SiFive

RISC-V IOMMU Architecture Overview

RISC-V Spring Week, May 5th

Vedvyas Shanbhogue, Rivos Inc. (Chair)
Perrine Peresse, SiFive (Vice-Chair)

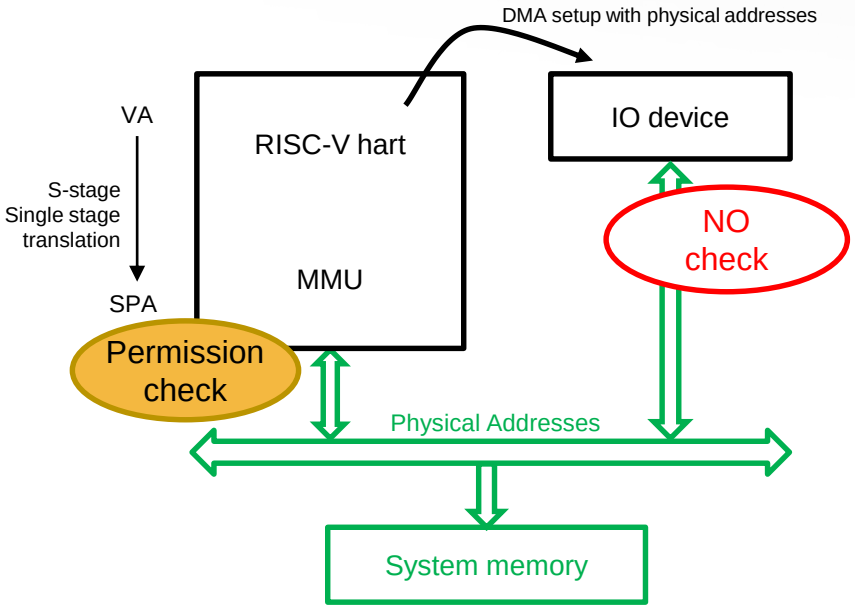
Agenda

- **What is an IOMMU ? Why do we need it ?**
  - IOMMU usage models
- **Baseline architecture overview**
  - Placement/features
  - Data structure
  - SW interface
- **IOMMU TG Goal / Status**
  - Join us !

What is an IOMMU? Why do we need it ?

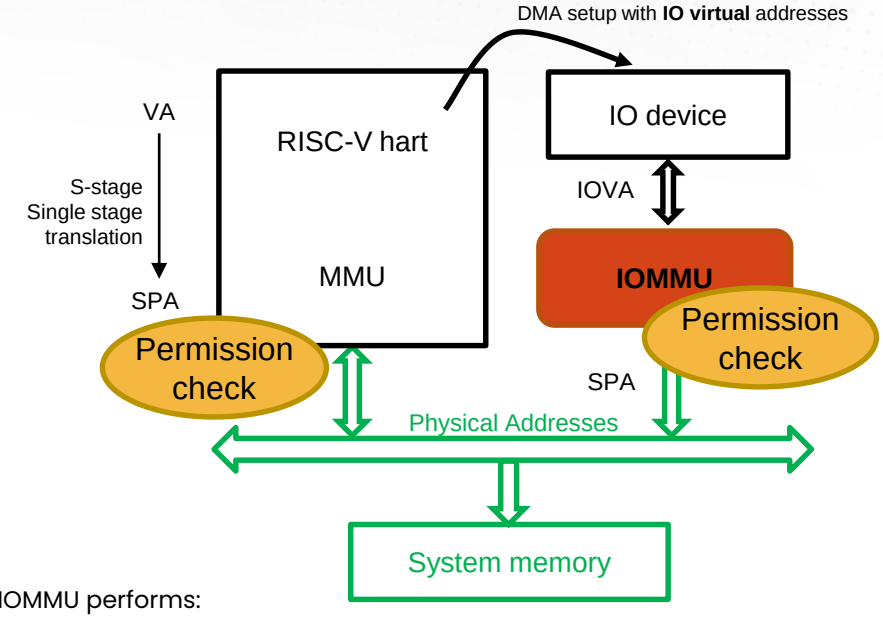
SiFive

Non-virtualized system without an IOMMU



- No protection from SW bugs in device driver or malicious driver or misbehaving IO device

