-400 reference manual are similar.
Domain DMA (FPD-DMA) works with a clock frequency $f_{\text{clk}} = 528$ MHz, the one that supervises the Low-Power Domain DMA (LPD-DMA) works with $f_{\text{clk}} = 495$ MHz, and the one that supervises the GEM Ethernet works with $f_{\text{clk}} = 247.5$ MHz.

This time conversion can be applied to Equations (1) or (2), to allow them to be used to account for the I/O-related memory traffic in a memory-aware response-time analysis for real-time tasks and/or hardware accelerators, for example References 36, 59, 61, 70, even without knowing the traffic patterns generated by the I/O devices, which may often be hard to predict.

4 | SW INFRASTRUCTURE TO CONTROL QOS-400 REGULATORS ON ULTRASCALE+

On the Ultrascale+ MPSoC, the ARM QoS-400 regulators are memory-mapped devices. Each QoS-400 regulator offers (among others) the following registers to control the regulation parameters introduced in the previous section:

- $\text{qos\_cntl}$ (bits 1–0): bit 0 to enable write regulation, and bit 1 to enable read regulation;
- $\text{aw\_p}$ (bits 31–24): write peak rate, as a 8-bit fraction of the number of transactions per cycle;
- $\text{aw\_b}$ (bits 15–0): integer value denoting the burstiness allowance for writes;
- $\text{aw\_r}$ (bits 31–20): write average rate, as a 12-bit fraction of the number of transactions per cycle;
- $\text{ar\_p}$, $\text{ar\_b}$, $\text{ar\_r}$: used to regulate peak rate, burstiness, and average rate for reads, with the same bit intervals as in the case of writes.

To distinguish the specific regulator, the aforementioned registers are provided with a prefix: for example, the GEM Ethernet QoS regulator uses the $\text{gem3M\_intiou\_}$ prefix (i.e., the corresponding enable register is labeled as $\text{gem3M\_intiou\_qos\_cntl}$ in the headers provided by Xilinx).

The Ultrascale+ MPSoC supports Linux distributions generated by Petalinux, a suite of tools provided by Xilinx to build and customize Linux distributions to properly run on Xilinx platforms. Unfortunately, we found that by using Petalinux and the standard software stack provided by Xilinx it is not possible to access the QoS-400 regulators. By investigating this issue it emerged that the regulators are reserved as privileged resources that are not exposed to the APU of the Ultrascale+, hence preventing to be used from either the Linux kernel or by a resource manager running upon Linux.

To overcome this issue, a suitable software infrastructure has been designed and implemented.

The Ultrascale+ is based on the Armv8-A processor architecture, which implements four distinct privilege levels for the software execution, called Exception Levels (EL). Each EL, from EL3 to EL0, is usually targeted for a specific software component, that is: EL3 for secure platform monitors; EL2 for hypervisors; EL1 for operating systems; and EL0 for user applications. The Linux kernel is conceived to run in EL1 (and optionally EL2), assuming the Arm Trusted Firmware (ATF) is also running at EL3. ATF is a reference implementation to securely handle processor-related low-level features, such as power management, and orchestrating the two processor worlds powered by the Arm TrustZone technology, namely normal-and secure-world (not relevant to be discussed here).

ATF is a project officially maintained by Arm, although a lot of contributions are continuously provided by chip vendors to support low-level features of their platforms. In the case of the Ultrascale+, Xilinx introduced a platform-dependent module to abstract a number of low-level platform features through Secure Monitor Calls (SMCs).

The proposed solution is illustrated in Figure 3. To allow the access to Qos-400 regulators from Linux, it leverages both ATF and the Platform Management Unit (PMU), a co-processor available on the SoC managing some key aspects of the lifecycle of the platform. On the Ultrascale+, the PMU is a dedicated user-programmable Microblaze processor that handles the board initialization, power management, and error handling. The PMU firmware provides an API that is exposed to the other processing units, such as APUs and RPs, to perform general platform management operations. This API can be accessed by means of inter-processor interrupts (IPI) and messages sent through dedicated buffers shared between the PMU and the requesting unit.

Most interesting to us, being a highly-privileged component of the platform, the PMU offers the possibility to perform, on behalf of another component, a read or write operation to the memory-mapped registers of any device available in the system. The functions that implement these operations in the PMU API are called $\text{mmio\_read}$ and $\text{mmio\_write}$, and can hence be used to access the memory-mapped registers of QoS-400 regulators on behalf of the APU. The behavior of
**FIGURE 3** Software infrastructure realized for managing QoS regulation from user-space Linux programs

`mmio_read` and `mmio_write` is controlled by an access table stored within the PMU firmware. The access table defines, for each processing unit, specific read and write permissions for a set of memory regions.

In the proposed solution, the application software running above Linux on the APU accesses the QoS-400 regulators by means of the PMUs. However, as the APU can interact with the PMU only at EL3 where ATF resides, while user-space applications run at EL0, we leverage ATF to mediate the access to the QoS regulation registers on behalf of the application. Luckily, the ATF module provided by Xilinx already offers a wrapper to call the PMU API via SMCs. Nevertheless, note that it is not possible to directly interact with ATF by EL0, since SMCs are privileged instructions executable at EL1 or EL2 only. To overcome this issue, we developed a Linux kernel module to bridge access requests to the QoS-400 regulators from user space to ATF. The module exposes a character device that can be easily accessed by software running at EL0. In this way, both user-space programs or resource managers running at EL0 can require to set new regulation values by interacting with this device. The module also exposes a direct interface to control the regulation from a kernel module, for example, useful to implement scheduling algorithms in the Linux kernel that also control the regulators.

