


# 1 The Omnivisor: A real-time static partitioning 2 hypervisor extension for heterogeneous core 3 virtualization over MPSoCs.

4 **Daniele Ottaviano** ✉ 

5 Università degli Studi di Napoli Federico II, Italy

6 **Francesco Ciralo** ✉ 

7 Boston University, MA, USA

8 **Renato Mancuso** ✉ 

9 Boston University, MA, USA

10 **Marcello Cinque** ✉ 

11 Università degli Studi di Napoli Federico II, Italy

## 12 — Abstract —

---

13 Following the needs of industrial applications, virtualization has emerged as one of the most effective  
14 approaches for the consolidation of mixed-criticality systems while meeting tight constraints in  
15 terms of space, weight, power, and cost (SWaP-C). In embedded platforms with homogeneous  
16 processors, a wealth of works have proposed designs and techniques to enforce spatio-temporal  
17 isolation by leveraging well-understood virtualization support. Unfortunately, achieving the same  
18 goal on heterogeneous MultiProcessor Systems-on-Chip (MPSoCs) has been largely overlooked.  
19 Modern hypervisors are designed to operate exclusively on main cores, with little or no consideration  
20 given to other co-processors within the system, such as small microcontroller-level CPUs or soft-cores  
21 deployed on programmable logic (FPGA). Typically, hypervisors consider co-processors as I/O  
22 devices allocated to virtual machines that run on primary cores, yielding full control and responsibility  
23 over them. Nevertheless, inadequate management of these resources can lead to spatio-temporal  
24 isolation issues within the system. In this paper, we propose the Omnivisor model as a paradigm  
25 for the holistic management of heterogeneous platforms. The model generalizes the features of  
26 real-time static partitioning hypervisors to enable the execution of virtual machines on processors  
27 with different Instruction Set Architectures (ISAs) within the same MPSoC. Moreover, the Omnivisor  
28 ensures temporal and spatial isolation between virtual machines by integrating and leveraging a  
29 variety of hardware and software protection mechanisms. The presented approach not only expands  
30 the scope of virtualization in MPSoCs but also enhances the overall system reliability and real-time  
31 performance for mixed-criticality applications. A full open-source reference implementation of the  
32 Omnivisor based on the Jailhouse hypervisor is provided, targeting ARM real-time processing units  
33 and RISC-V soft-cores on FPGA. Experimental results on real hardware show the benefits of the  
34 solution, including enabling the seamless launch of virtual machines on different ISAs and extending  
35 spatial/temporal isolation to heterogenous cores with enhanced regulation policies.

36 **2012 ACM Subject Classification** Computer systems organization → Real-time system architecture

37 **Keywords and phrases** Mixed-Criticality, Embedded Virtualization, Real-Time Systems, MPSoCs.

38 **Digital Object Identifier** 10.4230/LIPIcs.ECRTS.2024.6

39 **Supplementary Material** *Software (Source Code)*: <https://github.com/DanieleOttaviano/Omnivisor>

40 **Funding** This work is partially supported by the Italian Ministry of Enterprises and Made in Italy (MIMIT) under  
41 the GENIO Project (CUP B69J23005770005), and it has been carried out within the EUROfusion Consortium,  
42 funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200  
43 - EUROfusion) and by the National Science Foundation (NSF) under grant number CNS-2238476. Views and  
44 opinions expressed are however those of the author(s) only and do not necessarily reflect those of the EU or the  
45 European Commission or the NSF. Neither any of earlier can be held responsible for them.



© Daniele Ottaviano, Francesco Ciralo, Renato Mancuso, and Marcello Cinque;  
licensed under Creative Commons License CC-BY 4.0

36th Euromicro Conference on Real-Time Systems (ECRTS 2024).

Editor: Rodolfo Pellizzoni; Article No. 6; pp. 6:1–6:27

Leibniz International Proceedings in Informatics



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

46 **1** Introduction

47 The current approach to address the increasing number of functional requirements in industries  
48 that deal with safety-critical systems such as automotive [8], avionics [73], and nuclear  
49 fusion [25] is toward an integrated development model rather than a federated one, where  
50 several services with varying degrees of criticality coexist on shared hardware platforms. These  
51 software architectures are usually referred to as mixed-criticality systems (MCSs) [18, 17].  
52 Developing mixed-criticality systems on multiprocessor architectures to reduce the size,  
53 weight, power, and cost (SWaP-C) is a challenge that, despite strong community interest,  
54 has not yet found a unique standard solution [3, 45, 19]. Among the proposed approaches,  
55 one of the most valuable in the scope of high-performance real-time systems is the use of  
56 real-time virtualization [22]. While traditional virtualization is a feature-rich technology  
57 that allows efficient resource utilization, real-time virtualization leans toward minimalist  
58 architectures focusing on safety, security, and predictability. In the wide spectrum of real-  
59 time virtualization technologies [23], the minimal and safest virtualization approach is static  
60 partitioning [64]. This partitioning-based approach is suitable for critical systems where the  
61 lack of determinism can significantly increase the validation and certification cost.

62 This virtualization approach has shown outstanding isolation performance in symmetric  
63 multi-core architectures, as evidenced in recent studies [47]. However, alongside symmetric  
64 platforms, asymmetric architectures are increasingly gaining traction in the market; the  
65 complexity and heterogeneity of multi-core systems and Commercial Off-The-Shelf (COTS)  
66 boards are gradually increasing to meet the requirements of bleeding-edge industrial ap-  
67 plications. Therefore, we are currently witnessing the growing adoption of asymmetric  
68 MultiProcessor Systems-on-Chip (MPSoCs) in various industrial applications from auto-  
69 motive [8, 16, 65, 39], to avionics [73], and nuclear fusion [25, 7]. With the increase in  
70 hardware complexity within these systems, the already well-known challenges with predict-  
71 ability and security are further exacerbated. Modern MPSoCs, such as AMD/Xilinx Zynq  
72 UltraScale+ [78], Versal [77], NVIDIA Orin [56] and Xavier [57], Google Coral [33] and NXP  
73 i.MX8 [58], embed a heterogeneous set of processing elements. These include general-purpose  
74 microprocessor-level CPUs, sometimes referred to as Application Processing Units (APUs),  
75 and microcontroller-level CPUs that are simpler and more predictable, such as those within  
76 the ARM Cortex-M/R families. Additionally, some of these systems incorporate accelerators  
77 (e.g., Graphical Processing Units—GPUs, and/or Tensor Processing Units—TPUs), and,  
78 in some cases, also Field-Programmable Gate Arrays (FPGAs), that is, re-programmable  
79 hardware capable of integrating various types of special-purpose accelerators or additional  
80 cores (e.g., RISC-V soft-cores). All of these processing elements in the system are intricately  
81 interconnected and share numerous platform resources. From now on, to be consistent with  
82 ARM’s terminology, we will utilize the term *"managers"* to denote all hardware capable  
83 of initiating memory transactions. Additionally, we will refer to all cores that are not  
84 general-purpose application cores (main cores), as *"remote cores"* to be compliant with the  
85 terminology used by Linux (e.g. `remoteproc` driver [44]).

86 To provide code running on such complex architectures with real-time guarantees, re-  
87 searchers have focused on mitigating temporal interference due to resource contention across  
88 MPSoCs. Over the years, considerable effort has been invested in exerting control over  
89 the memory hierarchy, including the last-level cache [42], DDR memory [80], and memory  
90 controller [82]. Significant attention has also gone into minimizing interrupt latency [29] and  
91 managing the sharing of memory channels among modules in the programmable logic [27].  
92 However, comparatively less attention has gone into the inherent limitation of static parti-

tioning hypervisors in efficiently managing heterogeneous platforms. Specifically, modern architectures present cores that manufacturers provide ad-hoc to execute specialized software. Examples include Real-Time Processing Units (RPU) used to run critical applications and Deep Learning Processing Units (DPU) used to improve the performance of AI applications. In a mixed-criticality system, we expect the execution time of code running on RPU to remain unaffected by other independent applications, such as AI workload running on DPU.

Currently, remote cores are not managed by the hypervisors in the same way as the main CPUs; rather, these cores are either ignored entirely or, at best, treated as I/O devices allocated to virtual machines (VM) running on primary cores. This means a VM controlling one or more remote cores can load and execute any code on them. Unfortunately, a remote processing core usually possesses enough privileges to access critical platform resources, becoming a threat to the other VM running on the board from a spatial and temporal isolation point of view. In contrast, a hypervisor designed for heterogeneous MPSoC should:

- Offer a unified and transparent interface to the user to flexibly deploy virtual machines on any core within the platform, regardless of the Instruction Set Architecture (ISA).
- Guarantee comprehensive spatial and temporal isolation between VM across the platform.

**Research Question.** The question that inspired this paper is: *Can next-generation real-time static partitioning hypervisors adapt to the evolving landscape of modern heterogeneous platforms? Specifically, can they offer seamless and flexible mechanisms for deploying VM across heterogeneous processing cores, all while ensuring robust isolation guarantees for mixed-criticality deployment?*

**Contribution.** To tackle such a question, in this paper, we propose the *Omnivisor* model. This model extends the traditional static partitioning hypervisor paradigm to take control over heterogeneous cores in MPSoC platforms. Thus, we make the following contributions:

- We propose a novel model that generalizes the features of real-time static partitioning hypervisors to integrate the management of heterogeneous cores, improving their flexibility and usability in MPSoC platforms.
- We show how a combination of various hardware-software protection mechanisms can be seamlessly orchestrated at runtime by our *Omnivisor* to ensure high isolation between VM running on heterogeneous cores.
- We provide an open source reference implementation [61] and an evaluation of the proposed model on a COTS board (AMD/Xilinx's UltraScale+) by extending Jailhouse, a real-time static partitioning hypervisor, to run virtual machines over remote cores with different ISAs (Aarch32 RPU and RISC-V soft-cores).

Experimental results on the board show that a user can seamlessly launch a VM on heterogeneous cores via the *Omnivisor* with comparable boot times. These experiments highlight the *Omnivisor*'s flexibility which enables compelling scenarios such as real-time live migration [41], reboot after failure [51], system rejuvenation [1], and over-the-air (OTA) updates [28, 36]. Experiments also demonstrate the isolation capabilities of the *Omnivisor* by executing critical workload on remote cores in the presence of severe disturbances generated by the other cores and the FPGA on the same board. Finally, by using realistic benchmarks, we show how the *Omnivisor* can enforce a controlled degradation policy to keep real-time guarantees while not limiting the overall system performance.

**Paper Structure.** In Sec. 2, we review modern hardware protection mechanisms on MPSoC and discuss traditional hypervisor models' limitations. Sec. 3 introduces the *Omnivisor* model, highlighting its benefits and differences from traditional models. We also discuss *Omnivisor*'s requirements, responsibilities, and features. In Sec. 4, we walk through the implementation

140 of the Omnivisor on a Xilinx Ultrascale+ board, assessing strengths and weaknesses. Sec. 5  
 141 and 6 present experimental analysis and practical use cases. Sec. 7 compares Omnivisor with  
 142 related works. Conclusive remarks and future works are provided in Sec. 8.

## 143 **2 Background and Motivations**

144 Considering the high heterogeneity of processing elements deployed on MPSoCs that act as  
 145 managers—i.e., heterogeneous CPUs, GPUs, DMAs, and FPGAs sharing system resources  
 146 like the memory controller, memory storage, I/O devices—hardware manufacturers provide  
 147 a robust suite of hardware protection mechanisms to improve both spatial and temporal  
 148 isolation guarantees. **Spatial isolation** ensures that a processing element accessing a shared  
 149 resource prevents other processing elements from accessing its private data. **Temporal**  
 150 **isolation** guarantees that the time behavior of a processing element is not affected by (or  
 151 has a bounded effect on) the behavior of other processing elements, even if those (partially)  
 152 access the same shared resources.

153 This section aims to provide a comprehensive summary and categorization of the various  
 154 processor types and protection mechanisms employed on state-of-the-art MPSoCs, shedding  
 155 light on their roles and scope within the considered class of platforms. Following that, we  
 156 explain how traditional static partitioning hypervisors utilize these mechanisms only to  
 157 a limited extent, highlighting why this presents a significant constraint compared to the  
 158 extensive capabilities provided by modern COTS platforms.

