RISC-V Virtualization for a CVA6-based SoC

Bruno Sá *, Luca Valente †, José Martins *, Davide Rossi †, Luca Benini † ‡, Sandro Pinto *
* Centro Algoritmi - University of Minho DEI † DEI, University of Bologna, Italy ‡ IIS lab, ETH Zurich, Switzerland
bruno.sa@algoritmi.uminho.pt, luca.valente@unibo.it, jose.martins@dei.uminho.pt,
davide.rossi@unibo.it, lbenini@iis.ee.ethz.ch, sandro.pinto@dei.uminho.pt

***Abstract*—In this work, we describe the implementation of the latest version of the RISC-V Hypervisor extension (v1.0) specification in a RISC-V CVA6-based (64-bit) SoC. We also report the results of performing an extensive evaluation on the current design and we share our experience about the design space exploration for a few microarchitectural optimizations to the memory subsystem. To complete, we have also enhanced the timer infrastructure by implementing the privileged timer Sstc extension. All these efforts we conducted in an attempt to improve performance without compromising area and power.**

I. Introduction and Contextualization

In the past, embedded systems were typically built of sparse components, physically isolated from each other, with a specific purpose and low computational power. Nowadays, embedded systems are becoming more complex, with software stacks steadily targeting heterogeneous multicore platforms with characteristics typically encountered in general-purpose systems and endowed with domain-specific accelerators. In this sense, virtualization has been playing a major role in the embedded industry and has been seen as a key enabling technology to consolidate and isolate various systems with different levels of critically - a.k.a. mixed-critically systems (MCS) - into the same hardware platform [1].

This work reports the architectural and microarchitectural implementation of the hardware virtualization support in RISC-V CVA6-based [2] (64-bit) SoC, in compliance with the RISC-V Hypervisor extension 1.0. We also performed an extensive evaluation and describe a set of optimizations to the memory and timer subsystems to enhance performance. The ultimate goal of the project is to develop a fully open-sourced RISC-V-based SoC architecture with virtualization support (at the core and system level) and accompanying software stack for adoption on a broad set of embedded and MCS applications, e.g., drones.

II. Project Status

The project started in May 2021 and about six months later we had already completed the mandatory part of the RISC-V Hypervisor extension specification. By that time, we started to complement the design with some optional features of H-extension, e.g. support for VMIDs in the memory management unit (MMU). Once we had a stable implementation, we focus our efforts on the evaluation and identification of performance bottlenecks in the design and possible solutions to mitigate them. As expected, we identified two subsystems as the major sources of performance degradation: (i) MMU subsystem, naturally due to the nested translation, and (ii) timer subsystem, mainly due to the trap and emulation burden while multiplexing timer access to other privilege modes (since the timer is only accessible in machine mode). For the MiBench automotive suite, we observed an average relative performance overhead of 8%, with 14% for the susanc-small benchmark. After careful consideration, we proposed different solutions to tackle the overhead caused by the nested MMU and the timer emulation, e.g. introducing a dedicated G-Stage TLB (G-TLB) to intermediate translations on the Page Table Walker (PTW) and implementing the recent Sstc extension proposal, which extends the timer facility to the HS- and VS-modes. For reference, the addition of the G-TLB has reduced the observed worst case relative performance overhead by approx. 25%, at the cost of an increase of approx. 1.5% of the hardware.

The validation and evaluation processes were carried out during all the development stages using: i) the Verilator RTL simulator tool and ii) an FPGA prototyping. We also conducted the PPA analysys targeting Global Foundries 22 nanometer FDSOI technology, resulting the hardware virtualization support in an 8.2% area increase. The software used to achieve fully-functional validation encompassed: (i) an ad-hoc baremetal framework; (ii) the nested MMU test-suite of the Xvisor white-box framework; and (iii) multiple hypervisors which already leverage the RISC-V Hypervisor extension, i.e. Bao [3], Xvisor, and KVM.

III. Conclusion and Next Steps

The project is still in is preliminary stages and, for now, our focus has been at the hart level. Nevertheless, there is still much work to be done at the system level to design a secure and fully virtualizable platform. For example, the current design provides only memory virtualization and protection at the core level which means, DMA capable devices can access memory freely. To tackle this, we plan to augment the SoC with an IOPMP and IOMMU. Furthermore, the current interrupt controller, i.e. PLIC, has no virtualization-awareness and is now being replaced by the Advanced Interrupt Architecture (AIA) [4].

References

[1] B. Sa, J. Martins, and S. E. S. Pinto, "A First Look at RISC-V Virtualization from an Embedded Systems Perspective," *IEEE Transactions on Computers*, pp. 1–1, 2021.

[2] 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.

[3] 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.

[4] "RISC-V Advanced Interrupt Architecture (AIA)," RISC-V, Mar. 2022, available online at: https://github.com/riscv/riscv-aia, last accessed on 29.03.2022.