RISC-V IOMMU 架构规格

IOMMU 工作组

版本 v1.0.0，2023-07-25：已批准

译者：Song Cunjie

songcunjie@163.com

目录

序言

批准本文件。

不允许更改。任何希望或需要的更改都可以成为后续新延期的主题。已批准的延期永不修改。

版权和许可证信息

贡献者

对本 RISC-V 规范做出直接或间接贡献的有（按字母顺序排列）：

Aaron Durbin, Allen Baum, Anup Patel, Daniel Gracia Pérez, David Kruckemyer, Greg Favor, Ahmad Fawal, Guerney D Hunt, John Hauser, Josh Scheid, Matt Evans, Manuel Rodriguez, Nick Kossifidis, Paul Donahue, Paul Walmsley, Perrine Peresse, Philipp Tomsich, Rieul Ducousso, Scott Nelson, Siqi Zhao, Sunil V. L, Tomasz Jezach, Vassilis Papaefstathiou, Vedvyas Shanbhogue.L, Tomasz Jeznach, Vassilis Papaefstathiou, Vedvyas Shanbhogue

第 1 章 导言

输入输出内存管理单元 (IOMMU)，有时也称为系统 MMU (SMMU)，是一个系统级内存管理单元 (MMU)，用于将具有直接内存访问功能的输入/输出 (I/O) 设备连接到系统内存。

对于通过 IOMMU 连接到系统的每个 I/O 设备，软件都可以在 IOMMU 上配置设备上下文，将特定的虚拟地址空间和其他针对设备的参数与设备关联起来。通过在 IOMMU 为每个设备提供独立的设备上下文，可以为每个设备单独配置一个操作系统，这个操作系统可以是 Guest OS，也可以是主（主机）操作系统。在设备发起的每一次内存访问中，IOMMU 都会通过某种形式的唯一设备标识符来识别发起访问的设备，然后 IOMMU 会使用该标识符在软件提供的数据结构中找到相应的设备上下文。例如，对于 PCIe [1]，IOMMU 可通过 PCI 总线号（8 位）、设备号（5 位）和功能号（3 位）的 16 位三元组（统称路由标识符或 RID）以及最多 8 位的段号来识别始发设备。本规范将这种唯一的设备标识符称为 device_id，并支持多达 24 位宽的标识符。

层次结构是一种 PCI Express I/O 互连拓扑结构，其中的配置空间地址（即总线/设备/功能编号的元组）是唯一的。在某些情况下，层次结构也称为分段，在 Flit 模式中，分段编号有时包含在功能 ID 中。

某些设备可能支持共享虚拟寻址，即与设备共享进程地址空间的功能。与设备共享进程地址空间可以依靠核心内核内存管理进行 DMA，从而消除应用程序和设备驱动程序的一些复杂性。绑定到设备后，应用程序可指示其在静态或动态分配的缓冲区上执行 DMA。为支持此类寻址，软件可将一个或多个进程上下文配置到设备上下文中。此类设备启动的每次内存访问都会伴随一个唯一的进程标识符，IOMMU 将该标识符与唯一的设备标识符结合使用，以定位软件在设备上下文中配置的适当进程上下文。例如，对于 PCIe，进程上下文可由唯一的 20 位进程地址空间标识符 (PASID) 标识。本规范将此类唯一进程标识符称为 process_id，并支持多达 20 位宽的标识符。

IOMMU 采用两阶段地址转换流程，将 IOVA 转换为 SPA，并对 DMA 实施内存保护。为了执行地址转换和内存保护，IOMMU 在第一阶段和第二阶段地址转换中使用与 CPU 的 MMU 相同的页表格式。使用与 CPU MMU 相同的页表格式，可以消除 DMA 在内存管理方面的一些复杂性。使用相同的格式还允许 CPU MMU 和 IOMMU 同时使用相同的页表。

虽然没有禁用两阶段地址转换的选项，但可以通过将该阶段的虚拟内存方案配置为 Bare（即不执行地址转换或内存保护）来有效禁用任一阶段。

IOMMU 采用的虚拟内存方案可为每个设备单独配置 IOMMU。设备使用 I/O 虚拟地址 (IOVA) 执行 DMA。根据为设备选择的虚拟内存方案，设备使用的 IOVA 可能是 Supervisor 物理地址 (SPA)、Guest 物理地址 (GPA) 或虚拟地址 (VA)。

如果为两个阶段选择的虚拟内存方案都是 Bare，那么 IOVA 就是 SPA。IOMMU 不进行地址转换或保护。

如果为第一阶段选择的虚拟内存方案是 Bare，而第二阶段的方案不是 Bare，那么 IOVA 就是一个 GPA。第一阶段实际上被禁用。第二阶段将 GPA 转换为 SPA，并执行配置的内存保护。当设备控制传递给虚拟机，但虚拟机中的 Guest OS 不使用第一阶段地址转换来进一步限制此类设备的内存访问时，通常会采用这种配置。与 RISC-V Hart 相比，这种配置类似于 RISC-V Hart 上的两级地址转换，G 级处于激活状态，VS 级设置为 Bare。

如果为第一阶段选择的虚拟内存方案不是 Bare，但为第二阶段选择的方案是 Bare，那么 IOVA 就是 VA。第二阶段实际上被禁用。第一阶段将 VA 转换为 SPA，并执行配置的内存保护。当本地操作系统使用 IOMMU 或 Hypervisor 本身保留对设备的控制时，通常会采用这种配置。与 RISC-V 硬件相比，这种配置类似于 RISC-V 硬件上的单级地址转换。