Non-virtualized system with an IOMMU

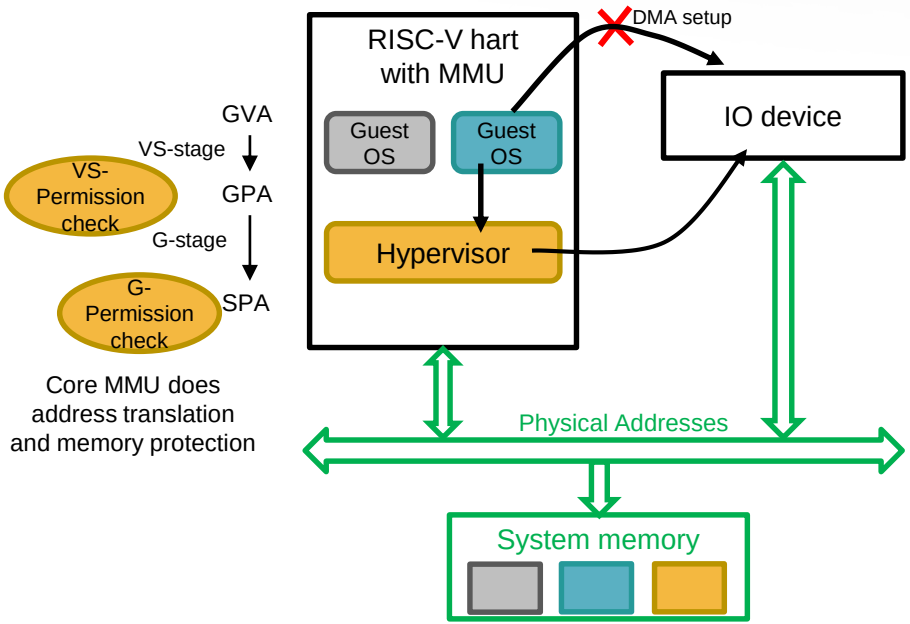


- IOMMU performs:
  - Single stage **address translation and permission checks**
- IOMMU provides:
  - **Memory protection** from IO device DMA
  - Mapping of contiguous IOVA to an underlying fragmented PA (avoidance of scatter/gather lists)
  - Enable 32b legacy IO device to access > 4GB (no bounce buffers)

Why do we need an IOMMU in a virtualized system?