In addition, the PMU firmware has been modified by customizing the access table to allow the APU to access the memory areas of the memory-mapped registers of the QoS regulators connected to the I/O devices of interest.

Note that this solution to interact with ATF from EL0 is not limited to I/O regulators and may also be used to overcome privilege-related issues in ARM-based platforms.

The character device exposed by the realized kernel module can be accessed with the standard Linux API, as shown in Listing 1.

```c
int fd = open("/dev/qos_regulator", O_RDWR);
write(fd, command, strlen(command) + 1);
close(fd)
```

Listing 1: Using the regulator from user-space Linux programs. [Correction added on 10 March 2022, after first online publication: The unconverted LaTeX codes in Listing 1 have been corrected in this version.]

The textual command (variable `command` in Listing 1) provided to the device has the following interface:

- `qos-[device-name]-read-enable`: enables read regulation for a specific device with identifier `device-name`; for example, the command `qos-gem3-read-enable` enables the read regulation for the GEM ethernet device (identified by `gem3`).
- `qos-[device-name]-read-r:[value]`: sets the value of the regulation register `ar_r` to `[value]` for device `device-name` (average-rate regulation); for example, `qos-gem3-read-r:0x20`.
- `qos-[device-name]-read-b:[value]`: sets the value of the regulation register `ar_b` to `[value]` for device `device-name` (burstiness allowance); for example, `qos-gem3-read-b:0x20`. 

![Diagram of software infrastructure for QoS regulation](image-url)
FIGURE 4  Distribution of time required (in $\mu s$) for modifying the average rate register of the QoS-400 regulator via SMC call

$qos$-[device-name]-read-p:[value]: sets the value of the regulation register $ar_p$ to [value] for device [device-name] (peak-rate regulation);
for example, qos-gem3-write-p:0x20.

The textual commands qos-[device-name]-write-enable, qos-[device-name]-write-r:[value], qos-[device-name]-write-b:[value], qos-[device-name]-write-p:[value] implement the same behavior as the previous register, but for write regulation.

In addition, a user-space library has been developed to encapsulate the interaction with the character device and simplify the user interface.

Applicability. The Linux-based interface proposed in this article offers a flexible way to manage the QoS-400 regulator provided by the Ultrascale+ MPSoC. Indeed, it allows both a static setup of the regulators, occurring at the system startup, or a more complex dynamic one where the parameters assigned to each regulator may change at runtime. We believe static regulation to be more common for most real-time systems; in all these cases, the overheads introduced by our implementation are negligible, as the regulation parameters are specified only once, and then they remain constant during execution. If the regulation values are modified at runtime, the overhead introduced by the proposed approach mainly consists of the time required by the SMC to execute and update the memory-mapped registers through the PMU. To empirically quantify such an overhead, we performed an experiment on the Zynq Ultrascale+ MPSoC, where the user-space library function corresponding to the qos-[device-name]-write-r:[value] has been called 10,000 times. The histogram reporting the distribution of the measured delays is illustrated in Figure 4. The observed time required to change the regulation value ranges from 15 to 40 $\mu s$, with most samples lying in the interval [15, 17] $\mu s$. Thus, the measured overhead is also broadly compatible with most online scheduling activities that would generally occur with a higher time grain (e.g., tents of milliseconds).

5 | EXPERIMENTAL EVALUATION

This section reports on an extensive experimental evaluation we performed to empirically study the effectiveness of QoS-400 regulators in predictably controlling I/O-related memory contention and assess their impact on the timing performance of software applications and I/O devices. To this end, we studied different system configurations and different workloads.

Experimental setting. We considered two scenarios: (i) the case in which QoS regulators are directly controlled by a bare-metal application (i.e., without operating systems) to conduct validation experiments in a minimal setting, and (ii) the case in which regulators are accessed from Linux using the software infrastructure of Section 4.

In the latter case, we adopted an experimental setup (shown in Figure 5) where the Clare hypervisor, a type-1 real-time hypervisor developed in our laboratory, is serving the execution of two virtual machines (VMs), a firmware VM (i.e., bare-metal), and a Linux VM (based on a distribution produced with Petalinux 2020.1). The Linux VM executes benchmark applications and provides support for QoS regulation by means of the Linux kernel module presented in Section 4. The firmware VM handles I/O devices and profiles their behavior. In particular, it also performs the initial
configuration for the I/O devices and APMs, and manages the corresponding interrupt service routines. This allows taking more precise measurements related to I/O devices without the effect of the complex infrastructure of Linux device drivers.

