RISC-V IOMMU Specification Document

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 separate 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 up to 8-bit segment number when the IOMMU supports multiple segments. This specification refers to such unique device identifier as `device_id` and supports up to 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 up to 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 duplication 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 Translation 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 resources 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)
4. Dynamic redirection of interrupts
5. Support shared virtual addressing

In the absence of an IOMMU, a device driver must program devices with Physical Addresses, which implies that DMA from a device could be used to access any memory, such as privileged memory, and cause malicious or unintended corruptions. This may be caused by hardware bugs, device driver bugs, or by malicious software.

The IOMMU offers a mechanism for the OS to defend against such unintended corruptions by limiting the memory that can be accessed by devices using DMA. Indeed, the Operating System configures the IOMMU to use the S-stage page table to translate IOVA to SPA and thereby limit the addresses that may be accessed.

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 themselves.

Legacy 32-bit devices cannot access the memory above 4 GiB. The integration of the IOMMU, through its address remapping capability, offers a simple mechanism for the DMA to directly access any address in the system (with appropriate access permission). Without an IOMMU, the OS must resort to copying data through buffers (also known as bounce buffers) allocated in memory below 4 GiB 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 a fragmented physical addresses.

The IOMMU can be used to 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.

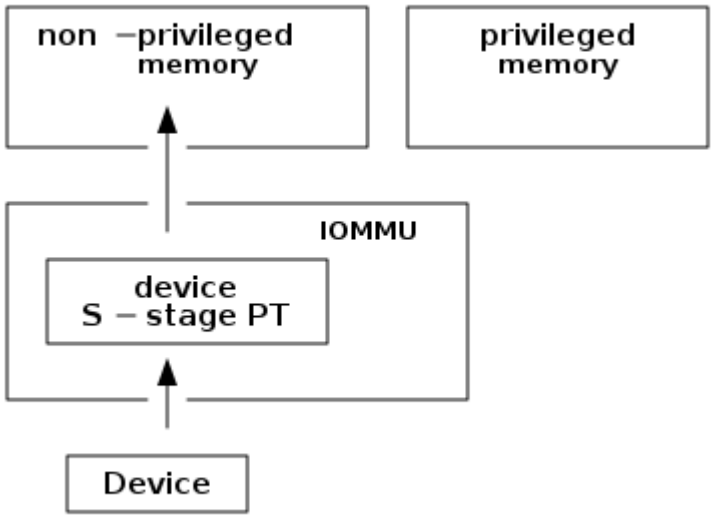


Figure 1. Device isolation in non-virtualized OS

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.

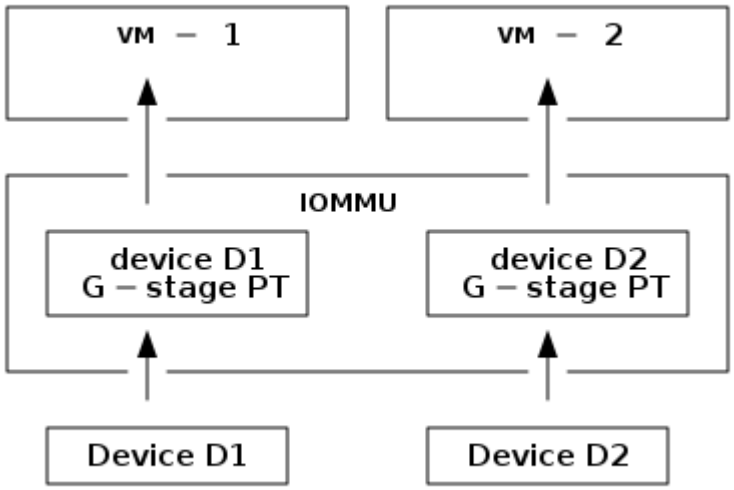


Figure 2. DMA translation to enable direct device assignment

1.2.3. Guest OS

The presence of an IOMMU allows each device to be individually configured for a different software master, usually a guest OS or the main (host) OS.

On implementations of the IOMMU that support two stages of translation (VS-stage and G-stage), the G-stage translation (or second stage of translation) is intended to virtualize device DMA to the Guest OS physical address space. Devices can be assigned to Guest OS which can directly program the device to do 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. The use of the G-stage page tables limits the physical memory accessible by a device controlled by the guest OS to the memory allocated to its virtual machine.

The Hypervisor may then provide a virtual IOMMU facility, through hardware emulation or by enlightening the Guest OS to use a software interface with the Hypervisor (also known as para-virtualization). The Guest OS may then use the facilities provided by the virtual IOMMU to avail the same benefits as those discussed for a Non-virtualized OS. The Guest OS employs a page table, really a VS-stage page table, to perform similar configurations for the device a Non-virtualized OS.

With two-stage address translations enabled, 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 SPA using the G-stage page tables managed by the hypervisor.

The following diagram illustrates the concept. The IOMMU is configured to perform two-stage address translation translation (VS-stage and G-stage) for the device (D1), is configured to to perform G-stage only translation for another device (D2). The host OS or hypervisor may also retain a device, such as D3, for its own use and for configure the IOMMU to perform a single-stage (S-stage) translation.

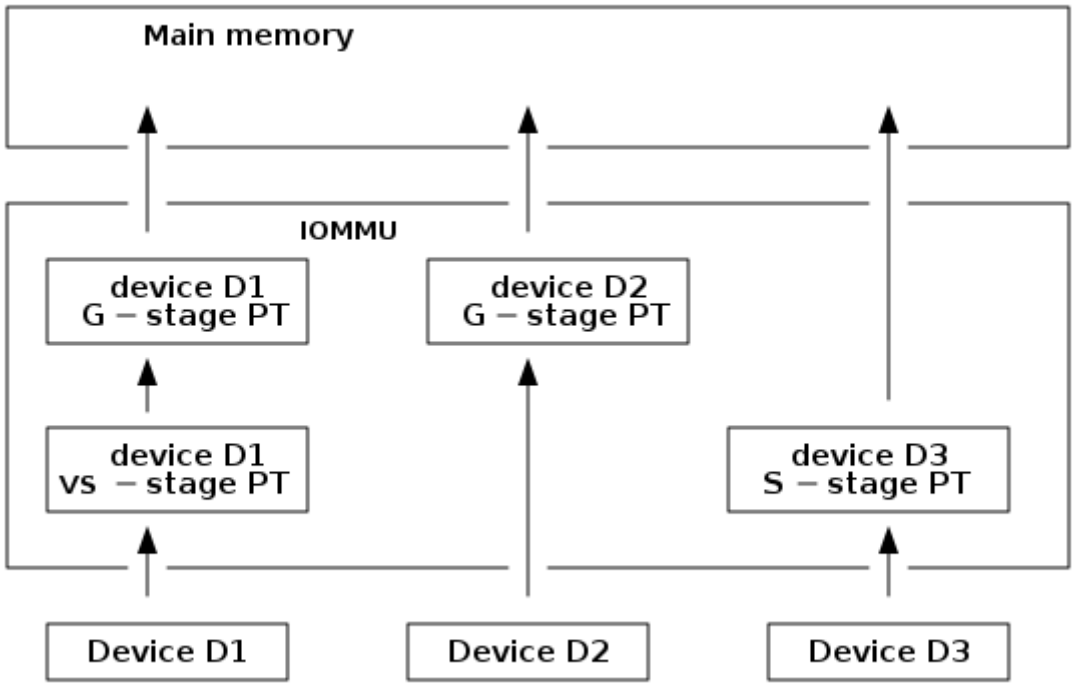


Figure 3. Address translation in IOMMU for Guest OS

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 and several IO devices. This SOC also incorporates two instances of the IOMMU. The device may be directly connected to the IO Bridge and the system interconnect or may be connected through a Root Port when a I/O protocol transaction to system interconnect transaction translation is required. In case of PCIe, for example, the Root Port is a PCIe port that maps a portion of a hierarchy through an associated virtual PCI-PCI bridge and maps the PCIe I/O protocol transactions to the system interconnect transactions.

The first instance, IOMMU 0 (associated with the IO Bridge 0), interfaces a Root Port to the system fabric. One or more endpoint devices are interface to the SoC through this Root Port. In case of PCIe, the Root Port incorporates an ATS interface to the IOMMU that is used to support the PCIe ATS protocol by the IOMMU. The example, shows an endpoint device with a device side ATC (devATC) that holds translations obtained by the device from IOMMU 0 using the PCIe ATS protocol.

When such I/O protocol to system fabric protocol translation using a Root Port is not required, the devices may interface directly with the system fabric. The second instance, IOMMU 1 (associated with the IO Bridge 1), illustrates interfacing devices (IO Devices A and B) to the system fabric without the use of a Root Port.

The IO Bridge is placed between the device(s) and the fabric/interconnect to process device originated DMA transactions. IO Devices may perform DMA transactions using IO Virtual Addresses (VA, GVA or GPA). The IO Bridge invokes the associated IOMMU to translate the IOVA to a System Physical

Addresses (SPA).

The IOMMU is not invoked for outbound transactions.

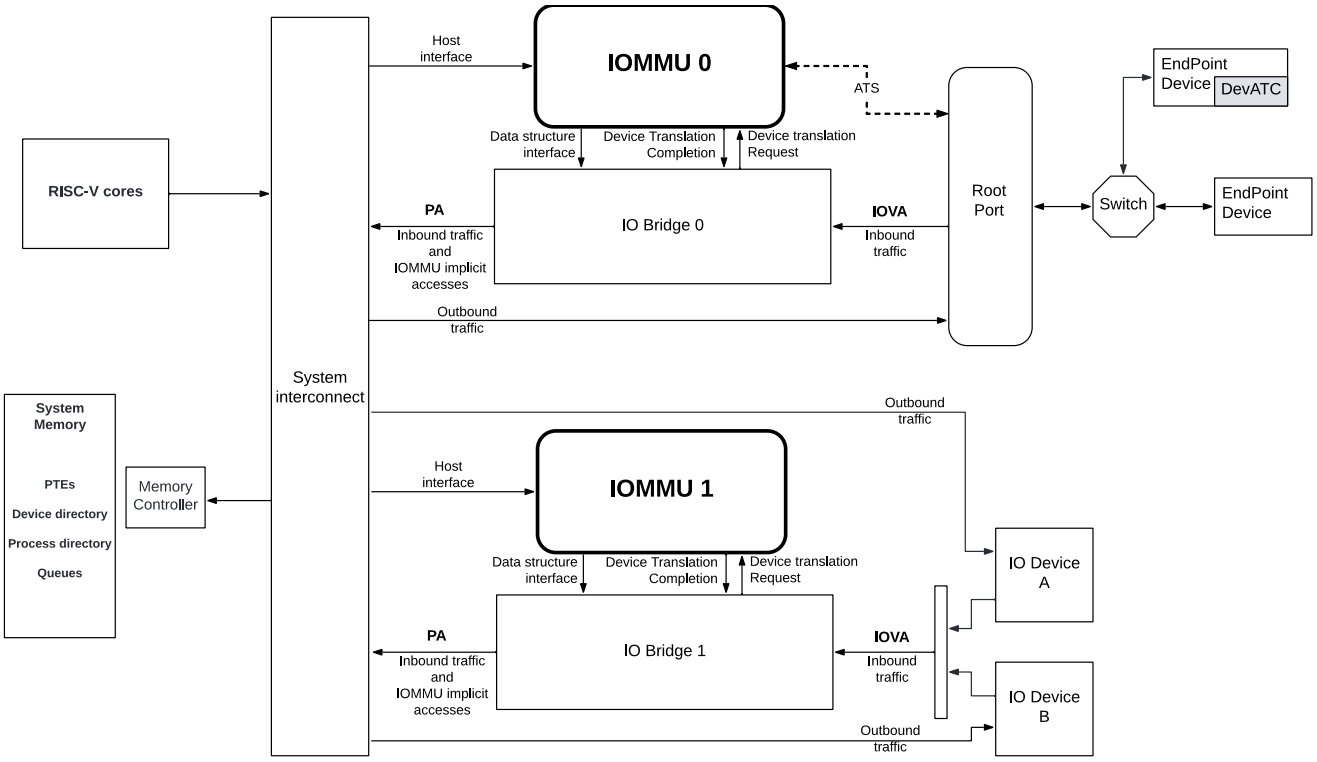


Figure 4. Example of IOMMUs integration in SoC.

The IOMMU is invoked by the IO bridge for address translation and protection for inbound transactions. The data associated with the inbound transactions is not processed by the IOMMU. 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 MMIO registers and perform global configuration and/or maintenance operations.
- Device Translation Request interface: it is a slave interface, which receives the translation requests from the IO Bridge. On this interface the IO Bridge provides information about the request such as:
  a. The hardware identities associated with transaction - the `device_id` and if applicable the `process_id`. The IOMMU uses the hardware identities to retrieve the context information to perform the requested address translations.
  b. The IOVA and the type of the transaction (Translated or Untranslated).
  c. Whether the request is for a read, write, execute, or an atomic operation.
  d. The privilege mode associated with the request when applicable.
  e. The number of bytes accessed by the request.
  f. The IO bridge may also provide some additional opaque information (e.g. tags) that are not interpreted by the IOMMU but returned along with the response from the IOMMU to the IO bridge. As the IOMMU is allowed to complete translation requests out of order, such information may be used by the IO Bridge to correlate completions to previous requests.
- The Data Structure interface: is used by the IOMMU for implicit access to memory. It is a master interface to the IO Bridge and is used 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 G-state and/or S/VS page table entries to translate the IOVA

c. The in-memory queues (command-queue, fault-queue, and page-request-queue) used to interface with software.

- Device Translation Completion interface: it is a master interface which provides the completion response from the IOMMU for previously requested address translations. The completion interface may provide information such as:

  a. Status of the request. Indicates if request completed successfully or a fault occurred.

  b. If the request was completed successfully; the System Physical Address (SPA).

  c. Opaque information (e.g. tags), if applicable, associated with the request.

  d. The page-based memory types (PBMT), if Svpbmt is supported, obtained from the IOMMU address translation page tables. When two-stage address translation is performed the IOMMU provides the page-based memory type as resolved between the G-stage and VS-stage page table entries.

- ATS interface: The ATS interface, if the optional PCIe ATS capability is supported by the IOMMU, is used to communicate with ATS capable endpoints through the PCIe Root Port. This interface is used to:

  a. To receive ATS translation request from the endpoints and to return the completions to the endpoints. The Root Port may provide an indication if the endpoint originating the request is a CXL type 1 or type 2 device.

  b. To send ATS "Invalidation Request" messages to the endpoints and to receive the "Invalidation Completion" messages from the endpoints.

  c. To receive "Page Request" and "Stop Marker" messages from the endpoints and to send "Page Request Group Response" messages to the endpoints.