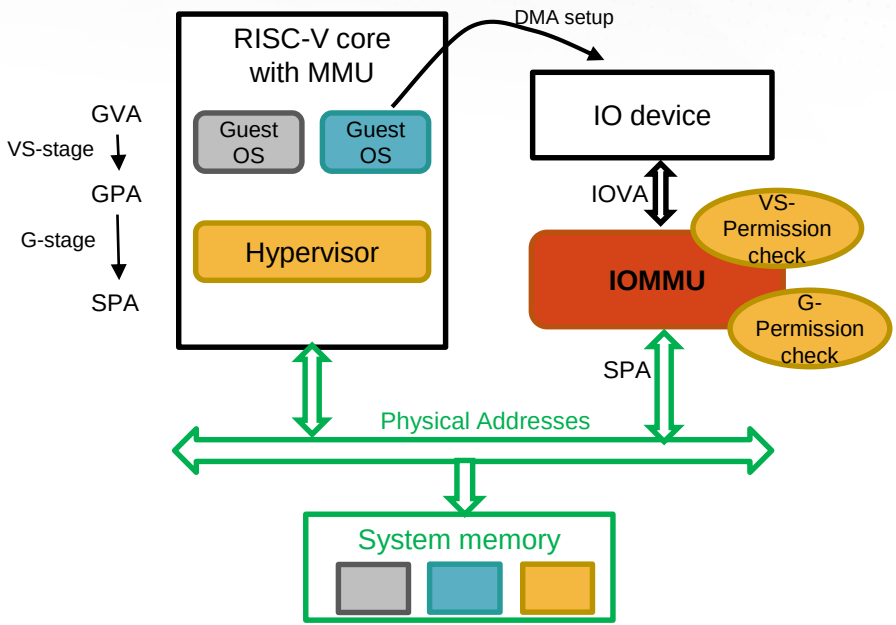
SiFive

Virtualized system without an IOMMU



Core MMU does address translation and memory protection

Virtualized system with an IOMMU

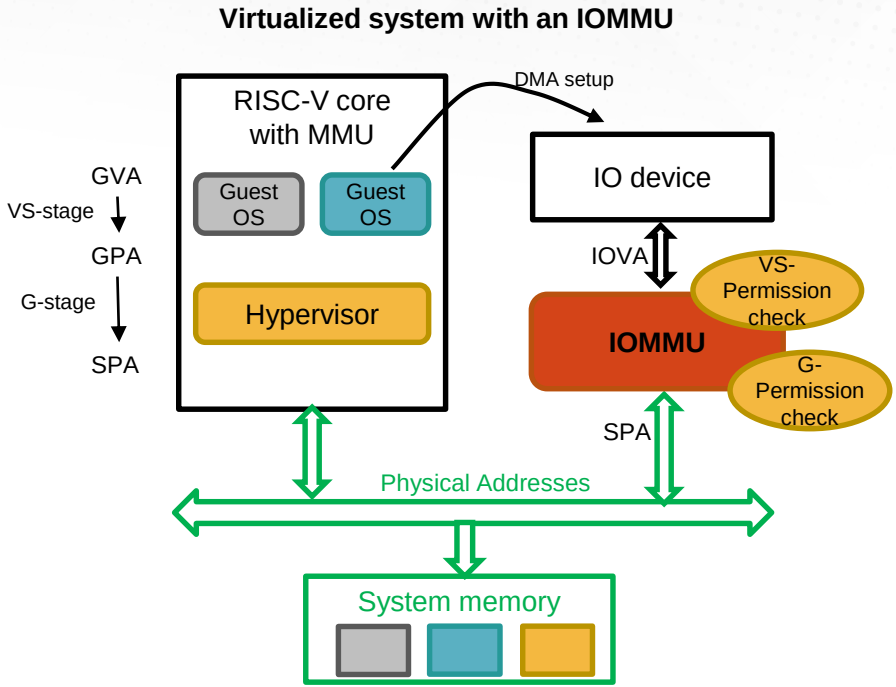


Virtualization on the core provides Guest OS memory isolation.

DMA operation on IO device must be mediated by hypervisor.

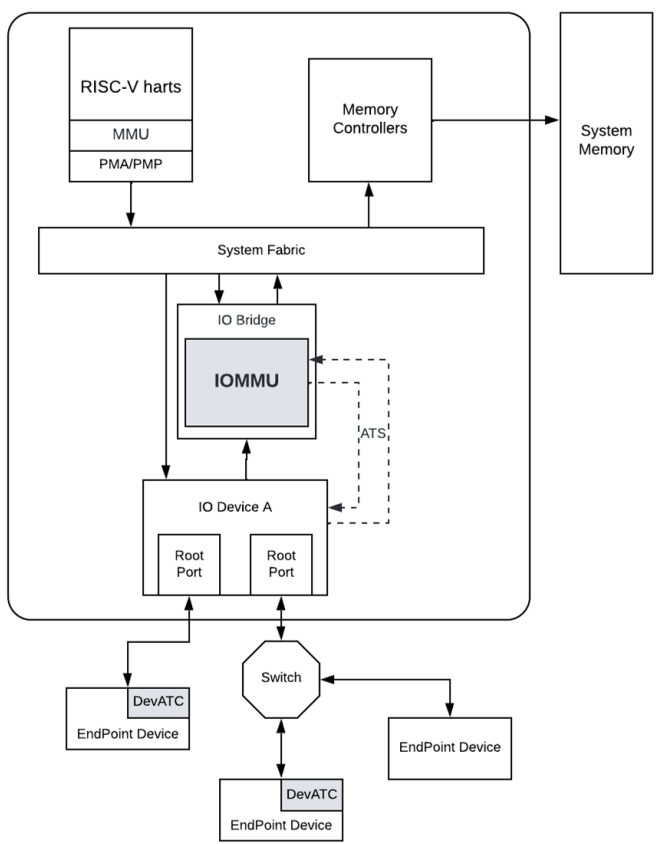
Why do we need an IOMMU in a virtualized system?

- Guest OS has a **direct access** to the IO device.
- IOMMU provides:
  - **Memory protection** from IO device DMA
  - **Virtual address translation** for IO device DMA
  - **Virtual address space sharing** between IO and CPU
  - **Interrupt remapping and virtualization**