### 159 **2.1 MPSoCs processors classes**

160 Embedded MPSoCs are nowadays characterized by heterogeneous clusters of CPUs that can  
 161 be categorized into three classes that feature different protection mechanisms:

- 162 ■ **microprocessor-level CPUs:** Fully featured general-purpose multi-core CPUs character-  
 163 ized by all the modern hardware optimization techniques such as prefetching, branch  
 164 prediction, cache coherence, as well as memory virtualization (MMU-based, see Sec. 2.2.1).  
 165 These processors present at least three privilege levels to differentiate permissions and  
 166 registers belonging to the hypervisor, the operating system, and the user-level applications.  
 167 These are often referred to as Application Processing Units (APUs); an example is the  
 168 cores belonging to the ARM Cortex-A family.
- 169 ■ **microcontroller-level CPUs:** Specific-purpose CPUs that do not have any mechanism  
 170 for memory virtualization (MPU-based). They exhibit reduced hardware optimization  
 171 techniques to improve simplicity and predictability. Furthermore, these microcontrollers  
 172 usually support less than three privileged levels. This is because the software deployed  
 173 on these CPUs is simpler and typically consists of a bare-metal application or, at most, a  
 174 real-time operating system (RTOS). An example includes the ARM Cortex-M and the  
 175 ARM Cortex-R family, and often referred to as Real-Time Processing Units (RPU).
- 176 ■ **programmable logic CPUs:** Highly specialized soft-cores deployed on re-programmable  
 177 hardware to run code with specific requirements. Although these processors are extremely  
 178 heterogeneous, their deployment on FPGA platforms enables communication with the rest  
 179 of the system, mediated by the SMMU (see Sec. 2.2.1). This category includes soft-cores  
 180 such as the AMD MicroBlaze [5], or the RISC-V Pico32 [79].

### 181 **2.2 MPSoCs Protection Mechanisms**

182 The MPSoCs protection mechanisms can be systematically categorized as follows.

### 183 2.2.1 Spatial Isolation

184 **Address Translation (MMU/SMMU):** The Memory Management Unit (MMU) is the  
185 most known and used memory isolation mechanism for address translation. It is a component  
186 integrated into most microprocessor-level CPUs, serving a fundamental role in virtual memory  
187 management. The MMU maps virtual addresses to physical addresses, enabling applications  
188 (or guest OSes) to access memory locations in a manner that is transparent and independent  
189 of the physical memory layout. In the context of heterogeneous MPSoCs, the System Memory  
190 Management Unit (SMMU) is an extension of the MMU, tailored to manage memory and  
191 address translation for DMA-capable devices and accelerators. However, not all processing  
192 elements that can potentially assume the role of a manager on these boards are equipped with  
193 an MMU/SMMU. Consequently, if not properly configured, certain managers can potentially  
194 access other managers' data in a manner that poses inherent security risks and/or results in  
195 poor fault containment, as evidenced in our evaluation.

196 **Accesses Protection (MPU/SMPU/SPPU):** Address translation mechanisms are not  
197 the only means of achieving spatial isolation. Microcontroller-level CPUs typically employed  
198 to run bare-metal software or Real-Time Operating Systems (RTOS) do not necessitate  
199 address translation mechanisms. This is due to both the inherent cost of such mechanisms  
200 in terms of space occupation and energy consumption and the temporal unpredictability  
201 that MMU-based mechanisms introduce [62]. In these scenarios, CPUs are equipped with  
202 more straightforward mechanisms known as Memory Protection Units (MPUs). These are  
203 implemented as hardware tables deployed between the manager (CPU) and the subordinate  
204 (Memory). Using the tables, an MPU enforces specific permissions to fixed address space  
205 regions. In heterogeneous MPSoCs, given that not all processing elements within these  
206 platforms possess address translation mechanisms, a comprehensive spatial isolation strategy  
207 is implemented by deploying system MPU-based protection mechanisms at the access port  
208 of important system resources. We term these system-level protection mechanisms System  
209 Memory Protection Units (SMPUs) when used to protect memory; we use the term System  
210 Peripheral Protection Units (SPPUs) when they are used to protect memory-mapped I/O.

### 211 2.2.2 Temporal Isolation

212 **Hardware Bandwidth Allocation:** In modern ARM-based platforms, Quality of Service  
213 (QoS) support offers a mechanism to manage memory traffic at the level of bus managers.  
214 Communication between a manager and a subordinate within an ARM-based platform is  
215 facilitated through the AXI protocol. The latest iteration of the AXI protocol, the AXI4  
216 standard, incorporates a set of signals, specifically ARQOS and AWQOS, which convey traffic  
217 prioritization details essential to enforce bandwidth regulation in QoS-aware on-chip memory.  
218 The QoS technology was initially introduced into MPSoCs with the primary objective of  
219 achieving load balancing. However, numerous studies have subsequently demonstrated its  
220 versatility and effectiveness in ensuring temporal isolation [67, 32]. However, there is a  
221 common trend in existing QoS-enabled platforms [69]: multi-core CPUs are typically treated  
222 as a unified manager. As a result, QoS support is primarily employed to regulate the  
223 aggregate traffic generated by all CPUs collectively. While this observation holds for main  
224 cores, it differs in the case of remote cores. These remote processors are usually equipped  
225 with distinct QoS ports for each CPU, a crucial distinction leveraged in the Omnivisor model  
226 to achieve temporal isolation between heterogeneous cores.

227 **Software Bandwidth Allocation:** Despite the QoS limitation in managing individual  
228 CPUs in a multi-core cluster, software solutions exist to regulate the bandwidth of the

229 multi-core processors, offering per-CPU granularity that an Omnivisor shall leverage [81] [82].

## 230 2.3 From Traditional to Static Partitioning Hypervisors

231 **Traditional Hypervisors.** In the traditional hypervisor model, a virtualization layer is set  
232 between multiple software environments, namely virtual machines (VMs), and the underlying  
233 hardware. The responsibility of this layer is to abstract the physical hardware resources  
234 to the VMs to give them the illusion of running alone on the platform. To realize such  
235 abstractions, modern hypervisors take advantage of a combination of software mechanisms,  
236 including *hypercalls* and the *trap-and-emulate* technique. In addition, they leverage hardware  
237 mechanisms such as advanced MMU systems with dual stages of translation and support for  
238 multiple privilege levels within processor cores. This approach is designed to ensure spatial  
239 isolation between VMs, preventing one VM from accessing the data belonging to another  
240 VM while striving to maintain high performance and resource utilization levels. On top of  
241 this layer, hypervisors provide an interface for managing the VMs, allowing a high-privilege  
242 user to create, stop, and control the resources assigned to VMs at run-time. Well-known  
243 open-source hypervisors that follow this model are KVM [40], Xen [11], and many others.  
244 These are widely used, and researchers have extended their capabilities to accommodate  
245 various use cases, including real-time scenarios [2, 30].

246 **Static Partitioning hypervisors.** Real-time static partitioning hypervisors (SPHs), such as  
247 Jailhouse [63], Bao [48], Xtratum [49], and Quest-V [74], moves from traditional hypervisor  
248 model by adding resource separation constraints bearing the cost of less efficient use of  
249 resources to meet the requirements of real-time applications. In the SPH model, temporal  
250 isolation is as important as spatial isolation; therefore, they statically partition hardware  
251 resources between VMs to minimize shared components and mitigate temporal interference.  
252 According to this model, each VM gets a subset of the platform's resources; therefore,  
253 the CPUs are statically assigned to the VMs, and so are the memory, I/O devices, and  
254 accelerators.

## 255 2.4 SPH Shortcoming over Asymmetric MPSoCs

256 SPHs are currently designed to operate exclusively on microprocessor-level CPUs, with  
257 little or no consideration given to remote cores within the system, such as microcontrollers  
258 or soft-cores on FPGAs. In this scenario, deploying code on remote cores requires the  
259 system programmer to manually load the code and start the core. This is currently possible  
260 using two approaches: (I) using the bootloader and thus at boot time or (II) using the  
261 Linux `remoteproc` driver on a VM at runtime. However, the former approach sacrifices  
262 the flexibility of dynamically halting and reloading code on the remote cores as needed,  
263 and the latter gives a VM full access to remote cores that can easily introduce time delays,  
264 interferences, or even system failures. Specifically, the remote cores are not isolated by default  
265 from the other virtual machines, and the code running on them can cause temporal and/or  
266 spatial isolation issues for the other VMs by accessing the shared resources. To address  
267 this, a system programmer can manually configure and enable platform-specific hardware  
268 protection mechanisms, such as SMPU/SPPU and QoS, to isolate the cores from the other  
269 VMs. Although effective, this approach diminishes the flexibility of the hypervisor and  
270 requires significant effort and specialized expertise. To actually maintain the isolation, every  
271 time a new VM is created, and every time a new code is loaded in the remote cores, the  
272 system developer must promptly reconfigure these mechanisms to isolate resources, otherwise  
273 risking data corruption or possible interference between cores.

274 An SPH on heterogeneous MPSoCs should ensure holistic protection across the entire  
275 board, transparently to the user. It should handle isolation seamlessly, avoiding the need for  
276 manual programming of specialized hardware protection mechanisms and providing a more  
277 user-friendly and robust solution for running code on asymmetric multi-core systems.

### 278 **3 The Omnivisor**

279 In this paper, we introduce the Omnivisor, a novel hypervisor model that generalizes static  
280 partitioning hypervisors to enable the transparent execution of VMs on heterogeneous cores  
281 over commercial off-the-shelf (COTS) MPSoCs. The model aims to streamline the deployment  
282 process and simplify the programming model of such complex architectures while providing  
283 strong spatial and temporal isolation as required by mixed-criticality systems.

284 **Model Purpose.** As depicted in Fig. 1, while conventional hypervisors are designed to  
285 manage microprocessor-level CPUs, our model extends its control to include microcontroller-  
286 level CPUs and soft-cores on programmable logic (FPGA). To achieve this, the Omnivisor  
287 assumes control over different hardware mechanisms to ensure isolation, both temporally  
288 and spatially, of the VMs. Three primary objectives underpin the Omnivisor model:

- 289 ■ **1.** To offer users a consistent, transparent, and easy-to-use interface for managing virtual  
290 machines on both primary and remote cores.
- 291 ■ **2.** To reorganize the privilege levels of the software running on heterogeneous cores in  
292 order to build a holistic privilege hierarchy across the platform.
- 293 ■ **3.** To seamlessly administer spatial and temporal isolation between virtual machines,  
294 regardless of the specific core on which they are deployed.

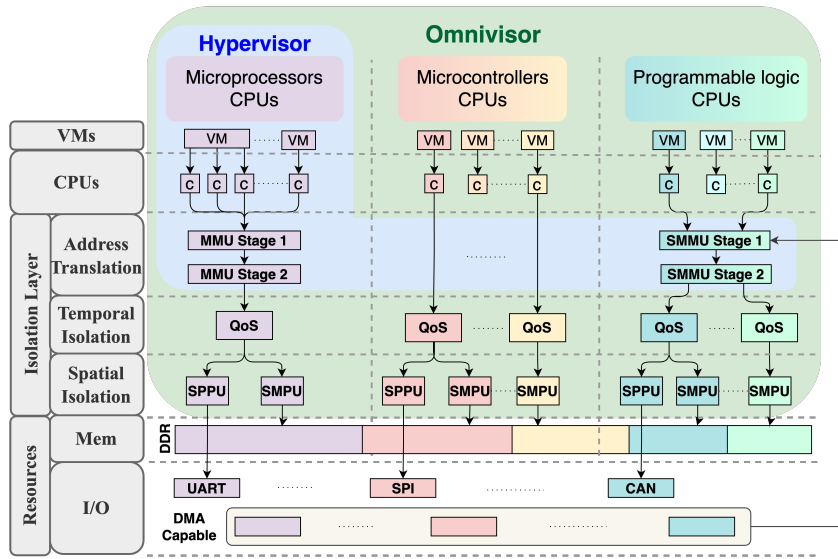
295 According to this novel model, remote cores are no longer mere I/O devices; instead, they  
296 are elevated to primary CPUs capable of running self-contained, strongly isolated VMs.

297 **Clarification of Terminology.** Before delving into the specifics of the Omnivisor, it is  
298 important to clarify why we chose to use the term "*Virtual Machine*" to denote the code  
299 executed by the Omnivisor on all the types of cores. We acknowledge that the code running on  
300 remote cores does not execute atop an actual hypervisor, meaning that there is no scheduler,  
301 and the code has complete control over the core itself. However, we have opted to label them  
302 VM for two main reasons. First, they are encapsulated by the Omnivisor, which is capable  
303 of isolating the accessible resources in the system, similar to how SPHs handle traditional  
304 VMs. Second, we provide users with a unified and transparent method for utilizing remote  
305 cores, mirroring the process of launching a VM on application cores.