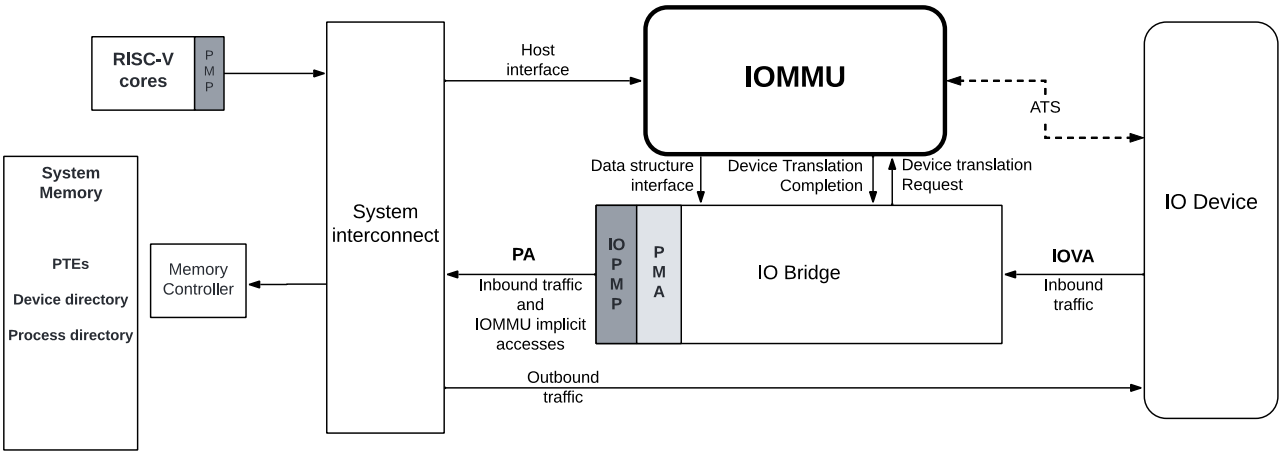


Figure 5. IOMMU interfaces.

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 PMP checkers is a platform choice.

PMA and PMP 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 may use the provided PBMT to override the PMA(s) for the associated memory pages.

The PMP may use the hardware ID of the bus master to determine physical memory access privileges. As the IOMMU itself is a bus master for its implicit accesses, the IOMMU hardware ID may be used by the PMP 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.
- Up to 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) that is traversed using the device_id is used to locate the DC.

The address space used by a device may require single-stage or two-stage address translation and protection. Two-stage address translation may required when the control of the device is passed through to a Guest OS. Single-stage address translation using a S-stage page table may be used when the control of the device is retained by the hypervisor or Host OS itself.

When two-stage address translation is used the DC holds the PPN of the root G-stage page table; a guest-soft-context-ID (GSCID), which facilitates fences on a per-virtual-machine basis; and the G-stage address translation scheme.

Some devices supports multiple contexts where each context may be associated with a different process and thus a different virtual address space. The context in such devices may be configured with a process_id that identifies the address space. When making a memory access, such devices signal the process_id along with the device_id to identify the accessed address space. An example of such a device is a GPU that supports multiple process contexts, where each context is associated with a different user process, such that the GPU may access memory using the virtual address provided by the user process itself. To support selecting an address space associated with the process_id, the DC holds the PPN of the root Process Directory Table (PDT), a radix-tree data structure, indexed using fields of the process_id to locate a data structure called the Process Context (PC).

When a PDT is active, the controls for S-stage or VS-stage address translation are held in the (PC).

When a PDT is not active, the controls for S-stage or VS-stage address translation are be held in the DC itself.

The S/VS-stage address translation controls include the PPN of the root S/VS-stage page table; a process-soft-context-ID (PSCID), which facilitates fences on a per-address-space basis; and the S/VS-stage address translation scheme.

To handle MSIs from a device controlled by a guest OS, an IOMMU must be able to redirect those MSIs 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 I/O MMU applies to other memory writes. However, the Advanced Interrupt Architecture defines IOMMU support to 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 Advanced Interrupt Architecture adds to device contexts an MSI address mask and address pattern, used together to recognize certain memory writes from the device as being MSIs; and the real physical address of an MSI page table for controlling the translation and/or conversion of MSIs from the device.

The DC further holds controls for the type of transactions that a device is allowed to generate. One example of such a control is whether the device is allowed to use the PCIe defined Address Translation Service (ATS).

Two formats of the device-context structure are supported:

- **Base Format** - is 32-bytes in size used when the special treatment of MSI as defined by the Advanced Interrupt Architecture is not supported by the IOMMU.
- **Extended Format** - is 64-bytes in size and extends the base format `DC` with additional fields to process MSIs as defined by the RISC-V Advanced Interrupt Architecture (AIA).

The DDT used to locate the `DC` 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:

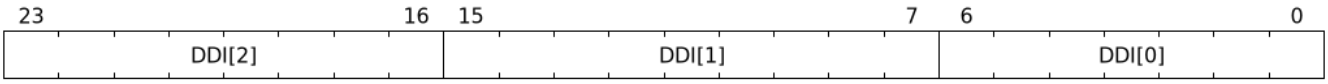


Figure 6. Base format `device_id` partitioning

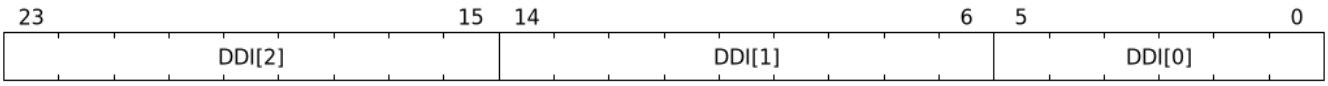


Figure 7. Extended format `device_id` partitioning

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 for that device. The partitioning of the `process_id` to obtain the process directory indices (PDI) to traverse the PDT radix-tree table are as follows:

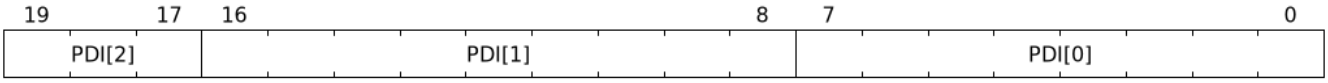


Figure 8. `process_id` partitioning for PDT radix-tree traversal

The `process_id` partitioning is designed to required a maximum of 4 KiB, a page, of memory for each process directory table. The root of the table when using a 20-bit wide `process_id` is not fully populated. The option of making the root table occupy 32 KiB was considered but not adopted as these tables are allocated at run time and contiguous memory allocation larger than a page may stress the Guest and hypervisor memory allocators.

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 but is not prohibited by this specification.

2.1. Device-Directory-Table (DDT)

DDT is up to 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 PPN of the root device-directory-table is held in a memory-mapped register called the device-directory-table pointer (`ddtp`).

Each valid non-leaf (`NL`) entry is 8-bytes in size and holds the PPN of the next device-directory-table.

A valid leaf device-directory-table entry holds the device-context (`DC`).

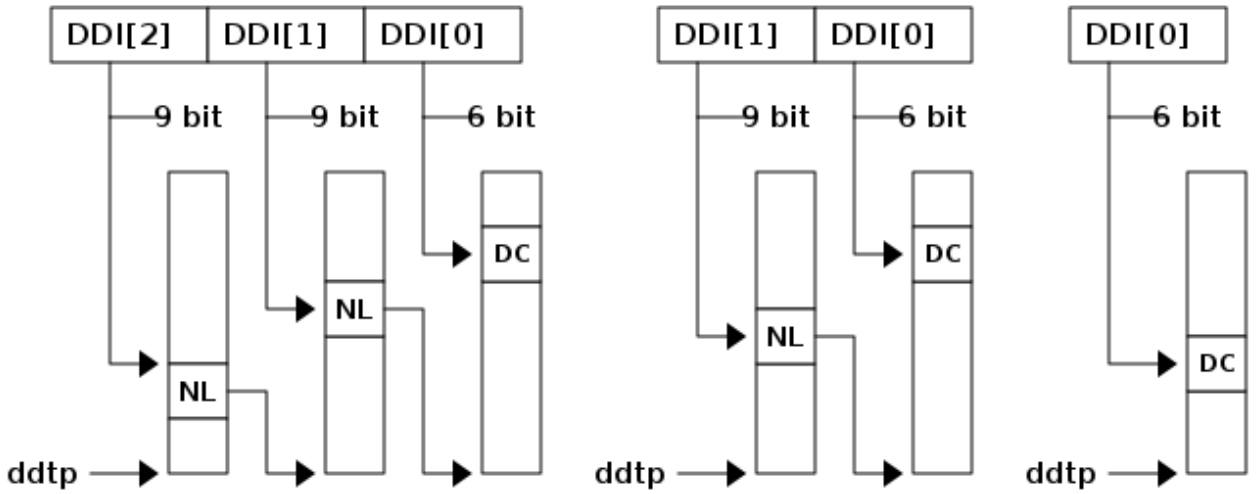


Figure 9. Three, two and single-level device directory with extended format `DC`

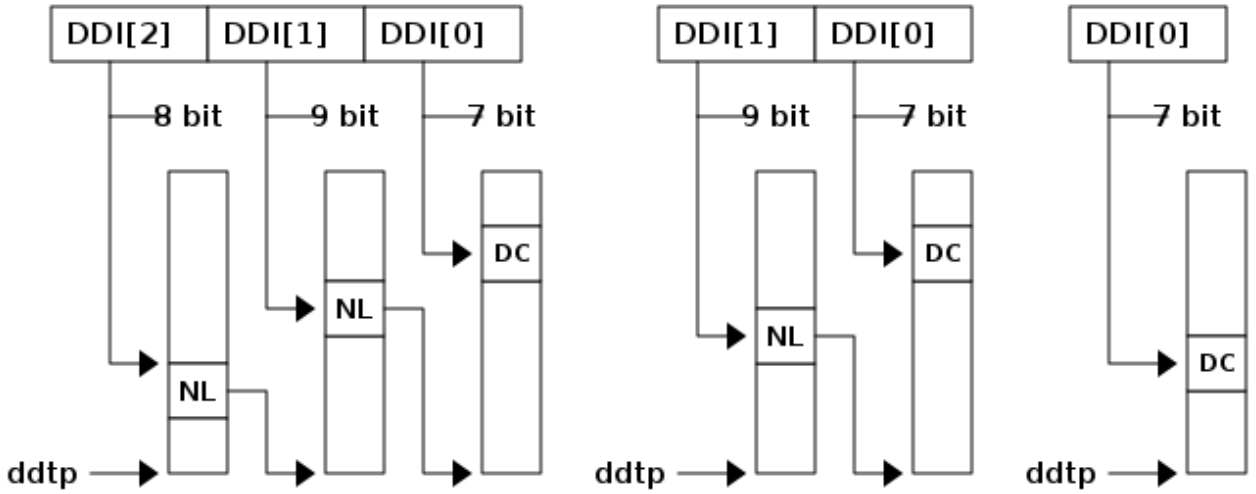


Figure 10. Three, two and single-level device directory with base format `DC`

2.1.1. Non-leaf DDT entry

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

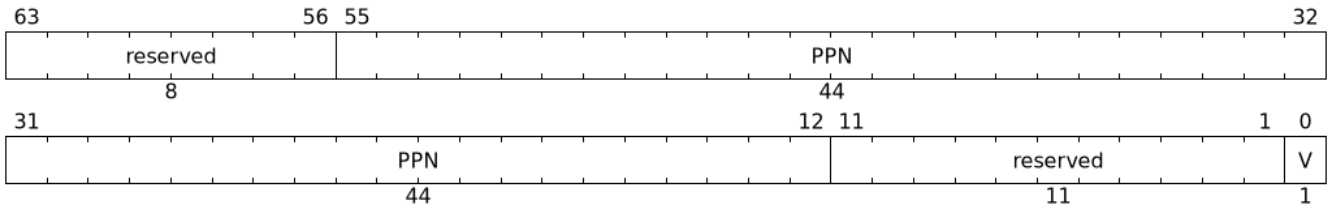


Figure 11. Non-leaf device-directory-table entry

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.

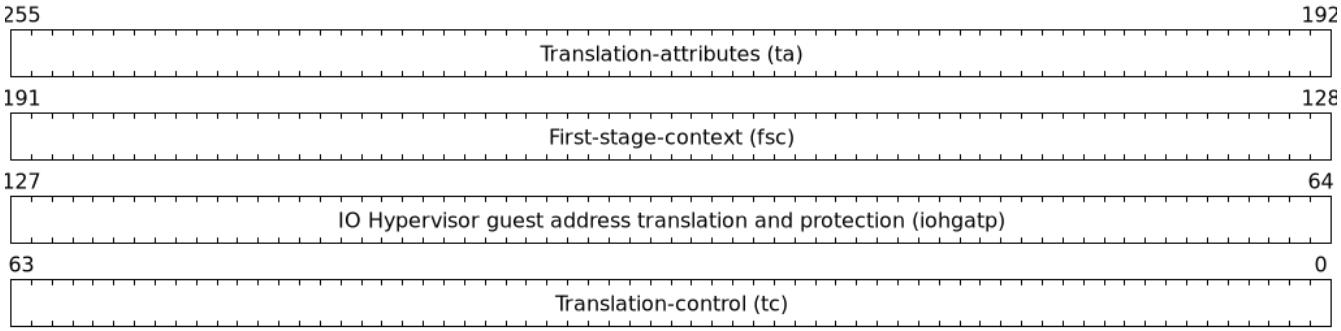


Figure 12. Base-format device-context

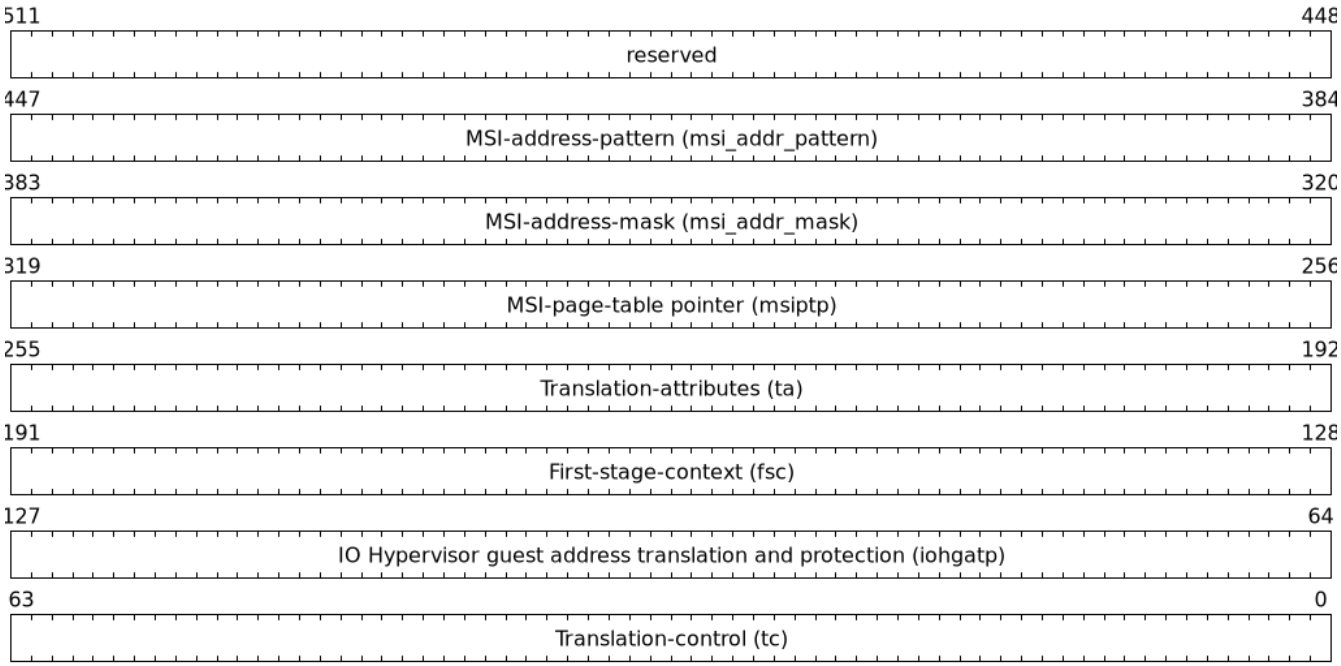


Figure 13. Extended-format device-context

2.1.3. Device-context fields

Translation control (tc)