The experiments have been performed on a ZCU102 board equipped with a Zynq UltraScale+ MPSoC by Xilinx and an MTA8ATF512 16-bank DDR memory. The VMs run on the cores of the APU only. Configurations with both caches turned on and off have been considered. We studied the traffic generated by three devices: the GEM Ethernet device and, as representative of the behavior of other I/O devices capable of generating intense bus traffic, the DMAs of the low-power and full-power domains (LPD-DMA and FPD-DMA) of the Zynq UltraScale+.

In all the reported experiments, we focus on the parameter \( r_i \) of the QoS-400 regulators, which is responsible for the long-run behavior provided by the regulators when I/O devices continuously issue transactions. In the following, this parameter is also referred to as the regulation control. The parameters burstiness allowance \( b_i \) and the peak rate \( p_i \) have been disabled.

We extensively evaluated the impact of QoS-400 regulators with different types of workloads, ranging from synthetic microbenchmarks (one consisting of a single memory access and one accessing a vector) to state-of-the-art benchmarks suites such as San Diego Vision Benchmark Suite (SD-VBS)\(^\text{16} \) and IsolBench.\(^\text{17} \) All of them have been used to quantify the capability of QoS-400 regulators in controlling contention delays and hence the running time of the benchmarks.

The IsolBench suite consists of two benchmarks, Bandwidth and Latency. Both are memory-intensive programs: the first one accesses memory to maximize the memory traffic, while the second one iterates through a linked list. SD-VBS consists of real-world applications based on computer vision. Table 1 summarizes the content of SD-VBS.

### 5.1 Preliminary analysis and validation

We start by studying the behavior of QoS-400 regulators from a bare-metal setup where the GEM is configured to work in loop-back mode. We measured with the APMs the traffic generated by each of the three devices with no regulation,

| Name      | Description                          |
|-----------|--------------------------------------|
| disparity | Depth data using dense stereo        |
| tracking  | Motion extraction from images        |
| pca       | Principal component analysis         |
| sift      | Feature transformation               |
| localization | Detect current location             |
| svm       | Supervised learning                  |
| mser      | Blob detection                       |
| stitch    | Image overlapping                    |
| texture synth | Texture synthesis from images     |
TABLE 2 Memory traffic generated by the GEM ethernet, LPD-DMA, and FPD-DMA without traffic regulation

|                | Write traffic | Read traffic |
|----------------|---------------|--------------|
| GEM            | 125 MB/s      | 295 MB/s     |
| LPD-DMA        | 3.5 GB/s      | 3.5 GB/s     |
| FPD-DMA        | 5 GB/s        | 5 GB/s       |

FIGURE 6 Observed memory bandwidth regulation versus theoretical bound for different devices

instructing each of them to continuously move large portions of data (1,000,000 bytes in the case of the DMAs, and 80 packets of 10,240 bytes for the GEM Ethernet). Table 2 reports these measurements. The FPD-DMA generates the highest amount of traffic, while the GEM Ethernet generates a considerably lower amount of traffic, with respect to both the DMA devices. Furthermore, we observe that the generated traffic is the same for reads and writes for the DMAs, while GEM generates a lower number of writes.

Validation. Figure 6 shows the observed behavior in output from the regulators for the three devices (specified in the caption above each chart) as a function of the maximum number of AXI transactions allowed by the regulation value stored in the aw_r and ar_r registers (parameter r_i expressed in transactions per seconds). In all the reported experiments, the regulation value used for reads is the same regulation value used for writes. These experiments are meant to match the analytical model presented in Section 3 with the measurements obtained from the platform. To this purpose, the plots also report the theoretical bound that is expected to be guaranteed by the regulators.

When the regulation is permissive enough, the three I/O devices are allowed to perform reads and writes without being limited by the regulator. This occurs for regulations corresponding to about $2 \times 10^7$ transactions/second for the FPD-DMA, and $3 \times 10^7$ transactions/second for the LPD-DMA. It is worth observing that, although the FPD-DMA generates a higher amount of traffic than the LPD-DMA (see Table 2), the regulation takes effect at higher values for the LPD-DMA. This is because the two devices are characterized by a different beat size, equal to 128 and 64 bits for the FPD and LPD DMAs, respectively. Figure 6C shows the effects of the regulation on the GEM Ethernet. Since the Ethernet is capable to produce an asymmetric maximum memory traffic, the regulation takes effect at different values for reads and writes.

Note that, in all the tested cases, the QoS-400 regulators are capable to predictably enforce a regulation of the memory traffic that is bounded by the theoretical curve introduced by Equation 1 in Section 3, hence proving to be suitable for being used in real-time systems.

Figure 7 reports on the read and write speed, in MB/s, achieved with different values of the regulation control as provided in registers aw_r and ar_r. In all the tested cases, for values above $0 \times 100$, the regulation does not take effect, allowing to reach the same speed observed in Table 2. For lower regulation values, the speed is reduced due to the effect of the regulation employed by the QoS-400 as expected.

