RISC-V IOMMU Specification Document (Ziommu)

IOMMU Task Group

Version 0.1, 3/2022: This document is in development. Assume everything can change. See http://riscv.org/spec-state for details.

Table of Contents

Preamble

This document is in the Development state

Assume everything can change. This draft specification will change before being accepted as standard, so implementations made to this draft specification will likely not conform to the future standard.

Copyright and license information

Contributors

This RISC-V specification has been contributed to directly or indirectly by (in alphabetical order):

Aaron Durbin, Allen Baum, Daniel Gracia Pérez, Greg Favor, Nick Kossifidis, Perrine Peresse, Philipp Tomsich, Rieul Ducousso, Siqi Zhao, Tomasz Jeznach, Vassilis Papaefstathiou, Vedvyas Shanbhogue

Chapter 1. Introduction

The IOMMU (sometimes referred to as a system MMU) is a system-level memory management unit (MMU) that connects direct-memory-access-capable I/O devices to system memory.

For each I/O device connected to the system through an IOMMU, software can configure at the IOMMU a device context, which associates with the device a specific virtual address space and other per-device parameters. By giving devices each their own seperate device context at an IOMMU, each device can be individually configured for a different software master, usually a guest OS or the main (host) OS. On every memory access made from a device, hardware indicates to the IOMMU the originating device by some form of unique device identifier, which the IOMMU uses to locate the appropriate device context within data structures supplied by software. For PCIe, for example, the originating device may be identified by the unique 16-bit triple of PCI bus number (8-bit), device number (5-bit), and function number (3-bit) (collectively known as routing ID (RID) and optionally upto 8-bit segment number when the IOMMU supports multiple segments. This specification refers to such unique device identifier as `device_id` and supports upto 24-bit wide IDs.

Some devices may support shared virtual addressing which is the ability to share process address space with devices. Sharing process address spaces with devices allows to rely on core kernel memory management for DMA, removing some complexity from application and device drivers. After binding to a device, applications can instruct it to perform DMA on buffers obtained with malloc. To support such addressing, software can configure one or more process context into the device context. On every memory access made from a device, the hardware indicates to the the IOMMU a unique process identifier, which the IOMMU uses in conjunction with the unique device identifier to locate the appropriate process context configured by software in the device context. For PCIe, for example, the process context may be identified by the unique 20-bit process address space ID (PASID). This specification refers to such unique process identifiers as `process_id` and supports upto 20-bit wide IDs.

Using the same S/VS-stage and G-stage page table formats in IOMMU for address translation and protections as the CPU's MMU removes some complexity from the core kernel memory management for DMA. Use of an identical format also allows the same G and S/VS-stage tables to be used by both MMU and the IOMMU.

DMA address translation in the IOMMU has certain performance implications for DMA accesses as DMA access time may be lengthened due to the time required to resolve the supervisor physical address using software provided data structures. Similar overheads in the CPU MMU are mitigated typically through the use of a translation look-aside buffer (TLB) to cache these address translations such that they may be re-used to reduce the translation overhead on subsequent accesses. The IOMMU may employ similar address translation caches (IOATC). The IOMMU provides mechanisms for software to synchronize the IOATC with the memory resident data structures used for address translation when they are modified. Software may configure the device context with a software defined context ID called Guest-soft-context-ID (`GSCID`) to indicate that a collection of devices are assigned to the same VM and thus access a common virtual address space. Software may configure the process context with a software defined context ID called Process-soft-context-ID (`PSCID`) to identify a collection of process ID that share a common virtual address space. The IOMMU may use the `GSCID` and `PSCID` to tag entries in the IOATC to avoid duplications and simplify invalidation operations.

Some devices may participate in the translation process and provide a device side ATC (DevATC) for its own memory accesses. By providing a DevATC, the device shares the translation caching responsibility and thereby reduce probability of "thrashing" in the IOATC. The DevATC may be sized by the device to suit its unique performance requirements and may also be used by the device to optimize latency by prefetching translations. Such mechanisms require close cooperation of the device and the IOMMU using a protocol. For PCIe, for example, the Address Trnaslation Services (ATS) protocol may be used by the device to request translations to cache in the DevATC and to synchronize it with updates made by software address translation data structures. The device participating in the address translation process also enables the use of I/O page faults to avoid the core kernel memory manager from having to make all physical memory that may be accessed by the device resident at all times. For PCIe, for example, the device may implement the Page Request Interface (PRI) to dynamically request the memory manager to make a page resident if it discovers the page for which it request a translation was not available. An IOMMU may support the interfaces to software and the protocols with the device to enable services such as PCIe ATS and PCIe PRI.

In systems built with an Incoming Message-Signaled Interrupt Controller (IMSIC), the IOMMU may be programmed by the hypervisor to direct message-signaled interrupts (MSI) from devices controlled by the guest OS to a guest interrupt file in an IMSIC. Because MSIs from devices are simply memory writes, they would naturally be subject to the same address translation that an IOMMU applies to other memory writes. However, the Advanced Interrupt Architecture requires that IOMMUs treat MSIs directed to virtual machines specially, in part to simplify software, and in part to allow optional support for memory-resident interrupt files. The device context is configured by software with parameters to identify memory writes as MSI and to be translated using a MSI address translation table configured by software in the device context.

1.1. Glossary

Table 1. Terms and definitions

Term	Definition
ATS	Address Translation Services - a PCIe protocol to support DevATC.
DC	Device Context
DDT	Device-directory-table: A radix-tree structure traversed using the unique device identifier to locate the Device Context structure.
DDI	Device-directory-index: a sub-field of the unique device identifier used as a index into a leaf or non-leaf DDT structure.
Device Context	A hardware representation of state that identifies a device and the VM to which the device is assigned.
Device ID	A identification number that is up to 24-bits to identify the source of a DMA or interrupt request. For PCIe devices this is the routing-ID.
DevATC	A address translation cache at the device.
DMA	Direct Memory Access
GPA	Guest Physical Address: an address in the virtualized physical memory space of a virtual machine.

Term	Definition
GSCID	Guest soft-context identifier: An identification number used by software to uniquely identify a collection of devices assigned to a virtual machine. An IOMMU may tag IOATC entries with the GSCID. Device contexts programmed with same GSCID must also be programmed with identical G-stage page tables.
Guest	Software in a virtual machine.
Hypervisor	Software entity that controls virtualization.
IOATC	IOMMU Address Translation Cache: cache in IOMMU that caches data structures used for address translations.
IOVA	I/O Virtual Address: Virtual address for DMA by devices
MSI	Message Signaled Interrupts.
PASID	Process Address Space Identifier that identifies the address space of a process. The PASID value is provided in the PASID TLP prefix of the request.
PBMT	Page-Based Memory Types
PPN	Physical Page Number
PRI	Page Request Interface - a PCIe protocol that enables devices to request OS memory manager services to make pages resident.
PT	Page Table
PTE	Page Table Entry. A leaf or non-leaf entry in a page table.
PC	Process Context
PDI	Process-directory-index: a sub field of the unique process identifier used to index into a leaf or non-leaf PDT structure.
Process ID	A identification number that is up to 20-bits to identify a process context. For PCIe devices this is the PASID.
PSCID	Process soft-context identifier: An identification number used by software to identify a unique address space. The IOMMU may tag IOATC entries with PSCID.
PDT	Process-directory-table: A radix tree data structure traversed using the unique Process identifier to locate the process context structure.
Reserved	A register or data structure field reserved for future use. Reserved fields in data structures must be set to 0 by software. Software must ignore reserved fields in registers and preserve the value held in these fields when writing values to other fields in the same register.
SPA	Supervisor Physical Address: Physical address used to to access memory and memory-mapped resources.

Term	Definition
VA	Virtual Address
VM	Virtual Machine: An efficient, isolated duplicate of a real computer system. In this specification it refers to the collection of resoures and state that is accessible when a RISC-V hart executes with V=1.
VMM	Virtual Machine Monitor. Also referred to as hypervisor.
VS	Virtual Supervisor: supervisor privilege in virtualization mode.
WARL	Write any values, reads legal values: attribute of a register field that is only defined for a subset of bit encodings, but allow any value to be written while guaranteeing to return a legal value whenever read.
WPRI	Reserved Writes Preserve Values, Reads ignore Values: attribute of a register field that is reserved for future use.

1.2. Usage models

1.2.1. Non-virtualized OS

A non-virtualized OS may use the IOMMU for the following significant system-level functionalities:

1. Protect the operating system from bad memory accesses from errant devices
2. Support 32-bit devices in 64-bit environment (avoidance of bounce buffers)
3. Support mapping of contiguous virtual addresses to an underlying fragmented physical addresses (avoidance of scatter/gather lists)

In the absence of an IOMMU, a device driver must program IO devices with Physical Addresses, which implies that DMA-ed data from an IO device could be written to privileged memory. This can be caused by hardware, device driver bugs, or malicious software. Malicious software attacks can come from IO peripherals that make read/write accesses to the system memory or even to embedded memory in other peripherals through DMA transactions. The integration of IOMMU offers a mechanism to protect the OS from these software attacks and from buggy device driver because the IOMMU ensures that privileged memory of the OS is isolated from a device. Indeed, the Operating System configures the IOMMU to use the S-Stage page table to translate IOVA to PA. The OS may also use the MSI address translation capability to dynamically redirect interrupts from one RISC-V hart to another without needing to reprogram the devices themself.

In the absence of an IOMMU, legacy 32-bit devices cannot access the entire 64-bit system memory range. This forces the operating system to ensure that when this 32-bit IO device is used, buffers in the first 4GB memory region are allocated and then contents of this buffer are copied to the final destination in the system memory to be consumed afterwards. The integration of the IOMMU offers a simple mechanism for the DMA to directly access any address space in the system (with appropriate access permission) and thereby improves system performance.

The IOMMU can be useful as it permits to allocate large regions of memory without the need to be contiguous in physical memory. Indeed, a contiguous virtual address range can be mapped to fragmented physical addresses.

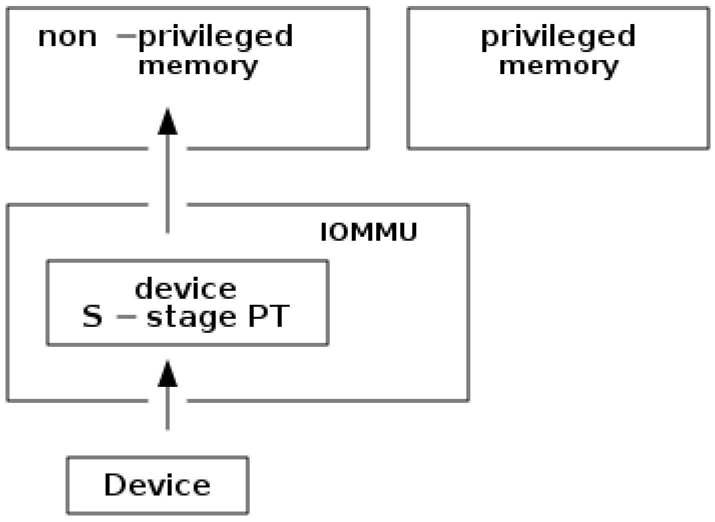


Figure 1. Device isolation in non-virtualized OS

1.2.2. Hypervisor

IOMMU makes it possible for a guest operating system, running in a virtual machine, to be given direct control of an I/O device with only minimal hypervisor intervention.

A guest OS with direct control of a device will program the device with guest physical addresses, because that is all the OS knows. When the device then performs memory accesses using those addresses, an IOMMU is responsible for translating those guest physical addresses into supervisor physical addresses, referencing address-translation data structures supplied by the hypervisor.

To handle MSIs from a device controlled by a guest OS, the hypervisor configures an IOMMU to redirect those MSIs to a guest interrupt file in an IMSIC or to a memory-resident interrupt file. The IOMMU is responsible to use the MSI address-translation data structures supplied by the hypervisor to perform the MSI redirection.

The following diagram illustrates the concept. The device D1 is directly assigned to VM-1 and device D2 is directly assigned to VM-2. The VMM configures the G-stage page table to be used by each device and restricts the memory that can be accessed by D1 to VM-1 associated memory and from D2 to VM-2 associated memory.

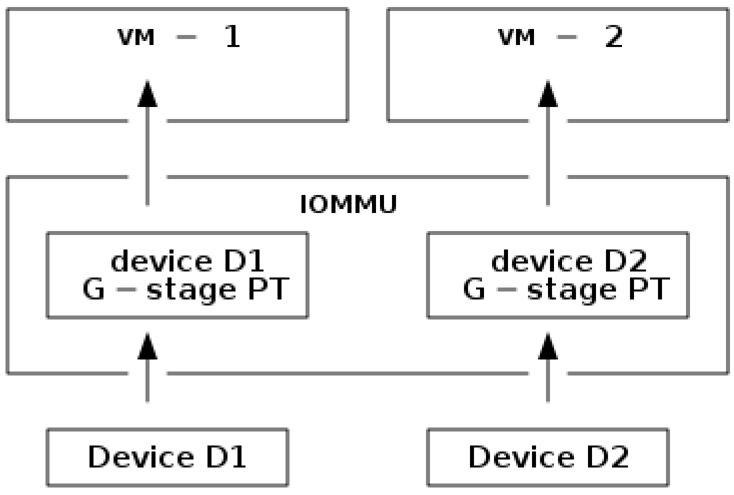


Figure 2. DMA translation to enable direct device assignment

1.2.3. Guest OS

The presence of IOMMU in SoC allows software to configure IO device DMA with virtual addresses instead of physical addresses. When an IO device is assigned to a Guest OS, the device can be configured directly using the VA address space and the Host configures IOMMU to perform single stage VA to PA translation. In a system with the Hypervisor Extension, the G-stage translation (or second stage of translation) is intended to virtualize IO device DMA to the Guest OS physical address space. IO Devices can be assigned to Guest OS which can directly set the DMA with its Guest Physical Addresses (GPA). The Hypervisor or Host OS will set up and configure the IOMMU to perform GPA to PA translation using G-stage page tables. It limits the physical memory accessible by a device controlled by the guest OS to the memory allocated to its virtual machine.

The two stages of translation (VS-stage and G-stage) are intended to be used by a software entity to provide isolation or translation to buffers within the entity, for example DMA isolation within an OS. The guest OS may use the VS-stage address translation functions to limit the device access to a subset of the guest memory. The IOVA may be first translated to a GPA using the VS-stage page tables managed by the guest OS and the GPA translated to a PA using the G-stage page tables managed by the hypervisor.

The figure below shows that the IOMMU can be configured to perform nested translation (VS-stage and G-stage translation) for one device (D1) and to perform G-stage only translation for another device (D2). The host or hypervisor could also retain a device for its own usage and thus the IOMMU can also be configured to perform a single stage (S-stage) translation for yet another device (D3).

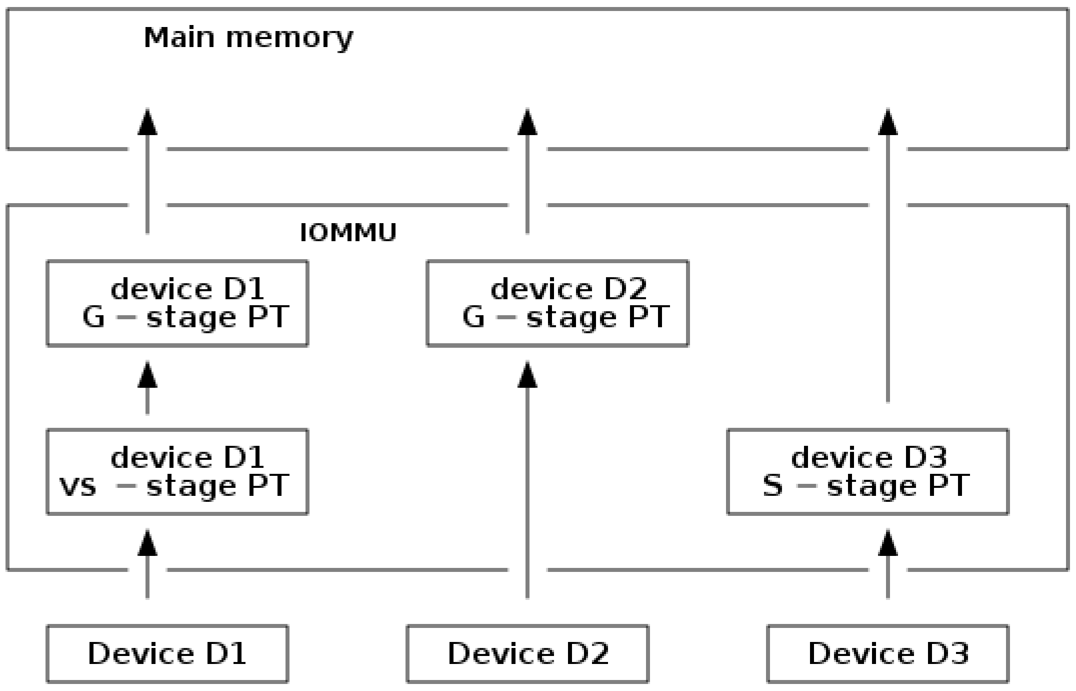


Figure 3. Address translation in IOMMU for Guest OS

The hypervisor may use the MSI address translation capability to dynamically redirect interrupts from guest controlled devices to the guest assigned interrupt register file of an IMSIC in the RISC-V hart.

1.3. Placement and data flow

The following figure shows an example of a typical SOC with RISC-V hart(s). The SOC incorporates memory controllers for DRAM storage and several IO devices. This SOC also incorporates two instances of IOMMU. The first instance interfaces one IO Device A to the system fabric and the second instance interfaces several clients (IO Devices B, and C) to the system fabric. As shown in this figure, the IO Bridge along with the IOMMU is placed between the device(s) and the fabric or interconnect. IO Devices can perform DMA transactions using Virtual Addresses (VA, GVA or GPA) or any other bus addresses in inbound transactions that the IOMMU will translate into Physical Addresses (PA).

The device may be directly connected to the IOMMU or may be connected through I/O ports using a protocol such as PCI Express (PCIe). The outbound transactions to the device do not transit through the IOMMU.

The IO Device A in the example is representing Root Ports connected to several endpoints with or without ATC support. The IOMMU may support ATS in order to manage the endpoint's device address translation cache (DevATC).

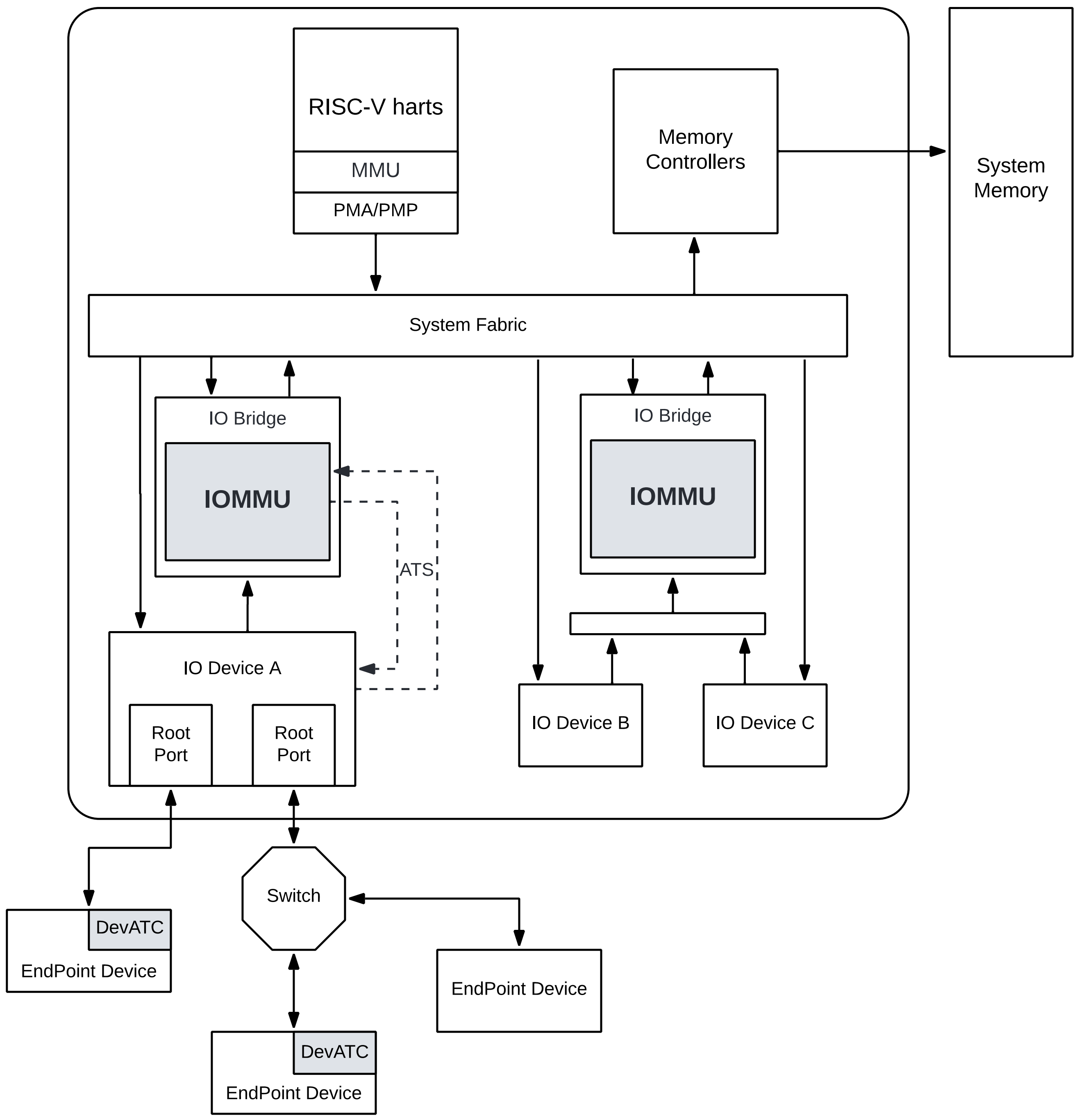


Figure 4. Example of IOMMUs integration in SoC.