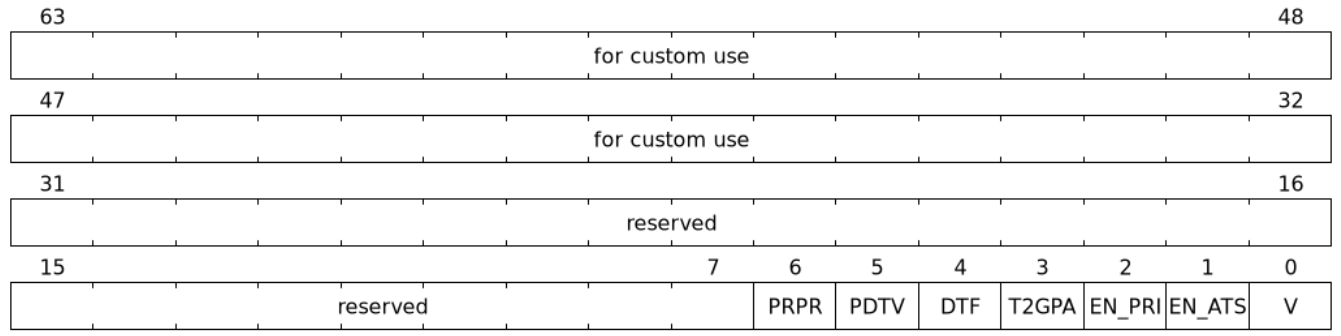


Figure 14. Translation control (tc) field

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

If the EN_ATS bit is 1 and the T2GPA bit is set to 1 the IOMMU returns a GPA , instead of a SPA, as the translation of an IOVA in response to a PCIe ATS Translation Request from the device. In this mode of operations, the ATC in the device caches a GPA as a translation for an IOVA and uses the GPA as the address in subsequent translated memory access transactions. Usually translated requests use a SPA and need no further translation to be performed by the IOMMU. However when T2GPA is 1, translated requests from a device use a GPA and are translated by the IOMMU using the G-stage page table to a SPA. The T2GPA control enables a hypervisor to contain DMA from a device, even if the device misuses the ATS capability and attempts to access memory that is not associated with the VM.

When T2GPA is enabled, the addresses provided to the device in response to a PCIe ATS Translation Request cannot be directly routed by the I/O fabric (e.g. PCI switches) that connect the device to other peer devices and to host. Such addresses are also cannot be routed within the device when 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/executable (Measurement) and hardware identities (Authentication) to establish such a trust relationship.

If EN_PRI bit is 0, then PCIe "Page Request" messages from the device are invalid requests. A "Page Request" message received from a device is responded to with a "Page Request Group Response" message. Normally, a software handler generates this response message. However, under some conditions the IOMMU itself may generate a response. For IOMMU generated "Page Request Group Response" messages the PRG-response-PASID-required (PRPR) bit when set to 1 indicates that the IOMMU response message should include a PASID prefix if the associated "Page Request" had a PASID prefix.

Functions that support PASID and have the "PRG Response PASID Required" capability bit set to 1, expect that "Page Request Group Response" messages will contain a PASID if the associated "Page Request" message had a PASID. If the capability bit is 0, the function does not expect PASID on any "Page Request Group Response" message and the behavior of the function if it receives the response with a PASID prefix is undefined. The `PRPR` bit should be configured with the value held in the "PRG Response PASID Required" capability bit.

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. The faults that are not reported when `DTF` is 1 are listed in Table 8.

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.

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

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. For PCIe, for example, PASID capable devices that have the PASID capability enabled, signal the `process_id` in the PASID TLP prefix of the TLP.

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.

The G-stage page table format and `MODE` encoding 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 or a subset thereof.

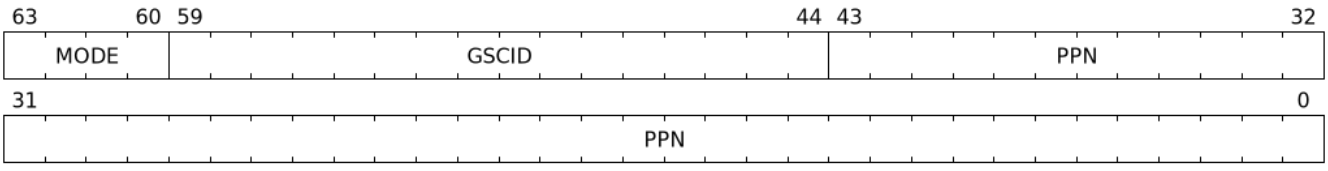


Figure 15. IO hypervisor guest address translation and protection (`iohgatp`) field

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 provide the controls for S-stage page table or VS-stage address translation and protection respectively.

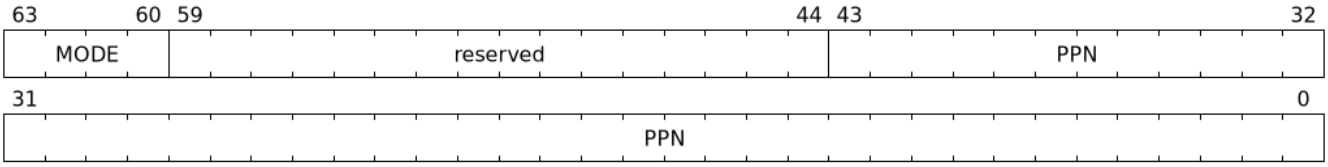


Figure 16. IO (Virtual)Supervisor addr. translation and prot. (iovsatp/iosatp) field (when PDTV is 0)

The encoding of the iosatp/iovsatp MODE field are as the same as the encoding 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 S/VS-stage page table and associated PSCID for virtual address translation and protection.

The pdtp field holds the PPN of the root PDT and the MODE field that determines the number of levels of the PDT.

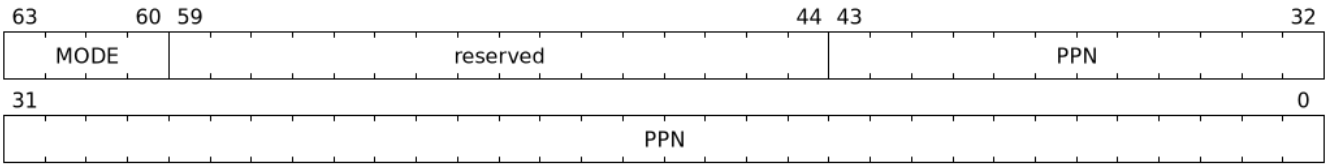


Figure 17. Process-directory table pointer (pdtp) field (when PDTV is 1)

When two-stage address translation is active (iohgatp.MODE != Bare), the PPN field holds a guest PPN. The GPA of the root PDT is then converted by guest physical address translation, as controlled by the iohgatp, into a supervisor physical address. Translating addresses of root PDT root through G-stage page tables, allows the PDT to be held in memory allocated by the guest OS and allows 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 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 with 256 entries.The bits 19:8 of process_id must be 0.
3-15	—	Reserved

Translation attributes (ta)

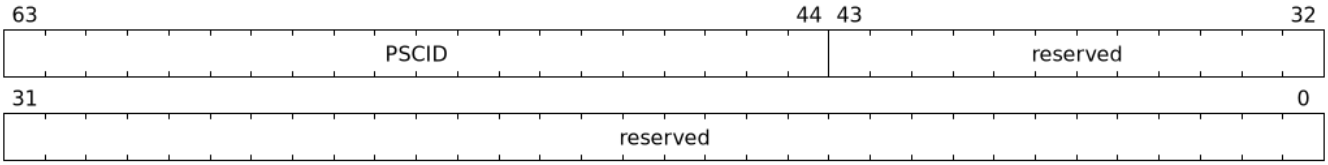


Figure 18. Translation attributes (ta) field

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 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.

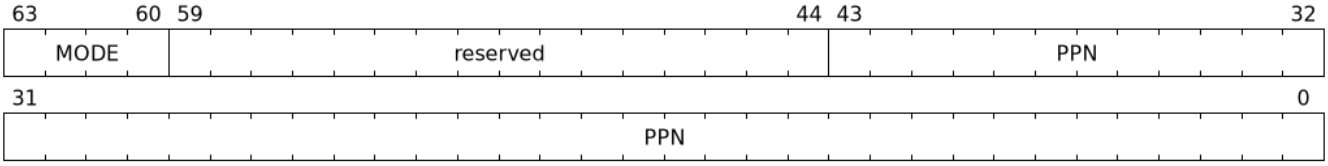


Figure 19. MSI page table pointer (msiptp) field

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.

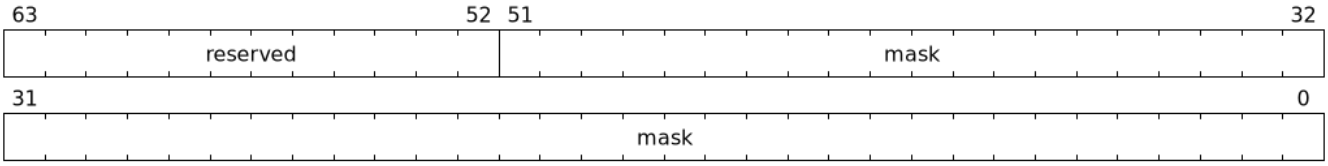


Figure 20. MSI address mask (msi_addr_mask) field

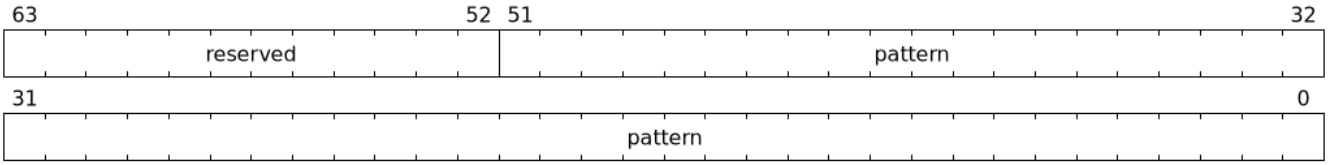


Figure 21. MSI address pattern (`msi_addr_pattern`) field

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. The leaf process-directory-table entry holds the process-context (`PC`).

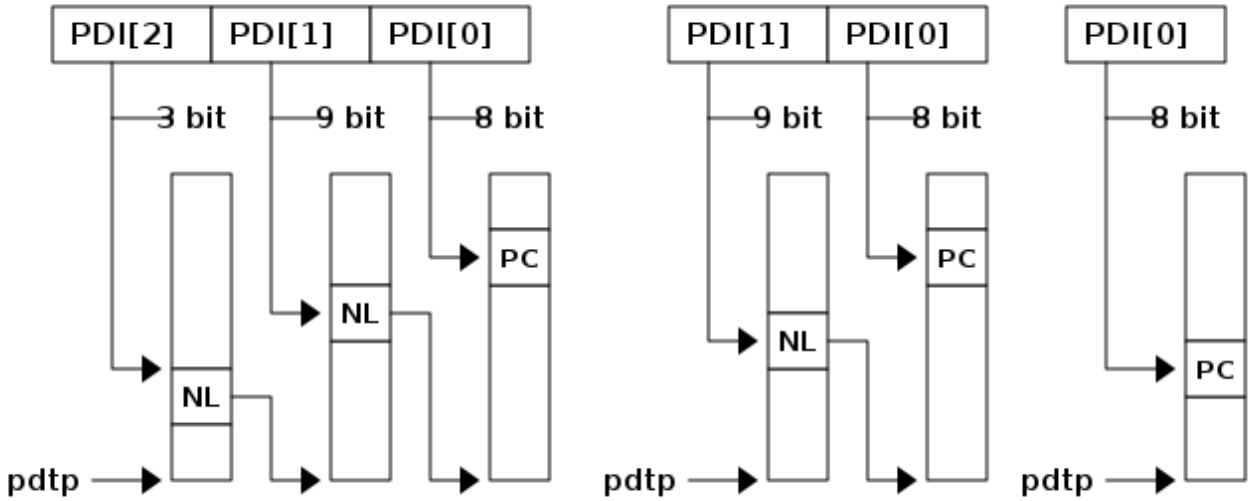


Figure 22. Three, two and single-level process directory

2.2.1. Non-leaf PDT entry

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

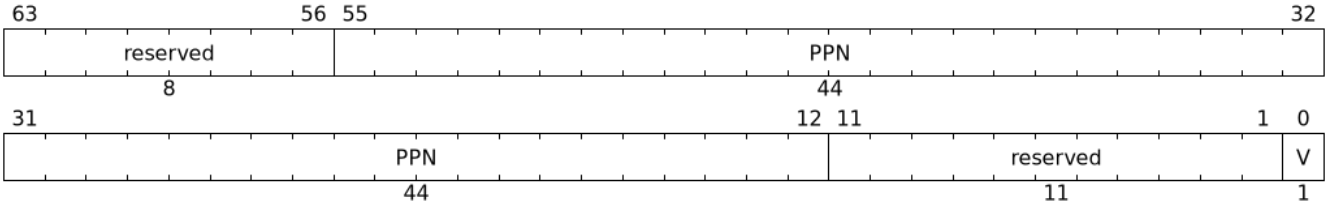


Figure 23. Non-leaf process-directory-table entry

2.2.2. Leaf PDT entry

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

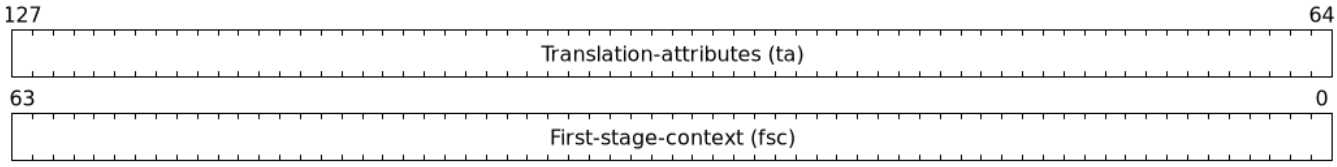


Figure 24. Process-context

2.2.3. Process-context fields

Translation attributes (ta)

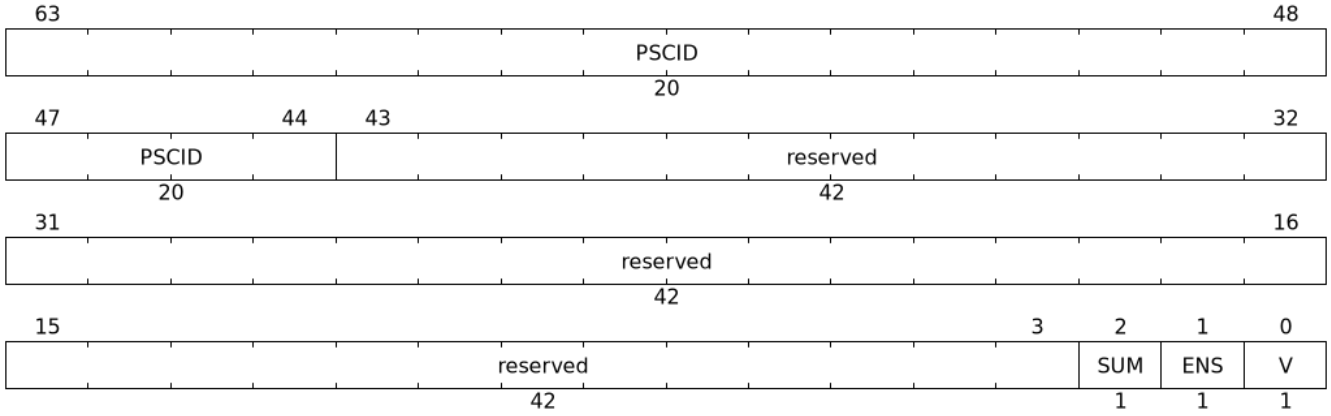


Figure 25. Translation attributes (ta) field

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 an 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 is 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 provide the controls for S-stage page or VS-stage address translation and protection respectively.

