

Surgical Software-less I/O Virtualization

Francesco Ciraolo

Boston University, USA
fciraolo@bu.edu

Mattia Nicoletta

Boston University, USA
mnico@bu.edu

Patrick Carpanedo

Boston University, USA
pfcarp21@bu.edu

Denis Hoornaert

TU München, Germany
denis.hoornaert@tum.de

Marco Caccamo

TU München, Germany
marco.caccamo@tum.de

Renato Mancuso

Boston University, USA
rmancuso@bu.edu

Abstract

Virtualizing I/O devices presents unique challenges compared to other system resources. Traditional approaches rely on software-based abstraction layers, which can be particularly complex to develop and often fail to provide guests with efficient access to the underlying hardware. The most common solutions involve dedicating an I/O device exclusively to a single guest or modifying/patching the guest software to utilize hardware-level I/O multiplexing. This paper introduces I/O Softwareless Nano-Virtualization (IO-SNV), a novel approach that achieves efficient and transparent I/O virtualization with minimal overhead and no software modifications. IO-SNV operates entirely at the hardware level, leveraging programmable logic to dynamically virtualize I/O devices while maintaining high performance. We present the conceptual model, a proof-of-concept implementation, and an evaluation of its feasibility. Our promising results demonstrate that IO-SNV can provide seamless I/O virtualization while preserving device access efficiency, making a compelling alternative to existing software-centric solutions.

ACM Reference Format:

Francesco Ciraolo, Mattia Nicoletta, Patrick Carpanedo, Denis Hoornaert, Marco Caccamo, and Renato Mancuso. 2025. Surgical Software-less I/O Virtualization. In *The 4th International Workshop on Real-time and IntelliGent Edge computing (RAGE '25)*, May 6–9, 2025, Irvine, CA, USA. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3722567.3727847>

1 Introduction

Prompted by a demand for performance and efficiency, modern embedded System-on-Chips (SoCs) are becoming increasingly heterogeneous. In addition to CPUs, they may feature GPUs, TPUs, fixed hardware accelerators, and re-programmable fabrics. SoCs featuring a tightly integrated

reprogrammable fabric (often an Field Programmable Gate Array—FPGA) have attracted attention in recent years due to their versatility, seeing concrete use cases in space exploration systems [16] and nuclear fusion plants [1]. These use cases exemplify safety-critical real-time systems, which are characterized by their ability to guarantee the execution of a desired set of workloads within strict timing constraints. The difficulty of managing heterogeneous SoC in safety-critical contexts has prompted the successful adoption of traditional and partitioning hypervisors. While software hypervisors offer efficient management of processing elements and memory resources [13, 14], one particular challenge remains: the virtualization of Input/Output (I/O) devices. This topic is subject to extensive research, with two main strategies being employed in practice. First, in a split-I/O model, the hypervisor manages the devices and exports a virtualized interface to the virtual machines (VM). Second, in a passthrough I/O model, the hypervisor exclusively maps an I/O device to the VM. As such, the methods proposed by previous research [20] to perform I/O virtualization are inherently software-based, offering access to the I/O devices of interest in three possible ways: (i) by granting direct access to the device as per the application needs (non-coordinated, passthrough); (ii) via abstraction layers provided by an OS (i.e., system calls); and (iii) via the hypervisor as a proxy (i.e., guest throwing hypercalls, split-I/O). Unfortunately, the aforementioned methods come at the cost of long access times or uncoordinated access. Approaches to establishing a better tradeoff have been studied [8, 17], as we briefly survey in Section 2. These approaches, however, still possess shortcomings, such as the less-than-ideal tradeoff between the inability of the OS/hypervisor to control the device and sizable I/O access overheads, paired with the explicit need for specialized hardware and/or additional software support. This paper proposes I/O Softwareless Nano-Virtualization (IO-SNV), a methodology to enact dynamic I/O virtualization in hardware by leveraging cache coherent programmable logic. IO-SNV allows seamlessly redirecting memory-mapped I/O accesses to a generic hardware proxy with negligible overhead. Thus, IO-SNV unlocks the ability to **virtualize a memory-mapped I/O device seamlessly by hijacking its mapping to a hardware proxy**. This