#### 306 **3.1 Requirements**

307 The Omnivisor model is based on the assumption of having at its disposal a fully featured  
308 MPSoC with the following characteristics:

- 309 ■ **Multiple Core Clusters:** Two or more heterogeneous clusters of cores, and at least one of  
310 the clusters is a multiprocessor-level CPU cluster.
- 311 ■ **Address Translation:** An MMU featuring two levels of translation in front of each  
312 multiprocessor-level CPU cluster and an SMMU placed between DMA-capable peripherals/  
313 accelerators and shared resources.
- 314 ■ **Accesses Protection:** SMPU/SPPU hardware protection mechanisms to shield shared  
315 resources (memory, system registers, and peripherals).
- 316 ■ **Bandwidth Regulation:** A hardware QoS-like bandwidth allocation mechanism for each  
317 core cluster and DMA-capable peripherals that access shared resources.



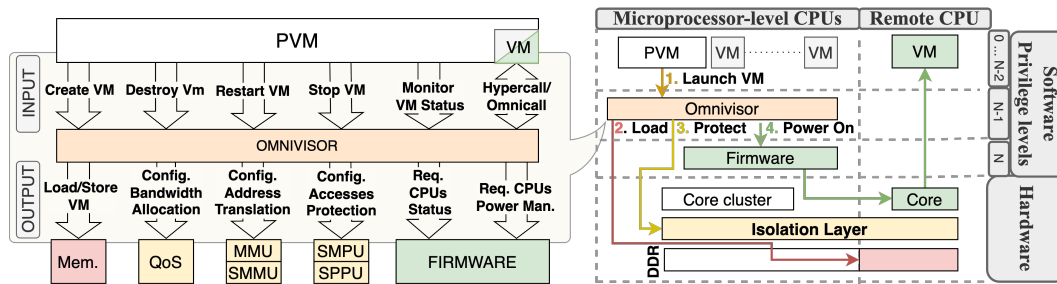
■ **Figure 1** A block diagram illustrates the Omnivisor model, showcasing varied temporal and spatial isolation mechanisms across CPU clusters, emphasizing their heterogeneity. The arrows indicate the flow of a request from an initiator to the accessed resource (memory or I/O).

318 ■ *Power Management Firmware*: A board-specific firmware that exposes an interface to  
 319 the hypervisor for heterogeneous cores power management.  
 320 These specified characteristics represent the foundational prerequisites for a platform to  
 321 be deemed *Omnivisor-ready*. Although these requirements may initially appear as limit-  
 322 ing factors, they effectively align with the design standards of modern embedded system  
 323 platforms [78, 77, 58], tailored to meet industrial demands.

### 3.2 Responsibilities

325 The Omnivisor operates as holistic software running at the highest privilege level on the  
 326 board. It delivers services to software running at lower privilege levels. Therefore, its primary  
 327 responsibility is to prevent the escalation of VM privileges, regardless of the used cores.  
 328 **AMP privilege enforcement.** The coexistence of multiple cores with varying architectures  
 329 in an MPSoC precludes the application of a Symmetric Multi-Processing (SMP) approach.  
 330 In SMP, all cores are orchestrated by a single software instance sharing a common address  
 331 space. Instead, MPSoCs imply the use of an Asymmetric Multi-Processing (AMP) approach,  
 332 where different core clusters operate independently, each with a unique address space. Given  
 333 that constraint, similar to traditional hypervisors in SMP configuration, the Omnivisor  
 334 must ensure that VMs running in AMP configuration do not access resources outside of the  
 335 boundaries of the partitions. However, while traditional hypervisors can leverage multi-core  
 336 hardware extensions to manage the privilege levels of VMs, the Omnivisor must employ  
 337 a combination of distinct hardware mechanisms tailored to the specific core cluster it is  
 338 managing. For instance, while soft-cores deployed on FPGA can be protected using the  
 339 SMMU, the Omnivisor must leverage SMPUs to shield the resources from the VMs running  
 340 on microcontroller-level CPUs. Consequently, as shown in Fig.2 (right), every time a new  
 341 VM is launched on a remote core, before starting the core, the Omnivisor must configure  
 342 an isolation layer that restrains the capabilities of the newly-created VM restricting access  
 343 to both higher privileged resources (e.g., system registers) as well as resources belonging  
 344 to different VMs (e.g., I/O peripherals, memory regions). The arrows in Fig.2 (right) are





■ **Figure 2** Omnivisor feature set (left) and remote core VM startup process (right).

color-coded based on the operation and are enumerated in temporal order.

**DMA-capable I/O:** The cores are not the only platform managers within the system. Indeed, DMA engines could have access to all system resources, potentially jeopardizing inter-partition spatio-temporal isolation. To address this risk, the Omnivisor must prevent:

- 1. DMA engines from having unrestricted and unregulated access to memory resources.
- 2. A core from programming the DMA to access memory regions it does not own.

As depicted in Fig. 1, the Omnivisor addresses the first issue by employing SMMU mechanisms to enforce address translation and access protection for DMA, much like traditional SPHs. Additionally, the QoS is employed to provide temporal isolation.

Typically, when an SPH allocates the DMA to a VM, it configures the SMMU to allocate the same memory regions to both. Therefore, a VM cannot exploit the DMA to access inaccessible regions. However, If a second virtual machine is running on a remote core without MMU/SMMU protection (microcontroller-level CPUs), it can freely access the address region of the DMA registers. Therefore, it could potentially program the DMA to gain unauthorized access to memory areas belonging to the first VM. To avoid that, addressing the second issue, the Omnivisor employs a strategy wherein SMPUs are configured to restrict access to the DMA registers exclusively to the Omnivisor itself and to the VM that is supposed to use it.

In a broader context, the Omnivisor applies a similar strategy to restrict permissions of remote cores to protect other critical address regions, including those for configuring the SMMU, SMPUs, and QoS.

### 3.3 Features

The Omnivisor provides a set of features that includes that of the traditional SPHs while expanding them to encompass heterogeneous processing elements (see Fig. 2). Given the diversity among existing hypervisors, defining the minimum feature set and how they are extended for effective operation on asymmetric architectures is crucial.

**The Privileged Virtual Machine (PVM) interface.** First, we introduce the *Privileged VM* (PVM), which is a known concept in hypervisor’s literature [54], and is the only VM with the ability to manage other VMs. A few examples are the *root-cell* in Jailhouse [63], and the *Dom0* in Xen [70]. The Omnivisor provides the PVM with the same interface for managing VMs for both the main and remote cores. For instance, as shown in Fig. 2, the PVM only needs to request the VM launch, and then the Omnivisor takes charge of programming the underlying resources to serve the request for the specified processor. Other than launching a VM, the Omnivisor provides methods for stopping and restarting a VM and an interface for monitoring the current status of the VMs.

**Omnical.** Most state-of-the-art hypervisors implement *hypercalls* to expose functionalities to virtual machines, akin to how operating systems implement system calls for processes.

381 Despite the current implementation of Omnivisor restricting this mechanism to virtual  
382 machines running on the APUs, we aim to propose a design for extending this service to  
383 VMs running on remote cores, which we will refer to as "*Omnicalls*". To implement this  
384 mechanism, the Omnivisor needs to provide three additional features:

- 385 ■ **1.** Event signaling from the Omnivisor to VMs on remote cores.
- 386 ■ **2.** Event signaling from VMs on remote cores to the Omnivisor.
- 387 ■ **3.** A real-time protocol for inter-VM communication.

388 For the first functionality, we need to differentiate between processing elements that support  
389 interrupt delivery, like APUs, and those that do not support them, such as hardly restricted  
390 soft-cores. To signal an event to the former category, the Omnivisor can leverage Software  
391 Generated Interrupts (SGI). Meanwhile, signaling events to the latter requires the remote  
392 VM to periodically check for Omnivisor-originated pending events (polling).

393 Regarding the second functionality, the Omnivisor can grant the VMs on remote cores  
394 access to a subset of the interrupt controller's configuration space, enabling the generation of  
395 SGIs toward the cores where the Omnivisor operates. Currently, the Omnivisor supports  
396 restricted access to the interrupt controller configuration space for these VMs.

397 Lastly, using shared memory for data exchange is already implemented in most legacy  
398 SPHs. We extended this feature to remote cores in the Omnivisor, but enhancing the  
399 real-time performance of the communications requires a tailored mechanism. To provide  
400 real-time guarantees, one existing solution consists of using an external processing element  
401 as a broker to orchestrate the communications between VMs. This has been theoretically  
402 proved and tested on a heterogeneous MPSoC by Schwärcke et al. [66], and the Omnivisor  
403 can easily integrate the broker as a VM running on a remote core while using its features to  
404 isolate it both temporally and spatially from the other VMs.

405 **Dynamic Address Translation.** In traditional hypervisors, when a new VM is created  
406 on the APU, address translation is typically implemented using the MMU. The Omnivisor  
407 extends this functionality to soft-cores by utilizing the SMMU. It's worth noting that  
408 the SMMU is already employed by SPHs to perform address translation for I/O devices  
409 associated with VMs. However, the Omnivisor changes the perspective and utilizes the  
410 same mechanism to implement self-contained translation specifically for soft-cores, which are  
411 treated as self-contained VMs in this context.

412 **Dynamic Accesses Protection.** Protection mechanisms on MPSoCs, such as SMPU/SPPU,  
413 are commonly configured statically at boot time by high-privilege and secure software (e.g.,  
414 first-stage bootloader). These configurations typically remain unchanged throughout the  
415 system's lifetime. However, to enable the seamless execution of isolated VMs on remote cores,  
416 the Omnivisor dynamically determines how to configure all access protection mechanisms.  
417 This approach ensures dynamic system-level protection that adapts during runtime based on  
418 the specific VMs currently active.

419 **Dynamic Bandwidth Allocation.** Traditional SPHs ensure that resource assignments  
420 remain static between PVM management calls. This implies that everything can be dynamic-  
421 ally reassigned by these calls, remaining static until the next call. The Omnivisor maintains  
422 consistency by applying the same approach to bandwidth allocation. Hence, every time  
423 a new VM is launched, it is possible to dynamically allocate the bandwidth to that VM.  
424 Moreover, to enable mission-critical reconfiguration scenarios and ease parameter tuning, the  
425 Omnivisor implements bandwidth allocation as a settling call that the user can leverage  
426 to modify the temporal behavior of the VMs to a new static configuration. Once more, the  
427 Omnivisor shifts the paradigm regarding resource utilization. Unlike SPHs, which primarily  
428 focus on protecting VMs solely on the APU, the Omnivisor extends its scope to encompass

429 VMs on other remote processors. Consequently, bandwidth regulation mechanisms like QoS  
430 are not only employed on accelerators to maintain service quality for APUs but also for  
431 remote cores, even if they are soft-core deployed on FPGA.

## 432 **4 Omnivisor Implementation**

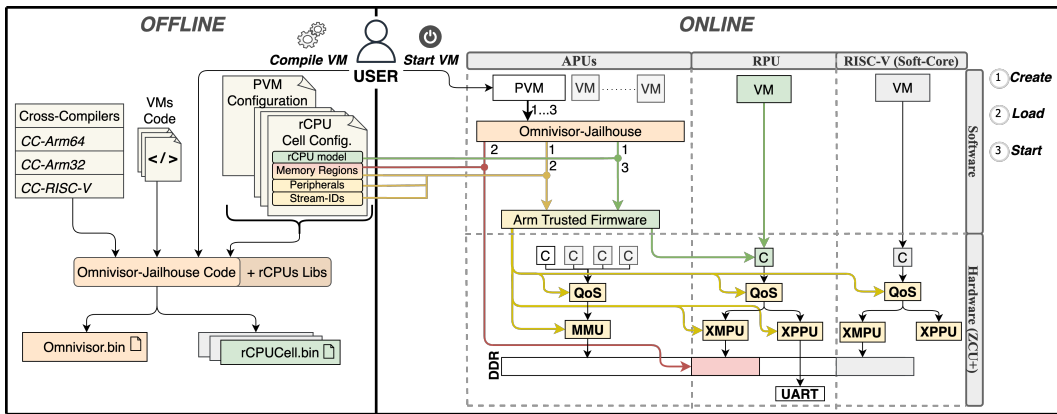
433 The Omnivisor model is designed to apply to a wide range of existing partitioning hypervisors;  
434 nonetheless, our reference implementation is built on top of the Jailhouse hypervisor [63]  
435 because it has low overhead [47] while maintaining an easy-to-use interface to manage VMs at  
436 runtime. Furthermore, the Jailhouse-RT branch, overseen by Minerva Systems [50], already  
437 implements MemGuard-like regulators for the APUs, page coloring, and basic SMMU drivers.  
438 It also provides a rudimental interface to control ARM Quality of Service (QoS) regulators.