The IOMMU handles only inbound address translation and protection. It does not manage the data of the inbound transactions. The IOMMU behaves like a look-aside IP to the IO bridge and has several interfaces:

- Host interface: it is a slave interface to the IOMMU for the harts to access its internal MMIO registers and perform its global configuration or maintenance.
- Device Translation Request interface: it is a slave interface, which receives the incoming virtual address requests from one or more IO devices. Along with the device address and attributes, a `device_id` (and `process_id`, if any) is provided to the IOMMU in order to uniquely identify the source of the request and potentially its process. Thanks to the unique hardware identifier {`device_id`, `process_id`}, the IOMMU is able to retrieve the context information to perform the requested address translation.
- The Data Structure interface for the IOMMU implicit access to memory: it is a master interface to

the system interconnect to fetch the required data structure from main memory. This interface is used to access:

a. the device and process directories to get the context information and translation rules

b. the page tables to get the translated address for S/VS and G-stage

c. the queues for software's interface (command queue, fault queue and page request queue)

- Device Translation Completion interface: it is a master interface which provides the requested translated Physical Address and an error status. As the IOMMU may not respond to requests in order, this interface will also provide some attributes as received on the Device Translation Request interface and the resolved Paged-Based Memory Type,if Svpbmt is supported (the resolved attribute is based on PBMT from first and/or second stage page tables).
- Optionally an ATS interface in order to communicate with the PCIe Root Port. This interface is used to service an ATS request, to invalidate cached entries in the device's ATC and to request pages through the PCIe Page Request Interface mechanism.

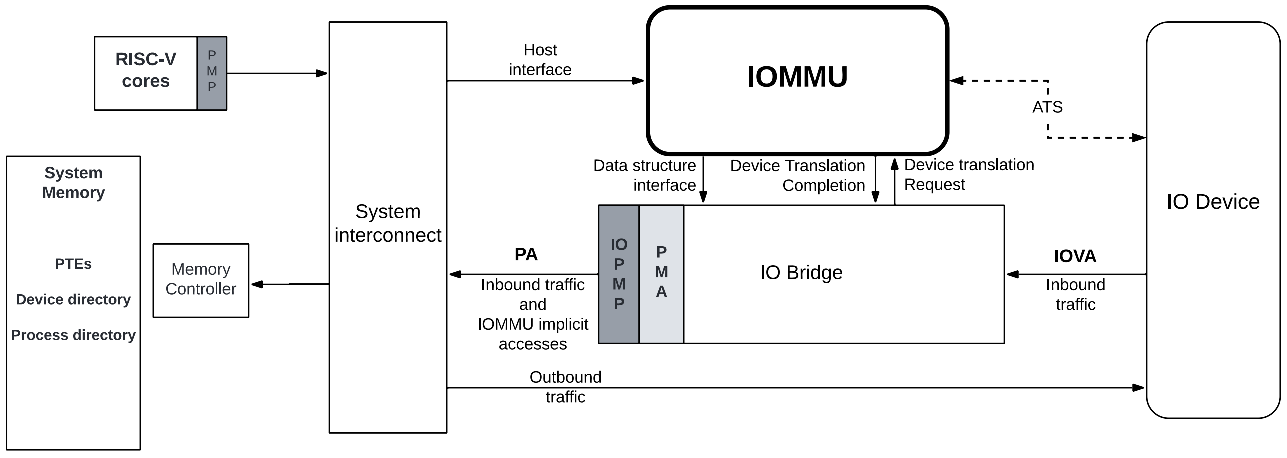


Figure 5. IOMMU interfaces.

Similar to the RISC-V harts, physical memory attributes (PMA) and physical memory protection (PMP) checks must be completed on any inbound IO transactions even when the IOMMU is in bypass (bare state). The placement and integration of the PMA and IOPMP (and other variants) checkers is a platform choice.

PMA and IOPMP checkers reside outside the IOMMU. The example above is showing them in the IO bridge.

Implicit accesses by the IOMMU itself through the data structure interface are checked by the PMA checker. PMAs are tightly tied to a given physical platform's organization, many details are inherently platform-specific.

The IOMMU provides the resolved PBMT (PMA, IO, NC) along with the translated address on the device translation completion interface to the I/O bridge. The PMA in I/O bridge resolves the final attributes using the overrides requested by PBMT and PMA type.

The IOPMP or other variants may use the hardware ID of the bus master to determine physical memory access privileges. Since the IOMMU itself is a bus master for its implicit accesses, the IOMMU hardware ID may be used by the IOPMP to select the appropriate access control rules.

1.4. IOMMU features

The version 1.0 of the RISC-V IOMMU specification supports the following features:

- Memory-based device context to locate parameters and address translations structures. The device context is located using the hardware provided unique `device_id`. The supported `device_id` width may be up to 24-bit. IOMMU is required to support at least one of the valid `device_id` widths as specified in Chapter 2.
- Memory-based process context to locate parameters and address translation structures using hardware provide unique `process_id`. The supported `process_id` may be up to 20-bit. IOMMU is required to support at least one of the valid `process_id` widths as specified in Chapter 2
- IOMMU must support 16-bit GSCIDs and 20-bit PSCIDs.
- An implementation may support only the VS/S-stage of address translation, only G-stage address translation, or two stage address translation.
- VS/S-stage and/or G-stage virtual-memory system as specified by the RISC-V privileged specification to allow software flexibility to use a common page table for CPU MMU as well as IOMMU or to use a separate page table for the IOMMU.
- Upto 57-bit virtual-address width and 59-bit guest-physical-address width.
- Support for hardware management of page-table entry Accessed and Dirty bits is optional for the IOMMU.
- Support for MSI address translation as specified by RISC-V Advanced Interrupt Architecture (AIA) is optional. When MSI address translation is supported using flat MSI page tables then supporting memory-resident-interrupt-files is optional.
- Supporting Svnapot extension is optional.
- Supporting Svpbmt extension is optional.
- IOMMU may optionally support the PCIe ATS and PRI services. When ATS is supported the IOMMU may optionally support the ability to translate to a GPA instead of a SPA in response to a translation request.
- IOMMU may optionally support an hardware performance monitoring unit (PMU). If a PMU is supported then the IOMMU must support the cycles counter and at least 7 hardware performance monitoring counters must be supported.
- The IOMMU may use MSI or wire-based-interrupts to request service from software. At least one method of generating interrupts from the IOMMU must be supported.

Software may discover the supported features using the `capabilities` register of the IOMMU.

Chapter 2. Data Structures

A data structure called device-context (DC) is used by the IOMMU to associate a device with an address space and to hold other per-device parameters used by the IOMMU to perform address translations. A radix-tree data structure called device directory table (DDT) traversed using the device_id is used to hold the DC.

Two formats of the device-context structure are supported:

- **Base Format** - A 32-byte DC used when MSI address translation using MSI flat page table and redirection of MSI to memory-resident interrupt files (MRIFs) are not supported.
- **Extended Format** - In the extended format a 64-byte device context is used and extends the base device context with additional information for processing MSIs.

DC associates the device, identified by device_id, with a guest-physical-address space by providing a pointer to a G-stage page table and a virtual machine identifier called the guest soft-context ID (GSCID).

To support devices with multiple process contexts, the DC points to a radix-tree data structure called the Process Directory Table (PDT) used to associate a process_id with a virtual address space. The PDT is used to hold the pointer to a S or VS-stage page table for a process_id.

The DDT may be configured to be a 1, 2, or 3 level radix table depending on the maximum width of the device_id supported. The partitioning of the device_id to obtain the device directory indexes (DDI) to traverse the DDT radix-tree table are as follows:

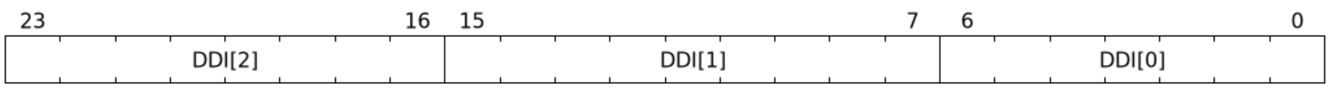


Figure 6. Base format device_id partitioning

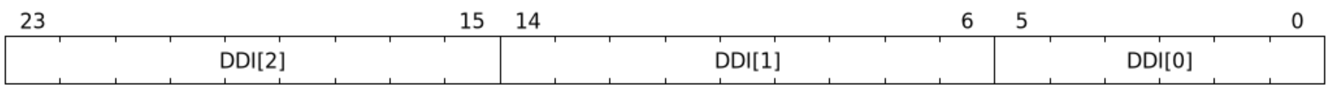


Figure 7. Extended format device_id partitioning

The PDT may be configured to be a 1, 2, or 3 level radix table depending on the maximum width of the process_id supported. Each process directory table entry is 16-byte wide and provides the pointer to a S or VS-stage page table, the address translation scheme and an identifier for the process address space called the Process soft-context ID (PSCID). The partitioning of the process_id to obtain the process directory indices (PDI) to traverse the PDT radix-tree table are as follows:

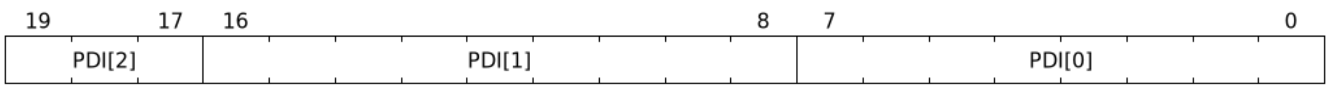


Figure 8. process_id partitioning for PDT radix-tree traversal

All RISC-V IOMMU implementations are required to support DDT and PDT located in main memory. Supporting data structures in I/O memory is not required.

2.1. Device-Directory-Table (DDT)

DDT is upto 3-level radix tree indexed using the device directory index (DDI) bits of the `device_id`.

The following diagrams illustrate the DDT radix-tree. The root device-directory page number is located using a memory-mapped control register called the device-directory-table pointer (`ddtp`). Each non-leaf (`NL`) entry provides the PPN of the next level device-directory-table page. The leaf device directory table entry holds the device-context (`DC`).

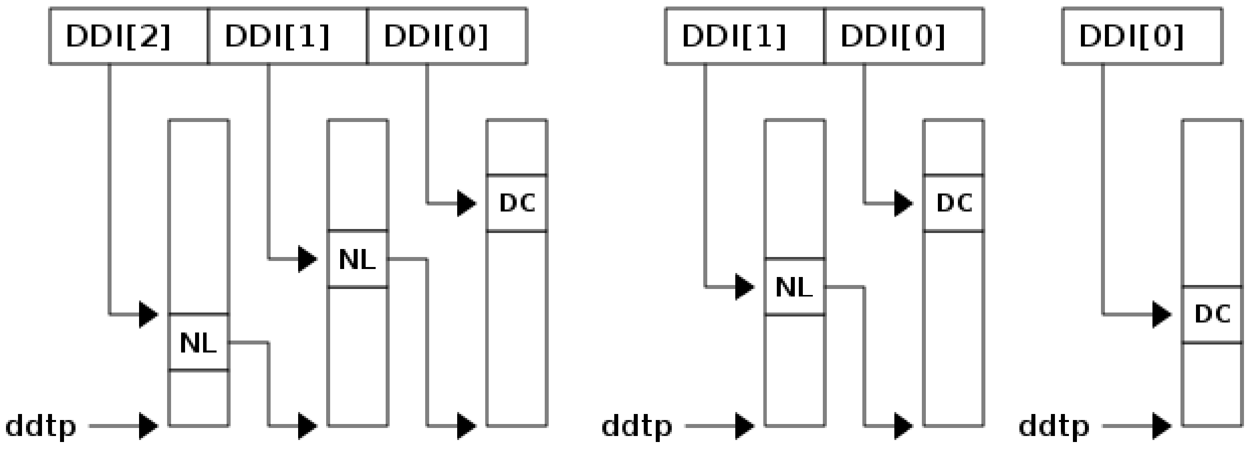


Figure 9. Three, two and single-level device directory

2.1.1. Non-leaf DDT entry

A valid (`V==1`) non-leaf DDT entry provides PPN of the next level DDT page.

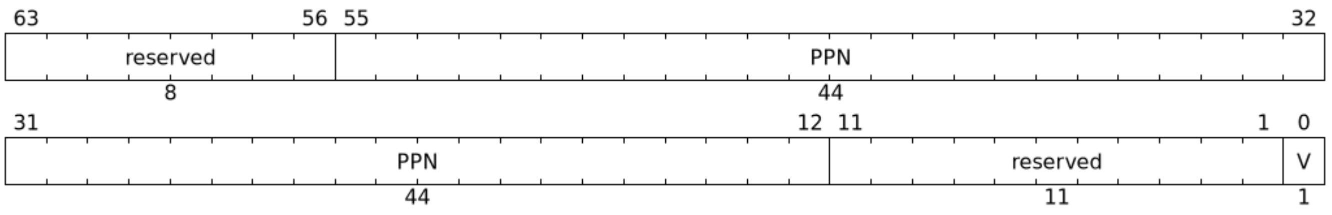


Figure 10. Non-leaf device-directory-table entry

2.1.2. Leaf DDT entry

The leaf DDT page is indexed by `DDI[0]` and holds the device-context (`DC`).

In base-format the `DC` is 32-bytes. In extended-format the `DC` is 64-bytes.

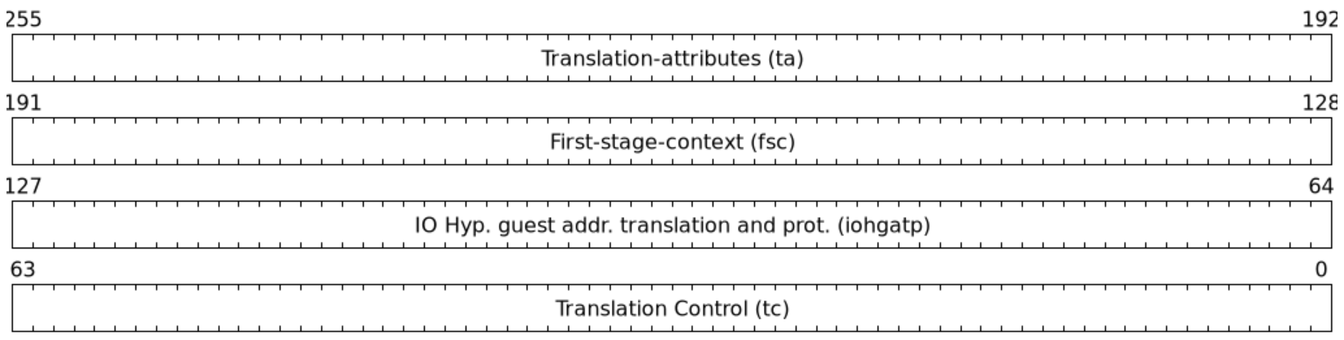

255
192
Translation-attributes (ta)
191
128
First-stage-context (fsc)
127
64
IO Hyp. guest addr. translation and prot. (iohgatp)
63
0
Translation Control (tc)

Figure 11. Base-format device-context

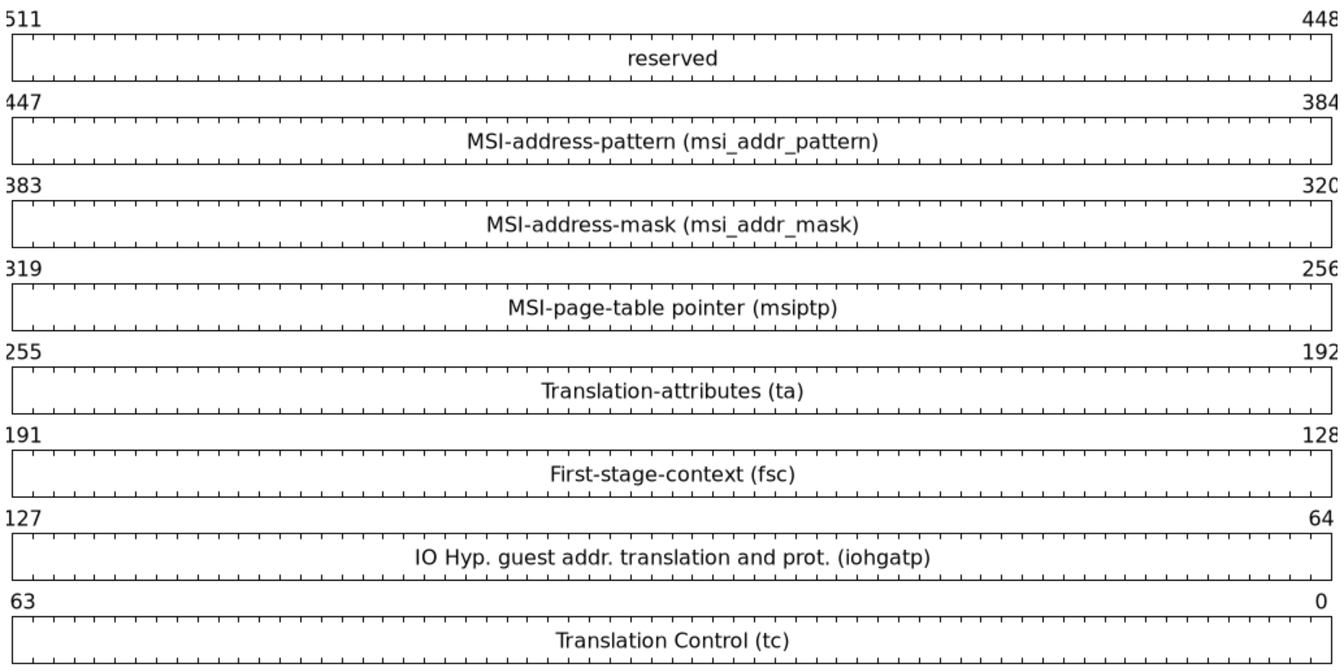

511
448
reserved
447
384
MSI-address-pattern (msi_addr_pattern)
383
320
MSI-address-mask (msi_addr_mask)
319
256
MSI-page-table pointer (msiptp)
255
192
Translation-attributes (ta)
191
128
First-stage-context (fsc)
127
64
IO Hyp. guest addr. translation and prot. (iohgatp)
63
0
Translation Control (tc)

Figure 12. Extended-format device-context

2.1.3. Device-context fields

Translation control (`tc`)

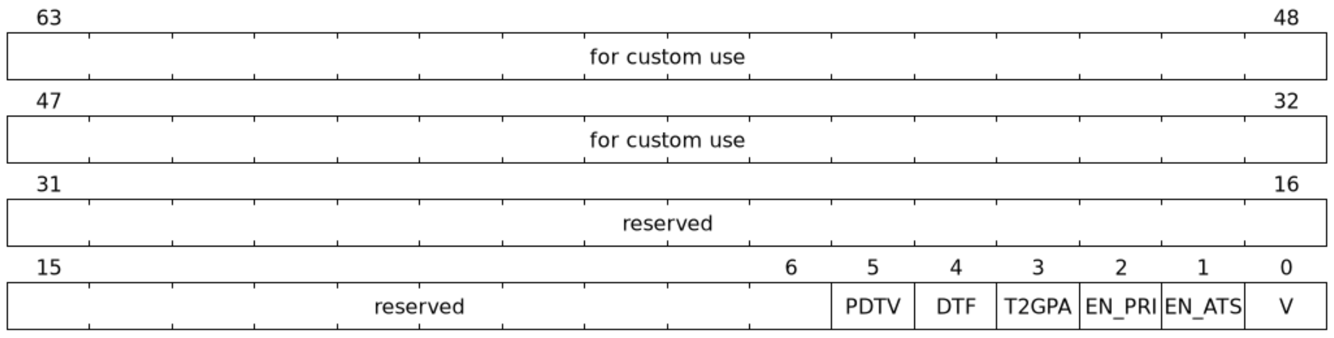


Figure 13. Translation control (`tc`) field

`DC` is valid if the `V` bit is 1; if it is 0, all other bits in `DC` are don't-care and may be freely used by software.

If the IOMMU supports PCIe ATS specification (see `capabilities` register), the `EN_ATS` bit is used to enable ATS transaction processing. If `EN_ATS` is set to 1, IOMMU supports the following inbound transactions; otherwise they are treated as unsupported transactions.