This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

RAGE '25, Irvine, CA, USA

© 2025 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-1611-9/2025/05

<https://doi.org/10.1145/3722567.3727847>

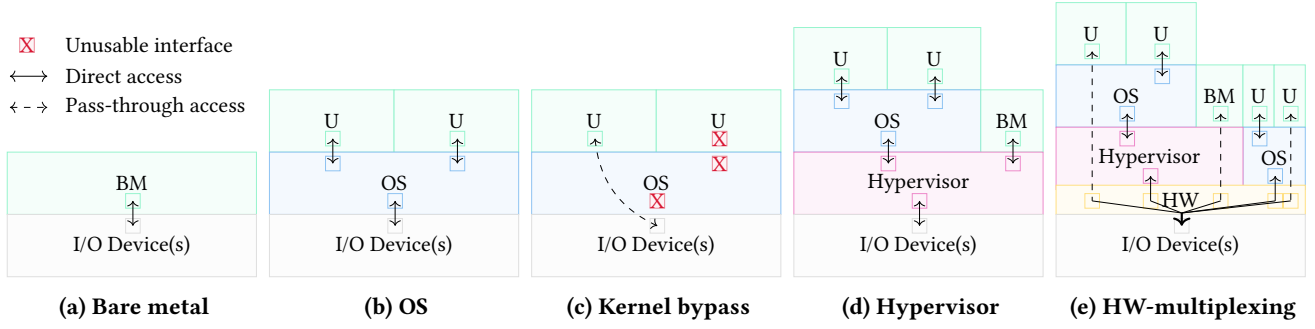


Figure 1: Overview of typical interaction paradigms with an I/O device; user-space software (U) differs from bare-metal (BM) as it accesses the I/O device via OS system calls instead of directly mapping its registers. Without IO-SNV, HW multiplexing (e) is only possible if the device offers virtualized sub-devices, e.g., using SR-IOV [17].

introduces negligible overhead and offers the VMs direct access to the I/O device. Moreover, it prevents the OS or hypervisor from completely relinquishing control of the device. We defined this method as *surgical* since IO-SNV does not require any virtualization layer and can be applied to a single device without changes to the overall system.

2 Motivation & related Works

Traditional methods for providing and accelerating virtualization/isolation rely on virtual interfaces accessed through hypervisor or library calls. Fig. 1 summarizes the main existing approaches. In bare-metal systems (Fig. 1a), no virtualization is performed, while a typical OS (Fig. 1b) or hypervisor implementing split-I/O drivers (Fig. 1d) directly interact with I/O devices while exporting high-level interfaces to user-space applications. Next, modern implementations of kernel bypass techniques such as DPDK [8], IX [2], and SPDK [19] correspond to Fig. 1c. They offer well-structured frameworks designed to streamline and scale up the development of user-space applications that directly interface with I/O devices to handle I/O requests efficiently. Hardware vendors recently introduced Single-Root I/O Virtualization (SR-IOV) depicted in Fig. 1e. The rationale of SR-IOV is to allow complex devices (e.g., network interfaces, GPUs) to be shared by multiple VMs. This is done in hardware by exporting multiple virtual devices from a single I/O device that can then be mapped in passthrough to guest VMs and user-space applications [17]. Support for SR-IOV is, therefore, device-specific and is implemented via hardware multiplexing through custom ASICs co-designed with the I/O device itself.

Although these traditional approaches to decreasing overhead in virtualized/isolated environments have matured, they offer tradeoffs that might compromise their usage on some platforms. DPDK and other kernel bypass approaches offer lower overheads but burden the programmer through

driver or library integration. They might also rely on expensive polling operations and offer limited multiplexing capabilities. SR-IOV's lack of overhead comes at the cost of single OS usage with hardware-embedded limits to multiplexing capabilities. The community has long debated the trade-offs between the different virtualization approaches in [7, 15]. As a response, a trend of FPGA-facilitated hardware multiplexing work has emerged, leveraging memory-mapped devices. FPGA-assisted Virtual Device Emulation (FVM) [11] provides hardware-level virtualization of block devices to alleviate contention and overhead to resolve the scaling of NVME-based storage. In a similar effort, but at a more granular level, the File System for Reconfigurable Fabrics (FSRF) [12] implements an I/O abstraction for OSes that are fully executed in FPGA. I/O-GUARD [10] proposes a system architecture that integrates a hypervisor microarchitecture, implemented in programmable hardware, for device drivers and virtualization logic to attain platform-wide SR-IOV performance with additional flexibility. BlueIO [9] instead delves into providing I/O drivers, isolation, and scheduling configured in FPGAs for embedded systems running RTOS. vFPIO [3] offers an approach similar to BlueIO but assumes that the FPGA is connected through PCIe. In contrast, IO-SNV tailors the SNV [5] capabilities to provide seamless redirection of memory-mapped I/O requests to a generic hardware proxy, allowing compatibility with previously mentioned HW/SW methods for I/O devices.