439 The implementation was carried out with testing focused on the ARM-based Zynq  
440 Ultrascale+ board from Xilinx. This MPSoC aligns with all the requirements outlined in  
441 Sec. 3.1: it features a quad-core ARM Cortex-A53 (APUs), a dual-core ARM Cortex-R5F  
442 (RPU), and a 16nm FinFET + Programmable Logic (FPGA). Additionally, the platform is  
443 equipped with protection mechanisms for both temporal isolation (QoS), address translation  
444 (MMU, SMMU), and access permissions (SMPUs, and SPPUs). From now on, we will refer  
445 to this platform with the *ZCU+* notation. Moreover, to use the correct terminology, the  
446 SMPUs/SPPUs on the board are named Xilinx Memory Protection Units (XMPU) and  
447 Xilinx Peripherals Protection Units (XPPU).

448 This section aims to illustrate key Omnivisor technical details, providing a comprehensive  
449 discussion of strengths and limitations. To achieve this, we first briefly describe the Jailhouse  
450 hypervisor, and the additional functionalities introduced by the Omnivisor extension. Then,  
451 we walk through the compiling and start processes of a VM from the user's perspective while  
452 explaining how the Omnivisor manages the system under the hood.

453 **Jailhouse in a Nutshell.** A pivotal design choice in Jailhouse is to initiate the hypervisor  
454 from a running Linux instance. Specifically, by utilizing a Linux kernel module, users can  
455 load the hypervisor into memory and initiate a series of procedures to prepare the system.  
456 Upon initialization on each core, the hypervisor takes control of the underlying hardware,  
457 transforming the running Linux into the first virtual machine within the system, referred  
458 to as the *root-cell*. For its bootstrap, the hypervisor requires only a configuration file that  
459 lists the resources allocated to the *root-cell*. Next, to create reservations (*cells* in Jailhouse  
460 jargon) for the creation of additional VMs (*inmates*), the hypervisor reallocates hardware  
461 resources (e.g., CPU(s), memory, PCI or MMIO devices) from Linux to the new cells as  
462 detailed in other cell-specific configuration files. From now on, we will use the term "*VM*" to  
463 refer to the cell plus inmate pair and "*PVM*" to refer to the root-cell.

464 **Omnivisor Extension Overview.** Starting from a vanilla Jailhouse, besides the small  
465 modifications integrated all over the code to transparently unify the interface of Jailhouse  
466 with the new services, the Omnivisor extends the hypervisor with new low-level functionalities.  
467 First, the power management of remote cores has been implemented, encompassing shut-down,  
468 stop, and start functionalities for both microcontroller-level and soft-cores. Second, spatial  
469 isolation management has been enhanced to include dynamic control of XMPUs/XPPUs.  
470 Moreover, temporal isolation management has been refined through the integration of QoS  
471 regulator control. Finally, the compiling procedure for remote cores VMs has been integrated  
472 into the hypervisor offline workflow. The usage of these functionalities is detailed below.



■ **Figure 3** Architectural view of VM compiling and start procedures using the Omnivisor implementation on top of Jailhouse and the Zynq Ultrascale+ board.

473 **4.1 Omnivisor Usage Workflow**

474 One of the key objectives of the Omnivisor is to simplify the utilization of complex hetero-  
 475 geneous architectures for users. Therefore, the Omnivisor provides a unified approach for  
 476 managing VMs on both main and remote cores. In our implementation, based on the ZCU+,  
 477 alongside the legacy APUs we have integrated all the necessary code to run VMs on two  
 478 types of remote cores: RPUs (ARM32-CortexR5F) and RISC-V soft-cores (Pico32 [79]). To  
 479 streamline our discussion, we will utilize the term "*rCPUs*" to refer to any remote core, while  
 480 we will delve into the implementation for RPU and RISC-V cores only when required.

481 **4.1.1 VM Compiling Process (Offline)**

482 The initial step involves the user compiling a specific VM application to run on a remote  
 483 core. The offline compiling procedure, along with its input and output, is depicted in Fig. 3.  
 484 Given the nature of the remote cores, the applications we run are either bare-metal or built  
 485 on top of simple RTOSes. In both cases, linking some libraries may be a requirement for the  
 486 code to work correctly on a specific core. For instance, the traditional compiling approach  
 487 for RPUs on ZCU+ entails using Xilinx-provided libraries. To streamline the utilization  
 488 of *rCPUs* and align with the Jailhouse methodology, we have integrated the libraries for  
 489 compiling VMs targeting RPU and RISC-V cores into the Omnivisor code. Consequently,  
 490 the user only needs to integrate the application-specific code into the Omnivisor code, as  
 491 all the necessary libraries are already provided, similar to how Jailhouse includes libraries  
 492 for compiling APU-based VMs. Additionally, the user must provide a configuration file for  
 493 the VM, specifying the required resources. This configuration should include details on the  
 494 core(s) used by the VM, whether they are main cores or remote cores, as well as information  
 495 about memory regions and peripherals the VM will access. Furthermore, the configuration  
 496 must list the IDs with which the VM's managers (e.g., CPUs/rCPUs and DMA-capable  
 497 devices) are recognized in the system. Once the user has prepared the application code and  
 498 the configuration file for the VM, they can be compiled together with the Omnivisor code.  
 499 To do it, the user must provide a list of cross-compilers, with one compiler designated for  
 500 each core with a different ISA in the system. For instance, in the case of the ZCU+, this  
 501 would entail using the AArch64 compiler for main cores, the AArch32 compiler for RPUs,  
 502 and the RISC-V 32-bit compiler for the soft-cores. The output after compilation will consist  
 503 of the Omnivisor binary along with the binary images for the VMs.

#### 4.1.2 VM Start-Up Process (Online)

**Omnivisor Enable.** Before starting an inmate, since the Omnivisor generalizes Jailhouse, we need to enable it from a Linux instance as explained in Sec. 4. Different from the vanilla Jailhouse, the configuration in our Omnivisor may also include a field for the *rCPUs*. If this is the case, the Omnivisor verifies whether the remote cores are already active, and if they are, it proceeds to shut them down. After that, it statically assigns their ownership to the PVM so that it can later assign the cores to other VMs. Additionally, the Omnivisor disables all the access permissions to the resources protected by the XMPUs such as memory and system registers. Then it configures the first entry of each XMPU's table to allow only the PVM to access those regions specified in its configuration file. This means that every manager outside the Omnivisor control can not access any resource in the system.

While the Omnivisor is up and running, launching a VM for the user involves using three simple commands, as depicted in Fig. 3: (1) create, (2) load, and (3) start, reviewed below.

**Create.** The create command takes as input the configuration file of the VM, which is parsed to generate per-VM data structures. Resources are then *carved out* from the PVM and mapped to the new VM. For example, the requested remote and main cores are hot-plugged and detached from the PVM to be assigned to the new VM. After that, the isolation layer is configured. First, the MMU is programmed to manage the APU's memory region accesses. However, the *rCPUs* lack protection from the MMU and, if left unprotected, have direct access to all memory-mapped regions. Therefore, XMPUs are dynamically re-programmed to allow access permissions only to the resources requested in the configuration, avoiding unexpected accesses to sensible memory-mapped registers (e.g. DMA registers) and memory regions belonging to other VMs. Finally, in the case of soft-cores over FPGA, the Omnivisor configures the address translation by leveraging the SMMU.

**Load.** The load command requires as input the VM image. Initially, it verifies the image size against the carved-out memory reservation. Then, if the available memory is sufficient, it loads the VM image into memory. Moreover, before starting the VM, the user can optionally regulate the memory bandwidth assigned to the managers to provide specific temporal guarantees to the VMs. The Omnivisor provides the knobs to do it, leveraging the aforementioned QoS and MemGuard interfaces in Jailhouse-RT [50]. This step is integrated into the load command during the start-up of a VM to avoid adding another PVM call between load and start. However, the Omnivisor also implements a PVM call for bandwidth allocation separately from the load to enable mission-critical reconfiguration scenarios. When selecting parameters for bandwidth allocation, it is the system integrator's responsibility to determine the suitable bandwidth for each VM, as this choice heavily relies on the application's requirements. However, using the tools offered by the Omnivisor, it is possible to empirically evaluate the parameters needed to enforce a specific maximum slowdown for a given VM. An example of a simple offline policy to automatize the choice of bandwidth parameters is provided in the experimental section.

**Start.** Finally, using the start command, the user initiates the VM start-up. Different MPSoC's architectures have different standards for power management of cores, such as the *ARM PSCI* [6] or the *Intel ACPI* [38]. However, the functionalities provided by these standards are similar. Therefore, the Omnivisor implements a series of generic power management procedures that are subsequently customized to the specific platform and core. We have implemented the procedures for the RPU<sub>s</sub> (ARM32-CortexR5F) and for a RISC-V soft-core (Pico32) deployed on the FPGA. In the ZCU+ the RPU<sub>s</sub> are overseen by the Platform Management Unit (PMU) core, which exercises control over their execution and power state. The only software with enough permission to call PMU services is the PSCI layer

552 within the *ARM trusted firmware*. Consequently, we implement a specific ZCU+ module to  
553 communicate with the PSCI to request the wake-up and power-off of the RPU. Regarding  
554 the soft-core(s), instead, we have implemented a memory-mapped configuration port in  
555 FPGA, and we expose this port to the Omnivisor to control the reset state of each soft-core.

## 556 **5 Use Cases**

557 In this section, we report a few use cases that inspired us toward the creation of the Omnivisor.  
558 **Real-time control in nuclear fusion power reactors.** Nuclear fusion is foreseen as a  
559 promising clean energy source for the next century, and the ITER tokamak reactor (iter.org)  
560 is set to be the first fusion device with a net-positive energy output. In a tokamak, magnetic  
561 confinement of the plasma is achieved using several magnetic fields generated by the electric  
562 current that flows in an array of external coils. These currents are controlled by the so-called  
563 plasma control system (PCS) [68], which is a complex and multi-input-multi-output control  
564 system. The PCS includes several subcomponents, each aiming to control a specific plasma  
565 feature with different requirements in terms of reliability, latency, and needed computational  
566 resources. The ITER project intends to use MPSoCs [7] to run multiple control loops and  
567 signal conditioning algorithms with different sampling times and reliability requirements  
568 on the same system [60]. Being an experimental facility, one of the missions of ITER is to  
569 test the efficiency of advanced control schemes, e.g., using reinforcement learning, running  
570 side-by-side with basic control loops, for safety reasons.

571 The use of the Omnivisor in this context can speed up the development and testing phases  
572 by enabling the deployment of advanced and computationally heavy control algorithms,  
573 launched as VMs on the APUs, along with stable safety controllers, launched as VMs on  
574 RPU or soft-cores, while assuring spatial and temporal isolation between them.

575 **MPSoCs for advanced system management research.** Researchers adopt heterogeneous  
576 architectures to run computation-intensive applications in safety-critical [19] and mission-  
577 critical scenarios, such as vision control units for self-driving vehicles [20]. Moreover, the  
578 real-time community has shown significant interest in leveraging MPSoCs resources, like  
579 remote cores, for monitoring and management. Executing monitoring or management tasks  
580 on the same platform as the monitored applications can introduce overhead and interference,  
581 while remote monitoring (e.g., over a network connection) suffers from communication  
582 latency [24]. Therefore, utilizing on-board resources, when available, is a good compromise.  
583 For instance, the work described in [82] utilizes RPU on a ZCU+ to finely monitor memory  
584 transactions and control the bandwidth of APUs using the board's debug infrastructure.  
585 In [32], instead, authors employ QoS setups on the memory controller to ensure high-degree  
586 isolation of critical applications across heterogeneous cores. Additionally, in [21], the progress  
587 of a critical application running on APUs is monitored by an RPU on a ZCU+ to provide  
588 online regulation based on the application's state.

589 Integrating an Omnivisor can greatly simplify utilizing these cutting-edge mechanisms in  
590 real-world industrial scenarios by providing an easy way to deploy and isolate the applications.  
591 Furthermore, it can speed up the experimental phase for researchers aiming to implement  
592 complex applications on heterogeneous platforms.