The transaction rate has been obtained from the value specified in the aw_r and ar_r registers (which store the least significant bits of a 12-bit fractional part of a number that expresses the amount of allowed transactions per clock cycle, as discussed in Section 4) by multiplying it by the clock frequency.
Figures 8 and 9 show how the observed execution time varies for the single-access and vector microbenchmarks, with caches disabled, as a function of the regulation of the device reported in the caption of each chart. As it can be noted from the figures, the regulation has a clear effect on both microbenchmarks but also a different impact on the two benchmarks. For instance, note that, with the exception of the first two regulation control values (from 100 to 80), which determine a 59% reduction of the longest-observed running time, single-access is limitedly impacted by the regulation of the GEM (Figure 8C). Conversely, vector is much more sensitive to the regulation of the same device but exhibits variations of a small order of magnitude only (about 5% of difference in the running time). As emerged in previous works focused on the analysis of memory contention, these results confirm the need for analytical methods capable of coping with both single and multiple memory accesses as a whole also for I/O-related memory traffic.
5.2 Effects of regulation of I/O devices on benchmarks

Next, we discuss the results of the extensive evaluation we performed on two benchmark suites: SD-VBS and IsolBench. In all the configurations presented next, the benchmarks have been executed on the Linux VM and caches have been turned on.

**San Diego Vision Benchmarks** Figures 10 and 11 present the results of the evaluation on the SD-VBS. In this experiment, for each of the tasks in the suite, one of them executes while the three devices under consideration (i.e., GEM, FPD-DMA and LPD-DMA) are continuously moving large amounts of data. We collected the longest-observed running time for each task in the benchmark while varying the regulation control for each device among 5 specific values (OFF, $0 \times 100$, $0 \times 80$, $0 \times 40$, and $0 \times 1$). These values have been chosen leveraging the results on the traffic generated by the devices presented in the previous subsection.

Each experiment has been executed ten times, collecting the maximum observed running time of each task. In the collected measurements, we noted an extremely low variance between the maximum and minimum values, which is generally below 0.2%. For each execution, $5^3 = 125$ different measurements have been collected, corresponding to the 5

![Figure 10](image1.png)  
**Figure 10** Maximum observed execution times (in milliseconds) for the pca, tracking, and disparity tasks from SD-VBS suite as a function of the value in the QoS average-rate regulation register of the devices reported in the caption above each chart. Configurations denoted as OFF considers the QoS regulation of the corresponding device turned off, that is, the device is free to perform AXI transactions

![Figure 11](image2.png)  
**Figure 11** Maximum observed execution times for the disparity, mser, sift, and localization tasks from SD-VBS suite (in milliseconds) as a function of the value in the QoS average-rate regulation registers of two devices reported on the axis of each chart.
settings of the regulation control for each of the 3 devices. The collection of these measurements required more than five days of runs.

Figure 10 shows how the longest-observed execution time varies when setting different values in the regulation control registers of the QoS-400. In Figure 10A,C, the pca and disparity tasks have been executed while varying the regulation control for the LPD-DMA. Three curves are reported, corresponding to three different regulations for the FPD-DMA (denoted as other DMA in the legend, that is, OTH-DMA). The effect of the regulation of the LPD-DMA is quantified in about 2.8% for pca and 6.5% for disparity, for the cases in which the FPD-DMA regulation is set to 0x40. If we consider the joint action of the FPD-DMA and the LPD-DMA under regulation, the reduction of the execution time reaches 3.4% and 9.3%, respectively. Figure 10B shows a similar experiment that considers the tracking task. In this case the regulation of the FPD-DMA is reported on the x-axis and the three lines represent different configurations of the LPD-DMA, which are reported in the legend.

Figure 11 reports 3D plots showing the maximum execution time as a function of the regulation of two devices. Figure 11A considers the maximum execution times collected for the disparity task with respect to the regulations of the GEM and the FPD-DMA. It is worth observing that the results are much more sensitive to the regulation control of the FPD-DMA, since the maximum traffic emitted by the device, when not regulated, is much higher itself. Figure 11B considers instead the mser task with respect to the two DMAs. Given the characteristic of the task, which is quite sensitive to the memory bandwidth reduction, the regulation can provide up to a 26% reduction of the running time when the regulation control of the two devices is set to the strictest value. Figure11C,D shows similar plots for the sift and localization tasks. While the results for sift indicates a similar trend as the one of mser, interestingly, Figure11D shows that the localization task exhibits a limited reduction of the running time for regulation values below 0x80.

**IsolBench.** Figures 12 and 13 report some representative configurations obtained from the execution of the Isolbench benchmark suite. As discussed at the beginning of the section, the suite includes two tasks, called bandwidth and latency. Figure 12 targets the bandwidth task and reports measurements of both the longest-observed running time and the value of the available memory bandwidth, which is computed by the benchmark by dividing the amount of read bytes by the time needed to complete the operation. Figure 13 targets instead the latency task. In this case, the memory latency and the longest-observed running time have been collected. The memory latency is obtained by Isolbench by computing the average time elapsed between the beginning and the end of several read operations. It is worth highlighting that all the accesses performed by Isolbench are carried out so as to avoid consecutive reads in the same cache line. This way, the benchmark guarantees each instruction to produce a cache miss and access the main memory. The experiments have been reported as a function of the configuration of the regulators for the considered devices.

Figure 12A,B shows an increase of the available memory bandwidth when reducing the allowed transaction rate for the FPD-DMA and the GEM Ethernet, respectively of 162% and 6% if considering the strictest regulation. Figure 12C shows the positive effect given by the regulation of the LPD-DMA, with respect to the execution time of the Bandwidth task, which decreases by up to 48%.