3 Background

3.1 CPU+FPGA platforms

CPU+FPGA SoCs are industry-standard enablers for complex cyber-physical systems. In these SoCs, a multi-core computing cluster (CPU) and programmable logic (FPGA) are co-located. Typically, the CPUs in the SoC share a single LLC cache. The SoC also includes a cache coherence manager that is tasked to resolve LLC misses by querying

other components in the system, typically the DRAM. The FPGA is usually connected to the CPU cluster and accessible through a memory-mapped interface at a specific physical address aperture. Recent platforms [4, 6] allow the FPGA to also be part of the same cache coherence domain as the main CPUs. Therefore, in this configuration, the FPGA is actively queried when CPU-originated LLC misses have to be resolved. Adopting the terminology of cache coherence protocols, we say that the FPGA is *snooped* when a CPU-side LLC access results in a cache miss.

3.2 Softwareless Nano-Virtualization

Softwareless Nano-Virtualization (SNV) is a recently-proposed idea [5] which refers to the possibility of manipulating the Virtual Address (VA) to Physical Address (PA) translation processes on-the-fly and with negligible overhead. This is not done by changing the underlying page tables nor by introducing extra translation stages as per traditional virtualization extensions. Instead, such a manipulation is enacted at the cache coherence level. Via SNV, one can modify the target PA with a custom one and/or alter the attributes of the mapping—e.g., access permissions, memory type, *etc.*

This alteration performed by SNV occurs in tight coupling with the Cache Coherent Interconnect (CCI) and offers some valuable perks: (i) it can be enacted to modify the VA-to-PA translation of any software layer—bare metal, OSs, hypervisors—that use virtual memory; (ii) it is applied at run time with no change in the software code; (iii) it does not alter the content of the page tables in main memory—therefore avoiding complex bookkeeping; (iv) rollbacks (e.g., deactivation of SNV) only requires to flush TLB and cache lines related to the translation(s) of interest.

4 System Model and Design

IO-SNV, as described in this paper, builds atop SNV. As such, we assume a CPU+FPGA platform. As discussed in Section 3, this corresponds to a SoC architecture featuring an FPGA (reprogrammable logic) tightly coupled with a set of CPUs (processing elements). We further assume that the latter support virtual memory via a Memory Management Unit (MMU). The FPGA is said to be tightly coupled as it can (i) be made part of the CPUs' coherence domain and (ii) directly interact with the rest of the SoC's hardware resources via memory transactions. A coherent interconnect (CCI) glues together the CPUs and the FPGA. The CCI enables the routing and monitoring of all the snoop transactions, regular memory requests, and (memory-mapped) I/O transactions. In particular, the inclusion of MMU-originated page table walks in the coherence domain is an essential requirement for IO-SNV. Indeed, this allows the FPGA to be snooped for any main

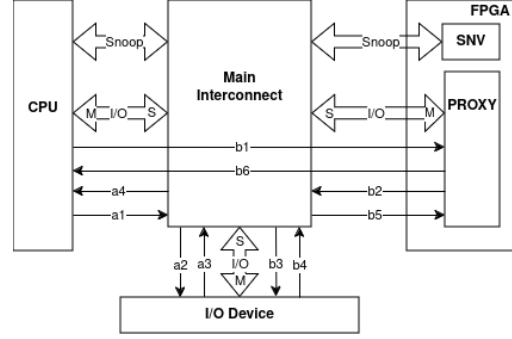


Figure 2: Data paths with and without IO-SNV.

memory access occurring during a page table walk. Overall, the system model assumed for IO-SNV is similar to SNV [5].