## 593 **6 Experimental Analysis**

594 In this section, we provide an evaluation of the Omnivisor model and its implementation.  
595 The reference implementation, along with a set of scripts to reproduce the experiments, is

596 openly available as open-source software [61]. The platform under test is the ZCU+ described  
597 in Sec. 4. The evaluation aims to address the following questions:

- 598 ■ *Is the boot time of a VM on a remote core comparable to that on main cores?*
- 599 ■ *What degree of spatio-temporal isolation does the Omnivisor guarantee for remote VMs?*
- 600 ■ *Can the Omnivisor be a turnkey solution to achieve controlled degradation?*

601 It's important to note that the additional functionalities introduced in our Omnivisor, as  
602 described in Section 4, are only invoked during the startup of newly created VMs, not at  
603 runtime. Therefore, the overhead of Omnivisor is consistent with prior findings on Jailhouse  
604 [47]. Consequently, we do not present runtime overhead results in this paper.

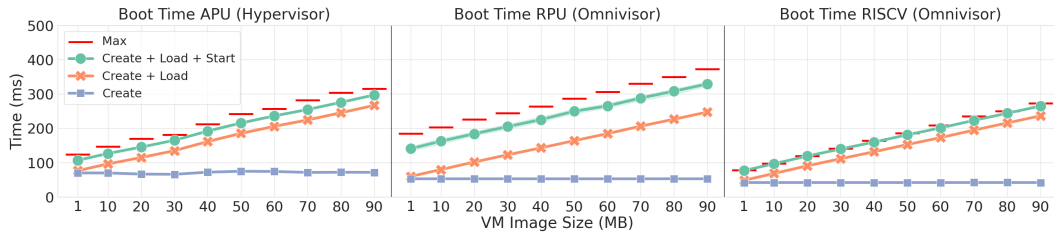
## 605 6.1 Boot Time Performance Assessment

606 This section shows that booting a VM on a remote core using Omnivisor is comparable in time  
607 to booting a VM via Jailhouse on a main core. Thus, with Omnivisor, users can deploy VMs  
608 on either main or remote cores with negligible differences in boot times, enabling flexibility  
609 for scenarios like real-time migration [41], reboot after failure [51], system rejuvenation [1]  
610 and OTA updates [28, 36].

611 Fig. 4 shows the boot times obtained by deploying a VM on RPU and RISC-V soft-core,  
612 using the Omnivisor, compared to the boot time on APU using vanilla Jailhouse. In each case,  
613 the binary contains the identical bare-metal application. However, running the application  
614 on the APU with the Jailhouse hypervisor necessitates linking a tiny 'inmates' library for  
615 initialization whose overhead is negligible during boot times. To obtain the boot time values,  
616 the root-cell acquires the initial value from a global platform timer just before initiating  
617 the new cell (Create). The same timer is used to measure the length of the load sequence  
618 (Create + Load). Finally, the newly started cell captures the third timer sample (Create +  
619 Load + Start), representing the boot time, and records it in a shared memory page. The  
620 described process has been repeated 100 times for ten different VM image sizes, specifically  
621 from 1 to 90 megabytes. It is possible to observe in Fig. 4 in more detail the three phases  
622 that comprise the boot times: create (blue line), load (orange line), and start (green line).

623 We first compare the boot times of a cell on APU and RPU. The results exhibit significant  
624 similarity, indicating that starting a VM on a microcontroller-level CPU does not result in  
625 performance losses. On the contrary, the RPU boot shows a slight speed advantage during  
626 the configuration phase. This difference arises because the APU needs to reorganize the page  
627 tables for the new cell, while the RPU does not use page tables. However, the final boot  
628 time is quite similar, partially due to the lower frequency of the RPU (600MHz) compared  
629 to the APU (1.5GHz), leading to the longer RPU wake-up procedure that involves both the  
630 PSCI and the PMU, as detailed in Sec. 4.1.2. The results for the RISC-V soft-core exhibit  
631 similar creation and loading times as the RPU, as there is no necessity for configuring the  
632 page tables in either case. Nonetheless, the boot time is notably faster. Despite the soft-core  
633 lower frequency (100Mhz), the boot time disparity arises because the soft-core is always  
634 powered on in the FPGA. Removing it from its reset mode via the FPGA's configuration  
635 port is a fast operation compared to the RPU boot procedure.

636 In the ITER project, a fusion experiment enforces multiple stages of the plasma: ramp-up,  
637 flattop, and ramp-down [35]. Each stage requires different controllers to effectively manage  
638 the plasma. Leveraging the Omnivisor ensures flexibility in dynamically reconfiguring these  
639 controllers, deployed as VMs on the board, by rebooting them on main or remote cores.



■ **Figure 4** Comparison of boot times across heterogeneous processors in the ZCU+ Platform

## 6.2 Omnivisor’s Isolation Capability

To demonstrate the Omnivisor isolation capabilities we initially highlight the vulnerabilities that arise when executing unprotected code across various cores within an MPSoC. Subsequently, we activate the Omnivisor with solely spatial protection mechanisms. Finally, we show the effectiveness of the full-fledged Omnivisor by enabling also temporal isolation.

**Experiment Setup.** Fig. 5a (left) depicts our experimental setup: we run a VM under test both on RPU-0 and RISC-V soft-core, while other managers, such as the APUs, the other RPU (RPU-1), and the FPGA, create interference by accessing the memory area owned by the VMs under test. The deployed application is the same on both remote cores. It involves a simple periodic task that reads an array from memory, calculates the sum of its values, and then writes the result back into memory at a different location. The only difference is that, due to the frequency difference between the soft-core (100Mhz) and the RPU (600Mhz), the matrix used in the soft-core application is smaller than that used in the RPU application to ensure comparable results in terms of execution time. The RPUs are configured with disabled caches and operate in split mode, where RPU-0 operates independently from RPU-1. The RISC-V soft-core is deployed on FPGA without any cache. Since the experiments are the same for both VMs under test, we will generally refer to both cores as “*rCPU*” for simplicity.

To assess the isolation capability of the Omnivisor against the vanilla Jailhouse hypervisor, we augment the Jailhouse hypervisor with minimal code necessary to execute applications on remote cores, a functionality not available by default. This enables us to compare the isolation achieved using Jailhouse alone with those obtained using an Omnivisor.

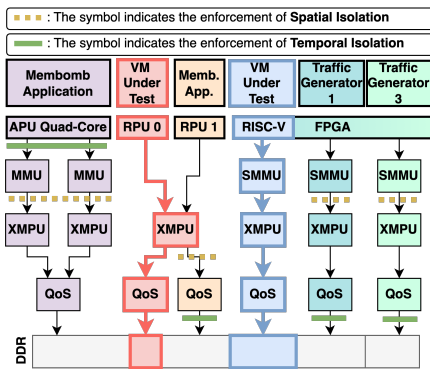
Employing a traditional hypervisor, as illustrated in Table 5b, will cause applications running on remote cores (RPU, RISC-V) to experience failures when other managers access their memory (e.g., RPU1 and FPGA). Conversely, when the Omnivisor extension is enabled, we expect these applications to continue functioning without failures.

**Spatial Isolation Evaluation** The results in Fig. 6 show that without explicitly programming the isolation layer, a manager can break both spatial and temporal isolation of VMs. In the test, the VM under test starts on one *rCPU* and, after two seconds of execution, an interfering application starts on one of the other managers. We repeat the test using Jailhouse vanilla (no protection mechanisms) and using the Omnivisor extension first with only spatial isolation and then the full-fledged version with temporal isolation too.

The interference application deployed on the APU is the well-known IsolBench *bandwidth* benchmark [71] from the RT-Bench framework [55]. The test is launched on three APU cores out of four and it reaches a utilization factor close to 1 on each processor. The free core is used to launch the scripts, start the tests, and save the results. On RPU-1, we deploy a synthetic bare-metal application mirroring the bandwidth benchmark behavior. Finally, in the FPGA, we deploy two instances of the AXI traffic generator IP from Xilinx [76].

We can observe the results when the RPU-1, FPGA, and APU managers are the source of interference in Figs. 6a, 6b, and 6c respectively. The lower bars represent the execution





(a) Architectural View of the experiments

Interference	Hypervisor	Omnivisor
APU(□)	No Failure	No Failure
RPU-1(□)	Crash	No Failure
FPGA(□, □)	Crash	No Failure

(b) Fault behavior of a VM under test

■ **Figure 5** Experimental configuration (left) and expected fault outcomes (right) of VMs running on rCPU subjected to interference from various sources (APU, RPU-1, FPGA): a comparative analysis between traditional Hypervisor and Omnivisor.

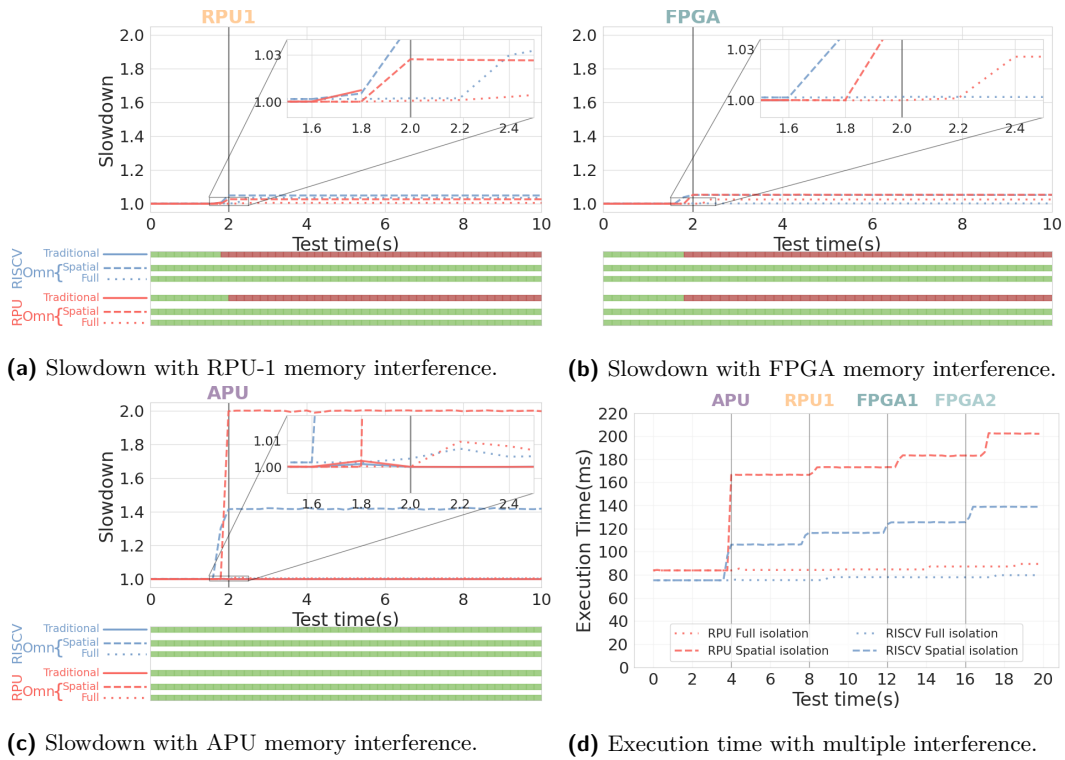
679 state of the VM, where green indicates that the application is running, while red denotes  
 680 that the application has failed. We can observe that, without the Omnivisor, all managers  
 681 can break the spatial containment of the cell, causing the virtual machine to fail except when  
 682 the APU is the source of interference (5b). This is because the Jailhouse hypervisor already  
 683 uses the MMU to protect the memory areas of the VMs. Since the APU is the only manager  
 684 that accesses memory using the MMU, it is also the only one for which spatial permissions  
 685 are enforced with traditional hypervisors. Notably, the access of the cell running on the APU  
 686 to the memory belonging to a different cell causes the APU-bound VM to be shut down by  
 687 the hypervisor while the latter continues undisturbed. That is the reason why, in Fig. 6c, the  
 688 execution time of the VM under test is not impacted when the vanilla hypervisor is deployed.

689 To run this evaluation, we have integrated the code to run VMs on remote cores in  
 690 Jailhouse. Without this upgrade, launching an application on remote cores at run-time is  
 691 possible with the `remoteproc` driver [44]. In that case, since the hypervisor has no vision of  
 692 the memory used by the RPU, it would not offer any form of isolation, leading to a fallback  
 693 in the same failing scenario, even when the APU is the source of traffic.

694 In real-world scenarios, like the ITER project, diverse applications, often developed by  
 695 separate groups, introduce the potential for bugs that can adversely affect other components.  
 696 For instance, a control application running on the RPUs might inadvertently overwrite  
 697 memory used by APUs for critical log information, resulting in the loss of invaluable insights  
 698 during expensive experiments. Thus, the containment level provided by the Omnivisor  
 699 emerges as a crucial feature, mitigating the risk of system failures caused by the malfunction  
 700 of individual applications and facilitating seamless integration.