- Translated read for execute transaction
- Translated read transaction
- Translated write/AMO transaction
- PCIe ATS Translation Request
- PCIe ATS Invalidation Completion Message
- PCIe ATS Page Request Message

If the `EN_ATS` bit is 1 and the `T2GPA` bit is set to 1 the IOMMU returns a GPA the translation of an IOVA in a PCIe ATS Translation Request from the device. When `T2GPA` is 1, the IOVA in translated memory accesses is a GPA and translated through the G-stage page table to a SPA. This control enables a hypervisor to contain DMA from a device directly controlled by the guest OS, even with ATS capability enabled and the device misuses the capability, to the VMs memory.

When `T2GPA` is enabled, the addresses provided to the device in response to a PCIe ATS Translation Request are not directly routable by the I/O fabric (e.g. PCI switches) that connect the device to other peer devices and to host. Such addresses are also not routable within the device even if peer-to-peer transactions within the device (e.g. between functions of a device) are supported.

Hypervisors that configure `T2GPA` to 1 must ensure through protocol specific means that translated accesses are routed through the host such that the IOMMU may translate the GPA and then route the transaction based on PA to memory or to a peer device. For PCIe, for example, the Access Control Service (ACS) may be configured to always redirect peer-to-peer (P2P) requests upstream to the host.

Use of `T2GPA` set to 1 may not be compatible with devices that implement caches tagged by the translated address returned in response to a PCIe ATS Translation Request.

As an alternative to setting `T2GPA` to 1, the hypervisor may establish a trust relationship with the device if authentication protocols are supported by the device. For PCIe, for example, the PCIe component measurement and authentication (CMA) capability provides a mechanism to verify the devices configuration and firmware/executables (Measurement) and hardware identities (Authentication) to establish such a trust relationship.

If `EN_PRI` bit is 0, then PCIe ATS Page Request messages from the device are invalid requests.

When SR-IOV VF is used as a unit of allocation, a hypervisor may disable page requests from one of the virtual functions by setting `EN_PRI` to 0. However the page-request interface is shared by the PF and all VFs. The IOMMU protocol specific logic is encouraged to classify this condition as a non-catastrophic failure in its response to avoid the shared PRI in the device being disabled for all PFs/VFs.

Setting the disable-translation-fault - `DTF` - bit to 1 disables reporting of faults encountered in the address translation process. Setting `DTF` to 1 does not disable error responses from being generated to the device in response to faulting transactions. Setting `DTF` to 1 does not disable reporting of faults from the IOMMU that are not related to the address translation process.

A hypervisor may set `DTF` to 1 to disable fault reporting when it has identified conditions that may lead to a flurry of errors such as due to an abnormal termination of a virtual machine that may require the hypervisor to reset the device.

The `fsc` field of `DC` holds the context for first-stage translations (S-stage or VS-stage). The field holds the pointer to a PDT if the `PDTV` bit is 1. If the `PDTV` bit is 0, the `fsc` field instead holds a pointer to a supervisor first-stage page table (i.e. `iosatp`) if `iohgatp.MODE` is `Bare` and holds a pointer to a virtual-supervisor first-stage page table (i.e. `iovsatp`) if `iohgatp.MODE` is not `Bare`.

The `PDTV` is expected to be set to 1 when `DC` is associated with a device that supports multiple process contexts and thus generates a valid `process_id` with its memory accesses.

IO hypervisor guest address translation and protection (`iohgatp`)

The `iohgatp` field holds the PPN of the root G-stage page table and a virtual machine identified by a guest soft-context ID (`GSCID`), to facilitate address-translation fences on a per-virtual-machine basis. If multiple devices are associated to a VM with a common G-stage page table, the hypervisor is expected to program the same `GSCID` in each `iohgatp`. The MODE field is used to select the G-stage address translation scheme.

This field controls the G-stage address translation and protection. The G-stage page table formats and `MODE` encodings follow the format defined by the privileged specification.

Implementations are not required to support all defined mode settings for `iohgatp`. The IOMMU only needs to support the modes also supported by the MMU in the harts integrated into the system.

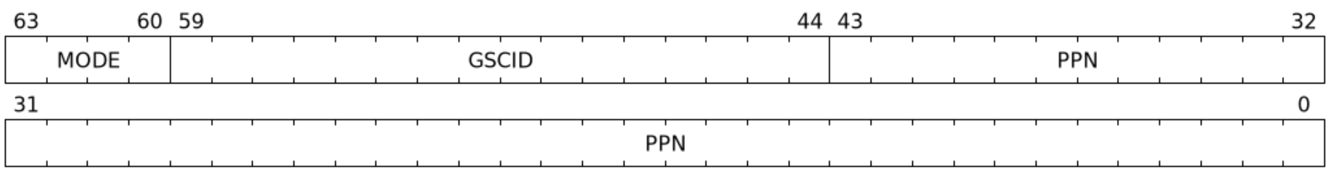


Figure 14. IO hypervisor guest address translation and protection (`iohgatp`) field

First-Stage context (`fsc`)

If `PDTV` is 0, the `fsc` field in `DC` holds the `iosatp` (when `iohgatp MODE` is `Bare`) or the `iovsatp` (when `iohgatp MODE` is not `Bare`) that points to a S-stage page table or VS-stage page table respectively.

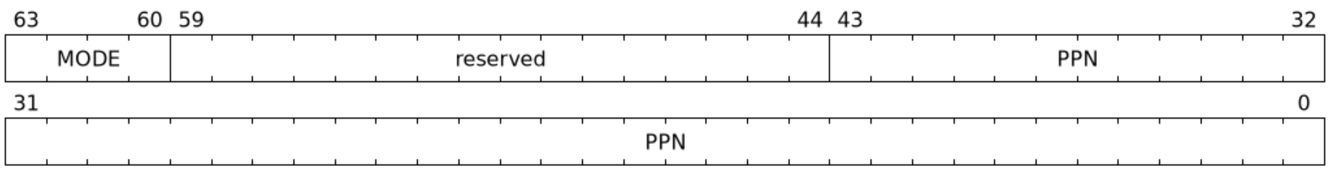


Figure 15. IO (Virtual)Supervisor addr. translation and prot. (`iovsatp`/`iosatp`) field (when `PDTV` is 0)

The encodings of the `iosatp`/`iovsatp` `MODE` field are as the same as the encodings for `MODE` field in the `satp` CSR.

When `PDTV` is 1, the `fsc` field holds the process-directory table pointer (`pdtp`). When the device supports multiple process contexts, selected by the `process_id`, the PDT is used to determine the first-stage page table and associated `PSCID` for virtual address translation and protection.

The PDT is a 1, 2, or 3-level radix tree indexed using the process directory index (`PDI`) bits of the process_id. The pdtp field holds the PPN of the root page of the PDT and the MODE field that determines the number of levels of the PDT.

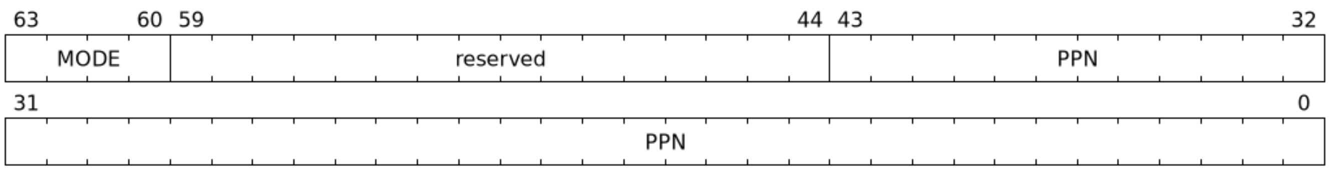


Figure 16. Process-directory table pointer (pdtp) field (when PDTV is 1)

When two-stage address translation is active (iohgatp.MODE != Bare), the PPN field holds a guest PPN. The guest physical address of the PDT root page is then converted by guest physical address translation, as controlled by the iohgatp, into a supervisor physical address. Translating addresses of PDT root page through G-stage page tables, allows the PDT to be mapped into the guest OS address space to allow the guest OS to directly edit the PDT to associate a virtual-address space identified by a VS-stage page table with a process_id.

Table 2. Encoding of pdtp.MODE field

Value	Name	Description
0	Bare	No translation or protection. First stage translation is not enabled.
1	PD20	20-bit process ID enabled. The directory has 3 levels. The root PDT page has 8 entries and the next non-leaf level has 512 entries.The leaf level has 256 entries.
2	PD17	17-bit process ID enabled. The directory has 2 levels. The root PDT page has 512 entries and leaf level has 256 entries. The bits 19:17 of process_id must be 0.
3	PD8	8-bit process ID enabled. The directory has 1 levels. The leaf level has 256 entries.The bits 19:8 of process_id must be 0.
3-15	—	Reserved

Translation attributes (ta)

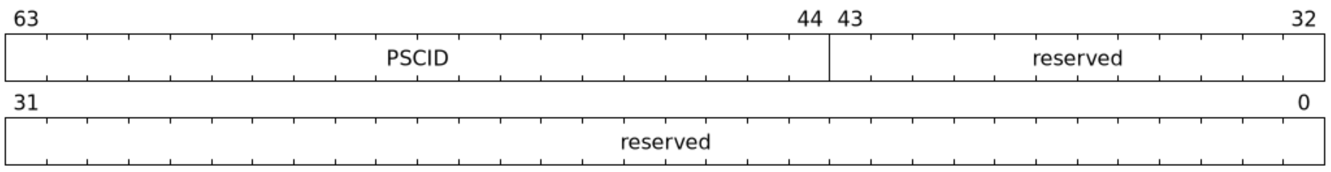


Figure 17. Translation attributes (ta) field

The PSCID field of ta provides the process soft-context ID that identifies the address-space of the process. PSCID facilitates address-translation fences on a per-address-space basis. The PSCID field in ta is used as the address-space ID if PDTV is 0 and the iosatp/iovsatp MODE field is not Bare.

MSI page table pointer (msiptp)

The msiptp field holds the PPN of the root MSI flat page table used to direct an MSI to a guest interrupt file in an IMSIC. The MSI page table format is defined in Section 9.5 of the Advanced Interrupt Architecture (AIA) specification.

The MODE field is used to select the MSI address translation scheme.

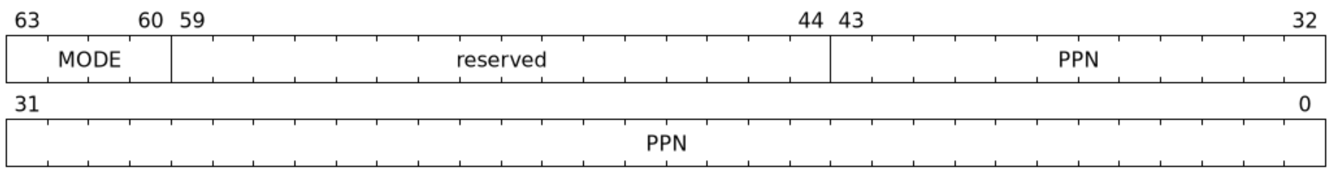


Figure 18. MSI page table pointer (msiptp) field

Table 3. Encoding of msiptp MODE field

Value	Name	Description
0	Bare	No translation or protection. MSI recognition using MSI address mask and pattern is not performed.
1	Flat	Flat MSI page table (see Section 9.5 of the AiA specification)

MSI address mask (msi_addr_mask) and pattern (msi_addr_pattern)

The MSI address mask (msi_addr_mask) and pattern (msi_addr_pattern) fields are used to recognize certain memory writes from the device as being MSIs. The use of these fields is as specified in Section 9.4 of the Advanced Interrupt Architecture specification.

2.2. Process-Directory-Table (PDT)

The PDT is a 1, 2, or 3-level radix tree indexed using the process directory index (PDI) bits of the process_id.

The following diagrams illustrate the PDT radix-tree. The root process-directory page number is located using the process-directory-table pointer (pdtp) field of the device-context. Each non-leaf (NL) entry provides the PPN of the next level process-directory-table page. The leaf process-directory table entry holds the process-context (PC).

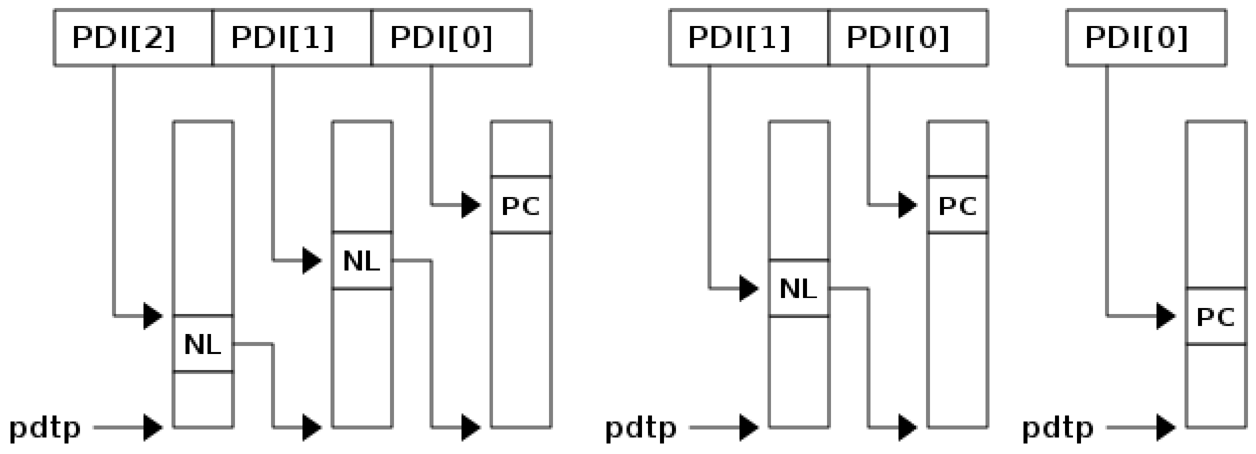


Figure 19. Three, two and single-level process directory

2.2.1. Non-leaf PDT entry

A valid (V==1) non-leaf PDT entry holds the PPN of the next-level PDT page.

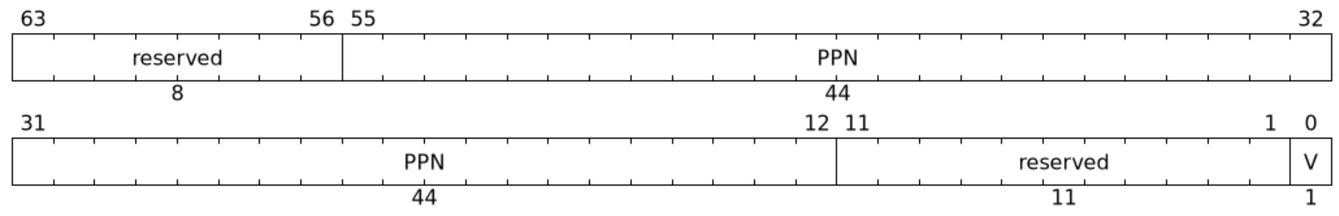


Figure 20. Non-leaf process-directory-table entry

2.2.2. Leaf PDT entry

The leaf PDT page is indexed by PDI[0] and holds the 16-byte process-context (PC).

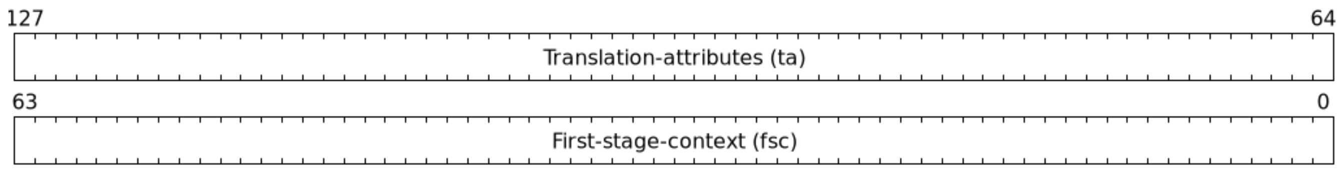


Figure 21. process-context

2.2.3. Process-context fields

Translation attributes (ta)

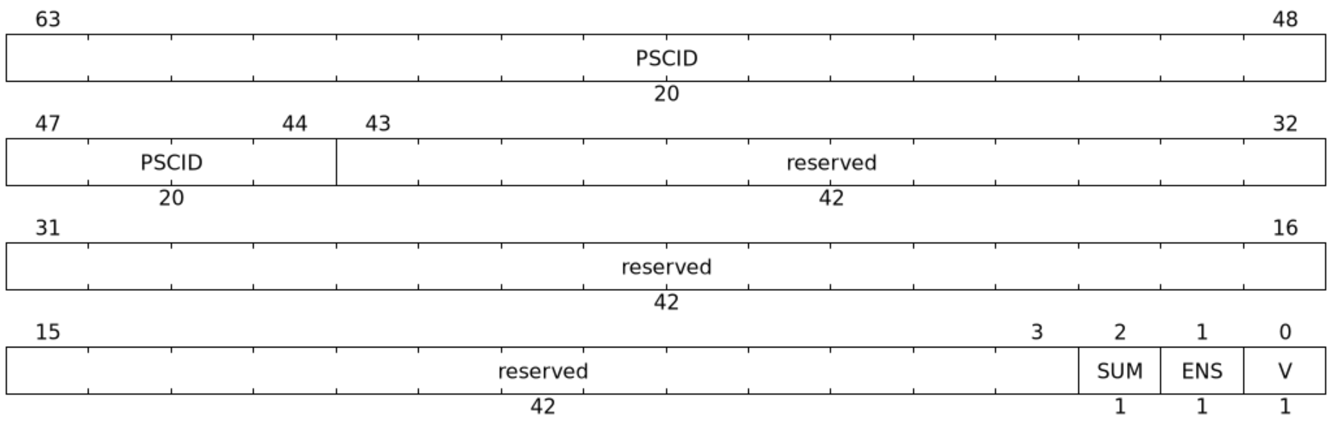


Figure 22. Translation attributes (ta) field

PC is valid if the V bit is 1; If it is 0, all other bits in PC are don't care and may be freely used by software.

When Enable-Supervisory-access (ENS) is 1, transactions requesting supervisor privilege are allowed with this process_id else the transaction is treated as a unsupported transaction.

When ENS is 1, the SUM (permit Supervisor User Memory access) bit modifies the privilege with which supervisor privilege transactions access virtual memory. When SUM=0, supervisor privilege transactions to pages mapped with U-bit in PTE set to 1 will fault.

When ENS is 1, supervisor privilege transactions that read with execute intent to pages mapped with U-bit in PTE set to 1 will fault, regardless of the state of SUM.

First-Stage context (fsc)

If PDTV is 0, the fsc field in DC holds the iosatp (when iohgatp MODE is Bare) or the iovsatp (when iohgatp MODE is not Bare) that points to a S-stage page table or VS-stage page table respectively.

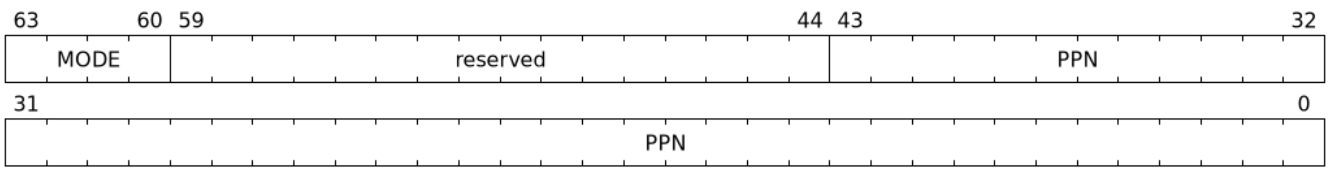


Figure 23. IO (Virtual)Supervisor addr. translation and prot. (iovsatp/iosatp) field (when PDTV is 1)

A valid (V=1) leaf PDT entry holds the PPN of the root page of a S/VS-stage page table and the MODE used to determine the S/VS-stage address translation scheme. The MODE field encodings are as defined for the MODE field in satp/vsatp CSR.

The software assigned process soft-context ID (PSCID) is used as the address space ID of the process identified by the S/VS-stage page table.

When two-stage address translation is active (`iohgatp.MODE != Bare`), the PPN field holds a guest PPN of the VS-stage page table. Addresses of the VS-stage page table entries is then converted by guest physical address translation, as controlled by the iohgatp, into a supervisor physical address. A guest OS may thus directly edit the VS-stage page table to limit access by the device to a subset of its memory and specify permissions for the device accesses.

2.3. Process to translate an IOVA

The process to translate an IOVA is as follows:

1. Use device_id to then locate the device-context as specified in Section 2.3.1.
2. If a process_id is present then locate the process-context as specified in Section 2.3.2.
3. If transaction is a write and the IOMMU support MSI address translation using MSI page tables then determine if the IOVA is a MSI address and translate it using MSI address translation process specified in Section 2.3.3.
4. If a G-stage page table is active in the device-context then use the two-stage address translation process specified in Section 8.5 of the privileged specification. The VS-stage page-table is in the process-context when process_id is valid else it is in the device-context itself.
5. If a G-stage page table is not active in the device-context then translate using the S-stage page table from device-context using the process specified in Section 4.3.2 of the privileged specification.

Exceptions detected during this process are reported using the fault/event reporting mechanism and fault record formats specified in Section 3.2.