Unlike the work in [5], we further assume the instantiation of a “I/O device proxy” module co-located with the SNV module in the FPGA. The objective of the module is to offer an alternative access route to the desired I/O device. We assume that said proxy offers a similar hardware-level interface to the I/O device behind it. We do not impose any assumptions on the exact interface or device type being exported and note that a number of previous works [8–11, 17] have already explored multiple alternatives. The element of novelty offered by the proposed IO-SNV, instead, is in its ability to dynamically activate/deactivate routing of software layers through I/O proxies without software intervention.

Indeed, a significant advantage of using SNV-like mechanisms is the ability to operate independently from the software layer. The only pieces of information required for SNV to work are (i) the translation base address of the address space of interest and (ii) the virtual address of the memory-mapped registers of the target I/O device. If SNV is performed to dynamically virtualize I/O devices used by a user-space process, these pieces of information can be provided to the SNV hardware by the OS. In other situations—e.g., when virtualizing the OS or hypervisor—they can be retrieved by inspecting CPU control registers and scanning the corresponding translation table trees. As such, we impose little-to-no assumptions about the nature of the software layers.

4.1 IO-SNV Design

In order to provide a high-level understanding of the operating principle at the core of the proposed IO-SNV, consider Fig. 2. The figure depicts the key module involved in a IO-SNV deployment. On the left-hand side of the figure, the CPU block refers to the CPU cluster where the target software component is deployed. Any CPU-originated cacheable transaction that missed in the cluster's LLC triggers a *snoop* request to the CCI—labeled as “Main Interconnect” in Fig. 2. The CCI forwards snoop transactions to the SNV module in

the cache-coherent FPGA (right-hand side). Non-cacheable I/O transactions are routed through the I/O interfaces.

Consider now a software component with direct access to the I/O device of interest. This can be a regular user-space program in bare-metal or using kernel-bypassing techniques, a typical OS-level device driver, or the hardware-facing side of a hypervisor implementing a split-I/O driver. One such software component would access the I/O device registers by performing *non-cacheable device reads/writes*. These are labeled as transactions *a1*, *a2*, *a3*, and *a4* in Fig. 2. Due to their non-cacheable nature, such operations are directly routed to the memory-mapped device and not snooped to the FPGA. Thus, the FPGA is unable to manipulate the content of such accesses.

To overcome the issue and enable IO-SNV, we leverage SNV to manipulate the VA-to-PA mapping of the I/O device to be virtualized in the target address space (user-space application, OS, or hypervisor). Upon activation, IO-SNV invalidates any TLB entry for the I/O device aperture. It also invalidates any CPU-side cache line containing page table entries involved in the VA-to-PA translation of the device aperture of interest. As such, the first translation for any of the I/O addresses after the activation of the IO-SNV module will be intercepted by SNV. This is because MMU-originated page table walks generate cacheable transactions that are snooped to the FPGA. At this stage, the SNV module manipulates the content of the entries returned to the MMU during the walk and redirects the results of the translation to the desired I/O proxy's aperture. This behavior can be dynamically forced at any time by issuing invalidation commands for the related PA's using the same coherence channel.

The software mapping the I/O device is unaware of the redirection. It continues to operate as expected, but it now maps to the I/O proxy aperture—transactions *b1* and *b6* in Fig. 2. Thus, it is fundamental for the proxy to have the same functionality as the originally mapped I/O device. Indeed, the proxy is now in charge of actually interacting with the mapped device—transactions *b2*, *b3*, *b4*, *b5* in Fig. 2.

Since the proxy isolates the two sides of the communication, not all the transactions *b1/b6* have to be forwarded to the I/O device, and the proxy can also generate internal transactions to the device itself. Different granularity and policies could coexist as independent proxies, and IO-SNV can quickly and seamlessly switch among them by manipulating the translation of the software pointing to one or another proxy's aperture.

