Holistic RISC-V Virtualization: CVA6-based SoC

Bruno Sá, Francisco Marques, Manuel Rodriguez, José Martins, Sandro Pinto
Centro ALGORITMI/LASI - University of Minho, Portugal
id10037@alunos.uminho.pt, *pg47200@alunos.uminho.pt*, *pg47436@alunos.uminho.pt*,
jose.martins@dei.uminho.pt, *sandro.pinto@dei.uminho.pt*

***Abstract*—This work describes our efforts to provide a holistic hardware RISC-V virtualization SoC based on the CVA6 core. At the core level, we implemented hardware support for virtualization through the ratified Hypervisor instruction set architecture (ISA) extension version 1.0. At the system level, we are working on providing reference open-source IPs for two non-ISA components needed to build a virtualization-aware platform: (i) the advanced interrupt architecture (AIA) to enable hardware support for interrupt virtualization; (ii) the input/output memory management unit (IOMMU) to protect memory accesses from direct memory access (DMA) devices. All these IPs will be open and freely available to the RISC-V community under permissive open-source licenses.**

I. Introduction

The use of virtualization technology has become ubiquitous in modern computing, spanning a diverse range of applications such as cloud computing, mobile devices, and embedded systems. In the embedded space, virtualization has emerged as a fundamental approach to consolidate and isolate several systems of different criticality levels - a.k.a mixed-criticality systems (MCS) - into a single platform to meet the market demands to minimize size, weight, power, and cost (SWaP-C) [1], [2].

The RISC-V architecture is a growing open-standard ISA that emerged with a promise to disrupt the computer hardware industry. Unlike its competitors, RISC-V is a royalty-free ISA built to be highly modular and customizable that can scale from small embedded computers to supercomputers.

In this work, we share our experience providing hardware support for virtualization in a RISC-V CVA6-based SoC. In summary, our contributions are as follows. At the core level, we extended the CVA6 core [3] to support the Hypervisor extension version 1.0 and the RISC-V supervisor-level timer extension, specifically "stimecmp/vstimecmp" (Sstc). Moreover, we complemented the design with a few virtualization-oriented microarchitecture enhancements: (i) a G-stage TLB (GTLB); and (ii) a second-level TLB (L2 TLB). At the system level, we are working on implementing two open-source IPs compliant with the RISC-V standard specifications: (i) the AIA; (ii) the IOMMU. Finally, we open-sourced the virtualization-enabled CVA6 [1] and upstreamed it to the OpenHW CVA6 main repository [2]. The IOMMU and AIA [3] will be finished and freely available with very permissive licenses in Q2 2023. Our main goal is to democratize virtualization in RISC-V and empower the next generation of CVA6-based SoC with virtualization capabilities.

[1]https://github.com/minho-pulp/cva6
[2]https://github.com/minho-pulp/cva6/tree/feat/hyp-upstream
[3]https://github.com/minho-pulp/aia

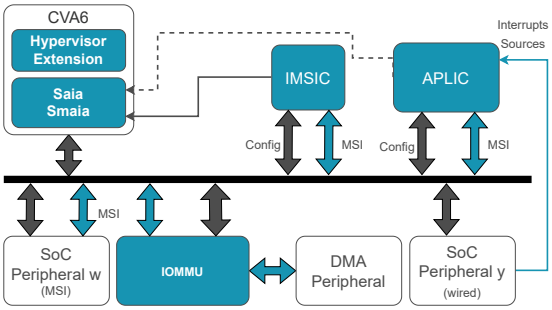

Fig. 1. Example of a single core CVA6-based SoC with virtualization capabilities (i.e., Hypervisor extension, IOMMU and AIA)

II. CVA6-based SoC Virtualization Support: Overview

The RISC-V architecture defines three crucial components to create a virtualization-aware platform, divided into core and system-level technologies. In Figure 1, we present an example of a possible CVA6-Based SoC with full virtualization support. At the core level, the RISC-V privileged architecture defines hardware support for virtualization through their Hypervisor extension. Hypervisors can leverage this extension to build efficient implementations. At the system level, the following two components are essential: (i) the AIA [4]; and (ii) the IOMMU [5]. The AIA is the new RISC-V standard interrupt controller specification that, unlike its predecessor platform-level interrupt controller (PLIC), enables hardware interrupt virtualization via messaged signal interrupt (MSI). This mechanism allows hypervisors to deliver interrupts directly to the guest avoiding overheads of emulating the interrupt controller [2]. Finally, the IOMMU is the RISC-V standard specification for translating and restricting memory accesses from bus masters devices (e.g., crypto accelerators). The IOMMU prevents malicious DMA-capable peripherals to access memory freely and break encapsulation. In a virtualized environment, among other features, the hypervisor can use the IOMMU to safely allow the direct assignment of these devices to the guest VM, thus avoiding device emulation [1].

III. CVA6 Core Virtualization Support: Status and Features

To support virtualization in the CVA6 [3] core, we implemented the following RISC-V standard extensions: (i) the Hypervisor extension; and (ii) the RISC-V Sstc extension. Although the Sstc extension is not mandatory for virtualization support, we argue that it is critical to achieve reasonable performance results (up to 11% performance speedup in our

[4]https://github.com/riscv/riscv-aia
[5]https://github.com/riscv-non-isa/riscv-iommu

TABLE I. AIA AND IOMMU FEATURE STATUS: ● FULLY-IMPLEMENTED; ◐ PARTIALLY IMPLEMENTED; ○ NOT IMPLEMENTED.