2.3.1. Process to locate the Device-context

The process to locate the Device-context for transaction using its device_id is as follows:

1. If `ddtp.iommu_mode == Off` then stop and report "All inbound transactions disallowed" (cause = 256).
2. Let a be `ddtp.PPN x` 2^{12} and let `i = LEVELS - 1`. When ddtp.iommu_mode is 3LVL, LEVELS is

three. When `ddtp.iommu_mode` is `2LVL`, `LEVELS` is two. When `ddtp.iommu_mode` is `1LVL`, `LEVELS` is one.

3. If `i == 0` go to step 8.
4. Let `ddte` be value of eight bytes at address `a + DDI[i]x8`. If accessing `ddte` violates a PMA or PMP check, stop and report "DDT entry load access fault" (cause = 257).
5. If `ddte.V == 0`, stop and report "DDT entry not valid" (cause = 258).
6. If if any bits or encodings that are reserved for future standard use are set within `ddte`, stop and report "DDT entry misconfigured" (cause = 259).
7. Let `i = i - 1` and let `a = ddte.PPN x` 2^{12}. Go to step 3.
8. Let `dc` be value of `DC_SIZE` bytes at address `a + DDI[0]*DC_SIZE`. If `capabilities.MSI_FLAT` is 1 then `DC_SIZE` is 64-bytes else it is 32-bytes. If accessing `dc` violates a PMA or PMP check, stop and report "DDT entry load access fault" (cause = 257).
9. If `dc.V == 0`, stop and report "DDT entry not valid" (cause = 258).
10. If any bits or encodings that are reserved for future standard use are set within `dc`, stop and report "DDT entry misconfigured" (cause = 259).
11. if any of the following conditions hold then stop and report "Transaction type disallowed" (cause = 260).
    a. Transaction type is a Translated request (read, write/AMO, read-for-execute) or is a PCIe ATS Translation request or is a ATS protocol message request and `dc.tc.EN_ATS` is 0.
    b. Transaction type is a PCIe Page Request Message and `dc.tc.EN_PRI` is 0.
    c. Transaction has a valid `process_id` and `dc.tc.PDTV` is 0
    d. Transaction has a valid `process_id` and `dc.tc.PDTV` is 1 and the `process_id` is wider than supported by `pdtp.MODE`.
    e. Is a trasaction type that is not supported by the IOMMU.
12. The device-context has been successfully located.

2.3.2. Process to locate the Process-context

The process to locate the Process-context for a transaction using its `process_id` is as follows:

1. Let `a` be `pdtp.PPN x` 2^{12} and let `i = LEVELS - 1`. When `pdtp.MODE` is `PD20`, `LEVELS` is three. When `pdtp.MODE` is `PD17`, `LEVELS` is two. When `pdtp.MODE` is `PD8`, `LEVELS` is one.
2. If `i == 0` go to step 8.
3. Let `pdte` be value of eight bytes at address `a + PDI[i]x8`. If accessing `pdte` violates a PMA or PMP check, stop and report "PDT entry load access fault" (cause = 265).
4. If `pdte.V == 0`, stop and report "PDT entry not valid" (cause = 266).
5. If if any bits or encodings that are reserved for future standard use are set within `pdte`, stop and report "PDT entry misconfigured" (cause = 267).
6. Let `i = i - 1` and let `a = pdte.PPN x` 2^{12}. Go to step 2.
7. Let `pc` be value of 16-bytes at address `a+PDI[0]*16`. If accessing `pc` violates a PMA or PMP check, stop and report "PDT entry load access fault" (cause = 265).
8. If `pc.ta.V == 0`, stop and report "PDT entry not valid" (cause = 266).

9. If any bits or encodings that are reserved for future standard use are set within pc, stop and report "PDT entry misconfigured" (cause = 267).
10. if any of the following conditions hold then stop and report "Transaction type disallowed" (cause = 260).
    a. The transaction requests supervisor privilege but pc.ta.ENS is not set.
11. The Process-context has been successfully located.

2.3.3. Process to translate addresses of MSIs

When MSI address translation using MSI PTE is supported, the process to identify a incoming 32-bit write made by a device as a MSI write and translating the write using the MSI page table is as follows:

1. Let A be a 32-bit aligned 32-bit write from from a device.
2. Let dc be the device-context located using the device_id of the device using the process outlined in Section 2.3.1.
3. If dc.msiptp.MODE == Bare, then MSI address translation using MSI page tables is not enabled. Stop this process and instead use the regular translation data structures to do the address translation.
4. If (A >> 12) & dc.msi_addr_mask is not equal to dc.msi_addr_pattern & dc.msi_addr_mask then this write is not a MSI write. Stop this process and instead use the regular translation data structures to do the address translation.
5. Let the interrupt file number I be extract((A >> 12), dc.msi_addr_mask). The extract function here is the same generic bit extract performed by RISC-V instruction BEXT.
6. Let a be (dc.msiptp.PPN x 2^{12}).
7. Let msipte be the value of sixteen bytes at address (a | (I x 16)). If accessing msipte violates a PMA or PMP check, stop and report "MSI PTE load access fault" (cause = 261).
8. If msipte.V == 0, then stop and report "MSI PTE not valid" (cause = 262).
9. If msipte.C == 1, then further process is to interpret the PTE is implementation defined. If msipte.C == 0 then the process is outlined in subsequent steps.
10. If msipte.W == 1 the PTE is write-through mode PTE and the translation process is as follows:
    a. If any bits or encodings that are reserved for future standard use are set within msipte, stop and report "MSI PTE misconfigured" (cause = 262).
    b. Translate the address as outlined in Section 9.5.1 of the Advanced Interrupt Architecture specification.
11. If msipte.W == 0 the PTE is a MRIF mode PTE and the translation process is as follows:
    a. If capabilities.MSI_MRIF == 0, stop and report "MSI PTE misconfigured" (cause = 262).
    b. If any bits or encodings that are reserved for future standard use are set within msipte, stop and report "MSI PTE misconfigured" (cause = 262).
    c. Perform the process as outlined in Section 9.5.2 of the Advanced Interrupt Architecture specification. If accessing the MRIF violates a PMA or PMP check, stop and report "MRIF access fault" (cause = 264).
12. MSI address translation process is complete.

2.4. Caching in-memory data structures

To speed up DIrect Memory Access (DMA) translations, the IOMMU may make use of translation caches to hold entries from device-directory-table, process-directory-table, S/VS and G-stage translation tables, MSI page tables. These caches are collectively referred to as the IOMMU Address Translation Caches (IOATC).

These IOATC do not observe modifications to the in-memory data structures using explicit loads and stores by RISC-V harts or by device DMA. Software must use the IOMMU commands to invalidate the cached data structure entries using IOMMU commands to synchronize the IOMMU operations to observe updates to in-memory data structures. Simpler implementation may not implement IOATC for some or for any of the in-memory data structures. The IOMMU commands may use one or more IDs to tag the cached entries to identify a specific entry or a group of entries.

Table 4. Identifiers used to tag IOATC enrties

Data Structure cached	IDs used to tag entries	Invalidation command
Device Directory Table	`device_id`	IODIR.INVAL_DDT
Process Directory Table	`device_id`, `process_id`	IODIR.INVAL_PDT
S/VS-stage page tables	`GSCID`, `PSCID`, and IOVA	IOTINVAL.VMA
G-stage page table	`GSCID`, `GPA`	IOTINVAL.GVMA
MSI page table	`device_id`, MSI-interrupt-file-number	IOTINVAL.MSI

Chapter 3. In-memory queue interface

Software and IOMMU interact using 3 in-memory queue data structures.

- A command-queue (CQ) used by software to queue commands to the IOMMU.
- A fault/event queue (FQ) used by IOMMU to bring faults and events to software attention..
- A page-request queue (PQ) used by IOMMU to report “Page Request” messages received from PCIe devices connected to the IOMMU. This queue is supported if the IOMMU supports PCIe ATS specification.

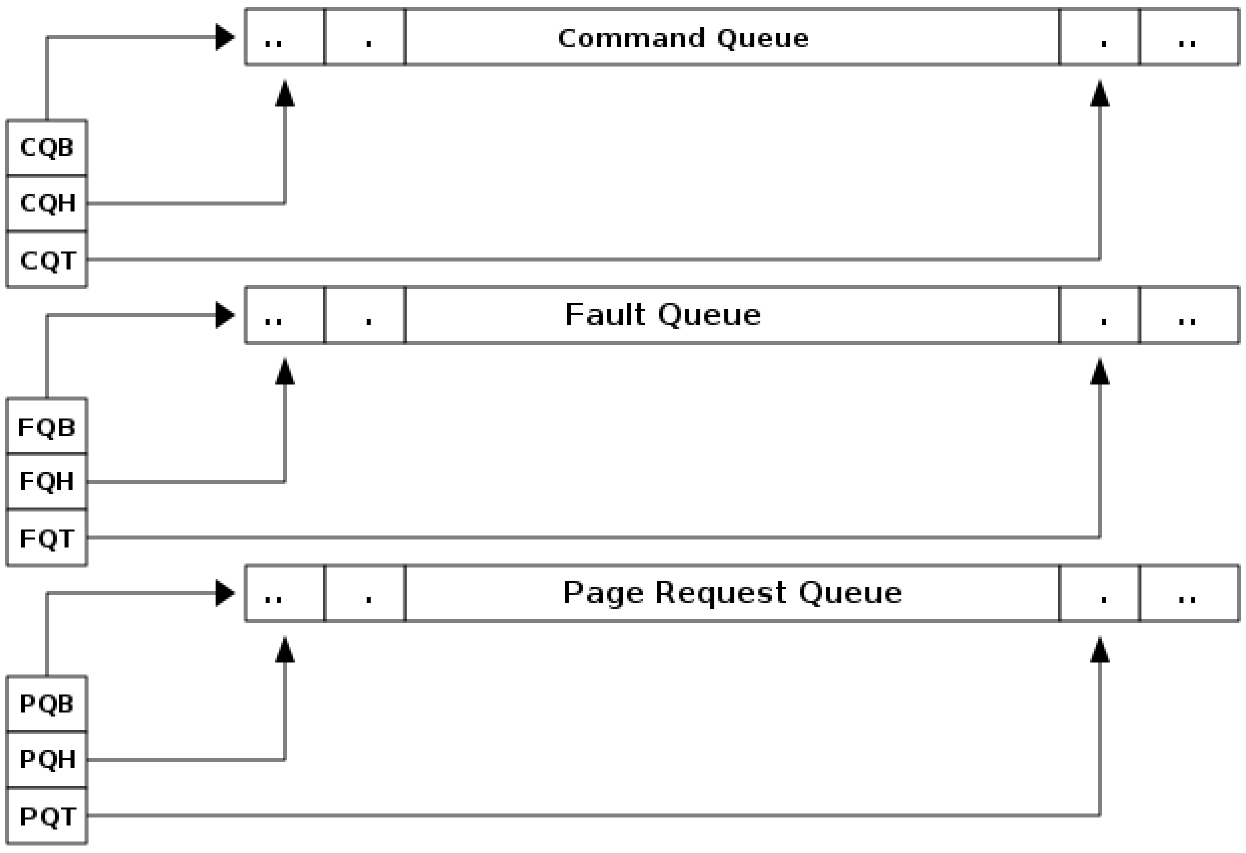


Figure 24. IOMMU in-memory queues

Each queue is a circular buffer with a head controlled by the consumer of data in the queue and a tail controlled by the producer of data into the queue. IOMMU is the producer of records into PQ and FQ and controls the tail register. IOMMU is the consumer of commands produced by software into the CQ and controls the head register.

A queue is empty if the head is equal to the tail. A queue is full if the tail is one minus the head. The head and tail wrap around when they reach the end of the circular buffer.

The producer of data must ensure that the data written to queue memory and the tail update are ordered such that the consumer that observes an update to the tail register must also observe all data produced into the queue between the offsets determined by the head and the tail.

All RISC-V IOMMU implementations are required to support in-memory queues located in main memory. Supporting in-memory queues in I/O memory is not required.

3.1. Command-Queue (CQ)

Command queue is used by software to queue commands to be executed by the IOMMU.

The PPN of the base of this in-memory queue and the size of the queue is configured into a memory-mapped register called command-queue base (`cqb`).

The tail of the command-queue resides in a software controlled read/write memory-mapped register called command-queue tail (`cqt`). The `cqt` is an index into the next command queue entry that software will write. Subsequent to writing the command(s), software advances the `cqt` by the count of the number of commands written.

The head of the command-queue resides in a read-only memory-mapped IOMMU controlled register called command-queue head (`cqh`). The `cqh` is an index into the command that IOMMU should process next. Subsequent to processing each command IOMMU advances the `cqh` by 1. If `cqh` == `cqt`, the command-queue is empty. If `cqt` == (`cqh` - 1) the command-queue is full.

IOMMU commands are grouped into a major command group determined by the `opcode` and within each group the `func3` field specifies the function invoked by that command. The `opcode` defines format of the operand fields. One or more of those fields may be used by the specific function invoked.

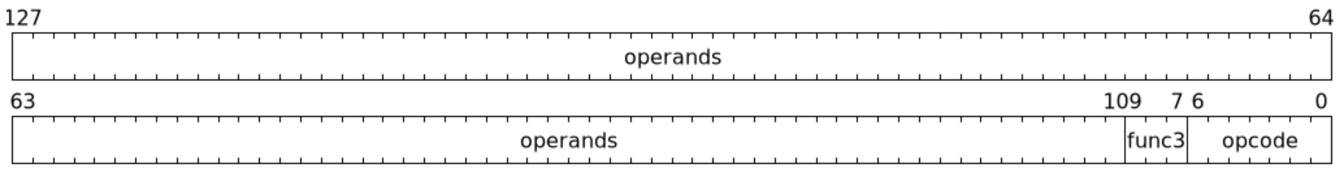
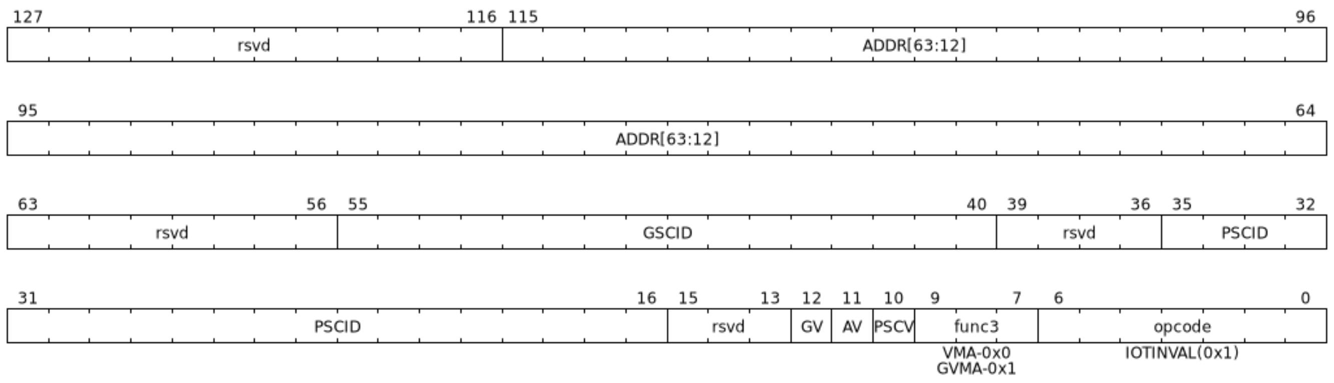


Figure 25. Format of an IOMMU command

3.1.1. IOMMU Page-Table cache invalidation commands



IOMMU operations cause implicit reads and writes to PDT, first-stage and second-stage page tables; however these implicit references are ordinarily not ordered with respect to explicit load and stores to these data structures by the RISC-V hart in the machine.

The IOMMU translation-table cache invalidation commands, IOTINVAL.VMA and IOTINVAL.GVMA synchronize updates to in-memory first-stage and second-stage page table data structures with the operation of the IOMMU and flush the matching IOATC entries.

The `GV` operand indicates if the Guest Soft-Context ID (`GSCID`) operand is valid. The `PSCV` operand indicates if the Process Soft-Context ID (`PSCID`) operand is valid. Setting `PSCV` to 1 is valid only with `IOTINVAL.VMA`. The `AV` operand indicates if the address (`ADDR`) operand is valid. When `GV` is 0, the translations associated with the host (i.e. those where the second-stage translation is not active) are operated on.

`IOTINVAL.VMA` guarantees that previous stores made to the first-stage page tables by the harts are observed before all subsequent implicit reads from IOMMU to the corresponding first-stage page tables.

Table 5. `IOTINVAL.VMA` operands and operations

`GV`	`AV`	`PSCV`	Operation
0	0	0	Invalidates cached information from any level of the S-stage page table, for all host address spaces.
0	0	1	Invalidates cached information from any level of the S-stage page tables, but only for the host address space identified by `PSCID` operand. Accesses to global mappings are not ordered.
0	1	0	Invalidates cached information from leaf S-stage page table entries corresponding to the IOVA in `ADDR`, for all host address spaces.
0	1	1	Invalidates cached information from leaf S-stage page table entries corresponding to the IOVA in `ADDR`, for the host address space identified by `PSCID` operand. Global mappings are not invalidated.
1	0	0	Invalidates cached information from any level of the VS-stage page table, for all VM address spaces associated with `GSCID`.
1	0	1	Invalidates cached information from any level of the VS-stage page tables, but only for the VM address space identified by `PSCID` and `GSCID` operand. Accesses to global mappings are not ordered.
1	1	0	Invalidates cached information from leaf VS-stage page table entries corresponding to the IOVA in `ADDR`, for all VM address spaces associated with the `GSCID` operand.
1	1	1	Invalidates cached information from leaf VS-stage page table entries corresponding to the IOVA in `ADDR`, for the VM address space identified by `PSCID` and `GSCID` operand. Global mappings are not invalidated.

`IOTINVAL.GVMA` guarantees that previous stores made to the G-stage page tables are observed before all subsequent implicit reads from IOMMU to the corresponding G-stage page tables. Setting `PSCV` to 1 with IOTINVAL.GVMA is illegal.

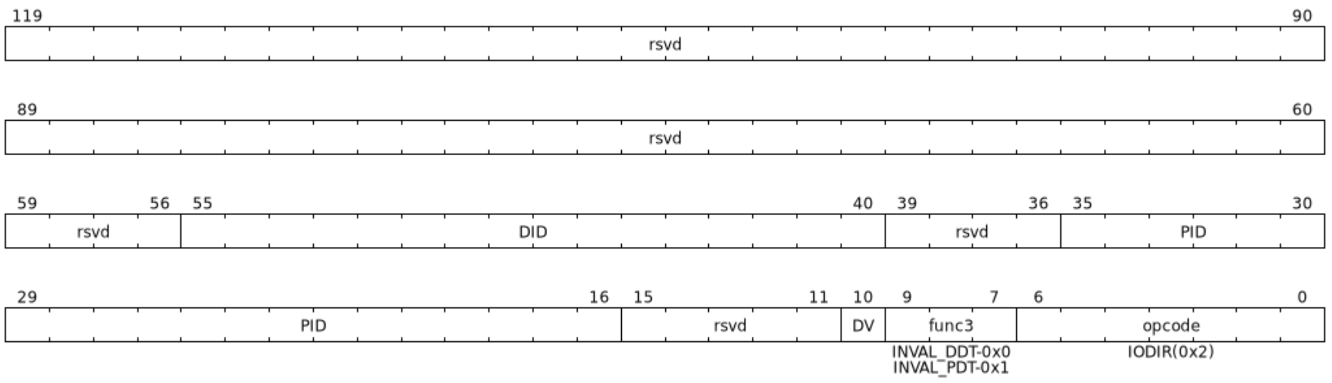
Table 6. IOTINVAL.GVMA operands and operations

GV	AV	Operation
0	n/a	Invalidates cached information from any level of the G-stage page table, for all VM address spaces.
1	0	Invalidates cached information from any level of the G-stage page tables, but only for all VM address spaces identified by the GSCID operand.
1	1	Invalidates cached information from leaf G-stage page table entries corresponding to the guest-physical-address in ADDR operand, for the all VM address spaces identified GSCID operand.

Implementations that cache VA to PA translations may ignore the guest-physical-address in ADDR operand of IOTINVAL.GVMA, when valid, and perform an invalidation of all virtual-addresses in VM address spaces identified by the GSCID operand if valid or all host address spaces if the GSCID operand is not valid.

Simpler implementations may ignore the operand of IOTINVAL.VMA and/or IOTINVAL.GVMA and always perform a global invalidation across all address spaces.

3.1.2. IOMMU directory cache commands



IOMMU operations cause implicit reads and writes to DDT and/or PDT; however these implicit references are ordinarily not ordered with respect to explicit load or store to DDT or PDT by the RISC-V harts in the machine.

The IOMMU DDT cache invalidation command, IODIR.INVAL_DDT synchronize updates to DDT with the operation of the IOMMU and flush the matching cached entries.

The IOMMU PDT cache invalidation command, IODIR.INVAL_PDT synchronize updates to PDT with the operation of the IOMMU and flush the matching cached entries.