5 Implementation

As a prototyping CPU+FPGA platform, we use the Zynq UltraScale+ MPSoC ZCU102 development board. The board's SoC is equipped with four ARM Cortex-A53 CPU cores and

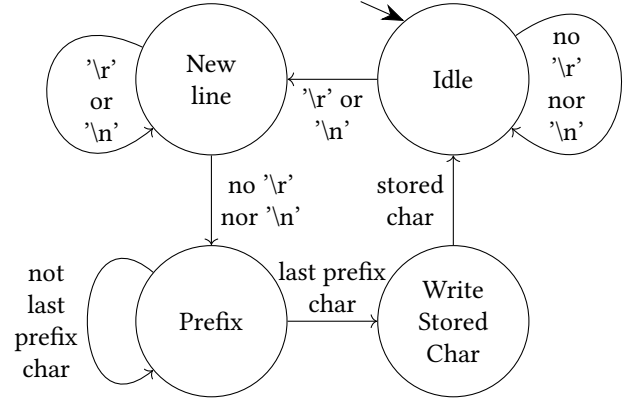


Figure 3: UART proxy state machine: prefix a string before printing it to the corresponding UART device.

1 MB of shared unified L2 cache. In addition, the SoC features an FPGA as reprogrammable fabric. Both elements are linked together and to the main memory (*i.e.*, DRAM controller) via a Cache Coherent Interconnect (the ARM CCI-400 [18]). The system's CCI implements and enforces coherence via two AXI Coherency Extension (ACE) ports connected to the CPU core cluster and the FPGA. It is the second port that allows the soft logic hosted on the FPGA to be integrated into the coherence domain. In addition, the CPU cores can directly access FPGA-hosted targets (and vice versa) via an exposed set of memory-mapped Advance eXtensible Interface (AXI) ports. These ports allow the FPGA to access various I/O devices present on the SoC, such as the UART controllers.

IO-SNV Hardware Prototype. To implement our prototype of IO-SNV, two modules must be implemented in the FPGA as soft logic. The first module is the SNV that is tethered to the FPGA-facing coherence port (*i.e.*, the ACE port). The second module is a I/O device proxy as defined in Section 4. It interfaces with the remaining of the system via two AXI ports: a *manager* and a *subordinate*. The manager port allows the proxy to access the address range and the registers of the I/O device it manages. Finally, the subordinate port receives various CPU-originated requests aiming at the virtualized I/O devices. Hence, the exposed PA range must be at least as wide as the memory range of the I/O device it virtualizes. The IO-SNV prototype operates at 100MHz and requires only a tiny share of FPGA resources (3% of LUTs, 2% of LUTRAM, 2% of FFs, and less than 1% of BUGF).

UART proxy. The UART proxies described in this section keep a 1-to-1 mapping between the actual I/O devices and

their physical aperture but can offer any layer of policies. In most existing approaches, if the hardware module offers different policies, the switch from one policy to another requires additional logic and some kind of configuration port to communicate the required one. IO-SNV transitions the proxies in a paradigm in which each one has its own policy, and multiple of them can coexist simultaneously. The selection between them is performed in a *stateless* fashion by simply replacing the PA of the virtualized devices with the address of one of the proxies. Since this work focuses on the aforementioned paradigm shift, the proxy functionalities are left unspecified since any policy could be applied. Nevertheless, for the experiment, we designed a simple proxy able to change policies based on the contents of the FIFO queue used to write in the UART device. the proof-of-concept proxy is designed to allow line-based prefixing on text-only communication. The proxy, shown in Figure 3, is in "Idle" state forwarding characters to the UART until a newline character is received; then it transitions to "New line" to handle further newline characters. When a different character is received after the newline, the state transitions to "Prefix" and the received character is stored; then the prefix will be forwarded to the UART, and the state transition to "Write stored char"; the stored character is forwarded and the state moves back to "Idle". Two HW proxies, **P0** and **P1**, are implemented using the state machine described above, one per each UART device on the prototyping platform, matching them by indexes.

IO-SNV Software Layer. As mentioned earlier, the proposed IO-SNV does not require drastic software modifications. Nonetheless, some precise configuration tweaks and specialized kernel modules are required for this specific experiment. The platform is operated by a vanilla Linux kernel (version 6.1), and no hypervisor is involved.

A custom kernel module must be inserted to pass a process' Page Global Directory address from the kernel to the user space. Moreover, the ARM Trusted Firmware must be altered to include the FPGA in the coherence domain; that way, snoop requests can be placed on the FPGA's ACE port by the CCI. The SNV module itself can be configured via a dedicated port that can be read and written using any program.

6 Evaluation

This section presents the initial set of experiments we carried out to investigate the feasibility of IO-SNV and discuss some limitations of our initial prototype.