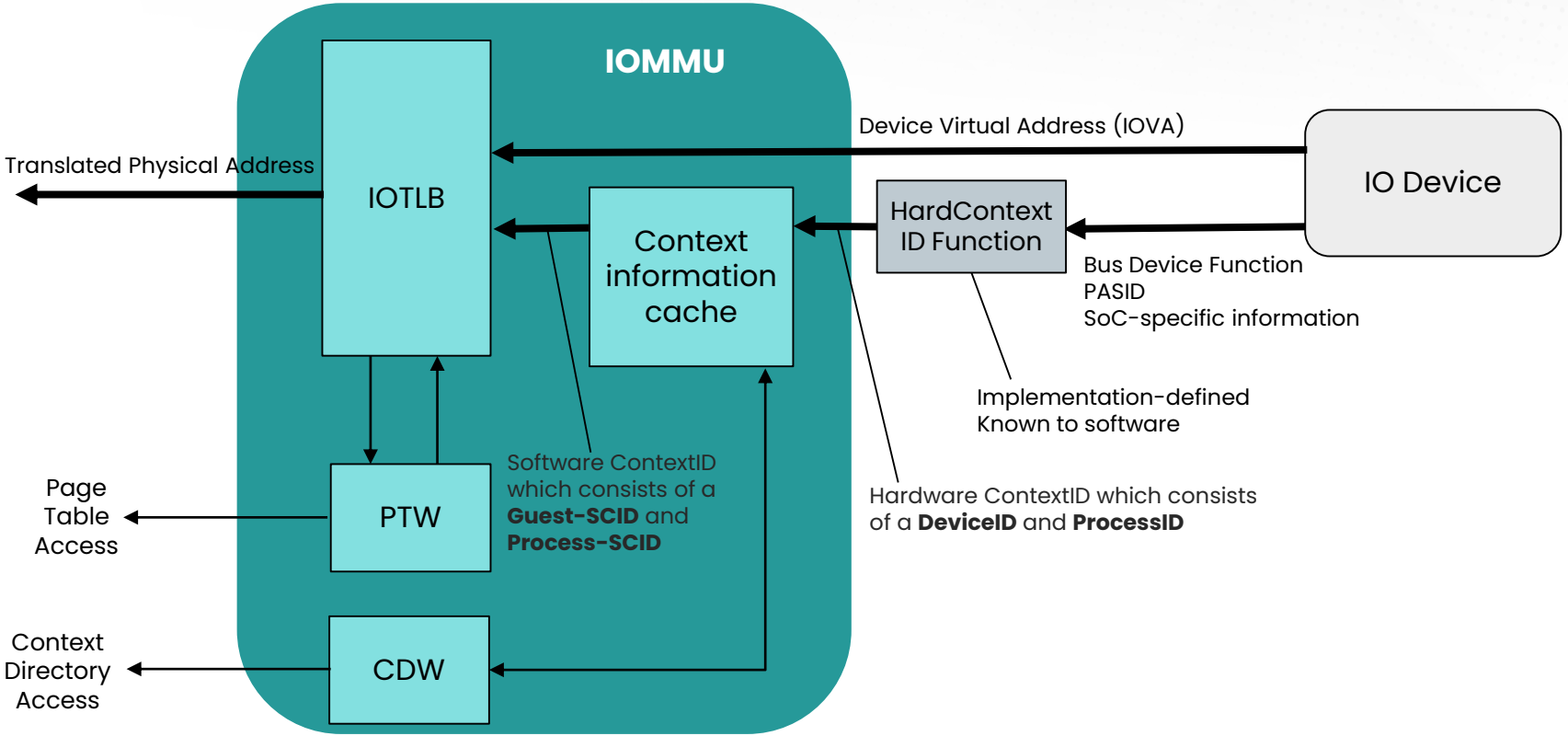


RISC-V IOMMU Placement and features

- RISC-V IOMMU 's main features:
  - **Address translation and protection**
    - single stage (or S-stage) equivalent to satp register behavior
    - two-stage translation (VS-stage and G-stage) equivalent to vsatp and hgatp register behavior
- Support **multiple concurrent devices** and translation contexts
- Support standard interfaces such as PCIe with **PASID, ATS** and PRI
- Compliant to the **RISC-V Privilege specification 1.12** (Hypervisor extension, Svpbmt, Svnapot)
- Compliant to **AIA specification** emphasizing MSI virtualization
- Assigning **physical memory types/attributes** to accesses from devices and the IOMMU
- Optional hardware **performance monitoring** unit
- Co-existence with physical memory protection mechanisms to isolate M-mode resources from access by devices and the IOMMU itself.



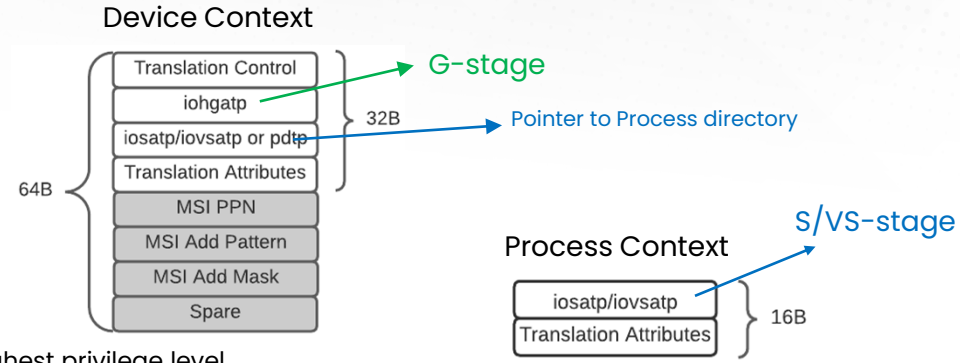
IOMMU High-Level Architecture for Address Translation