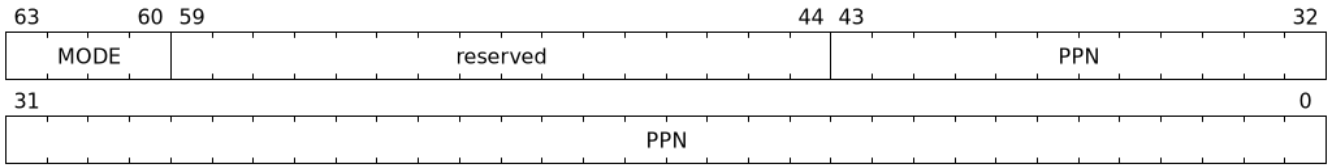


Figure 26. IO (Virtual)Supervisor addr. translation and prot. (iovsatp/iosatp) field (when PDTV is 1)

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 encoding are as defined for the MODE field in the satp/vsatp CSR.

The software assigned process soft-context ID (PSCID) is used as the address space ID for 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 root of a VS-stage page table. Addresses of the VS-stage page table entries are then converted by guest physical address translation process, 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 (DC) as specified in Section 2.3.1.
2. If DC.tc.pdtv is set to 0, the device_id does not support a process_id; go to step 4 with the following page table information:
   - If a G-stage page table is not active in the device-context (DC.iohgatp.mode is Bare) then S-stage page-table is defined in the first-stage context (DC.fsc) as it holds the iosatp. DC.ta holds the PSCID associated with the S-stage translation.
   - If a G-stage page table is active in the device-context (DC.iohgatp.mode is not Bare), then VS-stage page table is defined in the first-stage context (DC.fsc) as it holds the iovsatp and the G-stage page table is defined in DC.iohgatp, which also holds the GSCID. DC.ta holds the PSCID associated with the VS-stage translation.
3. If DC.tc.pdtv is set to 1, the device_id supports a process_id and DC.fsc holds the pdtp;locate the process-context (PC) as specified in Section 2.3.2.
   - If a G-stage page table is not active in the device-context (DC.iohgatp.mode is Bare) then S-stage page-table is defined in the first-stage context (PC.fsc) as it holds the iosatp. PC.ta holds the PSCID associated with the S-stage translation.
   - If a G-stage page table is active in the device-context (DC.iohgatp.mode is not Bare), then VS-stage page table is defined in the first-stage context (PC.fsc) as it holds the iovsatp and the G-stage page table is defined in DC.iohgatp, which also holds the GSCID. PC.ta holds the PSCID associated with the VS-stage translation.
4. If the transaction is a write and the IOMMU support MSI address translation using MSI flat 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.
5. If a G-stage page table is not active in the device-context then use the single stage address translation process specified in Section 4.3.2 of the RISC-V privileged specification. If a fault is detecting by the single stage address translation process then stop and report the fault.
6. 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 RISC-V privileged specification. If a fault is detecting by the single stage address translation process then stop and report the fault.

When the translation process reports a fault, and the request is a Untranslated request, a Translated request, or a message the IOMMU requests the IO bridge to abort the transaction. Guidelines for handling faulting transactions in the IO bridge are provided in Section 6.3. The fault may be reported using the fault/event reporting mechanism and fault record formats specified in Section 3.2.

If the fault was detected by a PCIe ATS Translation Request then the IOMMU may provide a PCIe protocol defined response instead of reporting fault to software or causing an abort. The handling of faulting PCIe ATS Translation Requests is specified in Section 2.5.

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. If `ddtp.iommu_mode == Bare` and 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.
   b. Transaction type is a PCIe "Page Request" Message.
   c. Transaction has a valid `process_id`
   d. Transaction type is not supported by the IOMMU in `Bare` mode.
3. 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.
4. If the `device_id` is wider than supported by `ddtp.iommu_mode`, then stop and report "Transaction type disallowed" (cause = 260).
5. If `i == 0` go to step 11.
6. Let `ddte` be value of eight bytes at address `a + DDI[i] x 8`. If accessing `ddte` violates a PMA or PMP check, then stop and report "DDT entry load access fault" (cause = 257).
7. If `ddte` access detects a data corruption (a.k.a. poisoned data), then stop and report "DDT data corruption" (cause = 268).
8. If `ddte.V == 0`, stop and report "DDT entry not valid" (cause = 258).
9. If if any bits or encoding that are reserved for future standard use are set within `ddte`, stop and report "DDT entry misconfigured" (cause = 259).
10. Let `i = i - 1` and let `a = ddte.PPN x` 2^{12}. Go to step 4.
11. 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, then stop and report "DDT entry load access fault" (cause = 257). If `dc` access detects a data corruption (a.k.a. poisoned data), then stop and report "DDT data corruption" (cause = 268).
12. If `dc.tc.V == 0`, stop and report "DDT entry not valid" (cause = 258).
13. If any bits or encoding that are reserved for future standard use are set within `dc`, stop and report "DDT entry misconfigured" (cause = 259).
14. The device-context has been successfully located and may be cached.
15. 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 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. Transaction type is not supported by the IOMMU.

2.3.2. Process to locate the Process-context

The device-context provides the PDT root page PPN (`pdtp.ppn`). When `DC.iohgatp.mode` is not bare, `pdtp.PPN` as well as `pdte.PPN` are Guest Physical Addresses (GPA) which must be translated into Supervisor Physical Addresses (SPA) using the G-stage page table pointed by `DC.iohgatp`.

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 `DC.iohgatp.mode != Bare`, then `a` is a GPA. Invoke the process to translate `a` to a SPA. If faults occur during G-stage address translation of `a` then stop and the fault detected by the G-stage address translation process. The translated `a` is used in subsequent steps.
3. If `i == 0` go to step 9.
4. Let `pdte` be value of eight bytes at address `a + PDI[i] x 8`. If accessing `pdte` violates a PMA or PMP check, then stop and report "PDT entry load access fault" (cause = 265).
5. If `pdte` access detects a data corruption (a.k.a. poisoned data), then stop and report "PDT data corruption" (cause = 269).
6. If `pdte.V == 0`, stop and report "PDT entry not valid" (cause = 266).
7. If if any bits or encoding that are reserved for future standard use are set within `pdte`, stop and report "PDT entry misconfigured" (cause = 267).
8. Let `i = i - 1` and let `a = pdte.PPN x 2^12`. Go to step 2.
9. Let `pc` be value of 16-bytes at address `a + PDI[0] x 16`. If accessing `pc` violates a PMA or PMP check, then stop and report "PDT entry load access fault" (cause = 265).If `pc` access detects a data corruption (a.k.a. poisoned data), then stop and report "PDT data corruption" (cause = 269).
10. If `pc.ta.V == 0`, stop and report "PDT entry not valid" (cause = 266).
11. If any bits or encoding that are reserved for future standard use are set within `pc`, stop and report "PDT entry misconfigured" (cause = 267).
12. 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.
13. The Process-context has been successfully located.

2.3.3. Process to translate addresses of MSIs

When MSI address translation using MSI flat 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 the write has a valid `process_id` (e.g., PASID prefix present), then the write is not determined to be a MSI write. Stop this process and instead use the regular translation data structures to do the address translation.
5. 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.
6. 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`.
7. Let `a` be $(dc.msiptp.PPN \times 2^{12})$.
8. Let `msipte` be the value of sixteen bytes at address `(a | (I x 16))`. If accessing `msipte` violates a PMA or PMP check, then stop and report "MSI PTE load access fault" (cause = 261).
9. If `msipte` access detects a data corruption (a.k.a. poisoned data), then stop and report "MSI PT data corruption" (cause = 270).
10. If `msipte.V == 0`, then stop and report "MSI PTE not valid" (cause = 262).
11. 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.
12. If `msipte.W == 1` the PTE is write-through mode PTE and the translation process is as follows:
    a. If any bits or encoding 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.
13. 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 encoding 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 MRIF violates a PMA or PMP check, then stop and report "MRIF access fault" (cause = 264). If the MRIF access detects a data corruption (a.k.a poisoned data), then stop and report "MSI MRIF data corruption" (cause = 271).
14. MSI address translation process is complete.

2.4. Faults from virtual address translation process

Faults detected during the S-stage or two-stage address translation specified in the privileged specification cause the IOVA translation process to stop and report the detected fault.

2.5. PCIe ATS translation request handling

ATS translation requests that encounter a configuration error results in a Completer Abort (CA) response to the requester. The following cause codes belong to this category:

- Instruction access fault (cause = 1)
- Read access fault (cause = 5)
- Write/AMO access fault (cause = 7)
- MSI PTE load access fault (cause = 261)
- MSI PTE misconfigured (cause = 263)
- PDT entry load access fault (cause = 265)
- PDT entry misconfigured (cause = 267)

If there is a permanent error or if ATS transactions are disabled then a Unsupported Request (UR) response is generated. The following cause codes belong to this category:

- All inbound transactions disallowed (cause = 256)
- DDT entry load access fault (cause = 257)
- DDT entry not valid (cause = 258)
- DDT entry misconfigured (cause = 259)
- Transaction type disallowed (cause = 260)

When translation could not be completed due to PDT entry being not present, MSI PTE being not present, or first and/or second stage PTE being not present or misconfigured then a Success Response with R and W bits set to 0 is generated. The translated address returned with such completions is undefined. The following cause codes belong to this category:

- Instruction page fault (cause = 12)
- Read page fault (cause = 13)
- Write/AMO page fault (cause = 15)
- Instruction guest page fault (cause = 20)
- Read guest-page fault (cause = 21)
- Write/AMO guest-page fault (cause = 23)
- PDT entry not valid (cause = 266)
- MSI PTE not valid (cause = 262)

If the translation request has a PASID prefix with "Privilege Mode Requested" field set to 0, or the request does not have a PASID prefix then the request does not target privileged memory. If the U-bit that indicates if the memory is accessible to user mode is 0 then a Success response with R and W bits set to 0 is generated.

If the translation request has a PASID prefix with "Privilege Mode Requested" field set to 1, then the request targets privileged memory. If the U-bit that indicates if the page is accessible to user mode is 1 and the `SUM` bit in `ta` field of the process-context is 0 then a Success response with R and W bits set to 0 is generated.

If the translation could be successfully completed but the requested permissions are not present (Execute requested but no execute permission; no-write not requested and no write permission; no read permission) then a Success response is returned with the denied permission (R, W or X) set to 0 and the other permission bits set to value determined from the page tables. The X permission is granted only if the R permission is also granted. Execute-only translations are not compatible with PCIe ATS as PCIe requires read permission to be granted if the execute permission is granted.

When a Success response is generated for a ATS translation request, no fault records are reported to software through the fault/event reporting mechanism; even when the response indicates no access was granted or some permissions were denied.

If the translation request has an address determined to be an MSI address using the rules defined by the AIA specification but the MSI PTE is configured in MRIF mode then a Success response is generated with R, W, and U bit set to 1. The U bit being set to 1 in the response instructs the device that it must only use Untranslated requests to access the implied 4 KiB memory range.

When a MSI PTE is configured in MRIF mode, a MSI write with data value D requires the IOMMU to set the interrupt-pending bit for interrupt identity D in the MRIF. A translation request from a device to a GPA that is mapped through a MRIF mode MSI PTE is not eligible to receive a translated address. This is accomplished by setting "Untranslated Access Only" (U) field of the returned response to 1.

When a Success response is generated for a ATS translation request, the setting of the Priv, N, CXL.io, and AMA fields is as follows:

- Priv field of the ATS translation completion is always set to 0 if the request does not have a PASID prefix. When a PASID prefix is present then the Priv field is set to the value in "Privilege Mode Requested" field as the permissions provided correspond to those the privilege mode indicate in the request.
- N field of the ATS translation completion is always set to 0. The device may use other means to determine if the No-snoop flag should be set in the translated requests.
- If requesting device is not a CXL device then CXL.io is set to 0.
- If requesting device is a CXL type 1 or type 2 device and the memory attribute, as determined by the Svpbmt extension, is NC or IO then the CXL.io bit is set to 1. If the Svpbmt extension is not supported then the setting of this bit is unspecified.
- The AMA field is by default set to 000b. The IOMMU may support an implementation specific method to provide other encodings.

2.6. PCIe ATS Page Request handling

To process a "Page Request" or "Stop Marker" message, the IOMMU first locates the device-context to determine if ATS and PRI are enabled for the requestor. If ATS and PRI are enabled, i.e. `EN_ATS` and `EN_PRI` are both set to 1, the IOMMU queues the message into an in-memory queue called the page-request-queue (`PQ`) (See Section 3.3). Following suitable processing of the "Page Request", a software handler may generate a "Page Request Group Response" message to the device.

When PRI is enabled for a device, the IOMMU may still be unable to report "Page Request" or "Stop Marker" messages through the `PQ` due to error conditions such as the queue being disabled, queue being full, or the IOMMU encountering access faults when attempting to access queue memory. These

error conditions are specified in Section 3.3.

If EN_PRI is set to 0, or EN_ATS is set to 0, or if the IOMMU is unable to locate the dc to determine the EN_PRI configuration, or the request could not be queued into PQ then the IOMMU behavior depends on the type of "Page Request".

- If the "Page Request" does not require a response, i.e. the "Last Request in PRG" field of the message is set to 0, then such message are silently discarded. "Stop Marker" messages do not require a response and are always silently discarded on such errors.
- If the "Page Request" needs a response, then the IOMMU itself may generate a "Page Request Group Response" message to the device.

When the IOMMU generates the response, the status field of the response depends on the cause of the error.

The status is set to Response Failure if the following faults are encountered:

- All inbound transactions disallowed (cause = 256)
- DDT entry load access fault (cause = 257)
- DDT entry misconfigured (cause = 259)
- DDT entry not valid (cause = 258)

The status is set to Invalid Request if the following faults are encountered:

- Transaction type disallowed (cause = 260)

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 classifies this condition (cause = 260) as a non-catastrophic failure, an Invalid Request, in its response to avoid the shared PRI in the device being disabled for all PFs/VFs.

A "Stop Marker" is encoded as a "Page Request" with a PASID prefix but with the L, W, and R fields set to 1, 0, and 0 respectively.

For IOMMU generated "Page Request Group Response" messages that have status Invalid Request, the PRG-response-PASID-required (PRPR) bit when set to 1 indicates that the IOMMU response message should include a PASID prefix if the associated "Page Request" had a PASID prefix.