For these two experiments, we instrument the SNV module to redirect accesses targeting the two UART controllers to the UART multiplexer described in Section 5. In other words, the experiments directly aim to demonstrate the ability of SNV to virtualize an I/O devices of choice. The two UART

controllers (noted UART0 and UART1) have been selected for three main reasons: (i) the interface is simple and straightforward; (ii) the module is resilient to mis-manipulations, and its misuse does not cause damages to the system; and (iii) the expected outcome can be immediately assessed using a serial terminal. Despite its simplicity, this proof-of-concept experiment provides an essential basic feasibility test. It also provides valuable insights into how the technique can be applied to more complex I/O devices in the near future.

6.1 User Space Virtualization

The experiment's objective is to demonstrate that at any given point in time, it is possible with IO-SNV to seamlessly insert an I/O proxy between a user-space process and the UARTs. To this end, we designed a simple user-space process named *screamer* that directly utilizes UART0 by memory-mapping its aperture (kernel bypassing) and writing data to the device every second. When started, both the screamer and the OS (*i.e.*, Linux) write on the UART0 without any established coordination. Upon enabling IO-SNV during the *screamer* execution, its expected output can appear on UART0 or UART1 with a prefix indicating also the UART's id.

The positive outcome of this experiment highlights the soundness of IO-SNV's model; furthermore, the system does not appear to be negatively affected by multiple consecutive remapping operations, and the UART's output behavior is always as expected.

6.2 Kernel Space Virtualization

To overcome SNV's current limitation in manipulating the kernel space's translations and test the IO-SNV ability to virtualize the kernel's I/O, we performed a raw alteration of translation tables in main memory.

This workaround circumvented the SNV's issue and showed that the I/O virtualization could also be applied kernel-side. However, this experiment also uncovered an additional problem: the PTE's manipulation has to be performed from an SSH console, and no input to the UART0 console can be provided while the IO-SNV module is active. If these constraints are respected, the UART virtualization works as expected.

The root of this issue can be found in the inadequate handling of the hardware interrupts generated by the I/O device. Indeed, the read path of the UART is not virtualized by our simple proxy module, and it could be performed in pass-through; on the other hand, the interrupts originated by the UART are not altered by our technique, and we believe this mismatch is responsible for the system's crashes.

Interrupts virtualization under SNV has not yet been explored and will be the subject of our future work.

7 Conclusions and Future work

In this groundbreaking work, we explored the potential and feasibility of the I/O Softwareless Nano-Virtualization (IO-SNV) and demonstrated its feasibility as a lightweight, flexible approach to I/O virtualization. The overhead of our technique will be tested on future and more mature drivers. The main strengths of IO-SNV are (i) its runtime adaptability, enabling rapid and seamless remapping of I/O devices, and (ii) its ability to function without requiring software modifications. These characteristics make IO-SNV a compelling alternative to conventional I/O virtualization techniques with promising future use cases such as hypervisor-free virtualization and transactional multiplexing for multiple virtual machines on the same physical device.

A natural next step is to extend our infrastructure to incorporate hardware interrupt handling. Addressing this challenge would significantly enhance IO-SNV's applicability to a broader range of I/O devices that heavily rely on interrupts—e.g., timers, event-driven peripherals. An evolution of the proposed technique that does not rely on a data path loop through the PL but only on the coherence channels is also on our roadmap. We envision that these improvements will push IO-SNV toward greater scalability and efficiency, opening new possibilities for high-performance, low-latency virtualization in embedded and cyber-physical systems.

Acknowledgments

This research was supported by the National Science Foundation (NSF) under grants number CSR-2238476 and SHF-2403012, and by Red Hat Research under award number 2025-01-RH01. Marco Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research.