701 **Temporal Isolation Evaluation** Fig. 6d illustrates the temporal behavior of the periodic  
 702 task running on the rCPU when all other managers access the memory. Every four seconds,  
 703 a new manager is activated and starts creating contention over the memory communication  
 704 channels. From the result, it is clear that hardware mechanisms such as MMU, SMMU, and  
 705 XMPUs/XPPUs can provide spatial isolation between VMs but cannot guarantee temporal  
 706 isolation and, therefore, cannot ensure real-time performance. This is intuitively due to  
 707 the resources that are still shared on board, such as the bus and the memory controller.  
 708 Therefore, temporal isolation can be enforced using mechanisms for bandwidth regulation.

709 As discussed in Sec. 4.1.2, the Omnivisor provides the knobs to regulate the memory  
 710 bandwidth of different managers in the system by leveraging a QoS and MemGuard imple-



■ **Figure 6** Execution time slowdown of simple periodic task running in a VM over both RPU-0 and RISC-V soft-core. The behavior of the applications under different sources of interference is shown first when using a plain jailhouse, then a partial Omnivisor implementation only with spatial isolation mechanisms enabled, and finally the full Omnivisor implementation.

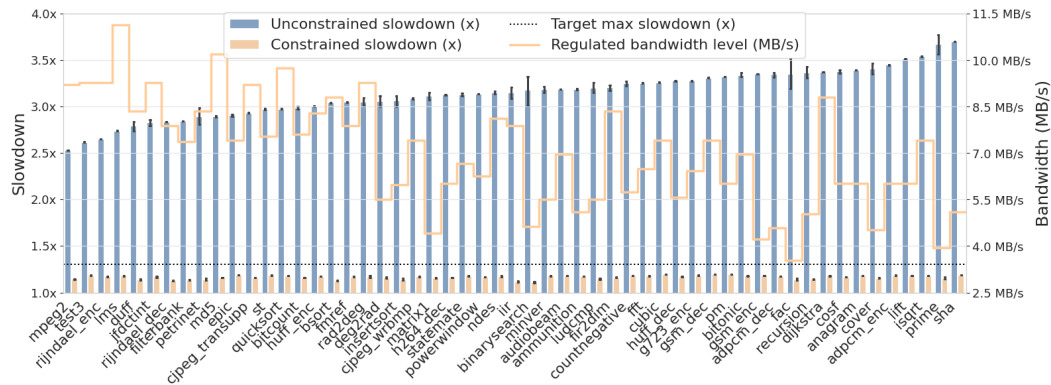
711 mentation. Since there are already papers exploring these mechanisms in detail [67, 32, 69],  
 712 in this experiment, we are interested in demonstrating that the Omnivisor can use these  
 713 mechanisms to reduce the temporal interference caused on a VM running on remote cores.

714 To isolate the VM running on rCPUs from the other managers, the Omnivisor first  
 715 configures the QoS for the FPGA and RPUs channels. In this experiment, each channel has  
 716 a request rate bounded to 11, which, using the formula from [69], translates to a memory  
 717 bandwidth of 4.7 MB/s. Regarding the APU, on the other hand, we enabled a MemGuard  
 718 regulation of 78 cache refills each millisecond for all the cores, which corresponds to having  
 719 4.997 MB/s of available bandwidth. Combining the two approaches strongly reduces the  
 720 performance impact on the rCPUs, as shown in Fig. 6d. Specifically, the maximum slowdown  
 721 drops from 142% to 7% on RPU and from 85% to 6% on RISC-V.

722 Integrating state-of-the-art monitoring and profiling applications into real safety-critical  
 723 systems is often sidestepped in favor of legacy methods. This hesitation primarily stems from  
 724 the difficulty in demonstrating that these applications don't disrupt the temporal behavior  
 725 of the critical application under observation. However, with the Omnivisor, integrating  
 726 such mechanisms becomes significantly easier, thanks to the utilization of a fully temporally  
 727 isolated VM running on remote cores.

### 728 6.3 Parameter Tuning for Controlled Degradation

729 To comprehensively evaluate and demonstrate the usability of the Omnivisor beyond synthetic  
 730 benchmarks, we execute a realistic benchmark suite on the remote cores. Specifically, our



(a) Benchmarks on VMs running on RPU.



(b) Benchmarks on VMs running on RISC-V.

■ **Figure 7** Comparative evaluation of TACLeBench: The bar plots depict the execution time slowdown with and without temporal constraints. Each benchmark showcases the bandwidth limitation imposed on other managers to achieve the desired 20% maximum degradation on the VM.

731 choice has gone towards using the benchmark set called TACLeBench provided in [31]. It  
 732 is a collection of 56 benchmark programs from several research groups and tool vendors  
 733 worldwide. However, while we were able to execute all the benchmarks on the RPU, due to  
 734 the limitations related to the absence of a floating-point extension of the RISC-V processor  
 735 (Pico32) deployed on FPGA, we used a subset of them for our RISC-V experiments.

736 The objective of this evaluation is twofold: first, to demonstrate how the Omnivisor  
 737 can induce controlled degradation in the execution time of a VM running on remote cores,  
 738 and second, to elucidate how the Omnivisor streamlines the parameter tuning process for  
 739 achieving an acceptable performance degradation level. Therefore, we first determine the  
 740 bandwidth allocation required to ensure unrestricted memory transactions on every manager.  
 741 Specifically, leveraging findings from [69] and experimental evaluations, we established that  
 742 a bandwidth limit of 950 MB/s for each manager is sufficient to maintain a comparable rate  
 743 of memory transfers to what was observed without regulation. Then, using these values as a  
 744 starting point, we developed a script iterating the execution of the benchmarks employing a  
 745 binary search algorithm to calculate the bandwidth allocation parameters. Specifically, we  
 746 search for those parameters that ensure a maximum slowdown of 20% for each benchmark.  
 747 Still, the script is generic and can be used to find the parameters for any value of degradation.  
 748 The slowdown is calculated in comparison with the observed maximum execution time over  
 749 thirty repetitions of the benchmarks without any interference. Furthermore, in between each  
 750 change of parameters, we execute thirty repetitions and consider the worst result as the target

751 value for the slowdown; when the target value is below the decided threshold, we consider  
 752 the bandwidth allocation quota used in that iteration as a possible candidate. However, we  
 753 stop the binary search after 15 iterations or when the slowdown is strictly between 19% and  
 754 20%. Fig. 7 presents the slowdown over thirty repetitions for each benchmark under two  
 755 scenarios. First, the slowdown without any bandwidth regulation is depicted. Next, the  
 756 case where the bandwidth is configured to incur at most a 20% degradation is shown. In  
 757 the same figure, we can also observe the level of bandwidth regulation in MB/s applied to  
 758 obtain the controlled degradation for each benchmark. The results demonstrate that it is  
 759 possible to achieve the desired slowdown even when the unconstrained slowdown exceeds  
 760 350%, provided you are willing to significantly constrain the rest of the system (e.g., max  
 761 bandwidth limit of 4MB/s).

762 Naturally, given specific application constraints, an ad-hoc policy that chooses the  
 763 parameters based on the importance of the VMs can be implemented to further improve the  
 764 utilization of the cores while maintaining the real-time guarantees of critical applications [32].

## 765 **7 Related**

766 **Partitioning Systems:** Numerous real-time hypervisors and microkernels proposed in the  
 767 literature are engineered with partitioning techniques aiming to explicitly meet certifications  
 768 such as ARINC-653 and AUTOSAR [72, 15]. Instead, the Omnivisor distinguishes itself  
 769 by offering partitioning with spatio-temporal isolation for a diverse range of processor  
 770 categories. Unlike works such as [83], which propose a partitioning microkernel-based design  
 771 targeting microcontrollers-level cores, and [72], which propose an ARINC-653 scheduling  
 772 on Xen focusing microprocessor-level cores, Omnivisor addresses the challenge of applying  
 773 partitioning to asymmetric core platforms by leveraging different isolation mechanisms for  
 774 each category in a coordinated manner. Although this work's focal point is not about  
 775 certification, the Omnivisor aims at establishing the blueprint of a partitioning hypervisor  
 776 for heterogeneous systems which is the first step for future certification endeavors.

777 **Asymmetric Multi-Core Architectures:** The management of asymmetric multi-core  
 778 architectures is a well-explored field within the systems software community, which has  
 779 proposed OS designs [9, 10, 43, 13] and hypervisors [37, 59] capable of fully leveraging  
 780 heterogeneous platforms. However, these existing works are not directly comparable to the  
 781 Omnivisor, since they often overlook the isolation challenges that heterogeneous cores can  
 782 introduce, making them unsuitable for mixed-criticality scenarios. In [12] the authors discuss  
 783 the challenges and opportunities of asymmetric architectures, proposing the OpenAMP  
 784 framework as a solution for remote core communication and power management. Despite  
 785 the framework is not meant for mixed-criticality, the works in [26, 60] and [4] explore the  
 786 possibility of using such a framework in critical scenarios. Both approaches focus on real-time  
 787 communication with remote cores, overlooking the interference between cores. In contrast,  
 788 the Omnivisor aims to provide spatio-temporal isolation between asymmetric cores, offering  
 789 a complementary solution that will incorporate real-time communication in the future.

790 **MPSoCs Hypervisors:** Some recent works have been proposing techniques to virtualize  
 791 heterogeneous platforms featuring programmable logic (FPGA) as well as heterogeneous  
 792 processors, to realize reliable mixed-criticality systems. Moratelli *et al.* propose a real-  
 793 time full-virtualization technique for MPSoCS [52]. While this work provides a solution  
 794 to run unmodified software on a traditional hypervisor with real-time requirements, the  
 795 Omnivisor is an extension for partitioning systems where the resources are statically allocated  
 796 to virtual machines and there is no need for schedulers. Gracioli *et al.* [34] explore the

797 capability to run mixed-criticality systems in MPSoCs where an SPH is deployed on APUs  
798 to isolate resources. The paper outlines how the rich hardware features provided by modern  
799 heterogeneous SoCs can reduce the contentions between partitioned applications. However,  
800 while this work analyzes the optimal utilization of heterogeneous resources such as diverse  
801 scratchpad memories, aspects not considered in our work, it overlooks the threat posed  
802 by unrestrained microcontroller-level CPUs. In contrast, Omnivisor focuses specifically on  
803 addressing temporal and spatial isolation issues between asymmetric cores and it also offers  
804 flexible and seamless control over remote cores through the hypervisor. CHIPS-AHOy is  
805 a predictable holistic hypervisor [53] that aims to satisfy temporal predictability and high-  
806 performance requirements of software running over MPSoCs while simultaneously handling  
807 energy efficiency, thermal bound, and system lifetime. The authors' goal is to address the  
808 most relevant source of unpredictability in MPSoCs, such as the memory hierarchy, the I/O  
809 subsystem, and the hardware variability, by using techniques such as cache coloring and I/O  
810 throttling. However, the authors do not provide a common interface to manage heterogeneous  
811 VMs and neither consider using bandwidth regulation mechanisms to improve temporal  
812 isolation. Biondi *et al.* present the SPHERE project [14], an integrated framework to abstract  
813 the hardware complexity of MPSoCs and simplify the management of heterogeneous hardware.  
814 The work explores the interesting possibility of using the dynamic partial reconfiguration  
815 of the FPGA to provide efficient implementations for cryptography modules, as well as  
816 hardware acceleration for deep neural networks in a hypervisor-based system. However, the  
817 authors do not explore asymmetric ISA cores as the Omnivisor, and instead focus solely  
818 on accelerators. While there is a strong effort in the literature to develop virtualization  
819 systems that utilize FPGA, existing works primarily focus on sharing the FPGA among  
820 Virtual Machines running on the main cores [75, 46]. In contrast, Omnivisor acknowledges  
821 the presence of cores in FPGA, which run entire and isolated VMs.

822 Although the Omnivisor model has similar objectives to those described in related work,  
823 that is, to realize a mixed-criticality system with strong real-time guarantees for critical  
824 VMs and to streamline the use of heterogeneous systems, it may be distinguished primarily  
825 by three points. First, it is the first hypervisor model that considers running isolated  
826 VMs on cores with heterogeneous ISAs as equal from the point of view of the hypervisor  
827 interface. This simplifies the adoption of such complex platforms and improves the overall  
828 system reliability. Secondly, unlike other solutions, it dynamically coordinates a combination  
829 of modern heterogeneous hardware protection mechanisms at runtime (including MMU,  
830 SMMU, SMPU/SPPU, and QoS) to provide spatial-temporal isolation to heterogeneous  
831 cores, transparently to the user. Finally, it is the first approach that considers using the  
832 soft-cores deployed on FPGA as isolated domains where to run VMs.