IOMMU	Memory-based device context (DC) and process context (PC)	●
	Two-stage address translation	●
	Address translation caches	●
	Memory-based queue interface to interact with software	◖
	Message-Signaled Interrupt (MSI) address translation	●
	Memory-Resident Interrupt Files (MRIF) support	○
	MSI or wired-signaled interrupt generation	◖
	Memory-mapped register interface	●
AIA	Smaia	●
	Ssaia	◖
	APLIC	●
	IMSIC	◖

evaluation [4]). Furthermore, we also complemented our implementation with a few micro-architectural enhancements to the nested-MMU subsystem.

A. CVA6 Architectural Extensions

The virtualization-aware CVA6 implementation complies with the mandatory features of standard RV64 Hypervisor extension specification version 1.0. Nevertheless, some optional features were left unimplemented or partially unimplemented mainly due to dependencies on other standard RISC-V extensions (e.g., for the *envcfg* CSRs, the CBIE bit depends on the standard RISC-V cache management instructions, not supported in the CVA6 yet). We also complemented the CVA6 implementation with support for the RISC-V Sstc extension to alleviate timer emulation overheads.

B. CVA6 Virtualization-Oriented Enhancements

The RISC-V Hypervisor extension defines a second stage of translation to translate guest physical addresses (GPA) to host physical addresses (HPA). In our work [4], we demonstrated that this extra stage is one of the major causes of performance loss in the CVA6 core [4]. To tackle this, we designed a collection of microarchitectural optimizations to reduce the virtualization overhead. Firstly, we designed a small second-stage TLB (GTLB) to store GPA to HPA and reduce the number of PTW iterations to complete a full double-stage translation. Secondly, we developed a set-associative second-level TLB (L2 TLB) designed to increase the TLB coverage and reduce the L1 TLB miss penalty. The L2 TLB follows a split design with two TLBs: (i) a larger TLB to store 4KiB page sizes; and (ii) a smaller TLB to hold 2MiB superpages. The size and associativity of each TLB are configurable, as well as the page size support (i.e., support 4KiB and/or 2MiB). We refer readers to [4] for more details on the performance speedup of these optimizations.

IV. RISC-V System-Level Virtualization

The RISC-V ISA has reached a new state of maturity, and the IOMMU and AIA specifications have now been ratified. As part of our efforts, we are developing two open-source IPs compliant with the RISC-V AIA and IOMMU standard specifications to be integrated into a CVA6-based SoC.

A. RISC-V IOMMU IP: Status and Features

The RISC-V IOMMU specification defines support for memory translation and protection for I/O devices with DMA capabilities while guaranteeing isolation between VMs and the hypervisor itself. For instance, the IOMMU can be used to allow the direct assignment of devices to a guest VM - a.k.a. device pass-through. As shown in Table I, our IOMMU IP is progressing towards a basic stable implementation with support for all mandatory features.

B. RISC-V AIA IP: Status and Features

The AIA is divided into three main components: (i) the APLIC which supersedes the PLIC; (ii) Core CSRS extensions (Smaia and Ssaia); and (iii) the IMSIC is a mandatory component to have hardware support for interrupt virtualization at the VS-level. So far, we have completed the implementation and functional validation of (i) the APLIC and (ii) the Smaia and Ssaia extensions. For the validation process, we used: (i) small functional tests; (ii) Linux baremetal (i.e., without a hypervisor), and (iii) Linux as VM guest atop the Bao hypervisor. Notwithstanding, the IMSIC IP has not been fully integrated and tested. As of this writing, we are working on integrating all three modules and performing more comprehensive validation tests.

V. Conclusion and Roadmap

We present our work in providing open-source IPs for building a CVA6-based SoC. We have completed the virtualization-aware CVA6 fully compliant with the Hypervisor extension version 1.0 and complemented the design with a set of virtualization-oriented enhancements to increase performance. Moving forward, we are finishing the implementation of the IOMMU and AIA IPs compliant with the RISC-V non-ISA standard specifications. Finally, we are focusing on evaluating and improving the AIA and IOMMU specifications in the context of MCS for the automotive domain. For instance, we want to explore if the AIA interrupt virtualization addresses the requirements of mixed-criticality embedded systems.

Acknowledgments

This work has been supported by Technology Innovation Institute (TII), and the FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope UIDB/00319/2020 and Scholarships Project Scope SFRH/BD/138660/2018 and SFRH/BD/07707/2021.

References

[1] J. Martins, A. Tavares, M. Solieri, M. Bertogna, and S. Pinto, "Bao: A Lightweight Static Partitioning Hypervisor for Modern Multi-Core Embedded Systems," in *Workshop on NG-RES*, vol. 77, 2020.

[2] B. Sa, J. Martins, and S. Pinto, "A First Look at RISC-V Virtualization from an Embedded Systems Perspective," *IEEE Transactions on Computers*, vol. 71, pp. 2177–2190, 2021.

[3] F. Zaruba and L. Benini, "The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, 2019.

[4] B. Sá, L. Valente, J. Martins, D. Rossi, L. Benini, and S. Pinto, "CVA6 RISC-V Virtualization: Architecture, Microarchitecture, and Design Space Exploration," 2023. [Online]. Available: https://arxiv.org/abs/2302.02969