For IOMMU generated "Page Request Group Response" with response code set to Response Failure, if the "Page Request" had a PASID prefix then response is generated with a PASID prefix.

2.7. 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).

This specification does not allow the caching of S/VS/G-stage PTEs whose V (valid) bit is clear, non-

leaf DDT entries whose `V` (valid) bit is clear, Device-context whose `V` (valid) bit is clear, non-leaf PDT entries whose `V` (valid) bit is clear, Process-context whose `V` (valid) bit is clear, or MSI PTEs whose `V` bit is clear.

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. A simpler implementation may not implement IOATC for some or 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 entries

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. This queue is supported if the IOMMU supports PCIe defined Page Request Interface.

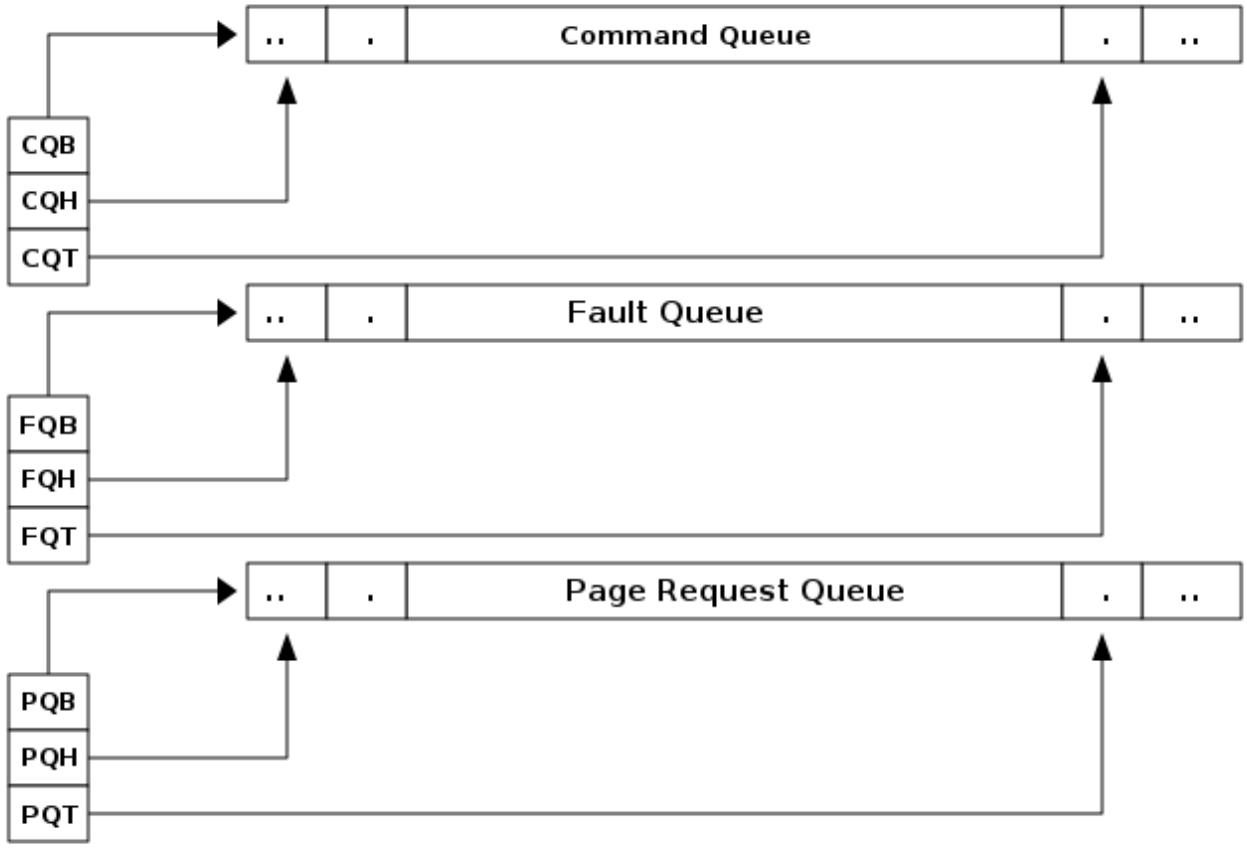


Figure 27. IOMMU in-memory queues

Each queue is a circular buffer with a head controlled by the consumer of data from 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. The tail register holds the index into the queue where the next entry will be written by the producer. The head register holds the index into the queue where the consumer will read the next entry to process.

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 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 but is not prohibited by this specification.

3.1. Command-Queue (CQ)

Command queue is used by software to queue commands to be processed by the IOMMU. Each command is 16 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 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 queue that IOMMU should process next. Subsequent to reading each command the 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 the format of the operand fields. One or more of those fields may be used by the specific function invoked.

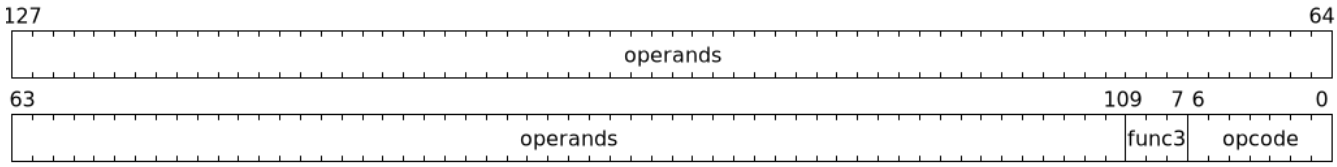
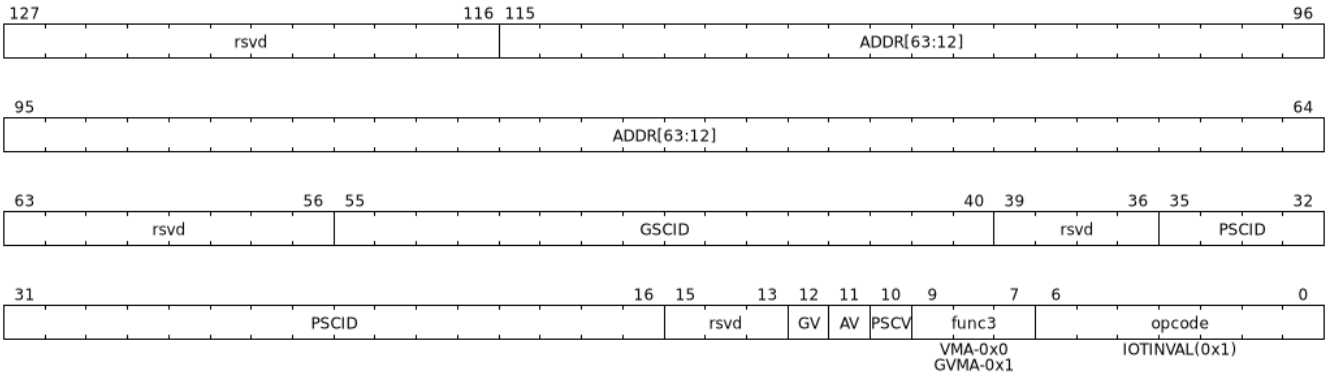


Figure 28. Format of an IOMMU command

3.1.1. IOMMU Page-Table cache invalidation commands



IOMMU operations cause implicit reads to PDT, first-stage and second-stage page tables. To reduce latency of such reads, the IOMMU may cache entries from the first and/or second-stage page tables in the IOMMU-address-translation-cache (IOATC). These caches may not observe modifications performed by software to these data structures in memory.

The IOMMU translation-table cache invalidation commands, IOTINVAL.VMA and IOTINVAL.GVMA synchronize updates to in-memory S/VS-stage and G-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 allowed only for 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 ensures that previous stores made to the first-stage page tables by the harts are observed by the IOMMU 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 may not be 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 may not be invalidated.

IOTINVAL.GVMA ensures 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.

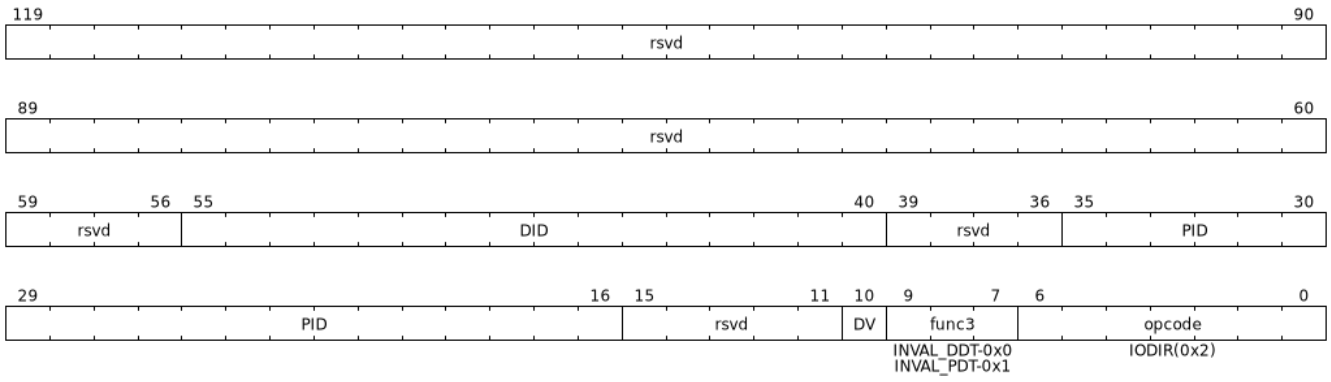
Table 6. IOTINVAL.GVMA operands and operations

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 to DDT and/or PDT. To reduce latency of such reads, the IOMMU may cache entries from the DDT and/or PDT in IOMMU directory caches. These caches may not observe modifications performed by software to these data structures in memory.

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

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

The DV operand indicates if the device ID (DID) operand is valid. The DV operand must be 1 for IODIR.INVAL_PDT.

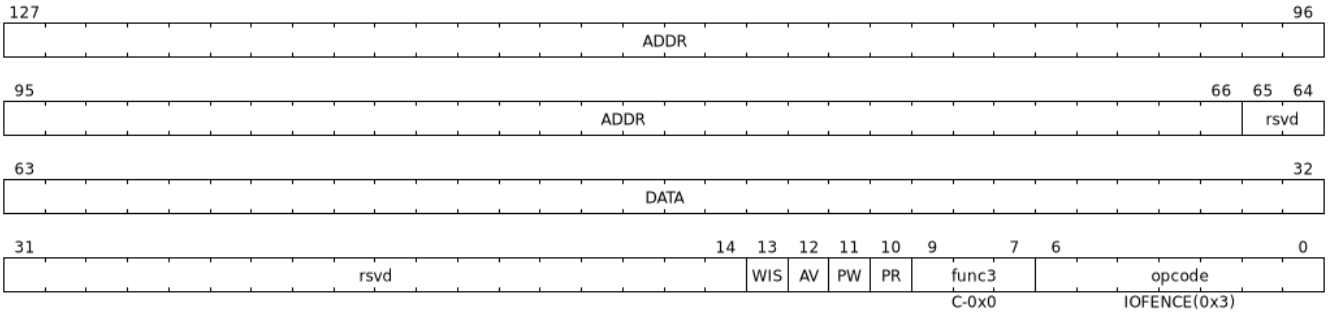
IODIR.INVAL_DDT guarantees that any previous stores made by a RISC-V hart to the DDT are observed before all subsequent implicit reads from IOMMU to DDT. If DV is 0, then the command invalidates all DDT and PDT entries cached for all devices. If DV is 1, then the command invalidates cached leaf level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

IODIR.INVAL_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 cached 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 issue the appropriate IODIR command as some implementations may choose to cache the translated pointers in the IOMMU directory caches.

IOTINVAL command 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 co-processors. 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 that was previously 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. 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.

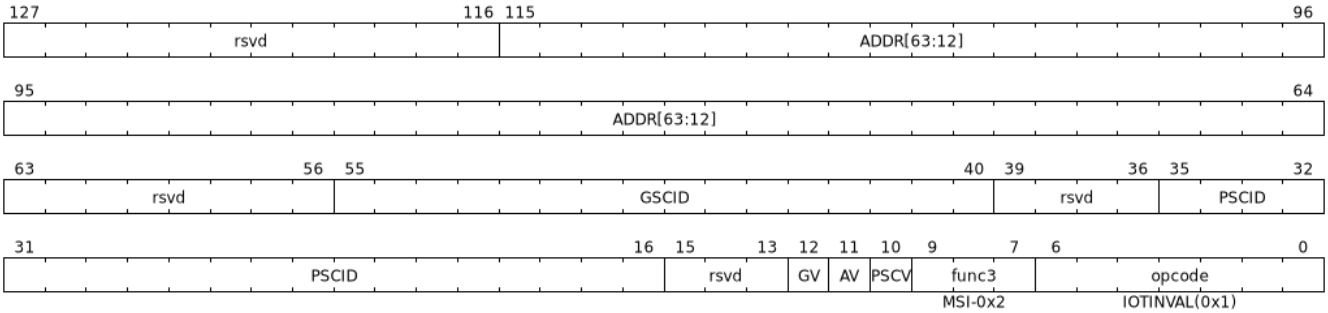
Simpler implementations may unconditionally order all previous memory accesses globally.

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* `seteipnum_le`/`seteipnum_be` *register 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 to 1.



IOMMU operations cause implicit reads to MSI page tables. To reduce latency of such reads, the IOMMU may cache entries from the MSI page tables. These caches may not observe modifications performed by software to these data structures in memory.

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

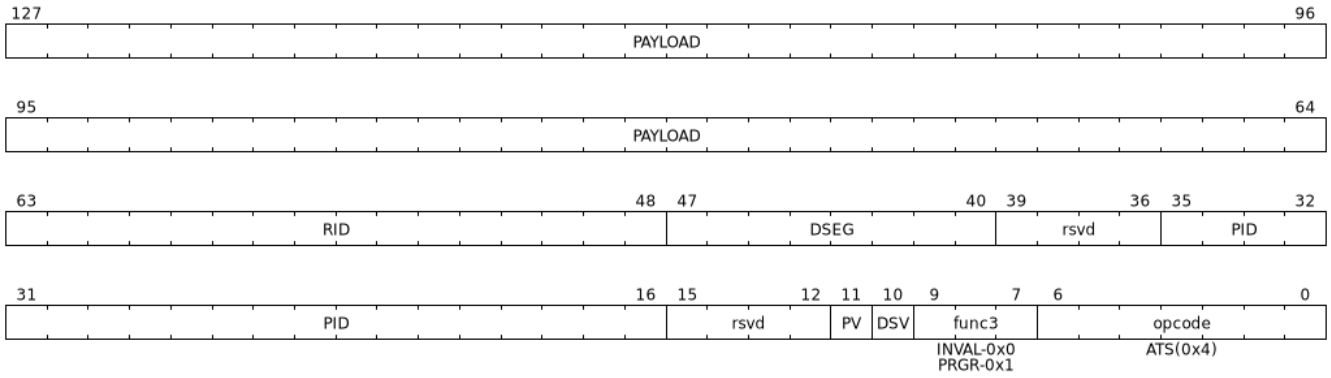
The `PSCV` operand is reserved and must be 0 for `IOTINVAL.MSI`.

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 to 1.



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 Request Group Response" message to the PCIe device function identified by the RID. The "Page Request 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.