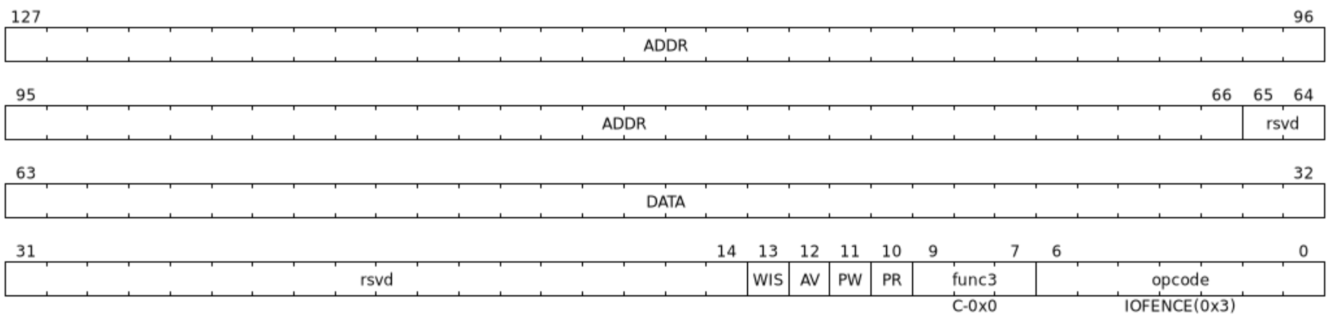
The DV operand indicates if the device ID (DID) operand is valid.

IODIR.INVAL_DDT guarantees that any previous stores made by a RISC-V hart to the DDT are observed before all subsequent implicit reads from IOMMU to DDT. If DV is 0, then the command invalidates all DDT and PDT entries cached for all devices. If DV is 1, then the command invalidates cached leaf level DDT entry for the device identified by DID operand and all associated PDT entries. The PID operand is reserved for IODIR.INVAL_DDT.

`IODIR.INVA_PDT` guarantees that any previous stores made by a RISC-V hart to the PDT are observed before all subsequent implicit reads from IOMMU to PDT. The command invalidates leaf PDT entry for the specified `PID` and `DID`.

Some pointers in the Device directory or Process Directory may be guest-physical addresses. If G-stage page table used for these translations are modified, software must send the appropriate IOTINVAL command and IODIR command to the Command-queue. Indeed, some implementations may choose to cache the translated pointers in the IOMMU directory caches. IOTINVAL has no effect on the IOMMU directory caches.

3.1.3. IOMMU Command-queue Fence commands



The IOMMU fetches commands from the CQ in order but the IOMMU may execute the fetched commands out of order. The IOMMU advancing `cqh` is not a guarantee that the commands fetched by the IOMMU have been executed or committed.

A `IOFENCE.C` command guarantees that all previous commands fetched from the CQ have been completed and committed.

The commands may be used to order memory accesses from I/O devices connected to the IOMMU as viewed by the IOMMU, other RISC-V harts, and external devices or coprocessors. The `PR` and `PW` bits can be used to request that the IOMMU ensure that all previous requests from devices that have already been processed by the IOMMU be committed to a global ordering point such that they can be observed by all RISC-V harts and IOMMUs in the machine.

The wired-interrupt-signaling (`WIS`) bit when set to 1 causes a wired-interrupt from the command queue to be generated on completion of `IOFENCE.C`. This bit is reserved if the IOMMU supports MSI.

Software should ensure that all previous read and writes processed by the IOMMU have been committed to a global ordering point before reclaiming memory made accessible to a device. A safe sequence for such memory reclamation is to first update the page tables to disallow access to the memory from the device and then use the `IOTINVAL.VMA` or `IOTINVAL.GVMA` appropriately to synchronize the IOMMU with the update to the page table. As part of the synchronization if the memory reclaimed was previously made read accessible to the device then request ordering of all previous reads; else if the memory reclaimed was previously made write accessible to the device then request ordering of all previous reads and writes to the IOFENCE. Ordering previous reads may be required if the reclaimed memory will be used to hold data that must not be made visible to the device.

The ordering guarantees are made for accesses to main-memory. For accesses to I/O memory, the ordering guarantees are implementation and I/O protocol defined.

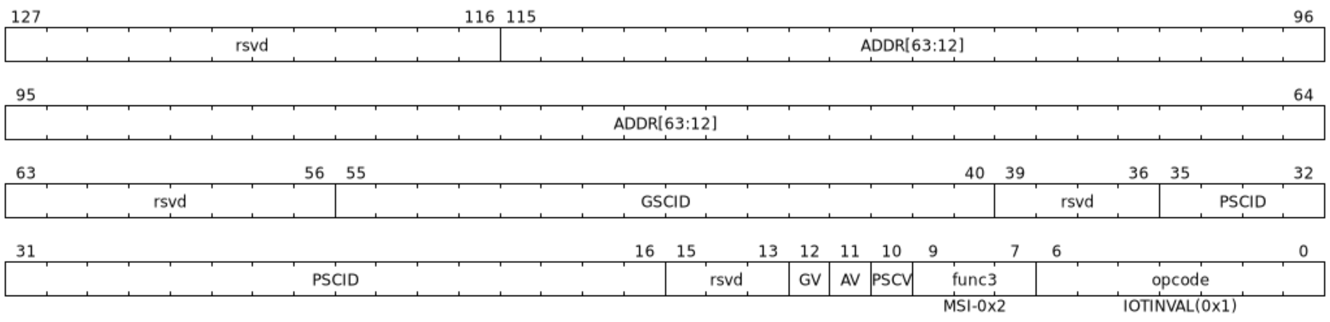
Simpler implementations may unconditionally order all previous memory accesses globally.

The `AV` command operand indicates if `ADDR` operand and DATA operands are valid. If `AV`=1, the IOMMU writes `DATA` to memory at a 4-byte aligned address in `ADDR` operand as a 4-byte store.

Software may configure the ADDR command operand to specify the address of the sereipnum_le/seteipnum_be in an IMSIC to cause an external interrupt notification on `IOFENCE.C` completion. Alternatively, software may program ADDR to a memory location and use `IOFENCE.C` to set a flag in memory indicating command completion.

3.1.4. IOMMU MSI table cache invalidation commands

This command is supported if `capabilities.MSI_FLAT` is set.



IOMMU operations cause implicit reads and writes to the MSI page table; however these implicit references are ordinarily not ordered with respect to explicit loads or stores to these data structures by the RISC-V harts in the machine.

`IOTINVAL.MSI` synchronizes updates to the MSI page table with the operation of the IOMMU and invalidates the matching cache entries.

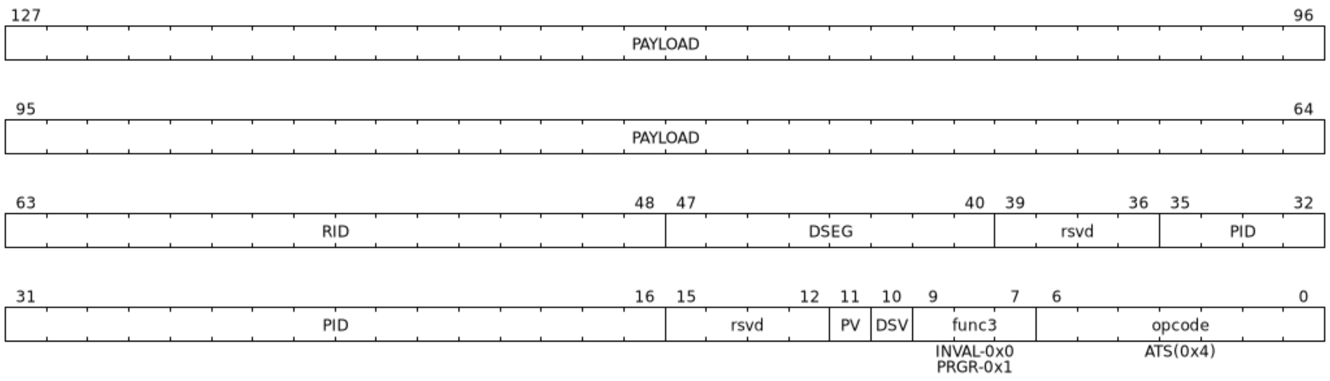
The `PSCV` operand is reserved and must be 0 for `IOTINVAL.MSI`.

Table 7. `IOTINVAL.MSI` operands and operations

AV	GV	Operation
0	0	Invalidates all cached MSI page table entries for host associated devices.
0	1	Invalidates MSI page table entry identified by INT_FILE_NUM for host associated devices.
1	0	Invalidates all cached MSI page table entries of VM identified by GSCID operand.
1	1	Invalidates MSI page table entry identified by INT_FILE_NUM of VM identified by GSCID operand.

3.1.5. IOMMU ATS commands

This command is supported if capabilities.ATS is set.



The ATS.INVAL command instructs the IOMMU to send a “Invalidation Request” message to the PCIe device function identified by RID. An “Invalidation Request” message is used to clear a specific subset of the address range from the address translation cache in a device function. The ATS.INVAL command completes when an “Invalidation Completion” response message is received from the device or a protocol defined timeout occurs while waiting for a response. The IOMMU may advance the cqh and fetch more commands from CQ while a response is awaited.

Software that needs to know if the invalidation operation completed on the device may use the IOMMU command-queue fence command (IOFENCE.C) to wait for the responses to all prior “Invalidation Request” messages. The IOFENCE.C is guaranteed to not complete before all previously fetched commands were executed and completed. A previously fetched ATS command to invalidate device ATC does not complete till either the request times out or a valid response is received from the device.

The ATS.PRGR command instructs the IOMMU to send a “Page Group Response” message to the PCIe device function identified by the RID. The “Page Group Response” message is used by system hardware and/or software to communicate with the device functions page-request interface to signal completion of a “Page Request”, or the catastrophic failure of the interface.

If the PV operand is set to 1, the message is generated with a PASID TLP prefix with the PASID field set to the PID operand.

The PAYLOAD operand of the command is used to form the message body.

If the DSV operand is 1, then a valid destination segment number is specified by the DSEG operand.

3.2. Fault/Event-Queue (FQ)

Fault/Event queue is an in-memory queue data structure used to report events and faults raised when processing transactions. Each fault record is 64 bytes.

The PPN of the base of this in-memory queue and the size of the queue is configured into a memory-mapped register called fault-queue base (fqb).

The tail of the fault-queue resides in a IOMMU controlled read-only memory-mapped register called fqt. The fqt is an index into the next fault record that IOMMU will write in the fault-queue. Subsequent to writing the record, the IOMMU advances the fqt by 1. The head of the fault-queue resides in a read/write memory-mapped software controlled register called fqh. The fqh is an index into the next fault record that SW should process next. Subsequent to processing fault record(s) software advances the fqh by the count of the number of fault records processed. If fqh == fqt, the fault-queue is empty. If fqt == (fqh - 1) the fault-queue is full.

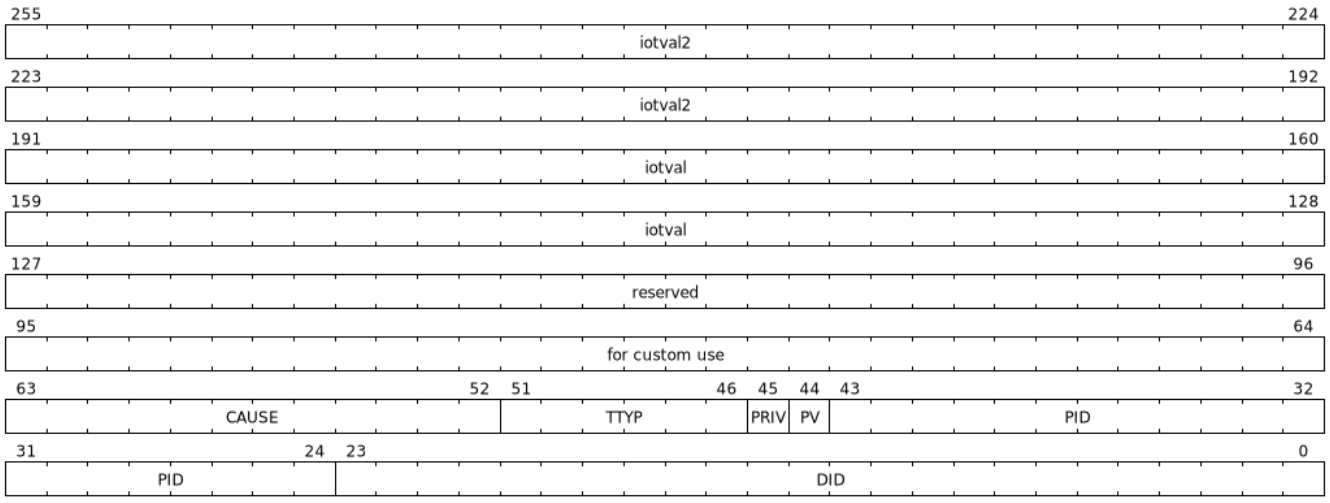


Figure 26. Fault-queue record

The CAUSE is a code indicating the cause of the fault/event.

Table 8. Fault record CAUSE field encodings

CAUSE	Description
0	Instruction address misaligned
1	Instruction access fault
4	Read address misaligned
5	Read access fault
6	Write/AMO address misaligned
7	Write/AMO access fault
12	Instruction page fault
13	Read page fault
15	Write/AMO page fault

CAUSE	Description
20	Instruction guest page fault
21	Read guest-page fault
23	Write/AMO guest-page fault
256	All inbound transactions disallowed (`ddtp.iommu_mode == Off`)
257	DDT entry load access fault
258	DDT entry not valid
259	DDT entry misconfigured - reserved fields not 0, unsupported encodings
260	Transaction type disallowed
261	MSI PTE load access fault
262	MSI PTE not valid
262	MSI PTE misconfigured
264	MRIF access fault
265	PDT entry load access fault.
266	PDT entry not valid
267	PDT entry misconfigured - reserved fields not 0, unsupported encodings

The `TTYP` field reports inbound transaction type.

Table 9. Fault record `TTYP` field encodings

TTYP	Description
0	None. Fault not caused by an inbound transaction.
1	Untranslated read for execute transaction
2	Untranslated read transaction
3	Untranslated write/AMO transaction
4	Translated read for execute transaction
5	Translated read transaction
6	Translated write/AMO transaction
7	PCIe ATS Translation Request
8	Message Request
9 - 15	Reserved
16 - 31	Reserved for custom use

If the `TTYP` is a transaction with an IOVA then its reported in `iotval`. If the `TTYP` is a message request then the message code is reported in `iotval`.

`DID` holds the `device_id` of the transaction. If `PV` is 0, then `PID` and `PRIV` are 0. If `PV` is 1, the `PID` holds a process_id of the transaction and if the privilege of the transaction was Supervisor then `PRIV` bit is 1 else its 0.

If the `CAUSE` is guest-page fault then the guest-physical-address right shifted by 2 is reported in `iotval[63:2]`. If bit 0 of `iotval2` 1, then guest-page-fault was caused by an implicit memory access for VS-stage address translation. If bit 0 of `iotval2` is 1, and the implicit access was a write then bit 1 is set to 1 else its set to 0.

The IOMMU may be unable to report faults through the fault-queue due to error conditions such as the fault-queue being full or the IOMMU encountering access faults when attempting to access the queue memory. A memory-mapped fault control and status register (`fqcsr`) holds information about such faults. If the fault-queue full condition is detected the IOMMU sets a fault-queue overflow (`fqof`) bit in fqcsr. If the IOMMU encounters a fault in accessing the fault-queue memory, the IOMMU sets a fault-queue memory access fault (`fqmf`) bit in fqcsr. While either error bits are set in fqcsr, the IOMMU discards the record that led to the fault and all further fault records. When an error bit is in the fqcsr changes state from 0 to 1 or when a new fault record is produced in the fault-queue, fault interrupt pending (`fip`) bit is set in the `fqcsr`.

3.3. Page-Request-Queue (PQ)

Page-request queue is an in-memory queue data structure used to report PCIe ATS “Page Request” messages to software. The base PPN of this in-memory queue and the size of the queue is configured into a memory-mapped register called page-request queue base (`pqb`).

The tail of the queue resides in a IOMMU controlled read-only memory-mapped register called `pqt`. The `pqt` holds an index into the queue where the next page-request message will be written by the IOMMU. Subsequent to writing the message, the IOMMU advances the `pqt` by 1.

The head of the queue resides in a software controlled read/write memory-mapped register called `pqh`. The `pqh` holds an index into the queue where the next page-request message will be received by software. Subsequent to processing the message(s) software advances the `pqh` by the count of the number of messages processed.

If `pqh` == `pqt`, the page-request queue is empty.

If `pqt` == (`pqh` - 1) the page-request queue is full.

The IOMMU may be unable to report page-requests through the queue due to error conditions such as the queue being full or the IOMMU encountering access faults when attempting to access queue memory. A memory-mapped page-request queue control and status register (pqcsr) is used to hold information about such faults. If the queue full condition is detected, the IOMMU discards the message and may auto-complete the page-request as specified by the PCIe ATS specification. On a page queue full condition the page-request-queue overflow (`pqof`) bit is set in `pqcsr`. If the IOMMU encountered a fault in accessing o the queue memory, page-request-queue memory access fault (`pqmf`) bit in `pqcsr`. While either error bits are set in `pqcsr`, the IOMMU discards subsequent page-request messages and may auto-complete them using an IOMMU generated “Page Group Response” message as specified by PCIe ATS specifications.

When an error bit is in the pqcsr changes state from 0 to 1 or when a new message is produced in the queue, page-request-queue interrupt pending (`pip`) bit is set in the `pqcsr`.

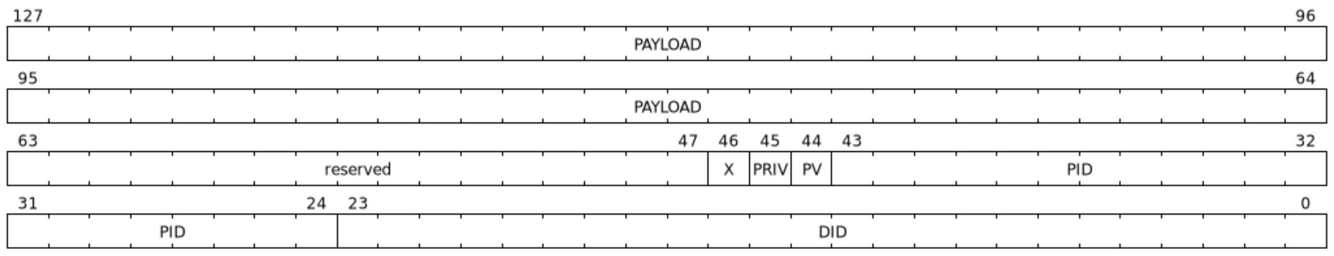


Figure 27. Page-request-queue record

The `DID` field holds the requester ID from the message. The `PID` field is valid if `PV` is 1 and reports the PASID from the PASID TLP prefix of the message. `PRIV` is set to 0 if the message did not have a PASID TLP prefix, otherwise it holds the “Privilege Mode Requested” bit from the PASID TLP prefix. `X` bit is set to 0 if the message did not have a PASID TLP prefix, otherwise it reports the “Execute Requested” bit from the PASID TLP prefix. All other fields are set to 0. The payload of the “Page Request” message (bytes 0x08 through 0x0F of the message) is held in the `PAYLOAD` field.

Chapter 4. Memory-mapped register interface

The IOMMU provides a memory-mapped programming interface. The memory-mapped registers of each IOMMU are located within a naturally aligned 4-KiB region (a page) of physical address space.

The IOMMU behavior for register accesses where the address is not aligned to the size of the access or if the access spans multiple registers is undefined.

The reset default value of register fields is 0 unless explicitly noted otherwise in the register definition.

Register fields are assigned one of the attributes described in following table.

Table 10. Register and Register bit-field types

Attribute	Description
RW	Read-Write - Register bits are read-write and are permitted to be either Set or Cleared by software to the desired state. If the optional feature that is associated with the bits is not implemented, the bits are permitted to be hardwired to Zero.
RO	Read-only - Register bits are read-only and cannot be altered by software. Where explicitly defined, these bits are used to reflect changing hardware state, and as a result bit values can be observed to change at run time. If the optional feature that would Set the bits is not implemented, the bits must be hardwired to Zero
RW1C	Write-1-to-clear status - Register bits indicate status when read. A Set bit indicates a status event which is Cleared by writing a 1b. Writing a 0b to RW1C bits has no effect. If the optional feature that would Set the bit is not implemented, the bit must be read-only and hardwired to Zero
WPRI	Reserved Writes Preserve Values, Reads Ignore Values. See RISC-V privileged specification for a detailed definition.
WARL	Write Any Values, Reads Legal Values. See RISC-V privileged specification for a detailed definition.