## 833 **8 Conclusions**

834 The increasing complexity of next-generation industrial applications has led to the widespread  
835 adoption of feature-rich heterogeneous MPSoCs. However, as the number of features within a  
836 single hardware platform increases, so does the complexity of deployment and the challenges  
837 of maintaining temporal guarantees for software. In this paper, we have introduced the  
838 Omnivisor, a novel model that extends static partitioning hypervisors to manage heterogen-  
839 eous processing elements within asymmetric architectures. Our experimental results have  
840 demonstrated that deploying this model on a real system enables the seamless deployment  
841 of virtual machines on cores with heterogeneous ISAs (ARM and RISC-V) within a single  
842 platform, even if some or all are implemented as soft-cores in FPGA. Furthermore, the solu-

tion ensures robust spatial and temporal isolation of VMs, achieved through a combination of software/hardware mechanisms. Additionally, we have showcased how the Omnivisor enhances the user’s control over MPSoCs. Specifically, we utilized Omnivisor features to precisely regulate the degradation of a real-time virtual machine executing on a remote core.

For future research directions we intend to (1) integrate a library of remote core utilities sourced from open-source scientific works in order to enhance the monitoring and management capabilities of MPSoCs. Following this (2), we aim to elevate the flexibility of these platforms to the next level by introducing dynamic FPGA hardware reconfiguration at the hypervisor level. Our objective is to integrate the capability to reconfigure portions (tiles) of the programmable logic as an additional Omnivisor feature, enabling the instantiation of soft-cores ad-hoc and on the fly to launch a VM with specific requirements.

Overall, our work showcases the potential of the Omnivisor in addressing the challenges posed by modern industrial applications, offering a promising solution for the efficient utilization of heterogeneous MPSoCs.

## References

- 1 Fardin Abdi Taghi Abad, Renato Mancuso, Stanley Bak, Or Dantsker, and Marco Caccamo. Reset-based recovery for real-time cyber-physical systems with temporal safety constraints. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2016.
- 2 Luca Abeni and Dario Faggioli. Using xen and kvm as real-time hypervisors. *Journal of Systems Architecture*, 106:101709, 2020.
- 3 Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I Davis. A comprehensive survey of industry practice in real-time systems. *Real-Time Systems*, 58(3):358–398, 2022.
- 4 Sara Alonso, Jesus Lazaro, Jaime Jimenez, Leire Muguira, and Unai Bidarte. Evaluating the OpenAMP framework in real-time embedded SoC platforms. In *2021 XXXVI Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 1–6. IEEE, 2021.
- 5 AMD. Microblaze reference guide. [https://www.amd.com/content/dam/xilinx/support/documents/sw\\_manuals/xilinx2021\\_2/ug984-vivado-microblaze-ref.pdf](https://www.amd.com/content/dam/xilinx/support/documents/sw_manuals/xilinx2021_2/ug984-vivado-microblaze-ref.pdf). [Accessed 21-02-2024].
- 6 ARM. PSCI specification — developer.arm.com. <https://developer.arm.com/Architectures/Power%20State%20Coordination%20Interface>. [Accessed 20-02-2024].
- 7 Giuseppe Avon, Arturo Buscarino, André C Neto, and Filippo Sartori. MARTe2 embedded signal processing unit for the ITER magnetics diagnostics. In *IECON 2021–47th Annual Conference of the IEEE Industrial Electronics Society*, pages 1–6. IEEE, 2021.
- 8 Victor Bandur, Gehan Selim, Vera Pantelic, and Mark Lawford. Making the case for centralized automotive e/e architectures. *IEEE Transactions on Vehicular Technology*, 70(2):1230–1245, 2021.
- 9 Antonio Barbalace, Robert Lyster, Christopher Jelesnianski, Anthony Carno, Ho-Ren Chuang, Vincent Legout, and Binoy Ravindran. Breaking the boundaries in heterogeneous-ISA datacenters. *ACM SIGARCH Computer Architecture News*, 45(1):645–659, 2017.
- 10 Antonio Barbalace, Marina Sadini, Saif Ansary, Christopher Jelesnianski, Akshay Ravichandran, Cagil Kendir, Alastair Murray, and Binoy Ravindran. Popcorn: Bridging the programmability gap in heterogeneous-ISA platforms. In *Proceedings of the Tenth European Conference on Computer Systems*, pages 1–16, 2015.
- 11 Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *ACM SIGOPS operating systems review*, 37(5):164–177, 2003.

- 891 12 Felix Baum and Arvind Raghuraman. Making full use of emerging ARM-based heterogeneous  
892 multicore SoCs. In *8th European Congress on Embedded Real Time Software and Systems*  
893 (*ERTS 2016*), 2016.
- 894 13 Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon  
895 Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhanian. The multikernel: a new  
896 os architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd*  
897 *symposium on Operating systems principles*, pages 29–44, 2009.
- 898 14 Alessandro Biondi, Daniel Casini, Giorgiomaria Cicero, Niccolò Borgioli, Giorgio Buttazzo,  
899 Gaetano Patti, Luca Leonardi, Lucia Lo Bello, Marco Solieri, Paolo Burgio, et al. SPHERE:  
900 A Multi-SoC architecture for next-generation cyber-physical systems based on heterogeneous  
901 platforms. *IEEE Access*, 9:75446–75459, 2021.
- 902 15 Gedare Bloom and Joel Sherrill. Harmonizing arinc 653 and realtime posix for conformance  
903 to the face technical standard. In *2020 IEEE 23rd international symposium on real-time*  
904 *distributed computing (ISORC)*, pages 98–105. IEEE, 2020.
- 905 16 Paolo Burgio, Marko Bertogna, Nicola Capodieci, Roberto Cavicchioli, Michal Sojka, Přemysl  
906 Houdek, Andrea Marongiu, Paolo Gai, Claudio Scordino, and Bruno Morelli. A software stack  
907 for next-generation automotive systems on many-core heterogeneous platforms. *Microprocessors*  
908 *and Microsystems*, 52:299–311, 2017.
- 909 17 Alan Burns and Robert I Davis. A survey of research into mixed criticality systems. *ACM*  
910 *Computing Surveys (CSUR)*, 50(6):1–37, 2017.
- 911 18 Alan Burns and Robert Ian Davis. Mixed criticality systems-a review:(february 2022). York,  
912 2022.
- 913 19 Jon Perez Cerrolaza, Roman Obermaisser, Jaume Abella, Francisco J Cazorla, Kim Grüttner,  
914 Irune Agirre, Hamidreza Ahmadian, and Imanol Allende. Multi-core devices for safety-critical  
915 systems: A survey. *ACM Computing Surveys (CSUR)*, 53(4):1–38, 2020.
- 916 20 Ravikumar V Chakaravathy, Hyun Kwon, and Hua Jiang. Vision control unit in fully self  
917 driving vehicles using Xilinx MPSoC and opensource stack. In *Proceedings of the 26th Asia*  
918 *and South Pacific Design Automation Conference*, pages 311–317, 2021.
- 919 21 Weifan Chen, Ivan Izhibirdeev, Denis Hoornaert, Shahin Roozkhosh, Patrick Carpanedo,  
920 Sanskriti Sharma, and Renato Mancuso. Low-overhead online assessment of timely progress  
921 as a system commodity. In *35th Euromicro Conference on Real-Time Systems (ECRTS 2023)*.  
922 Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.
- 923 22 Alessandro Cilardo, Marcello Cinque, Luigi De Simone, and Nicola Mazzocca. Virtualization  
924 over multiprocessor system-on-chip: an enabling paradigm for industrial iot. *arXiv preprint*  
925 *arXiv:2112.15404*, 2021.
- 926 23 Marcello Cinque, Domenico Cotroneo, Luigi De Simone, and Stefano Rosiello. Virtualizing  
927 mixed-criticality systems: A survey on industrial trends and issues. *Future Generation*  
928 *Computer Systems*, 2021.
- 929 24 Marcello Cinque, Luigi De Simone, Nicola Mazzocca, Daniele Ottaviano, and Francesco Vitale.  
930 Evaluating virtualization for fog monitoring of real-time applications in mixed-criticality  
931 systems. *Real-Time Systems*, 59(4):534–567, 2023.
- 932 25 Marcello Cinque, Gianmaria De Tommasi, Sara Dubbioso, and Daniele Ottaviano. Virtualizing  
933 real-time processing units in multi-processor systems-on-chip. In *2021 IEEE 6th International*  
934 *Forum on Research and Technology for Society and Industry (RTSI)*, pages 329–333. IEEE,  
935 2021.
- 936 26 Marcello Cinque, Gianmaria De Tommasi, Sara Dubbioso, and Daniele Ottaviano. RPUGuard:  
937 Real-time processing unit virtualization for mixed-criticality applications. In *2022 18th*  
938 *European Dependable Computing Conference (EDCC)*, pages 97–104. IEEE, 2022.
- 939 27 Edoardo Cittadini, Mauro Marinoni, Alessandro Biondi, Giorgiomaria Cicero, and Giorgio  
940 Buttazzo. Supporting ai-powered real-time cyber-physical systems on heterogeneous platforms  
941 via hypervisor technology. *Real-Time Systems*, pages 1–27, 2023.

- 942 **28** David J Coe, Jeffrey H Kulick, Aleksandar Milenkovic, and Letha Etkorn. Virtualized in situ  
943 software update verification: verification of over-the-air automotive software updates. *IEEE*  
944 *Vehicular Technology Magazine*, 15(1):84–90, 2019.
- 945 **29** Diogo Costa, Luca Cuomo, Daniel Oliveira, Ida Maria Savino, Bruno Morelli, Jose Martins,  
946 Fabrizio Tronci, Alessandro Biasci, and Sandro Pinto. IRQ coloring: Mitigating interrupt-  
947 generated interference on ARM multicore platforms. In *Fourth Workshop on Next Generation*  
948 *Real-Time Embedded Systems (NG-RES 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.
- 950 **30** Domenico Cotroneo, Luigi De Simone, and Roberto Natella. On temporal isolation assessment  
951 in virtualized railway signaling as a service systems. In *2022 IEEE Intl Conf on Dependable,*  
952 *Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl*  
953 *Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress*  
954 *(DASC/PiCom/CBDCCom/CyberSciTech)*, pages 1–5. IEEE, 2022.
- 955 **31** Heiko Falk, Sebastian Altmeyer, Peter Hellinckx, Björn Lisper, Wolfgang Puffitsch, Christine  
956 Rochange, Martin Schoeberl, Rasmus Bo Sørensen, Peter Wägemann, and Simon Wegener.  
957 Taclebench: A benchmark collection to support worst-case execution time research. In *16th*  
958 *International Workshop on Worst-Case Execution Time Analysis*, 2016.
- 959 **32** Sergio Garcia-Esteban, Alejandro Serrano-Cases, Jaume Abella, Enrico Mezzetti, and Fran-  
960 cisco J Cazorla. Quasi isolation QoS setups to control MPSoC contention in integrated software  
961 architectures. In *35th Euromicro Conference on Real-Time Systems (ECRTS 2023)*. Schloss  
962 Dagstuhl-Leibniz-Zentrum für Informatik, 2023.
- 963 **33** Google. Dev-Board Coral datasheet. <https://coral.ai/docs/dev-board/datasheet>. [Ac-  
964 cessed 21-02-2024].
- 965 **34** Giovanni Gracioli, Rohan Tabish, Renato Mancuso, Reza Mirosanlou, Rodolfo Pellizzoni, and  
966 Marco Caccamo. Designing mixed criticality applications on modern heterogeneous MPSoC  
967 platforms. In *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Schloss  
968 Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
- 969 **35** Yuri Gribov, Andrey Kavin, Victor Lukash, Rustam Khayrutdinov, Guido Huijsmans, Alberto  
970 Loarte, Joseph A. Snipes, and Luca Zabeo. Plasma vertical stabilisation in ITER. *Nuclear*  
971 *Fusion*, 55(7):073021, 2015.
- 972 **36** Domenik Helms, Patrick Uven, and Kim Grüttner. Modular over-the-air software updates for  
973 safety-critical real-time systems. *INSIGHT*, 25(4):85–88, 2022.
- 974 **37** Yu-Ju Huang, Hsuan-Heng Wu, Yeh-Ching Chung, and Wei-Chung Hsu. Building a kvm-based  
975 hypervisor for a heterogeneous system architecture compliant system. In *Proceedings of the 12th*  
976 *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages  
977 3–15, 2016.
- 978 **38** Intel. ACPI specification — intel.com. [https://www.intel.com/  
979 content/dam/www/public/us/en/documents/product-specifications/  
980 processor-vendor-specific-acpi-specification.pdf](https://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/processor-vendor-specific-acpi-specification.pdf). [Accessed 20-02-2024].
- 981 **39** Shravan Karthik, Karthik Ramanan, Nikhil Devshatwar, Subhajit Paul, Vishal Mahaveer,  
982 Sheng Zhao, Manoj Vishwanathan, and Chetan Matad. Hypervisor based approach for  
983 integrated cockpit solutions. In *2018 IEEE 8th International Conference on Consumer*  
984 *Electronics-Berlin (ICCE-Berlin)*, pages 1–6. IEEE, 2018.
- 985 **40** Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the linux virtual  
986 machine monitor. In *Proceedings of the Linux symposium*, volume 1, pages 225–230. Dttawa,  
987 Dntorio, Canada, 2007.
- 988 **41** Heiko Koziolok, Andreas Burger, and Abdulla Puthan Peedikayil. Fast state transfer for  
989 updates and live migration of industrial controller runtimes in container orchestration systems.  
990 *Journal of Systems and Software*, page 112004, 2024.
- 991 **42** Yoojin Lim and Hyoseung Kim. Cache-aware real-time virtualization for clustered multi-core  
992 platforms. *IEEE Access*, 7:128628–128640, 2019.