Similarly, Figure 13A,B present the observed reduction of the memory access latency when controlling the transactions of the LPD-DMA (reduction up to 37%) and the FPD-DMA (reduction up to 54%).

**FIGURE 12** Memory bandwidths (in MB/s) and maximum observed execution times (in milliseconds) for the bandwidth task from the Isolbench suite as a function of the value in the QoS average-rate regulation register of the device reported on the caption above each chart. The regulation of the other devices is set to $0 \times 01$. 

(A) FPD-DMA  
(B) GEM  
(C) LPD-DMA
FIGURE 13 Task latencies (in nanoseconds) and maximum observed execution times (in milliseconds) for the bandwidth task from the Isolbench suite as a function of the value in the QoS average-rate regulation register of the device reported in the caption above each chart. The regulation of the other devices is set to 0×01.

Finally, Figure 13C reports a 1.5% reduction of the execution time of the Latency task, with respect to the regulation applied to the Ethernet device.

On the whole, the collected data corroborates how the reduction of the memory bandwidth of the I/O devices effectively improves the memory bandwidth of the CPU, thus improving the execution time of the tasks running on the processor cores.

5.3 Effects of QoS-400 regulators on I/O devices

The results presented above showed the impact of the memory traffic generated by the I/O devices on the tasks running on processor cores. However, another interesting behavior to discuss is the effect of the QoS-400 regulators on the devices themselves.

To this end, we defined a specific performance metric: the time elapsed between two consecutive interrupts generated by each device, called transfer time.

The DMAs are configured to generate an interrupt every time a transfer is completed, while the GEM generates an interrupt every time the transmission of a packet is completed. As both operations involve memory accesses, the more a device is subject to regulation of its bus traffic the more the operations it performs can be delayed. As such, the interarrival time between consecutive interrupts can in turn be increased. Each device has been configured to start a new operation immediately after the previous one is concluded. As a consequence, by measuring the time interval between two consecutive interrupts we also estimate the duration of a complete transfer. For this reason, this value will be hereinafter referred to as transfer time. Figures 14 and 15 show how this quantity is affected by the average-rate regulation control offered by the QoS-400 for the three devices under analysis. More specifically, Figure 14 reports on the effect of the regulation of two of the devices on the transfer time of the third one, which is specified in the caption above the plot. For each curve of each chart, the regulation control of the device which is not involved in the measurement is set to 0×1, while the one of the device reported in the caption above the graph is turned off.

FIGURE 14 Effects of QoS regulation on the transfer time (in clock cycles) of other I/O devices
Figure 14 shows that when restricting the bandwidth of the other devices, the transfer time decreases, illustrating how also the peripherals are negatively influenced by the interference in the memory subsystem.

Finally, Figure 15 shows how every device is influenced by its own regulator. The regulation control of the other two devices not shown in the plot has been set to $0 \times 1$. In this scenario, the increase of the transfer time when the regulation gets stricter is significantly high, growing by 23% for the GEM, by 14776% for the FPD-DMA, and 22,254% for the LPD-DMA. This observation remarks the importance of finding a good trade-off for configuring the regulators to provide the best compromise between the advantage given to other tasks and I/O devices, and the performance degradation experienced by the regulated device.

6 CONCLUSIONS AND FUTURE WORK

This work studied the impact of I/O-related memory contention on a COTS embedded platform, namely the Zynq Ultrascale+ by Xilinx, and its regulation by means of Arm QoS-400 regulators provided by the platform. A software infrastructure to control QoS-400 regulators from Linux on the Ultrascale+ has been presented, overcoming several issues that prevented a direct access to the control interface of the regulators. The behavior of QoS-400 regulators has been analytically characterized and experimentally validated against theoretical bounds, proving that they are effective in predictably controlling I/O-related memory contention. Thanks to an extensive experimental evaluation, effects of the regulation of I/O-related memory contention have been investigated on longest-observed running times of software tasks, available memory bandwidth, memory access latency, and transfer times of I/O devices. The usage of different configurations of the regulators for I/O devices showed a variation up to 162% in the maximum observed execution time of tasks from benchmarks, and up to 22,254% in I/O device transfer times, remarking the relevance of I/O-related memory contention as well as the importance and effectiveness of the employed regulators.

A popular regulator and a representative heterogeneous platform have been considered to discuss a way more general problem for modern real-time systems: the mitigation of I/O-related memory contention. Although modern systems are more and more interconnected by means of an increasing number of I/O devices, for example, as it is the case for autonomous vehicles, the impact of I/O-related memory contention was often underestimated by researchers.

Furthermore, many other QoS regulators may benefit from the study of this article: indeed, as noted in Section 3, it follows the TSPEC specification, which other regulators may also implement.

This work constitutes a building block for future studies on I/O-related memory contention and can enable more general research based on sound models designed according to our experimental observations. For example, the analytical characterization of QoS-400 regulators validated in this work enables the derivation of memory-contention analysis techniques that also include I/O-related contention. The proposed software infrastructure may be used to devise design optimization strategies or even adaptive regulation algorithms to simultaneously optimize the timing constraints of real-time tasks and the performance requirements of I/O devices.