The format of the payload of an ATS "Invalidation Request" message is specified by the PCIe specification. Software specifies the untranslated address range to be invalidated in the payload.

The format of the payload of an ATS "Page Request Group Response" message is specified by the PCIe specification. The PAYLOAD[15:0] bits are used as the contents of the bytes 8 and 9 of the message.

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 32 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.

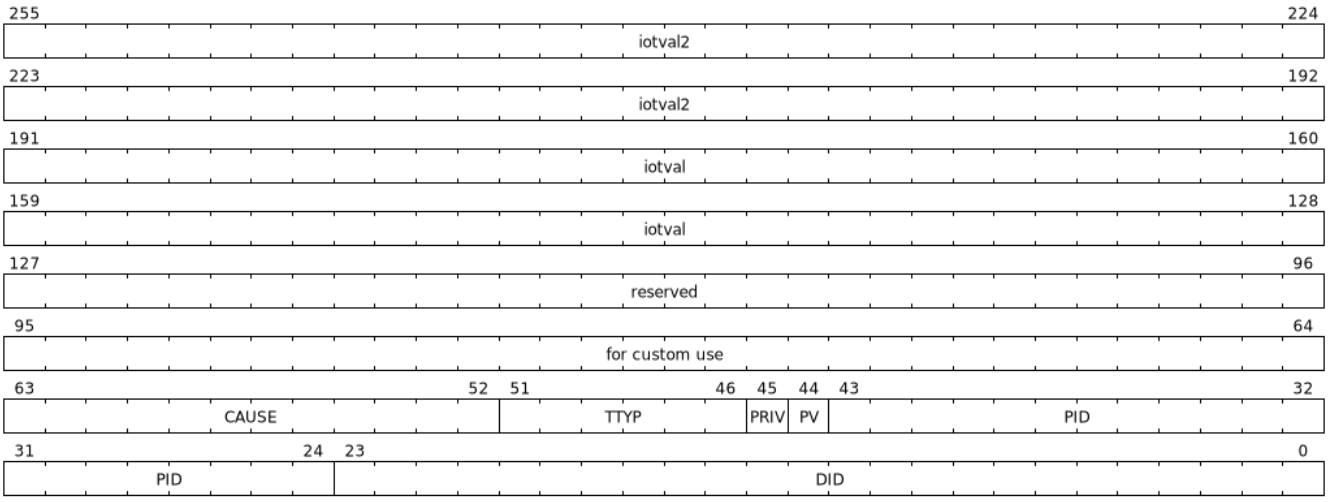


Figure 29. Fault-queue record

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

Table 8. Fault record CAUSE field encodings

CAUSE	Description	Reported if DTF is 1?
0	Instruction address misaligned	No
1	Instruction access fault	No

CAUSE	Description	Reported if DTF is 1?
4	Read address misaligned	No
5	Read access fault	No
6	Write/AMO address misaligned	No
7	Write/AMO access fault	No
12	Instruction page fault	No
13	Read page fault	No
15	Write/AMO page fault	No
20	Instruction guest page fault	No
21	Read guest-page fault	No
23	Write/AMO guest-page fault	No
256	All inbound transactions disallowed	Yes
257	DDT entry load access fault	Yes
258	DDT entry not valid	Yes
259	DDT entry misconfigured	Yes
260	Transaction type disallowed	No
261	MSI PTE load access fault	No
262	MSI PTE not valid	No
263	MSI PTE misconfigured	No
264	MRIF access fault	No
265	PDT entry load access fault	No
266	PDT entry not valid	No
267	PDT entry misconfigured	No
268	DDT data corruption	No
269	PDT data corruption	No
270	MSI PT data corruption	No
271	MSI MRIF data corruption	No
272	Internal datapath error	No
273	IOMMU MSI write access fault	Yes

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

TTYP	Description
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. If TTYP is 0, then the value reported in iotval and iotval2 fields is as defined by the CAUSE.

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. The DID, PV, `PID, and PRIV fields are 0 if TTYP is 0.

If the CAUSE is a guest-page fault then the guest-physical-address right shifted by 2 is reported in iotval2[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” and "Stop Marker" 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). Each Page-Request record is 16 bytes.

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 Request" messages through the queue due to error conditions such as the queue being disabled, 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. 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 the queue memory, page-request-queue memory access fault (pqmf) bit in pqcsr. While either error bits are set in pqcsr, the IOMMU discards all subsequent "Page Request" messages; including the message that caused the error bits to be set. "Page request" messages that do not require a response, i.e. those with the "Last Request in PRG" field is 0, are silently discarded. "Page request" messages that require a response, i.e. those with "Last Request in PRG" field set to 1 and are not Stop Marker messages, may be auto-completed by an IOMMU generated “Page Request Group Response” message as specified by PCIe ATS specifications. Such responses shall have the status set to Invalid Request.

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.

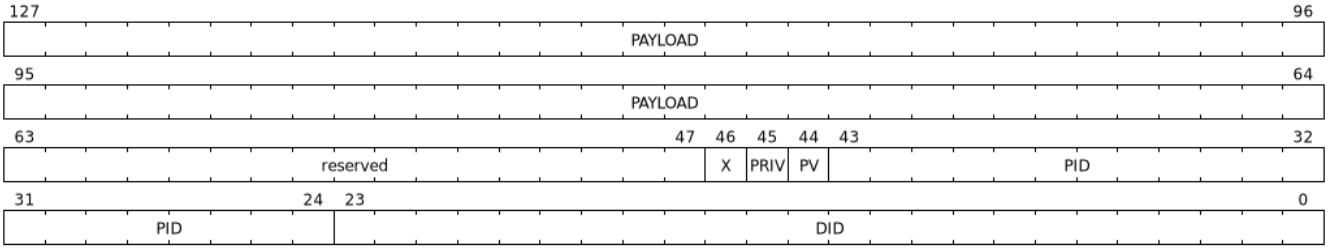


Figure 30. Page-request-queue record

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.

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
12	*custom*	4	*For custom use*

Offset	Name	Size	Description
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	Reserved	82	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. Reset behavior

The reset default value is 0 for the following registers. Reset value is implementation-defined for all other registers and/or fields.

- fctrl
- cqcsr
- fqcsr
- pqcsr

4.3. 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.

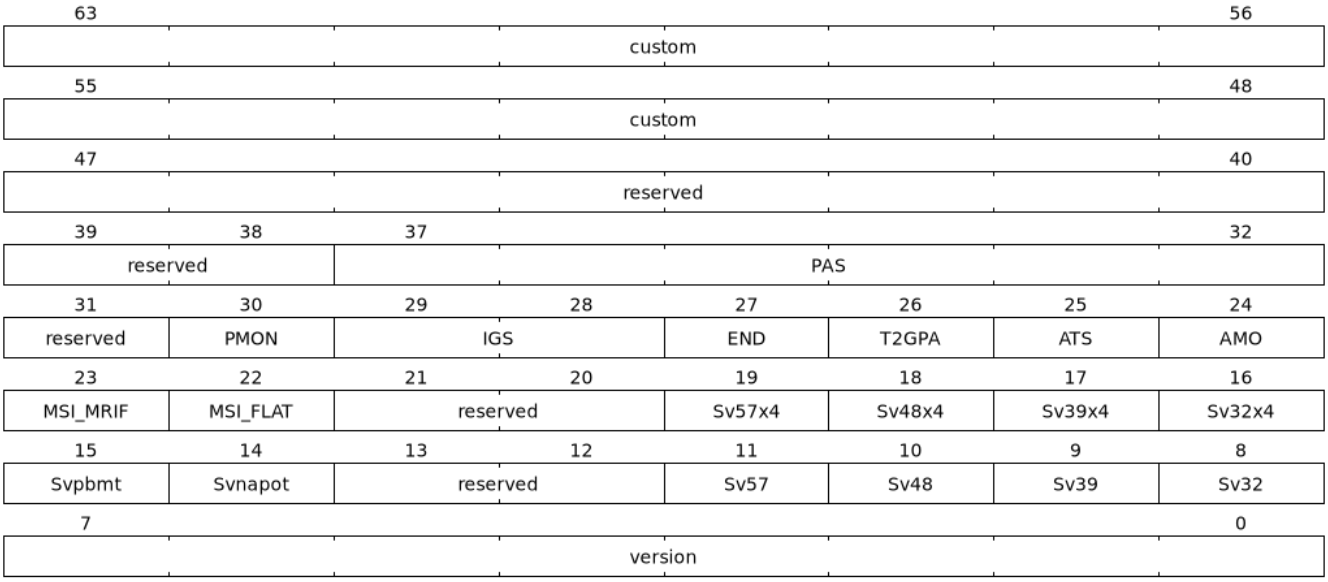


Figure 31. capabilities register fields

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.

Bits	Field	Attribute	Description
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.
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. 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
30	PMON	RO	IOMMU implements a performance-monitoring unit
31	RAS	RO	IOMMU implements the RISC-V RAS Registers
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*

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.

4.4. 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 writable 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.

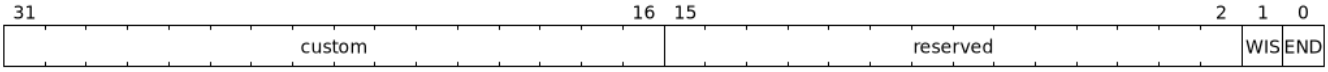


Figure 32. Features-control register fields

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.*

4.5. Device-directory-table pointer (ddtp)

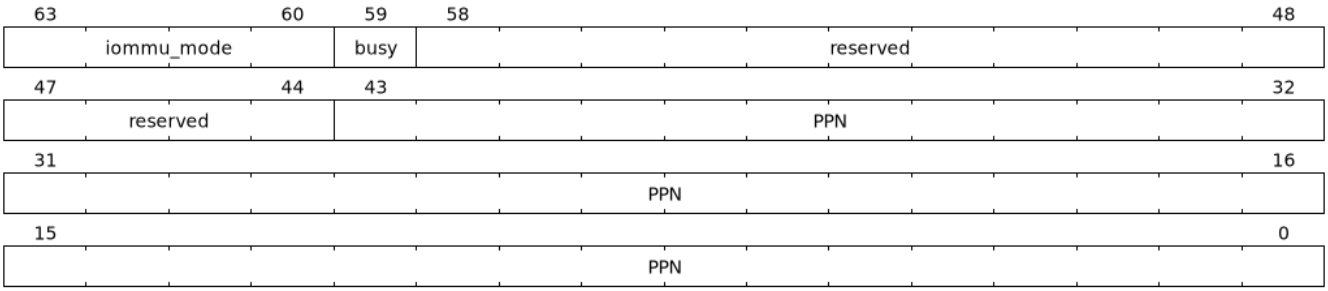


Figure 33. Device-directory-table pointer register fields

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
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 hard-wire 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. When in Bare mode only Untranslated requests are allowed. Translated requests, Translation request, and message transactions are unsupported.

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`.

4.6. 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. Each command is 16 bytes.

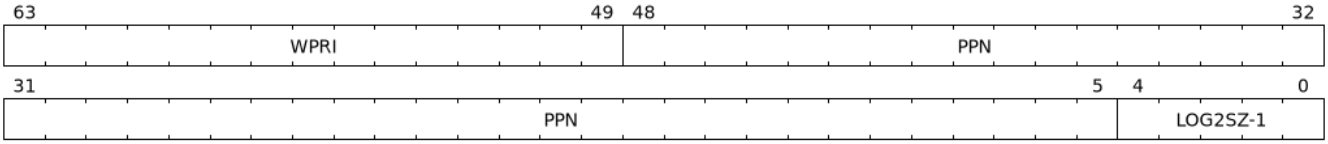


Figure 34. Command-queue base register fields

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

4.7. Command-queue head (cqh)

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

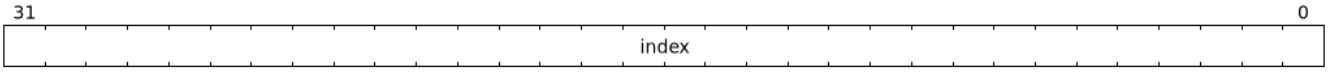


Figure 35. Command-queue head register fields

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.

4.8. 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.

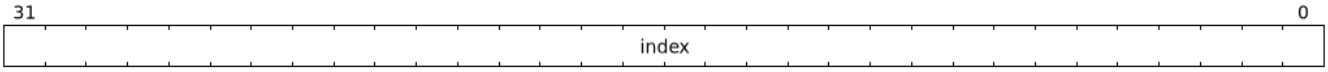


Figure 36. Command-queue tail register fields

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 writable when the queue is in enabled state (i.e., cqsr.cqon == 1).

4.9. 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. Each fault record is 32 bytes.

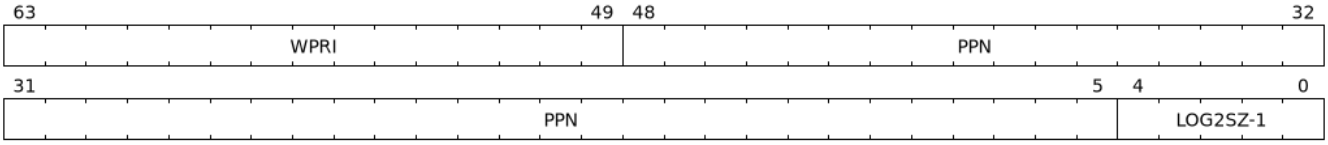


Figure 37. Fault queue base register fields

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 32-bytes. If the fault-queue has 128 or fewer entries then the base address of the queue is always aligned to 4-KiB. If the fault-queue has more than 128 entries then the fault-queue base address must be naturally aligned to 2^{LOG2SZ} x 32.
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

4.10. Fault queue head (fqh)

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

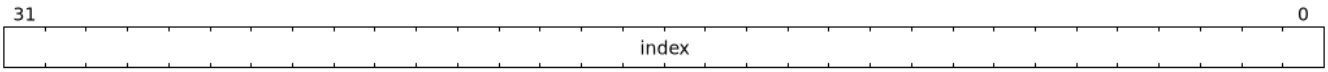


Figure 38. Fault queue head register fields

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 writable when the queue is in enabled state (i.e., fqsr.fqon == 1).

4.11. Fault queue tail (fqt)

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

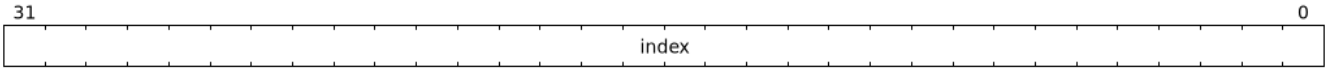


Figure 39. Fault queue tail register fields

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

4.12. 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. Each page-request message is 16 bytes.

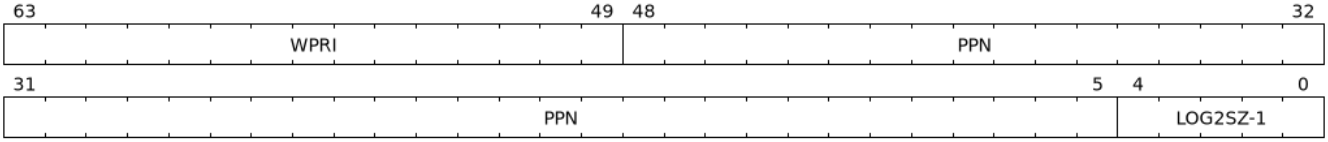


Figure 40. Page-Request-queue base register fields

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 $2^{LOG2SZ} \times 16$.
48:5	PPN	WARL	Holds the PPN of the root page of the in-memory page-request-queue used by IOMMU to queue "Page Request" messages.
63:49	WPRI	WPRI	Reserved for standard use

4.13. 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.