References

- [1] G. Avon, A. Buscarino, A. C. Neto, and F. 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, 2021.
- [2] A. Belay, G. Prekas, A. Klimovic, S. Grossman, C. Kozyrakis, and E. Bugnion. IX: A protected dataplane operating system for high throughput and low latency. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 49–65, Broomfield, CO, Oct. 2014. USENIX Association.
- [3] J. Chen, H. Unnibhavi, A. Koshiba, and P. Bhatotia. vFPIO: A virtual I/O abstraction for FPGA-accelerated I/O devices. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 1167–1184, Santa Clara, CA, July 2024. USENIX Association.
- [4] Y.-k. Choi, J. Cong, Z. Fang, Y. Hao, G. Reinman, and P. Wei. A Quantitative Analysis on Microarchitectures of Modern CPU-FPGA Platforms. In *Proceedings of the 53rd Annual Design Automation Conference, DAC '16*, New York, NY, USA, 2016. Association for Computing Machinery.
- [5] F. Ciraolo, M. Nicoletta, D. Hoornaert, M. Caccamo, and R. Mancuso. Light virtualization: a proof-of-concept for hardware-based virtualization, 2025. <https://arxiv.org/abs/2502.15738>.
- [6] D. Cock, A. Ramdas, D. Schwyn, M. Giardino, A. Turowski, Z. He, N. Hossle, D. Korolija, M. Licciardello, K. Martsenko, et al. Enzian: an open, general, cpu/fpga platform for systems software research. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 434–451, 2022.
- [7] H. Golestani, A. Mirhosseini, and T. F. Wenisch. Software data planes: You can't always spin to win. In *Proceedings of the ACM Symposium on Cloud Computing, SoCC '19*, page 337–350, New York, NY, USA, 2019. Association for Computing Machinery.
- [8] Intel Corporation, 6WIND. Optimized data plane processing solutions using the Intel DPDK. White paper, Intel Corporation, 2014.
- [9] Z. Jiang, N. Audsley, and P. Dong. Blueio: A scalable real-time hardware i/o virtualization system for many-core embedded systems. *ACM Trans. Embed. Comput. Syst.*, 18(3), Apr. 2019.
- [10] Z. Jiang, K. Yang, Y. Ma, N. Fisher, N. Audsley, and Z. Dong. I/o-guard: Hardware/software co-design for i/o virtualization with guaranteed real-time performance. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 1159–1164, 2021.
- [11] D. Kwon, J. Boo, D. Kim, and J. Kim. FVM: FPGA-assisted virtual device emulation for fast, scalable, and flexible storage virtualization. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 955–971. USENIX Association, Nov. 2020.
- [12] J. Landgraf, M. Giordano, E. Yoon, and C. J. Rossbach. Reconfigurable virtual memory for fpga-driven i/o. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ASPLOS 2023*, page 556–571, New York, NY, USA, 2023. Association for Computing Machinery.
- [13] J. Martins and S. 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, 2023.
- [14] D. Ottaviano, F. Ciraolo, R. Mancuso, and M. Cinque. The Omnivisor: A Real-Time Static Partitioning Hypervisor Extension for Heterogeneous Core Virtualization over MPSoCs. In R. Pellizzoni, editor, *36th Euromicro Conference on Real-Time Systems (ECRTS 2024)*, volume 298 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:27, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- [15] B. J. Patrick Kutch. SR-IOV for NFV Solutions Practical Considerations and Thoughts. Technical report, 2017. accessed on Feb. 2025.
- [16] A. Perez, A. Rodríguez, A. Otero, D. G. Arjona, Á. Jiménez-Peralo, M. Á. Verdugo, and E. de la Torre. Run-time reconfigurable mpso-based on-board processor for vision-based space navigation. *IEEE Access*, 8:59891–59905, 2020.
- [17] J. Suzuki, Y. Hidaka, J. Higuchi, T. Baba, N. Kami, and T. Yoshikawa. Multi-root share of single-root i/o virtualization (sr-iov) compliant pci express device. In *2010 18th IEEE Symposium on High Performance Interconnects*, pages 25–31, 2010.
- [18] Xilinx. ARM® CoreLink™ CCI-400 Cache Coherent Interconnect. Technical report, 2015.
- [19] Z. Yang, J. R. Harris, B. Walker, D. Verkamp, C. Liu, C. Chang, G. Cao, J. Stern, V. Verma, and L. E. Paul. Spdk: A development kit to build high performance storage applications. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 154–161, 2017.
- [20] B. Zhang, X. Wang, R. Lai, L. Yang, Y. Luo, X. Li, and Z. Wang. A survey on i/o virtualization and optimization. In *2010 Fifth Annual ChinaGrid Conference*, pages 117–123, 2010.