In this article, to highlight the relevance of I/O-related memory contention, we studied the performance of I/O devices and tasks independently. However, the performance of I/O and tasks may be, in some cases, deeply connected, as in the case of tasks requesting I/O data, where the availability of new I/O data that may be penalized by a too strict regulation of the corresponding device, thus, in turn, penalizing the task itself and reducing its benefit occurring from a lower I/O
contention. Defining this link is a flourishing research ground for future work. Furthermore, future work should also study the execution time variability in the presence of QoS regulation of I/O devices.

Finally, this study can enable the implementation of stronger isolation mechanisms for virtualized systems.72

ACKNOWLEDGMENTS
This work has been partially supported by Huawei and the Italian Ministry of University and Research (MIUR), under the SPHERE project funded within the PRIN-2017 framework (grant no. 93008800505).

DATA AVAILABILITY STATEMENT
Data available on request from the authors.

ORCID
Matteo Zini https://orcid.org/0000-0003-2750-7437
Daniel Casini https://orcid.org/0000-0003-4719-3631

REFERENCES
1. (CAST) CAST. position paper CAST-32A: multi-core processors; 2016.
2. Farshchi F, Huang Q, Yun H. BRU: bandwidth regulation unit for real-time multicore processors. Proceedings of the 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS); 2020.
3. Restuccia F, Biondi A, Marinoni M, Cicero G, Buttazzo G. AXI HyperConnect: a predictable, hypervisor-level AXI interconnect for hardware accelerators in FPGA SoC. Proceedings of the 57th ACM/ESDA/IEEE Design Automation Conference (DAC 2020); 2020; San Francisco, CA.
4. Yun H, Yao G, Pellizzoni R, Caccamo M, Sha L. MemGuard: memory bandwidth reservation system for efficient performance isolation in multi-core platforms. Proceedings of the 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS); 2013:55-64.
5. Gracioli G, Alhammad A, Mancuso R, Fröhlich AA, Pellizzoni R. A survey on cache management mechanisms for real-time embedded systems; 2015.
6. Kloda T, Solieri M, Mancuso R, Capodieci N, Valente P, Bertogna M. Deterministic memory hierarchy and virtualization for modern multi-core embedded systems. Proceedings of the 2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS); 2019:1-14.
7. Modica P, Biondi A, Buttazzo G, Patel A. Supporting temporal and spatial isolation in a hypervisor for ARM multicores platforms. Proceedings of the 2018 IEEE International Conference on Industrial Technology (ICIT); 2018: 1651-1657.
8. Suzuki N, Kim H, de Niz D, et al. Coordinated bank and cache coloring for temporal protection of memory accesses. Proceedings of the 2013 IEEE 16th International Conference on Computational Science and Engineering; 2013:685-692.
9. Yun H, Mancuso R, Wu Z, Pellizzoni R. PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms. Proceedings of the 2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS); 2014:155-166.
10. Pellizzoni R, Betti E & Bak S et al. A predictable execution model for cots-based embedded systems. Proceedings of the 2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium; 2011.
11. Durrieu G, Faugere M, Girbal S, Perez DG, Pagetti C & Puffitsch W Predictable flight management system implementation on a multicore processor. Embedded Real Time Software (ERTS’14); 2014.
12. Casini D, Pazzaglia P, Biondi A, Di Natale M & Buttazzo G Predictable memory-CPU co-scheduling with support for latency-sensitive tasks. Proceedings of the 57th Design Automation Conference (DAC); 2020.
13. Kim N, Tang S, Otterness N, Anderson JH, Smith FD, Porter DE. Supporting I/O and IPC via fine-grained OS isolation for mixed-criticality real-time tasks. Real-Time Syst. 2020;56(4):349-390.
14. Baidu Apollo. https://github.com/ApolloAuto/apollo
15. ARM. ARM CoreLink QoS-400 network interconnect advanced quality of service - supplement to ARM CoreLink NIC-400 network interconnect technical reference manual; 2016.
16. Venkata SK, Ahn I, Jeon D, et al. SD-VBS: the san diego vision benchmark suite. Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC); 2009:55-64.
17. Valsan PK, Yun H, Farshchi F. Taming non-blocking caches to improve isolation in multicore real-time systems. Proceedings of the 2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS); 2016:1-12.
18. Pellizzoni R, Caccamo M. Toward the predictable integration of real-time COTS based systems. Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS 2007); 2007:73-82.
19. Pellizzoni R, Caccamo M. Impact of peripheral-processor interference on WCET analysis of real-time embedded systems. IEEE Trans Comput. 2010;59(3):400-415.
20. Pellizzoni R, Bui BD, Caccamo M, Sha L. Coscheduling of CPU and I/O transactions in COTS-based embedded systems. Proceedings of the 2008 Real-Time Systems Symposium; 2008:221-231.
21. Bak S, Betti E, Pellizzoni R, Caccamo M, Sha L. Real-time control of I/O COTS peripherals for embedded systems. Proceedings of the 2009 30th IEEE Real-Time Systems Symposium; 2009:193-203.

22. Betti E, Bak S, Pellizzoni R, Caccamo M, Sha L. Real-time I/O management system with COTS peripherals. IEEE Trans Comput. 2013;62(1):45-58.