4.1. Register layout

Table 11. IOMMU Memory-mapped register layout

Offset	Name	Size	Description
0	`capabilities`	8	Capabilities supported by the IOMMU
8	`fctrl`	4	Features control

Offset	Name	Size	Description
12	*custom*	4	*For custom use*
16	ddtp	8	Device directory table pointer
24	cqb	8	Command-queue base
32	cqh	4	Command-queue head
36	cqt	4	Command-queue tail
40	fqb	8	Fault-queue base
48	fqh	4	Fault-queue head
52	fqt	4	Fault-queue tail
56	pqb	8	Page-request-queue base
64	pqh	4	Page-request-queue head
68	pqt	4	Page-request-queue tail
72	cqcsr	4	Command-queue control and status register
76	fqcsr	4	Fault-queue control and status register
80	pqcsr	4	Page-request-queue control and status register
84	ipsr	4	Interrupt pending status register
88	iocntovf	4	Performance-monitoring counter overflow status
92	iocntinh	4	Performance-monitoring counter inhibits
96	iohpmcycles	8	Performance-monitoring cycles counter
104	iohpmctr1 - 31	248	Performance-monitoring event counters
352	iohpmevt1 - 31	248	Performance-monitoring event selector
600	iocntconf	4	Counters reserved for confidential mode usage
604	Reserved	78	Reserved for future use (WPRI)
682	*custom*	78	*Reserved for custom use (WARL)*
760	icvec	4	Interrupt cause to vector register
768	msi_cfg_tbl	256	MSI Configuration Table
1024	Reserved	3072	Reserved for future use (WPRI)

4.2. IOMMU capabilities (capabilities)

The capabilities register is a read-only register reporting features supported by the IOMMU. Each field if not clear indicates presence of that feature in the IOMMU. At reset, the register shall contain the IOMMU supported features.

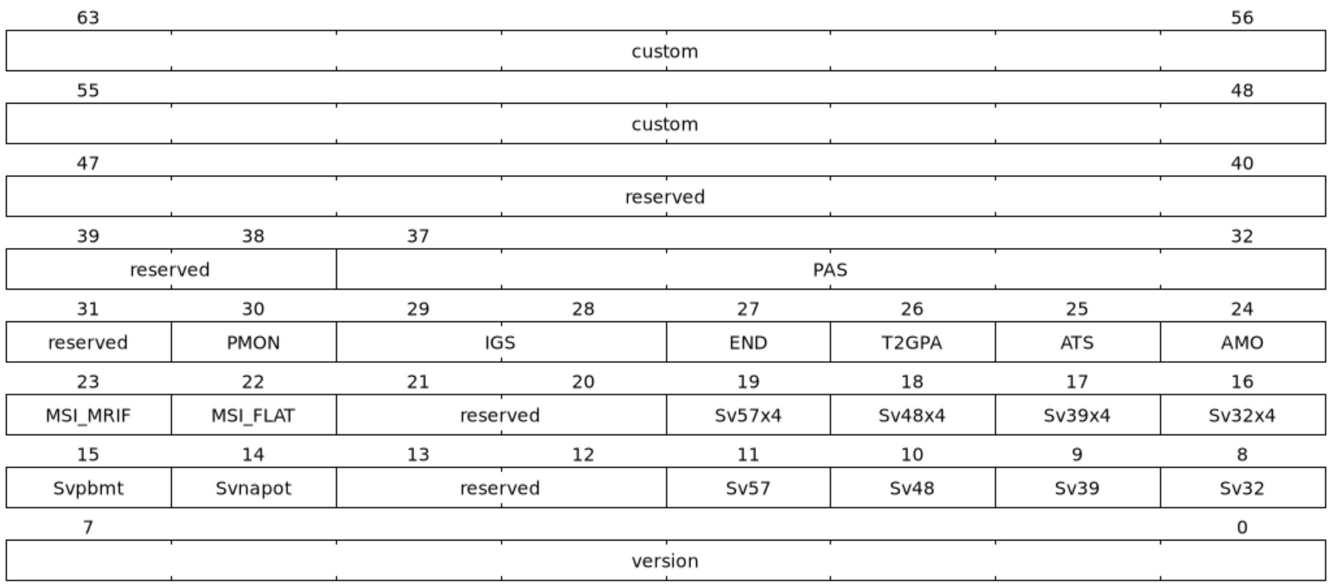


Figure 28. capabilities register fields

Bits	Field	Attribute	Description
7:0	`version`	RO	The `version` field holds the version of the specification implemented by the IOMMU. The low nibble is used to hold the minor version of the specification and the upper nibble is used to hold the major version of the specification. For example, an implementation that supports version 1.0 of the specification reports 0x10.
8	`Sv32`	RO	Page-based 32-bit virtual addressing is supported
9	`Sv39`	RO	Page-based 39-bit virtual addressing is supported
10	`Sv48`	RO	Page-based 48-bit virtual addressing is supported When `Sv48` field is set, `Sv39` field must be set.
11	`Sv57`	RO	Page-based 57-bit virtual addressing is supported When `Sv57` field is set, `Sv48` field must be set.
13:12	reserved	RO	Reserved for standard use.
14	`Svnapot`	RO	NAPOT translation contiguity.
15	`Svpbmt`	RO	Page-based memory types.
16	`Sv32x4`	RO	Page-based 34-bit virtual addressing for G-stage translation is supported.
17	`Sv39x4`	RO	Page-based 41-bit virtual addressing for G-stage translation is supported.
18	`Sv48x4`	RO	Page-based 50-bit virtual addressing for G-stage translation is supported.
19	`Sv57x4`	RO	Page-based 59-bit virtual addressing for G-stage translation is supported.
21:20	reserved	RO	Reserved for standard use.
22	`MSI_FLAT`	RO	MSI address translation using Write-through mode MSI PTE is supported.

Bits	Field	Attribute	Description
23	MSI_MRIF	RO	MSI address translation using MRIF mode MSI PTE is supported.
24	AMO	RO	Atomic updates to MRIF and PTE accessed (A) and dirty (D) bit is supported.
25	ATS	RO	PCIe Address Translation Services (ATS) and page-request interface (PRI) is supported.
26	T2GPA	RO	Returning guest-physical-address in ATS translation completions is supported.
27	END	RO	When 0, IOMMU supports one endianness (either little or big). When 1, IOMMU supports both endianness. The endianness is defined in fctrl register.
29:28	IGS	RO	IOMMU interrupt generation support.
30	PMON	RO	IOMMU implements a performance-monitoring unit
31	reserved	RO	Reserved for standard use
37:32	PAS	RO	Physical Address Size (value between 32 and 56)
47:38	reserved	RO	Reserved for standard use
63:48	*custom*	RO	*Reserved for custom use*

IGS values:

Value	Name	Description
0	MSI	IOMMU supports only MSI generation.
1	WIS	IOMMU supports only wire interrupt generation.
2	BOTH	IOMMU supports both MSI and wire interrupt generation. The interrupt generation method must be defined in fctrl register.
3	0	Reserved for standard use

Hypervisor may provide an SW emulated IOMMU to allow the guest to manage the VS-stage page tables for fine grained control on memory accessed by guest controlled devices.

A hypervisor that provides such an emulated IOMMU to the guest may retain control of the G-stage page tables and clear the SvNx4 fields of the emulated capabilities register.

A hypervisor that provides such an emulated IOMMU to the guest may retain control of the MSI page tables used to direct MSI to guest interrupt files in an IMSIC or to a memory-resident-interrupt-file and clear the MSI_FLAT and MSI_MRIF fields of the emulated capabilities register.

4.3. Features-control register (fctrl)

This register must be readable in any implementation. An implementation may allow one or more fields in the register to be writeable to support enabling or disabling the feature controlled by that field.

If software enables or disables a feature when the IOMMU is not OFF (i.e. `ddtp.iommu_mode == Off`) then the IOMMU behavior is `UNSPECIFIED`.

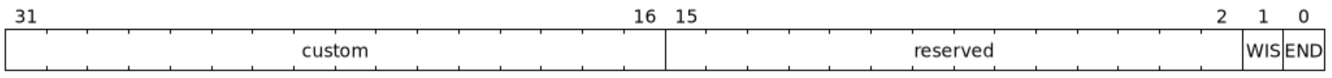


Figure 29. Features-control register fields

Bits	Field	Attribute	Description
0	END	WARL	When 0, IOMMU accesses to memory resident data structures (e.g. DDT, PDT, in-memory queues, S/VS and G stage page tables) are performed as little-endian accesses and when 1 as big-endian accesses.
1	WIS	WARL	When 1, IOMMU interrupts are signaled as wired-interrupts.
15:2	WPRI	WPRI	Reserved for standard use
31:16	*custom*		*These bits are reserved for custom use.*

4.4. Device-directory-table pointer (ddtp)

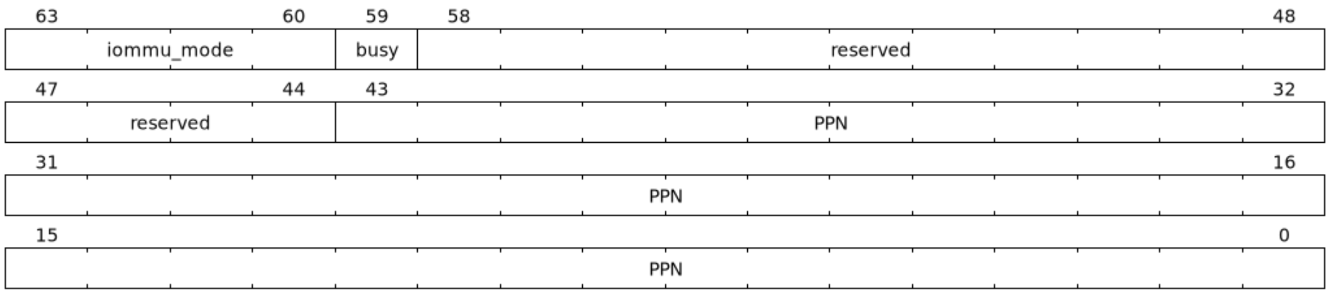


Figure 30. Device-directory-table pointer register fields

Bits	Field	Attribute	Description
43:0	PPN	WARL	Holds the PPN of the root page of the device-directory-table.
58:44	WPRI	WPRI	Reserved for standard use

Bits	Field	Attribute	Description
59	busy	RW	A write to ddtp may require the IOMMU to perform many operations that may not occur synchronously to the write. When a write is observed by the ddtp, the busy bit is set to 1. When the busy bit is 1, behavior of additional writes to the ddtp is implementation defined. Some implementations may ignore the second write and others may perform the actions determined by the second write. Software must verify that the busy bit is 0 before writing to the ddtp. If the busy bit reads 0 then the IOMMU has completed the operations associated with the previous write to ddtp. An IOMMU that can complete these operations synchronously may hardwire this bit to 0.
59	iommu_mode	RW	The IOMMU may be configured to be in following modes:

Value	Name	Description
0	Off	No inbound memory transactions are allowed by the IOMMU.
1	Bare	No translation or protection. All inbound memory accesses are passed through.
2	1LVL	One-level device-directory-table
3	2LVL	Two-level device-directory-table
4	3LVL	Three-level device-directory-table

The device-context is 64-bytes in size if capabilities.MSI_FLAT is 1 else it is 32-bytes.

When the 'iommu_mode' is 'Bare' or Off, the PPN field is don’t-care.

All IOMMU must support Off and Bare mode. An IOMMU is allowed to support a subset of directory-table levels and device-context widths. At a minimum one of the modes must be supported.

When the iommu_mode field value is changed the IOMMU guarantees that in-flight transactions from devices connected to the IOMMU will be processed with the configurations applicable to the old value of the iommu_mode field and that all transactions and previous requests from devices that have already been processed by the IOMMU be committed to a global ordering point such that they can be observed by all RISC-V hart, devices, and IOMMUs in the platform.

The reset default for the iommu_mode is recommended to be Off.

4.5. Command-queue base (`cqb`)

This 64-bits register (RW) holds the PPN of the root page of the command-queue and number of entries in the queue.

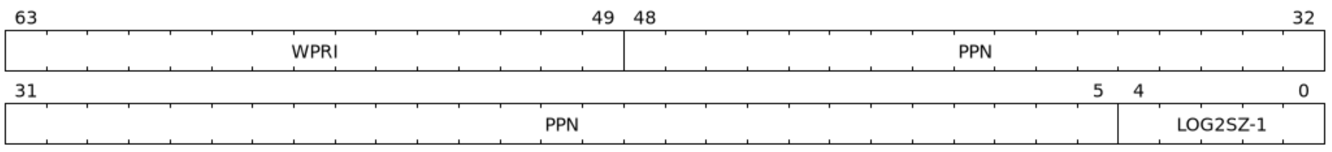


Figure 31. Command-queue base register fields

Bits	Field	Attribute	Description
4:0	`LOG2SZ-1`	WARL	The `LOG2SZ-1` field holds the number of entries in command-queue as a log to base 2 minus 1. A value of 0 indicates a queue of 2 entries. Each IOMMU command is 16-bytes. If the command-queue has 256 or fewer entries then the base address of the queue is always aligned to 4-KiB. If the command-queue has more than 256 entries then the command-queue base address must be naturally aligned to 2^{LOG2SZ} `x 16`.
48:5	`PPN`	WARL	Holds the `PPN` of the root page of the in-memory command-queue used by software to queue commands to the IOMMU.
63:49	`WPRI`	WPRI	Reserved for standard use

4.6. Command-queue head (`cqh`)

This 32-bits register (RO) holds the index into the command-queue where the IOMMU will fetch the next command.

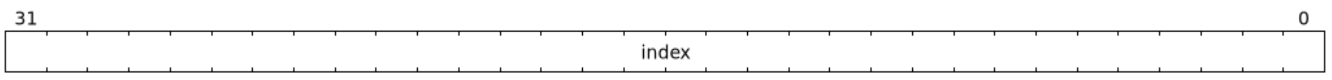


Figure 32. Command-queue head register fields

Bits	Field	Attribute	Description
31:0	`index`	RO	Holds the `index` into the command-queue from where the next command will be fetched next by the IOMMU.

4.7. Command-queue tail (`cqt`)

This 32-bits register (RW) holds the index into the command-queue where the software queues the next command for the IOMMU.

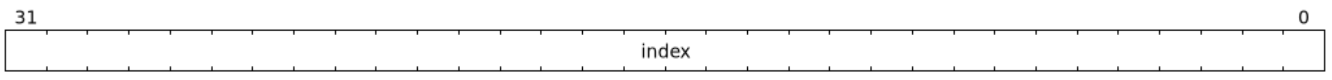


Figure 33. Command-queue tail register fields

Bits	Field	Attribute	Description
31:0	index	WARL	Holds the index into the command-queue where software queues the next command for IOMMU. Only LOG2SZ:0 bits are writeable when the queue is in enabled state (i.e., cqsr.cqon == 1).

4.8. Fault queue base (fqb)

This 64-bits register (RW) holds the PPN of the root page of the fault-queue and number of entries in the queue.

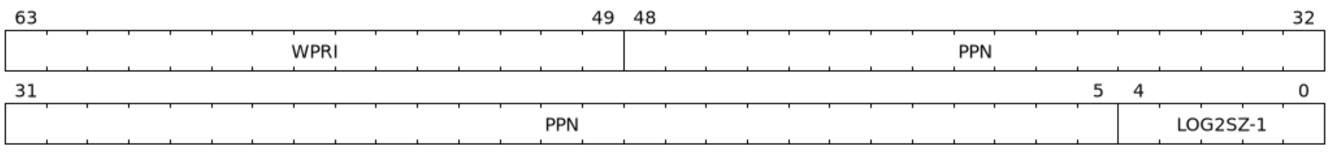


Figure 34. Fault queue base register fields

Bits	Field	Attribute	Description
4:0	LOG2SZ-1	WARL	The LOG2SZ-1 field holds the number of entries in fault-queue as a log-to-base-2 minus 1. A value of 0 indicates a queue of 2 entries. Each fault record is 64-bytes. If the fault-queue has 64 or fewer entries then the base address of the queue is always aligned to 4-KiB. If the fault-queue has more than 64 entries then the fault-queue base address must be naturally aligned to 2^{LOG2SZ} x 64.
48:5	PPN	WARL	Holds the PPN of the root page of the in-memory fault-queue used by IOMMU to queue fault record.
63:49	WPRI	WPRI	Reserved for standard use

4.9. Fault queue head (fqh)

This 32-bits register (RW) hodlds the index into fault-queue where the software will fetch the next fault reccord.

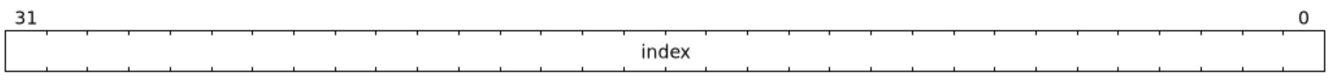


Figure 35. Fault queue head register fields

Bits	Field	Attribute	Description
31:0	index	WARL	Holds the index into the fault-queue from which software reads the next fault record. Only LOG2SZ:0 bits are writeable when the queue is in enabled state (i.e., fqsr.fqon == 1).

4.10. Fault queue tail (fqt)

This 32-bits register (RO) holds the index into the fault-queue where the IOMMU queues the next fault record.

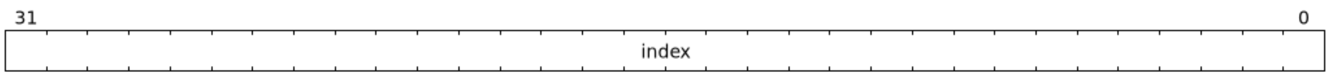


Figure 36. Fault queue tail register fields

Bits	Field	Attribute	Description
31:0	`index`	RO	Holds the `index` into the fault-queue where IOMMU writes the next fault record.

4.11. Page-request-queue base (`pqb`)

This 64-bits register (RW) holds the PPN of the root page of the page-request-queue and number of entries in the queue.

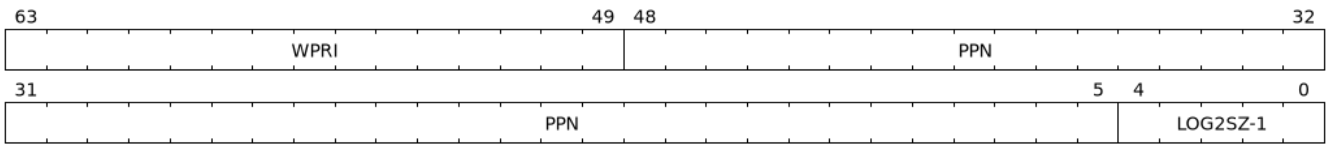


Figure 37. Page-Request-queue base register fields

Bits	Field	Attribute	Description
4:0	`LOG2SZ-1`	WARL	The `LOG2SZ-1` field holds the number of entries in page-request-queue as a log-to-base-2 minus 1. A value of 0 indicates a queue of 2 entries. Each page-request is 16-bytes. If the page-request-queue has 256 or fewer entries then the base address of the queue is always aligned to 4-KiB. If the page-request-queue has more than 256 entries then the page-request-queue base address must be naturally aligned to 2^{LOG2SZ} `x 16`.
48:5	`PPN`	WARL	Holds the `PPN` of the root page of the in-memory page-request-queue used by IOMMU to queue page requests.
63:49	`WPRI`	WPRI	Reserved for standard use

4.12. Page-request-queue head (`pqh`)

This 32-bits register (RW) holds the index into the page-request-queue where software will fetch the next page-request.

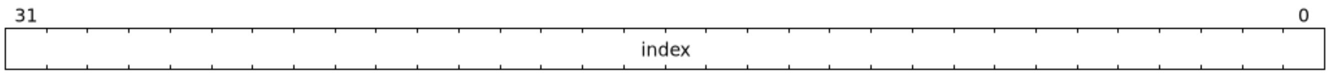


Figure 38. Page-request-queue head register fields

Bits	Field	Attribute	Description
31:0	`index`	WARL	Holds the `index` into the page-request-queue from which software reads the next page request. Only `LOG2SZ:0` bits are writeable when the queue is in enabled state (i.e., `pqsr.pqon == 1`).

4.13. Page-request-queue tail (pqt)

This 32-bits register (RO) holds the index into the page-request-queue where the IOMMU writes the next page-request.

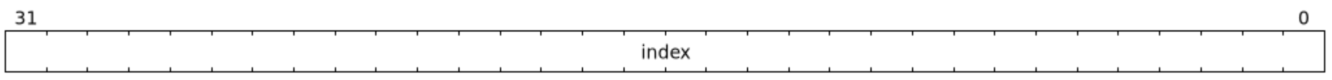


Figure 39. Page-request-queue tail register fields

Bits	Field	Attribute	Description
31:0	index	RO	Holds the index into the page-request-queue where IOMMU writes the next page request.

4.14. Command-queue CSR (cqcsr)

This 32-bits register (RW) is used to control the operations and report the status of the command-queue.

