Open-source RISC-V Input/Output Memory Management Unit (IOMMU) IP

Manuel Rodríguez*, Francisco Costa, Bruno Sá and Sandro Pinto

Centro ALGORITMI/LASI, Universidade do Minho, Portugal

Abstract

This work describes the design and implementation of an open-source IOMMU IP compliant with the ratified version of the RISC-V IOMMU specification (v1.0-rc1). So far, we have designed and implemented a basic IP encompassing only the mandatory features (of the spec) and support for virtualization. which has been successfully validated and evaluated on a single-core CVA6-based SoC. Moving forward, we plan to extend the IP with more advanced features, i.e., optional features such as a hardware performance monitor, memory-resident interrupt files support, etc. We will open-source our IP to the RISC-V community.

Introduction

The introduction of *Direct Memory Access (DMA)* transactions at the platform level has raised security concerns, as malicious peripherals with DMA capabilities could potentially undermine overall *System-on-Chips (SoC)* and their applications. To address this issue, the IOMMU hardware module started to be integrated into modern computers to mediate and manage device accesses to memory, performing permission checks and address translation on every DMA request. Recently, RISC-V froze its IOMMU specification, and the aim of this work is to implement an open-source IOMMU IP that meets this standard. We have implemented a basic IP that includes all mandatory features outlined in the specification and virtualization support. This version was validated in a single-core CVA6-based SoC [1]. We intend to make this basic IOMMU IP open-source under permissive licenses shortly, and furthermore, enhance it by adding advanced features.

RISC-V IOMMU in a Nutshell

The RISC-V IOMMU specification [2] defines a set of mandatory and optional features. Table 1 summarizes all the features considered for our implementation and categorizes them as basic or advanced.

Use Cases for Virtualization. The RISC-V IOMMU defines three methods for managing DMA-capable devices in virtualized systems. The first method, known as *device pass-through*, permits direct control of a device by a guest OS with minimal hypervisor intervention. Alternatively, a guest OS can share its process address space with devices, which allows guest applications to program the device using *IO Virtual Addresses (IOVA)*. Lastly, a host OS or hypervisor may choose to retain direct control of a device.

RISC-V IOMMU Feature	Classification
Memory-based device context (DC) and process context (PC).	★ ❶
Address translation caches.	❶
Two-stage address translation.	❶
Command Queue and Fault/Event Queue.	★ ❶
Message-Signaled Interrupt (MSI) address translation.	❶
Memory-Resident Interrupt Files (MRIF) support.	❷
Hardware Performance Monitor (HPM).	❷
MSI or Wired-Signaled Interrupt (WSI) generation.	★ ❶
Memory-mapped register interface.	★ ❶
Debug register interface.	❷

Table 1: *RISC-V IOMMU Features: (★) Mandatory; (❶) Included in both basic and advanced versions of the IOMMU IP; (❷) Included in advanced version only.*

Interaction with AIA. The IOMMU may optionally redirect MSIs from guest-controlled devices to the corresponding guest interrupt file in an IMSIC or to a MRIF. For this purpose, the IOMMU uses the MSI address translation data structures provided by the hypervisor and defined by the RISC-V AIA specification.

IOMMU IP: Features, Design and Status

The IOMMU IP's basic version includes all the essential features mandated by the RISC-V IOMMU specification, along with additional features needed to

*Corresponding author: `pg47436@alunos.uminho.pt`.

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

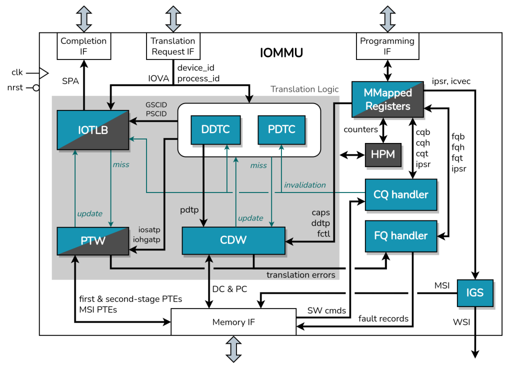

Figure 1: *IOMMU Internal Design: (●) Basic and Advanced; (●) Advanced only.*

provide virtualization support at the platform level. Figure 1 depicts the internal design of the IOMMU.