23. Jiang Z, Audsley NC. GIOCP: timing-accurate general purpose I/O controller for many-core real-time systems. Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE), 2017; 2017:806-811.

24. Xilinx VC709. https://www.xilinx.com/products/boards-and-kits/dk-v7-vc709-g.html

25. Jiang Z, Audsley N. VCDC: the virtualized complicated device controller. Proceedings of the 29th Euromicro Conference on Real-Time Systems (ECRTS 2017); 2017.

26. Jiang Z, Audsley NC, Dong P. BlueVisor: a scalable real-time hardware hypervisor for many-core embedded systems. Proceedings of the 2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS); 2018.

27. Jiang Z, Audsley N, Dong P. BlueIo: a scalable real-time hardware I/O virtualization system for many-core embedded systems. ACM Trans Embed Comput Syst. 2019;18(3):1–25.

28. Zhao S, Jiang Z, Dai X, Bate I, Habli I, Chang W. Timing-accurate general-purpose I/O for multi- and many-core systems: scheduling and hardware support. Proceedings of the 2020 57th ACM/IEEE Design Automation Conference (DAC); 2020:1-6.

29. Flodin J, Lampka K, Yi W. Dynamic budgeting for settling DRAM contention of co-running hard and soft real-time tasks. Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014); 2014:151-159.

30. Nowotsch J, Paulitsch M. Quality of service capabilities for hard real-time applications on multi-core processors. Proceedings of the 21st International Conference on Real-Time Networks and Systems (RTNS ’13); 2013:151-160.

31. Sohal P, Tabish R, Drepper U, Mancuso R. E-WarP: a system-wide framework for memory bandwidth profiling and management. Proceedings of the 2020 IEEE Real-Time Systems Symposium (RTSS); 2020:345-357.

32. Aghilinasab H, Ali W, Yun H, Pellizzoni R. Dynamic memory bandwidth allocation for real-time GPU-based SoC platforms. IEEE Trans Comput-Aided Des Integr Circuits Syst. 2020;39(11):3348-3360.

33. Capodieci N, Cavichiolri R, Valente P, Bertogna M. SiGAMMA: server based integrated GPU arbitration mechanism for memory accesses. Proceedings of the 21st International Conference on Real-Time Networks and Systems (RTNS ’17); 2017:48-57.

34. Pagani M, Rossi E, Biondi A, Marinoni M, Lipari G, Buttazzo G. A bandwidth reservation mechanism for AXI-based hardware accelerators on FPGAs. Proceedings of the 31st Euromicro Conference on Real-Time Systems (ECRTS 2019); 2019.

35. Serrano-Cases A, Reina JM, Abella J, Mezzetti E, Cazorla FJ. Leveraging hardware QoS to control contention in the Xilinx Zynq UltraScale+ MPSoC. Proceedings of the 33rd Euromicro Conference on Real-Time Systems (ECRTS 2021); 2021.

36. Davis RI, Altmeyer S, Indrusiak LS, Maiza C, Rihani H, Rivas JM, Goossens J, Altmejer M, Davis RI. A survey of timing verification techniques for multi-core real-time systems. ACM Comput-Aided Des Integr Circuits Syst. 2020;56(2):124-170.

37. Xiao J, Altmeyer S, Pimentel AD. Schedulability analysis of global scheduling for multicore systems with shared caches. IEEE Trans Comput. 2020;69(10):1487-1499.

38. Tessler C, Fisher N. BUNDLE: real-time multi-threaded scheduling to reduce cache contention. Proceedings of the 2016 IEEE Real-Time Systems Symposium (RTSS); 2016:279-290.

39. Maiza C, Rihani H, Rivas JM, Goossens J, Altmeyer S, Davis RI. A survey of timing verification techniques for multi-core real-time systems. ACM Comput Surv (CSUR). 2019;52(3):1-38.

40. Awan MA, Souto PF, Bletkas S, Akesson B, Tovar E. Mixed-criticality scheduling with memory bandwidth regulation. Proceedings of the 2018 Design, Automation Test in Europe Conference Exhibition (DATE); 2018:1277-1282.

41. Awan MA, Souto PF, Bletkas S, Akesson B, Tovar E. Memory bandwidth regulation for multiframe task sets. Proceedings of the 2019 IEEE 25th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA); 2019:1-11.

42. Agrawal A, Mancuso R, Pellizzoni R, Fohler G. Analysis of dynamic memory bandwidth regulation in multi-core real-time systems. Proceedings of the 2018 IEEE Real-Time Systems Symposium (RTSS); 2018:230-241.

43. Hebbache F, Brandner F, Jan M, Pautet L. Work-conserving dynamic time-division multiplexing for multi-criticality systems. Real-Time Syst. 2020;56(2):124-170.

44. Kim H, de Niz D, Andersson B, Klein M, Mutlu O, Rajkumar R. Bounding memory interference delay in COTS-based multi-core systems. Proceedings of the 2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS); 2014.

45. Wu S, Tao Z, Fan H, et al. Container lifecycle-aware scheduling for serverless computing. Softw Pract Exper. 2021.

46. Rouxel B, Skalrist S, Derrien S, Puaut I. Tightening contention delays while scheduling parallel applications on multi-core architectures. ACM Trans Embed Comput Syst. 2017;16:1–20.