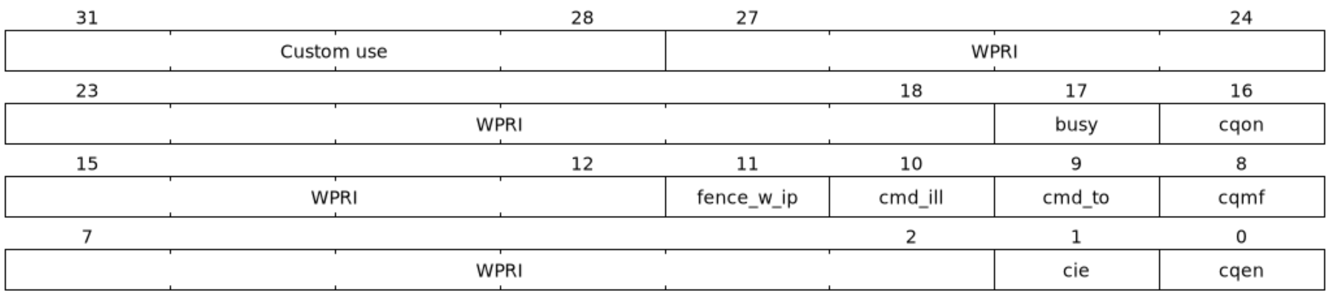


Figure 40. Command-queue CSR register fields

Bits	Field	Attribute	Description
0	cqen	RO	The command-queue-enable bit enables the command- queue when set to 1. Changing cqen from 0 to 1 sets the cqh, cqt to 0 and sets cqcsr bits cmd_ill, cmd_to, cqmf, fence_w_ip to 0. The command-queue may take some time to be active following setting the cqen to 1. When the command queue is active, the cqon bit reads 1. When cqen is changed from 1 to 0, the command queue may stay active till the commands already fetched from the command-queue are being processed and/or there are outstanding implicit loads from the command-queue. When the command-queue turns off, the cqon bit reads 0. When the cqon bit reads 0, the IOMMU guarantees that no implicit memory accesses to the command queue are in-flight and the command-queue will not generate new implicit loads to the queue memory.
1	cie	RW	Command-queue-interrupt-enable bit enables generation of interrupts from command-queue when set to 1.

Bits	Field	Attribute	Description
7:2	WPRI	WPRI	Reserved for standard use
8	cqmf	RW1C	If command-queue access leads to a memory fault then the command-queue-memory-fault bit is set to 1 and the command-queue stalls until this bit is cleared. When cqmf is set to 1, an interrupt is generated if an interrupt is not already pending (i.e., ipsr.cip == 1) and not masked (i.e. cqsr.cie == 0). To re-enable command processing, software should clear this bit by writing 1.
9	cmd_to	RW1C	If the execution of a command leads to a timeout (e.g. a command to invalidate device ATC may timeout waiting for a completion), then the command-queue sets the cmd_to bit and stops processing from the command-queue. When cmd_to is set to 1 an interrupt is generated if an interrupt is not already pending (i.e., ipsr.cip == 1) and not masked (i.e. cqsr.cie == 0). To re-enable command processing software should clear this bit by writing 1.
10	cmd_ill	RW1C	If an illegal or unsupported command is fetched and decoded by the command-queue then the command-queue sets the cmd_ill bit and stops processing from the command-queue. When cmd_ill is set to 1, an interrupt is generated if not already pending (i.e. ipsr.cip == 1) and not masked (i.e. cqsr.cie == 0). To re-enable command processing software should clear this bit by writing 1.
11	fence_w_ip	RW1C	An IOMMU that supports only wired interrupts sets fence_w_ip bit is set to indicate completion of a IOFENCE.C command. An interrupt on setting fence_w_ip if not already pending (i.e. ipsr.cip == 1) and not masked (i.e. cqsr.cie == 0) and fence_w_ip is 0. To re-enable interrupts on IOFENCE.C completion software should clear this bit by writing 1. This bit is reserved if the IOMMU uses MSI.
15:12	WPRI	WPRI	Reserved for standard use
16	cqon	RO	The command-queue is active if cqon is 1. IOMMU behavior on changing cqb when busy is 1 or cqon is 1 is implementation defined. The software recommended sequence to change cqb is to first disable the command-queue by clearing cqen and waiting for both busy and cqon to be 0 before changing the cqb.

Bits	Field	Attribute	Description
17	busy	RO	A write to cqcsr may require the IOMMU to perform many operations that may not occur synchronously to the write. When a write is observed by the cqcsr, the busy bit is set to 1. When the busy bit is 1, behavior of additional writes to the cqcsr is implementation defined. Some implementations may ignore the second write and others may perform the actions determined by the second write. Software must verify that the busy bit is 0 before writing to the cqcsr. An IOMMU that can complete controls synchronously may hardwire this bit to 0. An IOMMU that can complete these operations synchronously may hardwire this bit to 0.
27:18	WPRI	WPRI	Reserved for standard use
31:28	*custom*		*These bits are reserved for custom use.*

Command-queue being empty does not imply that all commands fetched from the command-queue have been completed. When the command-queue is requested to be disabled, an implementation may either complete the already fetched commands or abort execution of those commands. Software must use an IOFENCE.C *command to wait for all previous commands to be committed, if so desired, before turning off the command-queue.*

4.15. Fault queue CSR (fqcsr)

This 32-bits register (RW) is used to control the operations and report the status of the fault-queue.

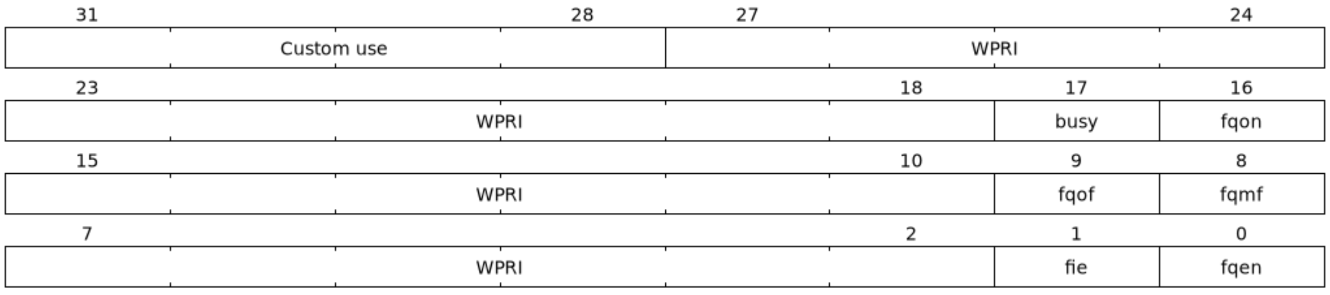


Figure 41. Fault queue CSR register fields

Bits	Field	Attribute	Description
0	`fqen`	RO	The fault-queue enable bit enables the fault-queue when set to 1. Changing `fqen` from 0 to 1, resets the `fqh` and `fqt` to 0 and clears `fqcsr` bits `fqmf` and `fqof`. The fault-queue may take some time to be active following setting the `fqen` to 1. When the fault queue is active, the `fqon` bit reads 1. When `fqen` is changed from 1 to 0, the fault-queue may stay active till in-flight fault-recording is completed. When the fault-queue is off, the `fqon` bit reads 0. The IOMMU guarantees that there are no in-flight implicit writes to the fault-queue in progress when `fqon` reads 0 and no new fault records will be written to the fault-queue.
1	`fie`	RW	Fault queue interrupt enable bit enables generation of interrupts from fault-queue when set to 1.
7:2	`WPRI`	WPRI	Reserved for standard use
8	`fqmf`	RW1C	The `fqmf` bit is set to 1 if the IOMMU encounters an access fault when storing a fault record to the fault queue. The fault-record that was attempted to be written is discarded and no more fault records are generated until software clears `fqmf` bit by writing 1 to the bit. An interrupt is generated if enabled and not already pending (i.e. `ispr.fip == 1`) and not masked (i.e. `fqsr.fie == 0`).
9	`fqof`	RW1C	The fault-queue-overflow bit is set to 1 if the IOMMU needs to queue a fault record but the fault-queue is full (i.e., `fqh == fqt - 1`) The fault-record is discarded and no more fault records are generated till software clears `fqof` by writing 1 to the bit. An interrupt is generated if not already pending (i.e. `ispr.fip == 1`) and not masked (i.e. `fqsr.fie == 0`).
10:15	`WPRI`	WPRI	Reserved for standard use
16	`fqon`	RO	The fault-queue is active if `fqon` reads 1. IOMMU behavior on changing `fqb` when `busy` is 1 or `pqon` is 1 implementation defined. The recommended sequence to change `fqb` is to first disable the fault-queue by clearing `fqen` and waiting for both `busy` and `fqon` to be 0 before changing `fqb`.

Bits	Field	Attribute	Description
17	`busy`	RO	Write to `fqcsr` may require the IOMMU to perform many operations that may not occur synchronously to the write. When a write is observed by the fqcsr, the `busy` bit is set to 1. When the `busy` bit is 1, behavior of additional writes to the `fqcsr` are implementation defined. Some implementations may ignore the second write and others may perform the actions determined by the second write. Software should ensure that the `busy` bit is 0 before writing to the `fqcsr`. An IOMMU that can complete controls synchronously may hardwire this bit to 0.
27:18	`WPRI`	WPRI	Reserved for standard use
31:28	*custom*		*These bits are reserved for custom use.*

4.16. Page-request-queue CSR (`pqcsr`)

This 32-bits register (RW) is used to control the operations and report the status of the page-request-queue.

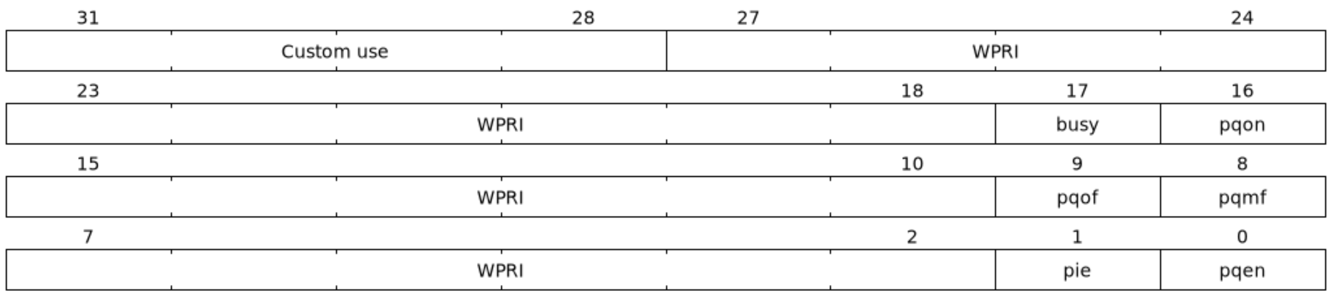


Figure 42. Page-request-queue CSR register fields

Bits	Field	Attribute	Description
16	pqon	RO	The page-request-enable bit enables the page-request-queue when set to 1. Changing pqen from 0 to 1, resets the pqh and pqt to 0 and clears pqcsr bits pqmf and pqof to 0. The page request queue may take some time to be active following setting the pqen to 1. When the page-request-queue is active, the pqon bit reads 1. When pqen is changed from 1 to 0, the page-request-queue may stay active till in-flight page-request writes are completed. When the page-request-queue turns off, the pqon bit reads 0. When pqon reads 0, the IOMMU guarantees that there are no older in-flight implicit writes to the queue memory and no further implicit writes will be generated to the queue memory. The IOMMU may respond to “Page Request” messages received when page-request-queue is off or in the process of being turned off, as having encountered a catastrophic error as defined by the PCIe ATS specifications
1	pie	RW	The page-request-queue-interrupt-enable (pie) bit when set to 1, enables generation of interrupts from page request queue.
7:2	WPRI	WPRI	Reserved for standard use
8	pqmf	RW1C	The pqmf bit is set to 1 if the IOMMU encounters an access fault when storing a page-request message to the page-request-queue. When pqmf is set to 1, an interrupt is generated if not already pending (i.e. `ipsr.pip == 1`) and not masked (i.e. `pqsr.pie == 1`). The page-request message that caused the pqmf or pqof error and all subsequent page-request messages are discarded till software clears the pqof and/or pqmf bits by writing 1 to it. The IOMMU may respond to “Page Request” messages that caused the pqof or pqmf bit to be set and all subsequent “Page Request” messages received while these bits are 1 as having encountered a catastrophic error as defined by the PCIe ATS specifications

Bits	Field	Attribute	Description
9	pqof	RW1C	The page-request-queue-overflow bit is set to 1 if the page-request queue overflows i.e. IOMMU needs to queue a page-request message but the page-request queue is full (i.e., pqh == pqt - 1). When pqof is set to 1, an interrupt is generated if not already pending (i.e. ipsr.pip == 1) and not masked (i.e. pqsr.pie == 1). The page-request message that caused the pqmf or pqof error and all subsequent page-request messages are discarded till software clears the pqof and/or pqmf bits by writing 1 to it. The IOMMU may respond to “Page Request” messages that caused the pqof or pqmf bit to be set and all subsequent “Page Request” messages received while these bits are 1 as having encountered a catastrophic error as defined by the PCIe ATS specifications
15:10	WPRI	WPRI	Reserved for standard use
16	pqon	RO	The page-request is active when pqon reads 1. IOMMU behavior on changing pqb when busy is 1 or pqon is 1 implementation defined. The recommended sequence to change pqb is to first disable the page-request queue by clearing pqen and waiting for both busy and pqon to be 0 before changing pqb.
17	busy	RO	A write to pqcsr may require the IOMMU to perform many operations that may not occur synchronously to the write. When a write is observed by the pqcsr, the busy bit is set to 1. When the busy bit is 1, behavior of additional writes to the pqcsr are implementation defined. Some implementations may ignore the second write and others may perform the actions determined by the second write. Software should ensure that the busy bit is 0 before writing to the pqcsr. An IOMMU that can complete controls synchronously may hardwire this bit to 0
27:18	WPRI	WPRI	Reserved for standard use
31:28	*custom*		*These bits are designated for custom use.*

4.17. Interrupt pending status register (ipsr)

This 32-bits register (RW1C) reports the pending interrupts which require software service. Each interrupt-pending bit in the register corresponds to a interrupt source in the IOMMU. When an interrupt-pending bit in the register is set to 1 the IOMMU will not signal another interrupt from that source till software clears that interrupt-pending bit by writing 1 to clear it.

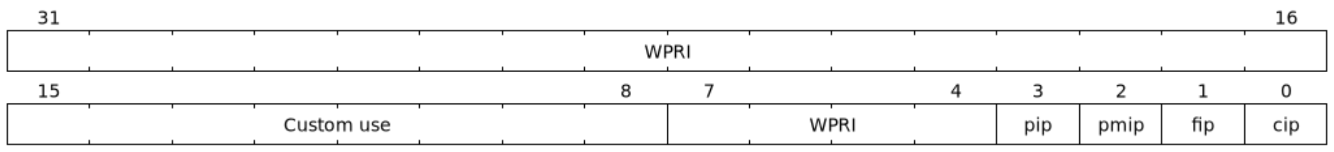


Figure 43. Interrupt pending status register fields

Bits	Field	Attribute	Description
0	cip	RW1C	The command-queue-interrupt-pending
1	fip	RW1C	The fault-queue-interrupt-pending
2	pmip	RW1C	The performance-monitoring-interrupt-pending
3	pip	RW1C	The page-request-queue-interrupt-pending
7:4	WPRI	WPRI	Reserved for standard use
15:8	*custom*		*These bits are designated for custom use.*
31:16	WPRI	WPRI	Reserved for standard use

4.18. Performance-monitoring counter overflow status (iocountovf)

The performance-monitoring counter overflow status is a 32-bit read-only register that contains shadow copies of the OF bits in the iohpmevt* registers - where iocntovf bit X corresponds to iohpmevtX and bit 0 corresponds to the OF bit of iohpmcycles.

This register enables overflow interrupt handler software to quickly and easily determine which counter(s) have overflowed.

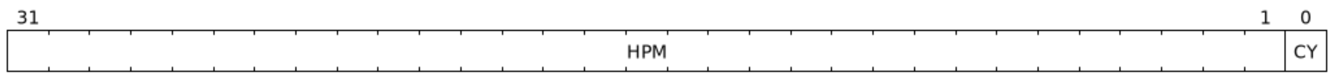


Figure 44. iocntovf register fields

Bits	Field	Attribute	Description
0	CY	RO	Shadow of iohpmcycles.OF
31:1	HPM	RO	Shadow of iohpmevt*.OF

4.19. Performance-monitoring counter inhibits (iocountinh)

The performance-monitoring counter inhibits is a 32-bits WARL register where that contains bits to inhibit the corresponding counters from counting. Bit X when set inhibits counting in iohpmctrX and bit 0 inhibits counting in iohpmcycles.

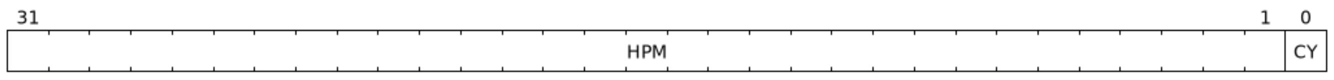


Figure 45. iocntinh register fields

Bits	Field	Attribute	Description
0	CY	RW	When set, iohpmcycles counter is inhibited from counting.
31:1	HPM	WARL	When bit X is set, then counting of events in iohpmctrX is inhibited.

When the iohpmcycles counter is not needed, it is desirable to conditionally inhibit it to reduce energy consumption. Providing a single register to inhibit all counters allows a) one or more counters to be atomically programmed with events to count b) one or more counters to be sampled atomically.

4.20. Performance-monitoring cycles counter (iohpmcycles)

This 64-bits register is a free running clock cycle counter. There is no associtated iohpmevt0.

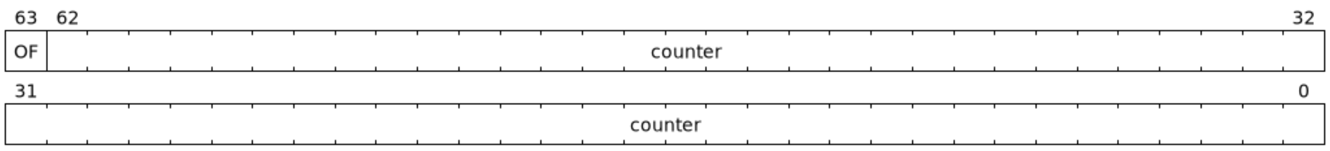


Figure 46. iohpmcycles register fields

Bits	Field	Attribute	Description
62:0	counter	WARL	Cycles counter value.
63	OF	RW	Overflow

When capabilities.HPM is set, the iohpmcycles register must be present and be at least a 32-bits wide.

4.21. Performance-monitoring event counters (iohpmctr1-31)

These registers are 64-bit WARL counter registers.

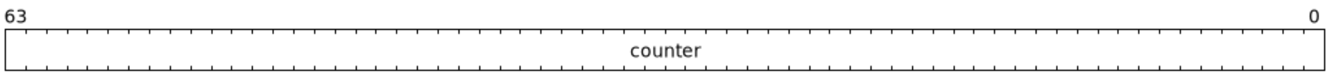


Figure 47. iohpmctr register fields*

Bits	Field	Attribute	Description
63:0	counter	WARL	Event counter value.

When capabilities.HPM is set, the iohpmctr1-7 registers must be present and be at least 32-bits wide.

4.22. Performance-monitoring event selector (iohpmevt1-31)

These performance-monitoring event registers are 64-bit RW registers. When a transaction processed by the IOMMU causes an event that is programmed to count in a counter then the counter is incremented. In addition to matching events the event selector may be programmed with additional filters based on device_id, process_id, GSCID, and PSCID such that the counter is incremented conditionally based on the transaction matching these additional filters. When such device_id based filtering is used, the match may be configured to be a precise match or a partial match. A partial match allows a transactions with a range of IDs to be counted by the counter.

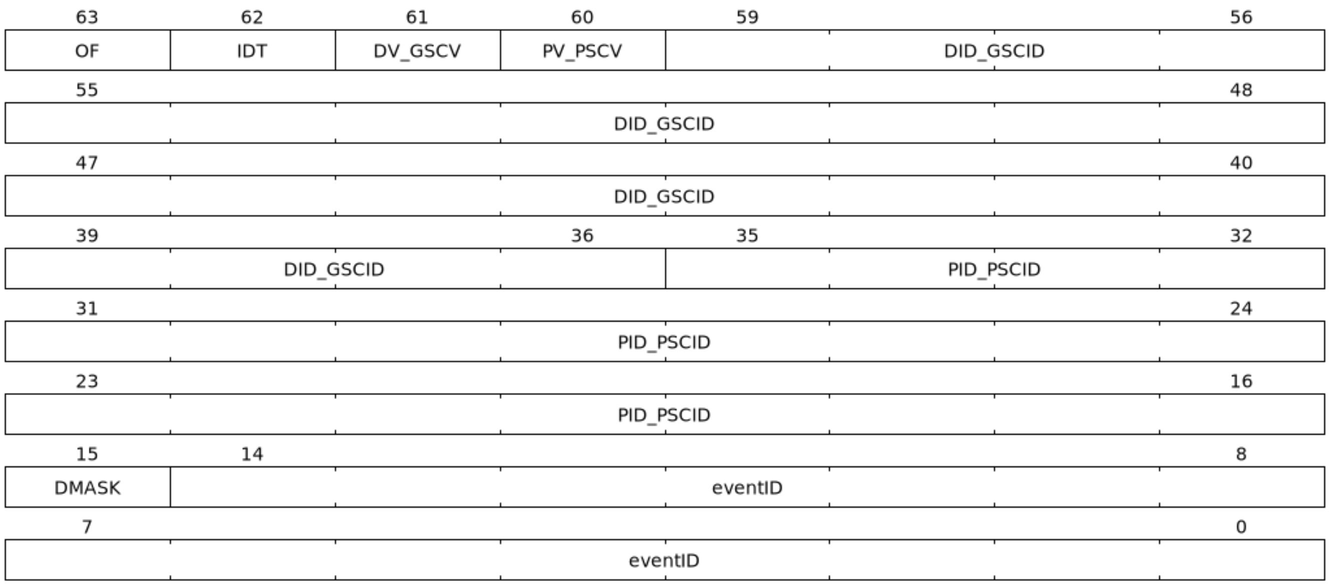


Figure 48. iohpmevt register fields*

Bits	Field	Attribute	Description
14:0	eventID	WARL	Indicates the event to count. A value of 0 indicates no events are counted. Other event IDs are implementation defined. When eventID is changed, including to 0, the counter retains its value.
15	DMASK	WARL	When set to 1, partial matching of the DID_GSCID is performed for the transaction. The lower bits of the DID_GSCID all the way to the first low order 0 bit (including the 0 bit position itself) are masked.
35:16	PID_PSCID	WARL	process_id if IDT is 0, PSCID if IDT is 1
59:36	DID_GSCID	WARL	device_id if IDT is 0, GSCID if IDT is 1.
60	PV_PSCV	WARL	If set, only transactions with matching process_id or PSCID (based on the Filter ID Type) are counted.
61	DV_GSCV	WARL	If set, only transactions with matching device_id or GSCID (based on the Filter ID Type) are counted.