IOMMU Basic IP. In the basic version of the IOMMU IP, the hardware *Page-Table Walker (PTW)* module translates IOVAs using memory page tables. It supports *Bare* and Sv39/Sv39x4 address translation schemes and MSI address translation. The *Context Directory Walker (CDW)* module locates the DC and PC in memory and supports up to three-level context directory tables. Additionally, three fully-associative TLBs were included to accelerate address translations. We have also developed a handler module for each memory-based queue. The command queue handler processes commands stored in the queue by software, whereas the fault queue handler writes a new entry whenever a fault or event occurs and generates an interrupt to request software service.

IOMMU Advanced IP. The advanced version of the IOMMU IP will integrate MRIF support into the PTW. An HPM will enable counting performance-related events and, finally, a debug register interface will be implemented, empowering software to request explicit address translations.

IOMMU IP Status. At present, we have finished the basic version of the IOMMU IP. We have also integrated the IP in a CVA6-based SoC with support for virtualization [3] and validated its operation using a DMA device. We are currently in the process of implementing required components in the software stack to enable the use of the IOMMU IP by the Bao hypervisor[4].

Preliminary Evaluation

Functional Validation. To test the basic version of the IOMMU IP, we created a baremetal test framework application, which utilize a basic DMA device integrated into the SoC. This application was initially executed in the verilated model of the SoC. Subsequently, we synthesized the RTL code and ran the tests physically on an FPGA platform.

Configuration	Resource	Utilization
No IOMMU	LUT	76667/203800
	FF	53017/407600
WSI support only	LUT	94187 (+9.65%)
	FF	62979 (+2.86%)
WSI and MSI	LUT	95417 (+10.29%)
	FF	63143 (+2.90%)
WSI, MSI and PC	LUT	95816 (+10.41%)
	FF	63464 (+2.98%)

Table 2: *Hardware resources used by a single-core CVA6-based SoC implemented in a Genesys2 FPGA board with different IOMMU configurations.*

Hardware Results. Table 2 illustrates the resource utilization for different IOMMU configurations, indicating the percentage increase relative to the total resources available on the FPGA. Our analysis revealed that most IOMMU resources are consumed by the Address Translation Caches, as these were implemented using flip-flops. In our future work, we will consider alternative technologies for implementing these caches (e.g., BRAM), which would reduce the number of used flip-flops at the cost of some additional cycles of latency.

Roadmap

Regarding the following steps, we plan to add support for the RISC-V IOMMU in the open-source Bao hypervisor. Moving forward, we will extend the IOMMU IP with advanced features as aforementioned. Finally, we will analyze and evaluate the implemented IP, focusing on *Mixed-Criticality Systems (MCS)* for automotive.

Conclusion

This project is currently in preliminary stages, and efforts have been primarily focused on designing and implementing a basic and functional IOMMU IP compliant with the RISC-V IOMMU specification. We intend to release the basic version open-sourced to promote collaboration within the RISC-V community for the implementation of advanced features.

References

[1] Florian Zaruba and Luca 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". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27.11 (2019), pp. 2629–2640.

[2] IOMMU Task Group. *RISC-V IOMMU Specification Document.* RISC-V. 2023. URL: `https://github.com/riscv-non-isa/riscv-iommu/blob/main/riscv-iommu.pdf`.

[3] Bruno Sá et al. *CVA6 RISC-V Virtualization: Architecture, Microarchitecture, and Design Space Exploration.* 2023. URL: `https://arxiv.org/abs/2302.02969`.

[4] José Martins et al. "Bao: A Lightweight Static Partitioning Hypervisor for Modern Multi-Core Embedded Systems". In: *Workshop on NG-RES*. 2020.