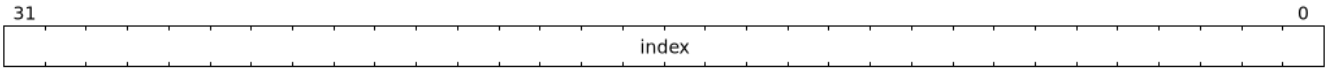


Figure 41. Page-request-queue head register fields

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

4.14. 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.

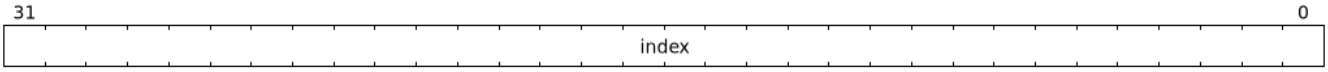


Figure 42. Page-request-queue tail register fields

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

4.15. Command-queue CSR (cqcsr)

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

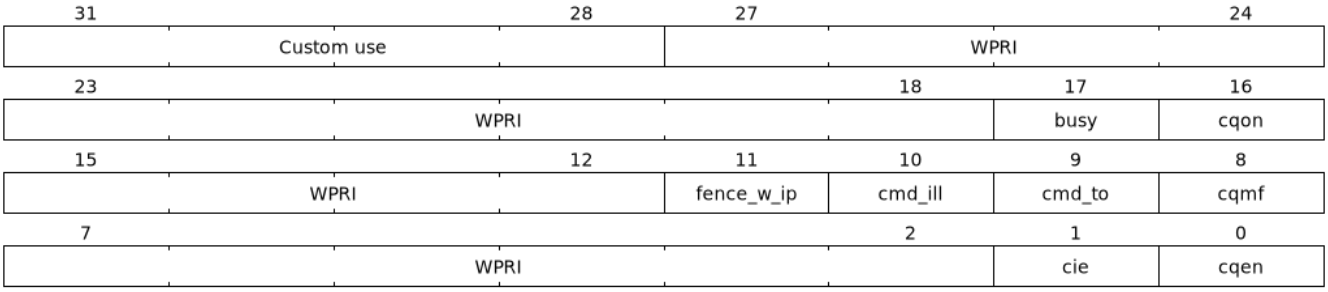


Figure 43. Command-queue CSR register fields

Bits	Field	Attribute	Description
0	cqen	RW	The command-queue-enable bit enables the command- queue when set to 1. Changing cqen from 0 to 1 sets the cqh and cqt 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, cqh is set to 0, cqt is set to 0 and the cqcsr bits cmd_ill, cmd_to, cqmf, fence_w_ip are set to 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.
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.

Bits	Field	Attribute	Description
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.
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 hard-wire this bit to 0. An IOMMU that can complete these operations synchronously may hard-wire 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.

4.16. Fault queue CSR (fqcsr)

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

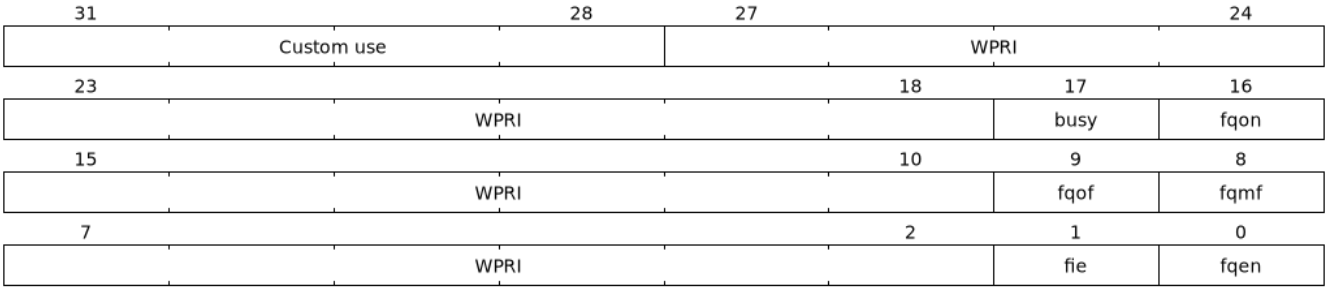


Figure 44. Fault queue CSR register fields

Bits	Field	Attribute	Description
0	fqen	RW	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. ipsr.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. ipsr.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 hard-wire this bit to 0.
27:18	WPRI	WPRI	Reserved for standard use
31:28	*custom*		*These bits are reserved for custom use.*

4.17. 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.

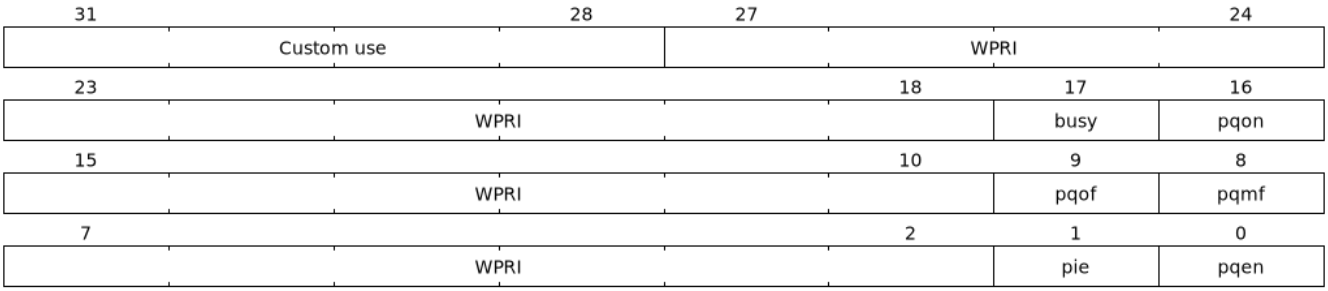


Figure 45. Page-request-queue CSR register fields

Bits	Field	Attribute	Description
16	pqon	RW	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 hard-wire this bit to 0
27:18	WPRI	WPRI	Reserved for standard use
31:28	*custom*		*These bits are designated for custom use.*

4.18. 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.

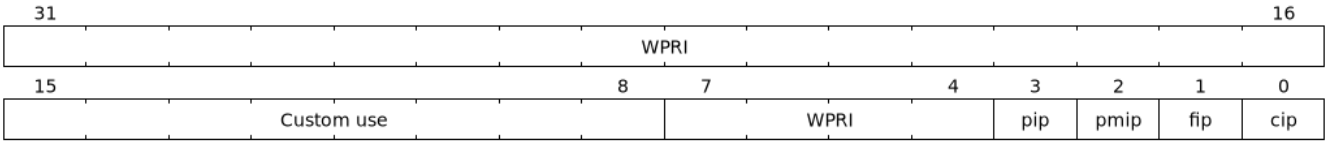


Figure 46. Interrupt pending status register fields

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

4.19. 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.

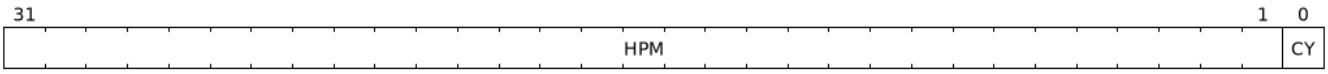


Figure 47. iocntovf register fields

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

4.20. 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.

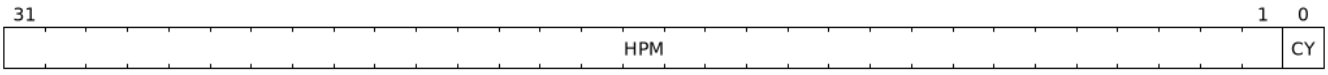


Figure 48. iocntinh register fields

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.

4.21. Performance-monitoring cycles counter (iohpmcycles)

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

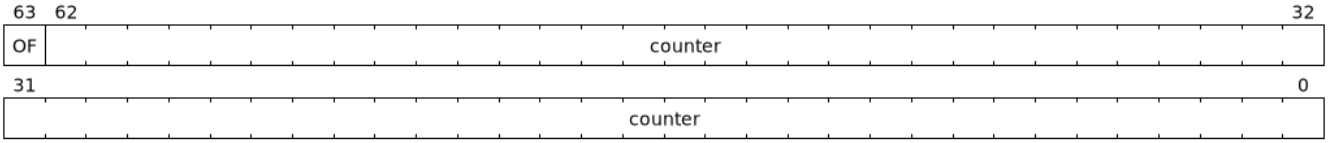


Figure 49. iohpmcycles register fields

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.

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

These registers are 64-bit WARL counter registers.

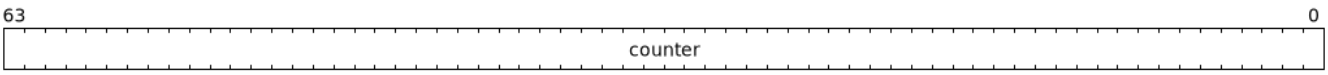


Figure 50. iohpmctr register fields*

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.

4.23. 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.

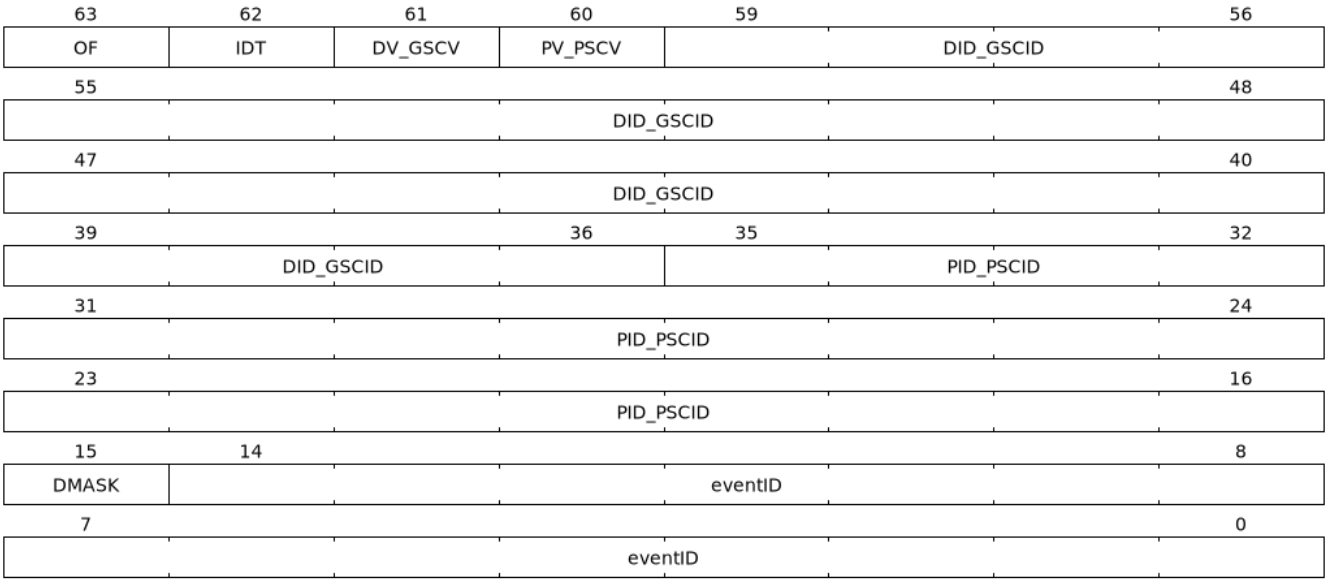


Figure 51. iohpmevt register fields*

Bits	Field	Attribute	Description
14:0	eventID	WARL	Indicates the event to count. A value of 0 indicates no events are counted. Encodings 1 to 16383 are reserved for standard events defined in the Table 14. Encodings 16384 to 32767 are for reserved for custom use and 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.

Bits	Field	Attribute	Description
61	DV_GSCV	WARL	If set, only transactions with matching device_id or GSCID (based on the Filter ID Type) are counted.
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.

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
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 following table lists the standard events that can be counted:

Table 14. Standard Events list

eventID	Event counted	IDT settings supported
0	Do not count	
1	Untranslated requests	0
2	Translated requests	0
3	ATS Translation requests	0
4	Device-context cache miss	0
5	Process-context cache miss	0
6	TLB miss	0/1
7	Device Directory Walks	0
8	Process Directory Walks	0
9	S/VS-stage Page Table Walks	0/1
10	G-stage Page Table Walks	0/1
11 - 16383	reserved for future standard	-

Some events types may be filtered by IDs. When a event type that does not support filtering by IDs is programmed then the associated counter does not increment.

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 ipsr 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.

4.24. 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 writable.

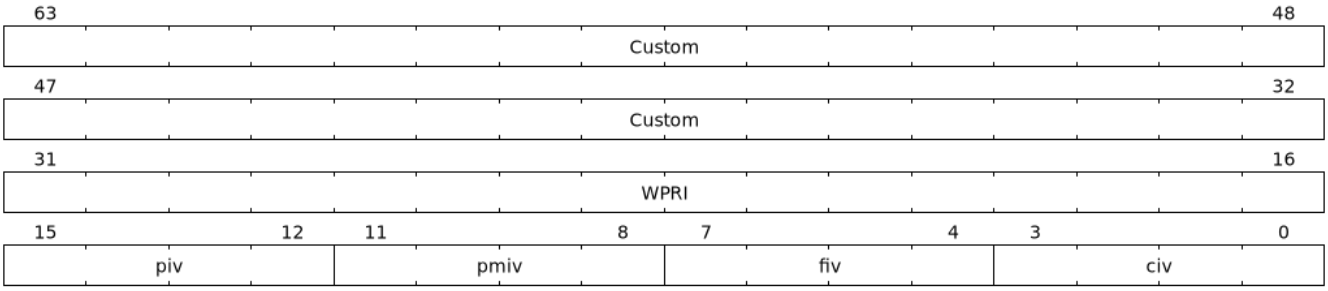


Figure 52. icvec register fields

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.

If an access fault is detected on a MSI write using msi_addr_x, then the IOMMU reports a "IOMMU MSI write access fault" (cause 273) fault, with TTYP set to 0 and iotval set to the value of msi_addr_x.

Table 15. 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

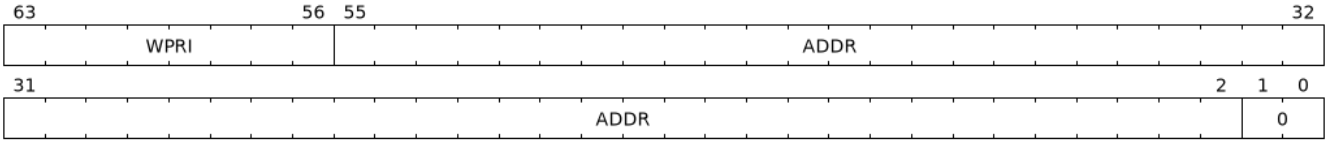


Figure 53. msi_addr_x register fields

Bits	Field	Attribute	Description
1:0	0	RO	Fixed to 0
55:2	ADDR	WARL	Holds the 4-byte aligned MSI address.
63:56	WPRI	WPRI	Reserved for future use.