47. Rouxel B, Skalrist S, Derrien S, Puaut I. Hiding communication delays in contention-free execution for SPM-based multi-core architectures. Proceedings of the 31st Euromicro Conference on Real-Time Systems (ECRTS 2019); 2019.

48. Tabish R, Mancuso R, Wasly S, et al. A real-time scratchpad-centric OS for multi-core embedded systems. Proceedings of the 2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS); 2016.

49. Tabish R, Mancuso R, Wasly S, Pellizzoni R, Caccamo M. A real-time scratchpad-centric OS with predictable inter/intra-core communication for multi-core embedded systems. Real-Time Syst. 2019;55(4):850–888.

50. Becker M, Dasari D, Nicolic B, Akesson B, Nélias V, Nolte T. Contention-free execution of automotive applications on a clustered many-core platform. Proceedings of the 2016 28th Euromicro Conference on Real-Time Systems (ECRTS); 2016.
52. Bonioli F, Cassé H, Noulard E, Pagetti C. Deterministic execution model on cots hardware. Proceedings of the International Conference on Architecture of Computing Systems; 2012:98-110.
53. Biondi A, Natale MD. Achieving predictable multicore execution of automotive applications using the let paradigm. Proceedings of the 2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS); 2018.
54. Pazzaglia P, Casini D, Biondi A, Di Natale M. Optimal memory allocation and scheduling for DMA data transfers under the LET paradigm. Proceedings of the 58th Design Automation Conference (DAC); 2021.
55. Mancuso R, Pellizzoni R, Caccamo M, Sha L, Yun H. WCET(m) estimation in multi-core systems using single core equivalence. Proceedings of the 2015 27th Euromicro Conference on Real-Time Systems; 2015.
56. Mancuso R, Pellizzoni R, Tokcan N, Caccamo M. WCET derivation under single core equivalence with explicit memory budget assignment. Proceedings of the 29th Euromicro Conference on Real-Time Systems (ECRTS 2017); 2017.
57. Hassane M, Pellizzoni R. Bounding DRAM interference in COTS heterogeneous MPSoCs for mixed criticality systems. IEEE Trans Comput Aided Des Integr Circuits Syst. 2018;37(11):2323-2336.
58. Hassan M, Pellizzoni R. Analysis of memory-contention in heterogeneous COTS MPSoCs. Proceedings of the 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020); 2020.
59. Yun H, Pellizzoni R, Valsan PK. Parallelism-Aware Memory Interference Delay Analysis for COTS Multicore Systems. Proceedings of the 2015 27th Euromicro Conference on Real-Time Systems; 2015.
60. Casini D, Biondi A, Nelissen G, Buttazzo G. A holistic memory contention analysis for parallel real-time tasks under partitioned scheduling. Proceedings of the Proceedings of the 26th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2020); 2020.
61. Akesson B, Goossens K. Architectures and modeling of predictable memory controllers for improved system integration. Proceedings of the 2011 Design, Automation Test in Europe; 2011:1-6.
62. Cruz R. A calculus for network delay. I. network elements in isolation. IEEE Trans Inf Theory. 1991;37(1):114-131.
63. Stiliadis D, Varma A. Latency-rate servers: a general model for analysis of traffic scheduling algorithms. IEEE/ACM Trans Netw. 1998;6(5):611-624.
64. Kim N, Tang S, Otterness N, Anderson JH, Smith FD, Porter DE. Supporting I/O and IPC via fine-grained OS isolation for mixed-criticality real-time tasks. Proceedings of the 21st International Conference on Real-Time Networks and Systems (RTNS’18); 2018.
65. Paolieri M, Quinones E, Cazorla FJ, Valero M. An analyzable memory controller for hard real-time CMPs. IEEE Embedded Syst Lett. 2009;1(4):86-90.
66. Akesson B, Goossens K, Ringhofer M. Predator: a predictable SDRAM memory controller. Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis; 2007:251-256.
67. Mirosanlou R, Hassan M, Pellizzoni R. Drambulism: balancing performance and predictability through dynamic pipelining. Proceedings of the 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS); 2020:82-94.
68. Zhonghai Lu, Millberg M, Jantsch A, Bruce A, van der Wolf P, Henriksson T. Flow regulation for on-chip communication. Proceedings of the 2009 Design, Automation Test in Europe Conference Exhibition; 2009.
69. Restuccia F, Pagani M, Biondi A, Marinoni M, Buttazzo G. Modeling and analysis of bus contention for hardware accelerators in FPGA SoCs. Proceedings of the 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020); 2020.
70. CLARE Software stack. http://clare.santannapisa.it
71. Casini D, Biondi A, Cicero G, Buttazzo G. Latency analysis of I/O virtualization techniques in hypervisor-based real-time systems. Proceedings of the 27th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2021); 2021.

How to cite this article: Zini M, Cicero G, Casini D, Biondi A. Profiling and controlling I/O-related memory contention in COTS heterogeneous platforms. Softw Pract Exper. 2022;52(5):1095-1113. doi: 10.1002/spe.3053