Bits	Field	Attribute	Description
62	IDT	WARL	Filter ID Type: This field indicates the type of ID to filter on. When 0, the DID_GSCID field holds a device_id and the PID_PSCID field holds a process_id. When 1, the DID_GSCID field holds a GSCID and PID_PSCID field holds a PSCID.
63	OF	WARL	Overflow status or Interrupt disable

When capabilities.HPM is set, the iohpmevt1-7 registers must be present.

Some events types may be filtered by IDs. When a event type that does not support filtering by IDs is programmed then the filtering options are ignored. The table below summarizes the filtering option for events that support filtering by IDs.

Table 12. filtering options

IDT	DV_GSCV	PV_PSCV	Operation
0/1	0	0	Counter increments. No ID based filtering.
0	0	1	If the transaction has a valid process_id, counter increments if process_id matches PID_PSCID.
0	1	0	Counter incremented if device_id matches DID_GSCID.
0	1	1	If the transaction does not have a valid process_id, counter increments if device_id matches DID_GSCID. If the transaction has a valid process_id, counter increments if device_id matches DID_GSCID and process_id matches PID_PSCID.
1	0	1	If the transaction has a valid process_id, counter increments if the PSCID of that process matches PID_PSCID.
1	1	0	Counter incremented if GSCID of the device matches DID_GSCID.
1	1	1	If the transaction does not have a valid process_id, counter increments if GSCID of the device matches DID_GSCID. If the transaction has a valid process_id, counter increments if GSCID of the device matches DID_GSCID and PSCID of the process matches PID_PSCID.

When filtering by device_id or GSCID is selected and the event supports ID based filtering, the DMASK field can be used to configure a partial match. When DMASK is set to 1, partial matching of the DID_GSCID is performed for the transaction. The lower bits of the DID_GSCID all the way to the first low order 0 bit (including the 0 bit position itself) are masked.

The following example illustrates the use of DMASK and filtering by device_id.

Table 13. DMASK with IDT set to device_id based filtering

DMASK	DID_GSCID	Comment
0	yyyyyyyy yyyyyyyy yyyyyyyy	One specific seg:bus:dev:func

DMASK	DID_GSCID	Comment
1	yyyyyyyy yyyyyyyy yyyyy011	seg:bus:dev - any func
1	yyyyyyyy yyyyyyyy 01111111	seg:bus - any dev:func
1	yyyyyyyy 01111111 11111111	seg - any bus:dev:func

The `OF` bit is set when the corresponding `iohpmctr*` overflows, and remains set until cleared by software. Since `iohpmctr*` values are unsigned values, overflow is defined as unsigned overflow. Note that there is no loss of information after an overflow since the counter wraps around and keeps counting while the sticky `OF` bit remains set.

If an `iohpmctr*` overflows while the associated `OF` bit is zero, then a HPM Counter Overflow interrupt is generated. If the `OF` bit is one, then no interrupt request is generated. Consequently the `OF` bit also functions as a count overflow interrupt disable for the associated `iohpmctr*`.

A pending HPM Counter Overflow interrupt (OR of all `iohpmctr*` overflows) is and reported through `ipcsr` register.

There are not separate overflow status and overflow interrupt enable bits. In practice, enabling overflow interrupt generation (by clearing the `OF` bit) is done in conjunction with initializing the counter to a starting value. Once a counter has overflowed, it and the `OF` bit must be reinitialized before another overflow interrupt can be generated.

4.23. Counters reserved for confidential mode usage (`iocntconf`)

Reserved for future standard use.

4.24. Interrupt-cause-to-vector register (`icvec`)

Interrupt-cause-to-vector register maps a cause to a vector. All causes can be mapped to same vector or a cause can be given a unique vector.

The vector is used:

1. By a MSI capable IOMMU to index into MSI configuration table (`msi_cfg_tbl`) to determine the MSI to generate. A IOMMU is capable of generating IOMMU generated interrupt as a MSI if `capabilities.IGS==MSI` or if `capabilities.IGS==BOTH` and `fctrl.WIS` is set to 1.
2. By a non-MSI capable IOMMU determine the wire to use to signal the interrupt

If an implementation only supports a single vector then all bits of this register may be hardwired to 0 (WARL). Likewise if only two vectors are supported then only bit 0 for each cause could be writeable.

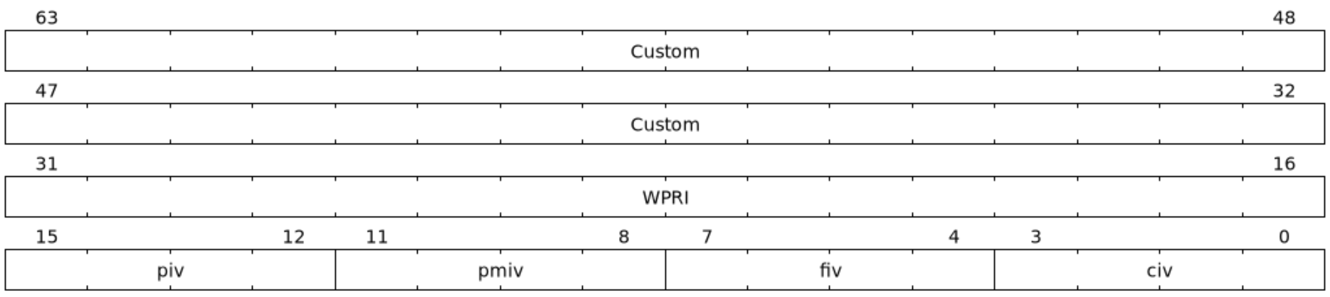


Figure 49. icvec register fields

Bits	Field	Attribute	Description
3:0	civ	WARL	The command-queue-interrupt-vector (civ) is the vector number assigned to the command-queue-interrupt.
7:4	fiv	WARL	The fault-queue-interrupt-vector (fiv) is the vector number assigned to the fault-queue-interrupt.
11:8	pmiv	WARL	The performance-monitoring-interrupt-vector (pmiv) is the vector number assigned to the performance-monitoring-interrupt.
15:12	piv	WARL	The page-request-queue-interrupt-vector (piv) is the vector number assigned to the page-request-queue-interrupt.
31:16	WPRI	WPRI	Reserved for standard use
63:32	*custom*	WARL	*Reserved for custom use*

4.25. MSI configuration table (msi_cfg_tbl)

IOMMU that supports MSI implements a MSI configuration table that is indexed by the vector from icvec to determine a MSI table entry. Each MSI table entry for interrupt vector x has three registers msi_addr_x, msi_data_x, and msi_vec_ctrl_x. These registers are hard wired to 0 if the IOMMU does not support MSI.

Table 14. MSI configuration table structure

bit 63	bit 0	Byte Offset
Entry 0: Message address		+000h
Entry 0: Vector Control	Entry 0: Message Data	+008h
Entry 1: Message address		+010h
Entry 1: Vector Control	Entry 1: Message Data	+018h
…		+020h

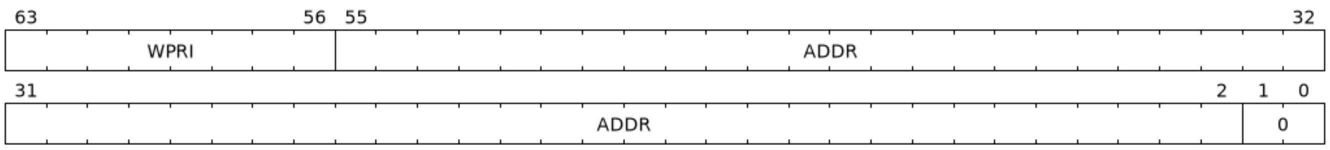


Figure 50. msi_addr_x register fields

Bits	Field	Attribute	Description
1:0	0	RO	Fixed to 0

Bits	Field	Attribute	Description
55:2	ADDR	WARL	Holds the 4-byte aligned MSI address.
63:56	WPRI	WPRI	Reserved for future use.

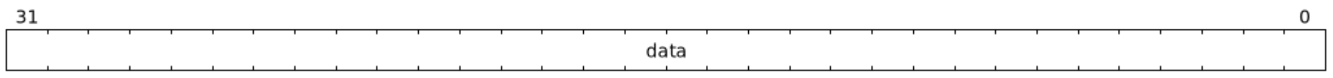


Figure 51. msi_data_x register fields

Bits	Field	Attribute	Description
31:0	data	RW	Holds the 4-byte MSI data

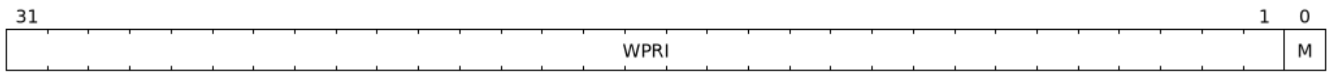


Figure 52. msi_vec_ctrl_x register fields

Bits	Field	Attribute	Description
0	M	RW	When the mask bit M is 1, the corresponding interrupt vector is masked and the IOMMU is prohibited from sending the associated message.
31:1	WPRI	WPRI	Reserved for future use.

Chapter 5. Software guidelines

This section provides guidelines to software developers on correct and expected sequence of using the IOMMU interfaces. The behavior of the IOMMU if these guidelines are not followed is implementation defined.

5.1. Guidelines for initialization

This section provides guidelines to software for the IOMMU initialization sequence:

1. Set up the interrupt related registers
   - If MSI generation is supported, `msi_addr_*`, `msi_data_*` and `msi_vec_ctrl_*` must be set along with `icvec`
   - If wire interrupt is supported, nothing specific is required as the registers above are in this case hardcoded to 0.
2. Set up the Fault queue base address and size (`fqb`) then enable the queue and associated interrupt if desired in `fqcsr`.
3. Set up the Command queue base address and size (`cqb`) then enable the queue and associated interrupt if desired in `cqcsr`.
4. Set up the hardware performance monitor if present and desired.
5. Set the Context table in system memory (Device Directory and Process Directory if supported)
6. Set the translation mode and the base address of the device directory in the `ddtp` register.
   - if 7-bit DeviceID or less is supported, set `ddtp.iommu_mode` to `1LVL`.
   - if 8-bit to 16-bit DeviceID is supported, set `ddtp.iommu_mode` to `2LVL`.
   - if 17-bit to 24-bit DeviceID or less is supported, set `ddtp.iommu_mode` to `3LVL`.

5.2. Guidelines for invalidations

This section provides guidelines to software on the invalidation commands to send to the IOMMU when modifying the IOMMU in-memory data structures used for address translations.

5.2.1. Changing device directory table entry

If software changes a leaf-level DDT entry i.e, a device context (DC), of device with `device_id=D` then the following invalidations must be performed:

- `IODIR.INVAL_DDT` with `DV=1` and `DID=D`
- If `DC.tc.PDTV==1`, `IODIR.INVAL_PDT` with `DV=1`, `PV=0`, and `DID=D`
- If `DC.iohgatp.MODE != Bare`
  - `IOTINVAL.VMA` with `GV=1`, `AV=PSCV=0`, and `GSCID=DC.iohgatp.GSCID`
  - `IOTINVAL.GVMA` with `GV=1`, `AV=0`, and `GSCID=DC.iohgatp.GSCID`
  - If `DC.msiptp.MODE != Bare`, `IOTINVAL.MSI` with `AV=0` and `GV=1`, and

    GSCID=DC.iohgatp.GSCID

- else
  - If `DC.tc.PDTV==1 || DC.tc.PDTV == 0 && DC.fsc.MODE == Bare`
    - `IOTINVAL.VMA` with `GV=AV=PSCV=0`
  - else
    - `IOTINVAL.VMA` with `GV=AV=0` and `PSCV=1`, and `PSCID=DC.ta.PSCID`
  - If `DC.msiptp.MODE != Bare`, `IOTINVAL.MSI` with `AV=GV=0`

If software changes a non-leaf-level DDT entry the following invalidations must be performed: * `IODIR.INVAL_DDT` with `DV=0`

Between change to the DDT entry and when an invalidation command to invalidate the cached entry is processed by the IOMMU, the IOMMU may use the old value or the new value of the entry.

5.2.2. Changing process directory table entry

If software changes a leaf-level PDT entry i.e, a process context (PC), for `device_id=D` and `process_id=P` then the following invalidations must be performed:

- `IODIR.INVAL_PDT` with `DV=1`, `PV=1`, `DID=D` and `PID=P`
- If `DC.iohgatp.MODE != Bare`
  - `IOTINVAL.VMA` with `GV=1`, `AV=0`, `PV=1`, `GSCID=DC.iohgatp.GSCID`, and `PSCID=PC.PSCID`
- else
  - `IOTINVAL.VMA` with `GV=0`, `AV=0`, `PV=1`, and `PSCID=PC.PSCID`

Between change to the PDT entry and when an invalidation command to invalidate the cached entry is processed by the IOMMU, the IOMMU may use the old value or the new value of the entry.

5.2.3. Changing MSI page table entry

If software changes a MSI page-table entry identified by by interrupt file number `I` then following invalidations must be performed:

- If `DC.iohgatp.MODE == Bare`
  - `IOTINVAL.MSI` with `GV=0`, `AV=1`, and `INT_FILE_NUM=I`
- else
  - `IOTINVAL.MSI` with `GV=AV=1`, `INT_FILE_NUM=I` and `GSCID=DC.iohgatp.GSCID`

To invalidate all cache entries from a MSI page table the following invalidations must be performed:

- If `DC.iohgatp.MODE == Bare`
  - `IOTINVAL.MSI` with `GV=0`, `AV=0`
- else
  - `IOTINVAL.MSI` with `GV=1`, `AV=0`, and `GSCID=DC.iohgatp.GSCID`

Between change to the MSI PTE and when an invalidation command to invalidate the cached PTE is processed by the IOMMU, the IOMMU may use the old PTE value or the new PTE value.

5.2.4. Changing G-stage page table entry

If software changes a leaf G-stage page-table entry of a VM where the change affects translation for a guest-PPN `G` then following invalidations must be performed:

- `IOTINVAL.GVMA` with `GV=AV=1`, `GSCID=DC.iohgatp.GSCID`, and `ADDR[63:12]=G`

If software changes a non-leaf G-stage page-table entry of a VM then following invalidations must be performed:

- `IOTINVAL.GVMA` with `GV=1`, `AV=0`, `GSCID=DC.iohgatp.GSCID`

The `DC` has fields that hold a guest-PPN. An implementation may translate such fields to a supervisor-PPN as part of caching the `DC`. If the G-stage page table update affects translation of guest-PPN held in the `DC` then software must invalidate all such cached `DC` using `IODIR.INVAL_DDT` with `DV=1` and `DID` set to the corresponding `device_id`. Alternatively, an `IODIR.INVAL_DDT` with `DV=0` may be used to invalidate all cached `DC`.

Between change to the G-stage PTE and when an invalidation command to invalidate the cached PTE is processed by the IOMMU, the IOMMU may use the old PTE value or the new PTE value.

5.2.5. Changing VS/S-stage page table entry

When `DC.iohgatp.MODE == Bare`, a `DC` may be configured with a S-stage page table (when `DC.tc.PDTV=0`) or a directory of S-stage page tables selected using `process_id` from a process-directory-table (when `DC.tc.PDTV=1`).

When `DC.iohgatp.MODE != Bare`, a `DC` may be configured with a VS-stage page table (when `DC.tc.PDTV=0`) or a directory of VS-stage page tables selected using `process_id` from a process-directory-table (when `DC.tc.PDTV=1`).

When a change is made to a S-stage page table then software must perform invalidations using `IOTINVAL.VMA` with `GV=0` and `AV` and `PSCV` operands appropriate for the modification as specified in Table 5.

When a change is made to a VS-stage page table then software must perform invalidations using `IOTINVAL.VMA` with `GV=1`, `GSCID=DC.iohgatp.GSCID` and `AV` and `PSCV` operands appropriate for the modification as specified in Table 5.

Between change to the S/VS-stage PTE and when an invalidation command to invalidate the cached PTE is processed by the IOMMU, the IOMMU may use the old PTE value or the new PTE value.

5.2.6. Accessed (A)/Dirty (D) bit updates and page promotions

When IOMMU supports hardware managed A and D bit updates, if software clears the A and/or D bit in the S/VS-stage and/or G-stage PTEs then software must invalidate corresponding PTE entries that may be cached by the IOMMU. If such invalidations are not performed, then the IOMMU may not set these bits when processing subsequent transactions that use such entries.

When software upgrades a page in S/VS-stage PTE and/or a G-stage PTE to a superpage without first clearing the original non-leaf PTEs valid bit and invalidating cached translations in the IOMMU then it is possible for the IOMMU to cache multiple entries that match a single address. The IOMMU may use either the old non-leaf PTE or the new non-leaf PTE but the behavior is otherwise well defined.

When promoting and/or demoting page sizes, software must ensure that the original and new PTEs have identical permission and memory type attributes and the physical address that is determined as a result of translation using either the original or the new PTE is otherwise identical for any given input. The only PTE update supported by the IOMMU without first clearing the V bit in the original PTE and executing a appropriate IOTINVAL command is to do a page size promotion or demotion. The behavior of the IOMMU if other attributes are changed in this fashion is implementation defined.

5.2.7. Device Address Translation Cache invalidations

When VS/S-stage and/or G-stage page tables are modified, invalidations may be needed to the DevATC in the devices that may have cached translations from the modified page tables. Invalidation of such page tables requires generating ATS invalidations using `ATS.INVAL` command. Software must specify the PAYLOAD following the rules defined in PCIe ATS specifications.

If software generates ATS invalidate requests at a rate that exceeds the average DevATC service rate then flow control mechanisms may be triggered by the device to throttle the rate and a side effect of this is congestion spreading to other channels and links and could lead to performance degradations. An ATS capable device publishes the maximum number of invalidations it can buffer before causing backpressure through the Queue Depth field of the ATS capability structure. When the device is virtualized using PCIe SR-IOV, this queue depth is shared among all the VFs of the device. Software must limit the number of outstanding ATS invalidations queued to the device adverstized limit.

The `RID` field is used to specify the routing ID of the ATS invalidation request message destination. A PASID specific invalidation may be performed by setting `PV=1` and specifying the PASID in `PID`. When the IOMMU supports multiple segments then the `RID` must be qualified by the destination segment number by setting `DSV=1` with the segment number provided in `DSEG`.

When ATS protocol is enabled for a device, the IOMMU may still cache translations in its IOATC in addition to providing translations to the DevATC. Software must not skip IOMMU translation cache invalidations even when ATS is enabled in the device context of the device. Since a translation request from the DevATC may be satisfied by the IOMMU from the IOATC, to ensure correct operation software must first invalidate the IOATC before sending invalidations to the DevATC.

5.2.8. Caching invalid entries

This specification does not the caching of S/VS/G-stage PTEs whose V (valid) bit is clear, DDT entries whose V bit is clear, PDT entries whose V bit is clear, or MSI PTEs whose V bit is clear. Software need not perform invalidations when changing the V bit in these entries from 0 to 1.

5.2.9. Reconfiguring PMAs

Where platforms support dynamic reconfiguration of PMAs, a machine-mode driver is usually provided that can correctly configure the platform. In some platforms that might involve platform-specific operations and if the IOMMU must participate in these operations then platform-specific operations in the IOMMU are used by the machine-mode driver to perform such reconfiguration.

5.2.10. Guidelines for handling interrupts from IOMMU

Placeholder

Chapter 6. Hardware guidelines

This section provides guidelines to the system/hardware integrator of the IOMMU in the platform.

6.1. Integrating an IOMMU as a PCIe device

The IOMMU may be constructed as a PCIe device itself and be discoverable as a dedicated PCIe function with PCIe defined Base Class 08h, Sub-Class 06h, and Programming Interface 00h.

Such IOMMU must map the IOMMU registers defined in this specification as PCIe BAR mapped registers.

The IOMMU may support MSI or MSI-X or both. When MSI-X is supported, the MSI-X capability block must point to the msi_tbl in BAR mapped registers such that system software can configure MSI address and data pairs for each message supported by the IOMMU. The MSI-X PBA may be located in the same BAR or another BAR of the IOMMU. The IOMMU is recommended to support MSI-X capability.

6.2. Debug

Placeholder

6.3. RAS

Placeholder

Index

Bibliography