- 993 43 Felix Xiaozhu Lin, Zhen Wang, and Lin Zhong. K2: A mobile operating system for heterogenous coherence domains. *ACM SIGPLAN Notices*, 49(4):285–300, 2014.
- 994
- 995 44 Remote Processor Framework; The Linux Kernel documentation — docs.kernel.org. <https://docs.kernel.org/staging/remoteproc.html>. [Accessed 07-02-2024].
- 996
- 997 45 Tamara Lugo, Santiago Lozano, Javier Fernández, and Jesus Carretero. A survey of techniques for reducing interference in real-time applications on multicore platforms. *IEEE Access*, 10:21853–21882, 2022.
- 998
- 999
- 1000 46 Jiacheng Ma, Gefei Zuo, Kevin Loughlin, Xiaohe Cheng, Yanqiang Liu, Abel Mulugeta Eneyew, Zhengwei Qi, and Baris Kasikci. A hypervisor for shared-memory fpga platforms. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 827–844, 2020.
- 1001
- 1002
- 1003
- 1004 47 José Martins and Sandro Pinto. Shedding light on static partitioning hypervisors for ARM-based mixed-criticality systems. In *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 40–53. IEEE, 2023.
- 1005
- 1006
- 1007 48 José Martins, Adriano Tavares, Marco Solieri, Marko Bertogna, and Sandro Pinto. Bao: A lightweight static partitioning hypervisor for modern multi-core embedded systems. In *Workshop on next generation real-time embedded systems (NG-RES 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
- 1008
- 1009
- 1010
- 1011 49 Miguel Masmano, Ismael Ripoll, Alfons Crespo, and J Metge. Xtratum: a hypervisor for safety critical embedded systems. In *11th Real-Time Linux Workshop*, volume 9. Citeseer, 2009.
- 1012
- 1013
- 1014 50 Minerva. Jailhouse-RT GitLab repository. [https://gitlab.com/minervasys/public/jailhouse/-/tree/minerva/public?ref\\_type=heads](https://gitlab.com/minervasys/public/jailhouse/-/tree/minerva/public?ref_type=heads). [Accessed 08-02-2024].
- 1015
- 1016 51 Eric Missimer, Richard West, and Ye Li. Distributed real-time fault tolerance on a virtualized multi-core system. *OSPERS 2014*, page 17, 2014.
- 1017
- 1018 52 Carlos Moratelli, Samir Zampiva, and Fabiano Hessel. Full-virtualization on mips-based mpsoes embedded platforms with real-time support. In *Proceedings of the 27th Symposium on Integrated Circuits and Systems Design*, pages 1–7, 2014.
- 1019
- 1020
- 1021 53 Tiago Mück, Antonio A Fröhlich, Giovanni Gracioli, Amir M Rahmani, João Gabriel Reis, and Nikil Dutt. CHIPS-AHOy: A predictable holistic cyber-physical hypervisor for MPSoCs. In *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 73–80, 2018.
- 1022
- 1023
- 1024
- 1025 54 Djob Mvondo, Boris Teabe, Alain Tchana, Daniel Hagimont, and Noel De Palma. Closer: A new design principle for the privileged virtual machine os. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 49–60. IEEE, 2019.
- 1026
- 1027
- 1028
- 1029 55 Mattia Nicoletta, Shahin Roozkhosh, Denis Hoornaert, Andrea Bastoni, and Renato Mancuso. Rt-bench: An extensible benchmark framework for the analysis and management of real-time applications. In *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, pages 184–195, 2022.
- 1030
- 1031
- 1032
- 1033 56 NVIDIA. Jetson AGX Orin technical brief. <https://www.nvidia.com/content/dam/en-zz/Solutions/gtc/gtc21/jetson-orin/nvidia-jetson-agx-orin-technical-brief.pdf>. [Accessed 21-02-2024].
- 1034
- 1035
- 1036 57 NVIDIA. Jetson Xavier Series. <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/>. [Accessed 21-02-2024].
- 1037
- 1038 58 NXP. i.MX8-series processors. <https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i-mx-applications-processors/i-mx-8-applications-processors:IMX8-SERIES>. [Accessed 07-02-2024].
- 1039
- 1040
- 1041 59 Pierre Olivier, Binoy Ravindran, and Antonio Barbalace. The multihype: Virtualizing heterogeneous-ISA architectures. In *9th Workshop on Systems for Multi-core and Heterogeneous Architectures (SFMA)*, 2019.
- 1042
- 1043

- 1044 **60** D Ottaviano, M Cinque, G Manduchi, and S Dubbioso. Virtualization of accelerators in  
 1045 embedded systems for mixed-criticality: RPU exploitation for fusion diagnostics and control.  
 1046 *Elsevier Fusion Engineering and Design*, 2023.
- 1047 **61** Daniele Ottaviano. The Omnivisor Source Code. [https://github.com/DanieleOttaviano/  
 1048 Omnivisor](https://github.com/DanieleOttaviano/Omnivisor), 2024. Accessed: May 7, 2024.
- 1049 **62** Shrinivas Anand Panchamukhi and Frank Mueller. Providing task isolation via tlb coloring.  
 1050 In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 3–13.  
 1051 IEEE, 2015.
- 1052 **63** Ralf Ramsauer, Jan Kiszka, Daniel Lohmann, and Wolfgang Mauerer. Look mum, no vm  
 1053 exits!(almost). *arXiv preprint arXiv:1705.06932*, 2017.
- 1054 **64** Ralf Ramsauer, Jan Kiszka, and Wolfgang Mauerer. A novel software architecture for mixed  
 1055 criticality systems. In *Digital Transformation in Semiconductor Manufacturing: Proceedings  
 1056 of the 1st and 2nd European Advances in Digital Transformation Conference, EADTC 2018,  
 1057 Zittau, Germany and EADTC 2019, Milan, Italy*, pages 121–128. Springer International  
 1058 Publishing, 2020.
- 1059 **65** Falk Rehm, Jörg Seitter, Jan-Peter Larsson, Selma Saidi, Giovanni Stea, Raffaele Zippo, Dirk  
 1060 Ziegenbein, Matteo Andreozzi, and Arne Hamann. The road towards predictable automotive  
 1061 high-performance platforms. In *2021 Design, Automation & Test in Europe Conference &  
 1062 Exhibition (DATE)*, pages 1915–1924. IEEE, 2021.
- 1063 **66** Gero Schwäricke, Rohan Tabish, Rodolfo Pellizzoni, Renato Mancuso, Andrea Bastoni, Al-  
 1064 exander Zuepke, and Marco Caccamo. A real-time VirtIO-based framework for predictable  
 1065 inter-VM communication. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, pages 27–40.  
 1066 IEEE, 2021.
- 1067 **67** Alejandro Serrano-Cases, Juan M Reina, Jaume Abella, Enrico Mezzetti, and Francisco J  
 1068 Cazorla. Leveraging hardware QoS to control contention in the Xilinx Zynq UltraScale+  
 1069 MPSoC. In *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*. Schloss  
 1070 Dagstuhl-Leibniz-Zentrum für Informatik, 2021.
- 1071 **68** J. A. Snipes et al. ITER plasma control system final design and preparation for first plasma.  
 1072 *Nuclear Fusion*, 2021.
- 1073 **69** Parul Sohal, Rohan Tabish, Ulrich Drepper, and Renato Mancuso. Profile-driven memory  
 1074 bandwidth management for accelerators and CPUs in QoS-enabled platforms. *Real-Time  
 1075 Systems*, 58(3):235–274, 2022.
- 1076 **70** Stefano Stabellini. Xen project blog. [https://xenproject.org/2019/12/16/  
 1077 true-static-partitioning-with-xen-dom0-less/](https://xenproject.org/2019/12/16/true-static-partitioning-with-xen-dom0-less/), 2019. [Accessed 27-02-2024].
- 1078 **71** Prathap Kumar Valsan, Heechul Yun, and Farzad Farshchi. Taming non-blocking caches to  
 1079 improve isolation in multicore real-time systems. In *2016 IEEE Real-Time and Embedded  
 1080 Technology and Applications Symposium (RTAS)*, pages 1–12. IEEE, 2016.
- 1081 **72** Steven H VanderLeest. Designing a future airborne capability environment (face) hypervisor for  
 1082 safety and security. In *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*,  
 1083 pages 1–9. IEEE, 2017.
- 1084 **73** Steven H VanderLeest and Dagan White. Mpsoc hypervisor: The safe & secure future of  
 1085 avionics. In *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, pages  
 1086 6B5–1. IEEE, 2015.
- 1087 **74** Richard West, Ye Li, Eric Missimer, and Matthew Danish. A virtualized separation kernel  
 1088 for mixed-criticality systems. *ACM Transactions on Computer Systems (TOCS)*, 34(3):1–41,  
 1089 2016.
- 1090 **75** Tian Xia, Ye Tian, Jean-Christophe Prévotet, and Fabienne Nouvel. Ker-one: A new hypervisor  
 1091 managing fpga reconfigurable accelerators. *Journal of Systems Architecture*, 98:453–467, 2019.
- 1092 **76** Xilinx. AXI Traffic Generator LogiCORE IP Product Guide (PG125). [https://docs.xilinx.  
 1093 com/r/en-US/pg125-axi-traffic-gen/Introduction](https://docs.xilinx.com/r/en-US/pg125-axi-traffic-gen/Introduction). [Accessed 12-02-2024].
- 1094 **77** Xilinx. Versal Device Technical Reference Manual. [https://docs.xilinx.com/r/en-US/  
 1095 am011-versal-acap-trm](https://docs.xilinx.com/r/en-US/am011-versal-acap-trm). [Accessed 07-02-2024].

- 1096 **78** Xilinx. Zynq Ultrascale+ Device Technical Reference Manual. <https://docs.xilinx.com/r/en-US/ug1085-zynq-ultrascale-trm>. [Accessed 07-02-2024].
- 1097
- 1098 **79** YosysHQ. picorv32. <https://github.com/YosysHQ/picorv32>. [Accessed 27-02-2024].
- 1099 **80** Heechul Yun, Renato Mancuso, Zheng-Pei Wu, and Rodolfo Pellizzoni. PALLOC: DRAM  
1100 bank-aware memory allocator for performance isolation on multicore platforms. In *2014 IEEE*  
1101 *19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 155–166.  
1102 IEEE, 2014.
- 1103 **81** Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. Memguard: Memory  
1104 bandwidth reservation system for efficient performance isolation in multi-core platforms. In  
1105 *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*,  
1106 pages 55–64. IEEE, 2013.
- 1107 **82** Alexander Zuepke, Andrea Bastoni, Weifan Chen, Marco Caccamo, and Renato Mancuso.  
1108 MemPol: Policing core memory bandwidth from outside of the cores. In *2023 IEEE 29th*  
1109 *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 235–248.  
1110 IEEE, 2023.
- 1111 **83** Alexander Zuepke, Marc Bommert, and Daniel Lohmann. Autobest: a united autosar-os and  
1112 arinc 653 kernel. In *21st IEEE real-time and embedded technology and applications symposium*,  
1113 pages 133–144. IEEE, 2015.