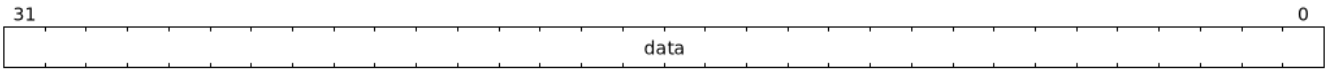


Figure 54. msi_data_x register fields

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

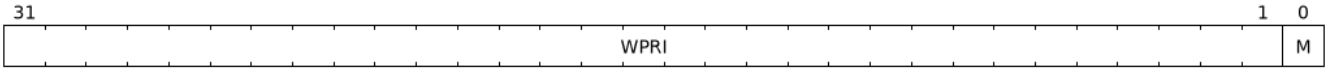


Figure 55. msi_vec_ctrl_x register fields

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 guidelines initializing the IOMMU are as follows:

1. Read the `capabilities` register to discover the capabilities of the IOMMU.
2. Stop and report failure if `capabilities.version` is not supported.
3. Read the feature control register (`fctrl`).
4. Stop and report failure if big-endian memory access is needed and the `capabilities.END` field is 0 (i.e. only one endianness) and `fctrl.END` is 0 (i.e. little endian).
5. If big-endian memory access is needed and the `capabilities.END` field is 1 (i.e. both endianness supported), set `fctrl.END` to 1 (i.e. big endian) if the field is not already 1.
6. Stop and report failure if wired-interrupts are needed for IOMMU initiated interrupts and `capabilities.IGS` is not `WIS`.
7. If wired-interrupts are needed for IOMMU initiated interrupts and `capabilities.IGS` is `BOTH`, set `fctrl.WIS` to 1 if the field is not already 1.
8. Stop and report failure if other required capabilities (e.g. virtual-addressing modes, MSI translation, etc.) are not supported.
9. The `icvec` register is used to program an interrupt vector for each interrupt cause. Determine the number of vectors supported by the IOMMU by writing 0xF to each field and reading back the number of writable bits. If the number of writable bits is `N` then the number of supported vectors is 2^N. For each cause `C` associate a vector `V` with the cause. `V` is a number between 0 and $(2^N - 1)$.
10. If the IOMMU is configured to use wired interrupts, then each vector `V` corresponds to an interrupt wire connected to a platform level interrupt controller (e.g. APLIC). Determine the interrupt controller configuration register to be programmed for each such wire using configuration information provided by configuration mechanisms such as device tree and program the interrupt controller.
11. If the IOMMU is configured to use MSI, then each vector `V` is an index into the `msi_cfg_tbl`. For each vector `V`, allocate a MSI address `A` and an interrupt identity `D`. Configure the `msi_addr_V` register with value `A`, `msi_data_V` register with value `D`. Configure the interrupt mask `M` in `msi_vec_ctrl_V` register appropriately.
12. To program the command queue, first determine the number of entries `N` needed in the command queue. The number of entries in the command queue is always a power of two. Let $N = 2^k$. If `N` is 256 or lower then allocate a memory buffer that is aligned to a 4-KiB page address and is of size `N` x 16-bytes. If `N` is greater than 256 then allocate a memory buffer that is naturally aligned to a `N` * 16-byte address boundary. Let the physical page number of the buffer be `B`. Program the command queue registers as follows.
    - Poll on `cqcsr.busy` till it reads 0

- `temp_cqb_var.PPN = B`
- `temp_cqb_var.LOG2SZ-1 = (k - 1)`
- `cqb = temp_cqb_var`
- `cqcsr.cqen = 1`
- Poll on `cqcsr.cqon` till it reads 1

13. To program the fault queue, first determine the number of entries N needed in the fault queue. The number of entries in the fault queue is always a power of two. Let $N = 2^k$. If `N` is 128 or lower then allocate a memory buffer that is aligned to 4-KiB page address and is of size `N` x 32-bytes. If `N` is greater than 128 then allocate a memory buffer that is naturally aligned to a `N` * 32-byte address boundary. Let the physical page number of the buffer be `B`. Program the fault queue registers as follows:
    - Poll on `fqcsr.busy` till it reads 0
    - `temp_fqb_var.PPN = B`
    - `temp_fqb_var.LOG2SZ-1 = (k - 1)`
    - `fqb = temp_fqb_var`
    - `fqcsr.fqen = 1`
    - Poll on `fqcsr.fqon` till it reads 1
14. To program the page-request queue, first determine the number of entries `N` needed in the page-request queue. The number of entries in the page-request queue is always a power of. Let $N = 2^k$. If `N` is 256 or lower then allocate a memory buffer that is aligned to 4-KiB page address and is of size `N` x 16-bytes. If `N` is greater than 256 then allocate a memory buffer at an address that is naturally aligned to a `N` * 16-byte address boundary. Let the physical page number of the buffer be `B`. Program the page-request queue registers as follows:
    - Poll on `pqcsr.busy` till it reads 0
    - `temp_pqb_var.PPN = B`
    - `temp_pqb_var.LOG2SZ-1 = (k - 1)`
    - `pqb = temp_pqb_var`
    - `pqcsr.pqen = 1`
    - Poll on `pqcsr.pqon` till it reads 1
15. To program the DDT pointer, first determine the supported `device_id` width `Dw` and the format of the device-context data structure. If `capabilities.MSI` is 0, then the IOMMU uses base format device-context else extended format device-context are used. Allocate a page (4 KiB) of memory to use as the root table of the DDT. Let `B` be the PPN of the allocated memory. Initialize the allocated memory to all 0. Determine the mode `M` of the DDT based on `Dw` and if the IOMMU requires base or extended format device-contexts.
16. If extended format device-context are used then
    a. If `Dw` is less than or equal to 6-bits then `M = 1LVL`
    b. If `Dw` is less than or equal to 15-bits then `M = 2LVL`
    c. If `Dw` is less than or equal to 24-bits then `M = 3LVL`
17. If base format device-context are used then

a. If Dw is less than or equal to 7-bits then M = 1LVL

b. If Dw is less than or equal to 16-bits then M = 2LVL

c. If Dw is less than or equal to 24-bits then M = 3LVL

18. Program the ddtp register as follows:

- temp_ddtp_var.MODE = M
- temp_ddtp_var.PPN = B
- ddtp = temp_ddtp_var

The IOMMU is initialized and may be now be configured with device-context for devices in scope of the IOMMU.

5.2. Guidelines for invalidation's

This section provides guidelines to software on the invalidation commands to send to the IOMMU through the CQ when modifying the IOMMU in-memory data structures. Software must perform the invalidation after the update is globally visible. The ordering on stores provided by FENCE instructions and the acquire/ release bits on atomic instructions also orders the data structure updates associated with those stores as observed by IOMMU.

A IOFENCE.C command may be used by software to ensure that all previous commands fetched from the CQ have been completed and committed.

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 invalidation's 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 invalidation's 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 invalidation's 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 invalidation's 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 invalidation's 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 invalidation's 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 invalidation's 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 invalidation's 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 invalidation's 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 invalidation's 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 super-page 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 invalidation's

When VS/S-stage and/or G-stage page tables are modified, invalidation's 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 invalidation's 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 degradation. An ATS capable device publishes the maximum number of invalidation's it can buffer before causing back-pressure 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 invalidation's queued to the device advertised 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 invalidation's 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 invalidation's to the DevATC.

5.2.8. Caching invalid entries

This specification does not allow the caching of S/VS/G-stage PTEs whose `V` (valid) bit is clear, non-leaf DDT entries whose `V` (valid) bit is clear, Device-context whose `V` (valid) bit is clear, non-leaf PDT entries whose `V` (valid) bit is clear, Process-context whose `V` (valid) bit is clear, or MSI PTEs whose `V` bit is clear.

Software need not perform invalidation's 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

IOMMU may generate an from the `CQ`, the `FQ`, the `PQ`, or the PMU. Each interrupt source may be configured with a unique vector or a vector may be shared among one or more interrupt sources. The interrupt may be delivered as a MSI or a wire-based-interrupt. The interrupt handler may perform the following actions:

1. Read the `ipsr` register to determine the source of the pending interrupts

2. If ipsr.cip bit is set then an interrupt is pending from the CQ.
   a. Read the cqcsr register.
   b. Determine if an error caused the interrupt and if so, the cause of the error by examining the state of the cmd_to, cmd_ill, and cqmf bits If any of these bits are set then the CQ encountered an error and command processing is temporarily disabled.
   c. If errors have occurred, correct the cause of the error and clear the bits corresponding to the corrected errors in cqcsr by writing 1 to the bits.
      i. Clearing all error indication bits in cqcsr re-enables command processing.
   d. An IOMMU that supports wired-interrupts may be requested to generate an interrupt from the command queue on completion of a IOFENCE.C command. This cause is indicated by the fence_w_ip bit. Note that command processing does not stop when fence_w_ip is set to 1. Software handler may re-enable interrupts from CQ on IOFENCE.C completions by clearing this bit by writing 1 to it.
3. If ipsr.fip bit is set then an interrupt is pending from the FQ.
   a. Read the fqcsr register.
   b. Determine if an error caused the interrupt and if so, the cause of the error by examining the state of the fqmf and fqof bits. If either of these bits are set then the FQ encountered an error and fault/event reporting is temporarily disabled.
   c. If errors have occurred, correct the cause of the error and clear the bits corresponding to the corrected errors in fqcsr by writing 1 to the bits.
      i. Clearing all error indication bits in cqcsr re-enables fault/event reporting.
   d. Read the fqt and fqh registers.
   e. If value of fqt is not equal to value of fqh then the FQ is not empty and contains fault/event reports that need processing.
   f. Process pending fault/event reports that need processing and remove them from the FQ by advancing the fqh by the number of records processed.
4. If ipsr.pip bit is set then an interrupt is pending from the PQ.
   a. Read the `pqcsr`register.
   b. Determine if an error caused the interrupt and if so, the cause of the error by examining the state of the pqmf and pqof bits. If either of these bits are set then the PQ encountered an error and "Page Request" reporting is temporarily disabled.
   c. If errors have occurred, correct the cause of the error and clear the bits corresponding to the corrected errors in pqcsr by writing 1 to the bits.
      i. Clearing all error indication bits in pqcsr re-enables "Page Request" reporting.
   d. Read the pqt and pqh registers.
   e. If value of pqt is not equal to value of pqh then the PQ is not empty and contains "Page Request" messages that need processing.
   f. Process pending "Page Request" messages that need processing and remove them from the PQ by advancing the pqh by the number of records processed.
      i. If a PQ overflow condition caused the IOMMU to automatically respond to a "Page Request" with the "Last Request in PRG" flag set to 1, then software may observe an incomplete page-request group. Software should ignore the group and not service such groups.

5. If `ipsr.pmip` bit is set then an interrupt is pending from the PMU.
   a. Process the performance monitoring counter overflows.
6. For each source that was serviced in this process, clear the interrupt pending bit for the source by clearing the corresponding bit in `ipsr`.

5.3. Guidelines for enabling and disabling ATS and/or PRI

To enable ATS and/or PRI:

1. Place the device in an idle state such that no transactions are generated by the device.
2. If the device-context for the device is already valid then first mark the device-context as invalid and queue commands to the IOMMU to invalidate all cache G/S/VS-stage page table entries, DDT entries, MSI PT entries (if required), and PDT entries (if required).
3. Program the device-context with `EN_ATS` set to 1 and if required the `T2GPA` field set to 1. Set `EN_PRI` to 1 if required.
4. Mark the device-context as valid.
5. Enable device to use ATS and if required PRI.

To disable ATS and/or PRI:

1. Place the device in an idle state such that no transactions are generated by the device.
2. Disable ATS and/or PRI at the device
3. Set `EN_ATS` and/or `EN_PRI` to 0 in the device-context.
4. Queue commands to the IOMMU to invalidate all cached G/S/VS-stage page table entries, DDT entries, MSI PT entries (if required), and PDT entries (if required).
5. Queue commands to the IOMMU to invalidate DevATC by generating Invalidation Request messages.
6. Enable DMA operations in the device

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. Faults from PMA and PMP

The IO bridge may invoke a PMA and/or an PMP checker on memory accesses from IO devices or those generated by the IOMMU implicitly to access the in-memory data structures. When a memory access violates a PMA check or violates an PMP check, the IO bridge may abort the memory access as specified in Section 6.3.

6.3. Aborting transactions

If the aborted transaction is an IOMMU initiated implicit memory access then the IO bridge signals such access faults to the IOMMU itself. The details of such signaling is implementation defined.

If the aborted transaction is a write then the IO bridge may discard the write; the details of how the write is discarded is implementation defined. If the IO protocol requires a response for write transactions (e.g., AXI) then a response as defined by the IO protocol may be generated by the IO bridge (e.g., SLVERR on BRESP - Write Response channel). For PCIe, for example, write transactions are posted and no response is returned when a write transaction is discarded.

If the faulting transaction is a read then the device expects a completion. The IO bridge may provide a completion to the device. The data, if returned, in such completion is implementation defined; usually it is a fixed value such as all 0 or all 1. A status code may be returned to the device in the completion to indicate this condition. For AXI, for example, the completion status is provided by SLVERR on RRESP (Read Data channel). For PCIe, for example, the completion status field may be set to "Unsupported Request" (UR) or "Completer Abort" (CA).

6.4. Reliability, Availability, and Serviceability (RAS)

The IOMMU may support the RISC-V RAS architecture that specifies the methods for enabling error detection, logging the detected errors (including their severity, nature, and location), and configuring means to report the error to an error handler. The `capabilities.RAS` enumerates if the IOMMU

complies with the RISC-V RAS architecture.

Some errors, such as those in the IOATC, may be correctable by reloading the cached in-memory data structures when the error is detected. Such errors are not expected to affect the functioning of the IOMMU.

Some errors may corrupt critical internal of the IOMMU and such errors may lead the IOMMU to a failed state. Examples of such state may include registers such as the `ddtp`, `cqb`, etc. On entering such a failed state, the IOMMU may request the IO bridge to abort all incoming transactions.

Some errors, such as corruptions that occur within the internal data paths of the IOMMU, may not be correctable but the effects of such error may be contained to the transaction being processed by the IOMMU.

As part of processing a transaction, the IOMMU may need to read data from in-memory data structures such as the DDT, PDT, or S/VS/G-stage page tables. The provider (a memory controller or a cache) of the data may detect that the data requested has a uncorrectable error and signal that the data is corrupted and defer the error to the IOMMU. Such technique to defer the handling of the corrupted data to the consumer of the data is also commonly known as data poisoning. The effects of such errors may be contained to the transaction that caused the corrupted data to be accessed.

In the cases where the error effects the transaction being processed but otherwise allow the IOMMU to continue providing service, the IOMMU may abort (see Section 6.3) the transaction and report the the fault by queuing a fault record in the `FQ`. The following cause codes are used to report such faulting transactions:

- DDT data corruption (cause = 268)
- PDT data corruption (cause = 269)
- MSI PT data corruption (cause = 270)
- MSI MRIF data corruption (cause = 271)
- Internal data-path error (cause = 272)
- For VS/S/G-stage PTE report the cause code defined by the RISC-V RAS specification.

If the IO bridge is not capable of signaling such deferred errors uniquely from other errors that prevent the IOMMU from accessing in-memory data structures then the IOMMU may report such errors as access faults instead of using the differentiated data corruption cause codes.

Index

Bibliography