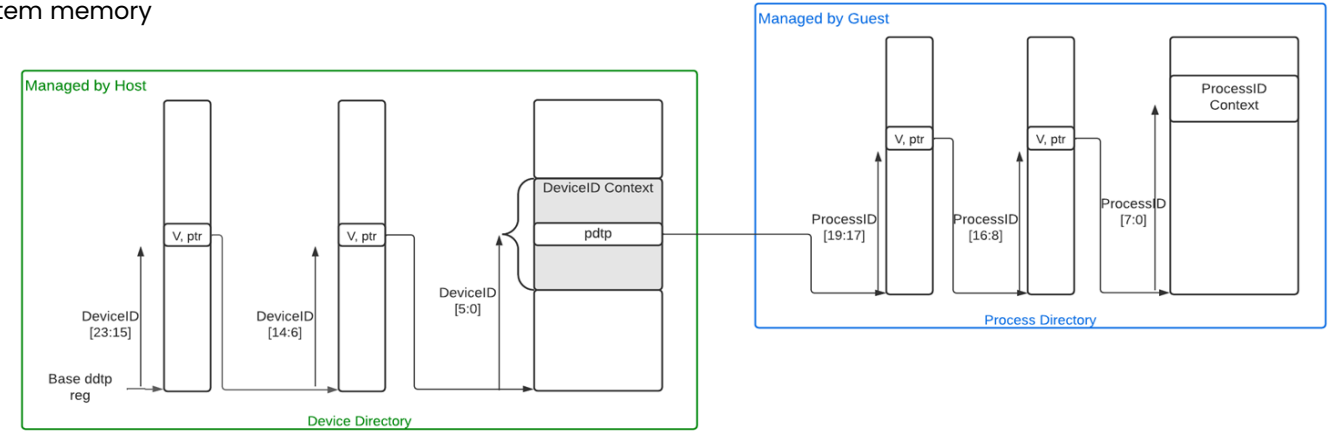
The IOMMU maintains translations for multiple contexts and HARTs

RISC-V IOMMU baseline data structures

- Unique hardware identification:
  - DeviceID up to 24 bits
  - ProcessID up to 20 bits
- Software Context Identification:
  - GSCID up to 16 bits (to identify VM)
  - PSCID up to 20 bits (to identify a process address space)
- Context information
  - Device Directory table is set by SW in system memory by highest privilege level.
  - Process Directory table can be set by guest.
- Multi-level hierarchical table in system memory



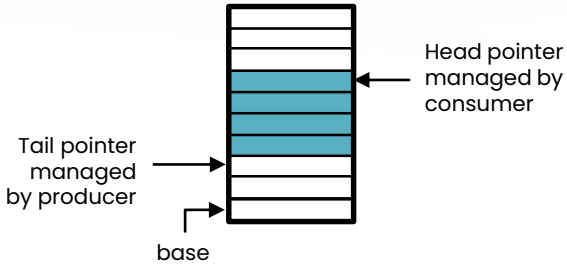
Example of an IO device with ProcessID in a virtualized system



Software Interface

In addition to the memory mapped register space for IOMMU 's configuration and the memory-based structures (Device, Process Directories and the Page Tables), software requires the following circular buffer queues in system memory:

- Command Queue
  - Why ? To improve performance, the IOMMU can cache some context or translation information locally. SW must be able to invalidate any entry.
  - Commands are submitted by SW
  - Commands are processed and completed by IOMMU
- Fault Queue
  - Why ? Errors happen asynchronously to the CPU and many faults can occur simultaneously.
  - IOMMU reports any fault conditions encountered - e.g., page faults
  - SW reads and handles any fault report
- Page Request Queue
  - Why ? for endpoint device to request page to be made present
  - Defined by PCIe ATS/PRI protocol
  - SW reads page request messages from device



Summary and call for action

- Creation:
  - Early December 2021
  - Four I/O MMU specifications were donated to RVI (Rivos Inc., T-head, SiFive and Thales)
- Goal: To develop a unified, advanced IOMMU architecture appropriate for RISC-V platforms

- Working draft specification v0.1:
  https://github.com/riscv-non-isa/riscv-iommu
- QEMU and linux driver development: in progress (Target August 2022).
  Need help and contribution !

- Future enhancements:
  - Hardware acceleration for virtualizing IOMMU
  - Confidential computing
  - Quality of service

Thank you

SIFIVE.COM