如果为两个阶段选择的虚拟内存方案都不是 Bare，那么 IOVA 就是 VA。两阶段地址转换生效。第一阶段将 VA 转换为 GPA，第二阶段将 GPA 转换为 SPA。每个阶段都执行配置的内存保护。当设备控制传递给虚拟

机，虚拟机中的 Guest OS使用第一阶段地址转换进一步限制此类设备访问的内存以及相关权限和内存保护时，通常会采用这种配置。与 RISC-V 硬件相比，这种配置类似于 RISC-V 硬件上的两级地址转换，G 级和 VS 级都处于激活状态（非裸机）。

IOMMU 中的 DMA 地址转换对 DMA 访问有一定的性能影响，因为使用软件提供的数据结构确定 SPA 所需的时间可能会延长访问时间。CPU MMU 中的类似开销通常是通过使用转换旁路缓冲器（TLB）来缓存这些地址转换，以便在后续访问中重复使用，从而减少转换开销。IOMMU 可以使用类似的地址转换缓存，即 IOMMU 地址转换缓存（IOATC）。当用于地址转换的内存驻留数据结构被修改时，IOMMU 为软件提供了使 IOATC 与之同步的机制。软件可使用软件定义的上下文标识符（称为 Guest 软件上下文标识符 (GSCID)）配置设备上下文，以指示设备集合被分配给同一虚拟机，从而访问共同的虚拟地址空间。软件可使用称为进程软件上下文标识符（PSCID）的软件定义上下文标识符配置进程上下文，以标识共享共同虚拟地址空间的进程集合。IOMMU 可使用 GSCID 和 PSCID 标记 IOATC 中的条目，以避免重复并简化失效操作。

有些设备可能会参与转换过程，并为自己的内存访问提供设备侧 ATC（DevATC）。通过提供 DevATC，设备可以分担转换缓存责任，从而降低 IOATC 中的 "颠簸（thrashing）"概率。DevATC 的大小可由设备决定，以满足其独特的性能要求，设备还可通过预取转换来优化 DMA 延迟。这种机制需要设备和 IOMMU 通过协议密切合作。例如，对于 PCIe，设备可使用地址转换服务（ATS）协议请求将转换内容缓存到 DevATC 中，并使其与软件地址转换数据结构的更新同步。参与地址转换过程的设备还可以使用 I/O 页故障，以避免核心内核内存管理器必须始终驻留设备可能访问的所有物理内存。例如，对于 PCIe，设备可以实现页面请求接口 (PRI)，以便在发现请求转换的页面不可用时，动态请求内存管理器驻留页面。IOMMU 可支持与设备的专用软件接口和协议，以实现 PCIe ATS 和 PCIe PRI 等服务 [1]。

在内置消息信号中断控制器（IMSIC）的系统中，Hypervisor 可对 IOMMU 进行编程，以便将 Guest OS 所控制设备的消息信号中断（MSI）导向 IMSIC 中的 Guest 中断文件。由于来自设备的 MSI 只是简单的内存写入，它们自然会受到 IOMMU 对其他内存写入所适用的相同地址转换。不过，RISC-V 高级中断架构[2]要求 IOMMU 对指向虚拟机的 MSI 进行特殊处理，部分原因是为了简化软件，部分原因是为了允许对内存驻留中断文件提供可选支持。设备上下文由软件配置参数，用于识别对虚拟中断文件的内存访问，并使用设备上下文中由软件配置的 MSI 地址转换表进行转换。

1.1. 术语表（Glossary）

表 1. 术语和定义

Term	定义
AIA	RISC-V 高级中断架构 [2]。
ATS / PCIe ATS	地址转换服务：支持 DevATC 的 PCIe 协议 [1]。
CXL	Compute Express Link 总线标准。
DC / Device Context	一种硬件状态表示法，用于识别设备和分配给该设备的虚拟机。
DDT	设备目录表：使用唯一设备标识符遍历的辐射树结构，用于定位设备上下文结构。
DDI	设备目录索引：唯一设备标识符的子字段，用作叶子或非叶子 DDT 结构的索引。
Device ID	一个长达 24 位的标识号，用于识别 DMA 或中断请求的来源。对于 PCIe 设备，这是路由标识符（RID）[1]。
DevATC	设备上的地址转换缓存。
DMA	直接内存访问。
GPA	Guest物理地址：虚拟机虚拟化物理内存空间中的地址。
GSCID	Guest 软件上下文标识符：软件用于唯一标识分配给虚拟机的设备集合的标识号。IOMMU 可以用 GSCID 标记 IOATC 条目。使用相同 GSCID 编程

Term	定义
	的设备上下文也必须使用相同的第二阶段页表。
Guest	虚拟机中的软件
HPM	硬件性能监控器。
Hypervisor	控制虚拟化的软件实体。
ID	标识符。
IMSIC	接收信息信号的中断控制器。
IOATC	IOMMU 地址转换缓存：IOMMU 中用于缓存地址转换数据结构的缓存。
IOVA	I/O 虚拟地址：设备 DMA 的虚拟地址。
MSI	信息信号中断
OS	操作系统。
PASID	进程地址空间标识符：它标识了进程的地址空间。PASID 值在请求的 PASID TLP 前缀中提供。
PBMT	基于页面的内存类型
PPN	物理页码。
PRI	页面请求接口（Page Request Interface）--一种 PCIe 协议，可让设备请求操作系统内存管理器服务，使页面常驻[1]。
PC	进程上下文。
PCIe	外围元件互连高速总线标准[1]。
PDI	进程目录索引：唯一进程标识符的子字段，用于索引页或非页 PDT 结构。
PDT	进程目录表：使用唯一进程标识符遍历的辐射树数据结构，用于定位进程上下文结构。
PMA	物理内存属性
PMP	物理内存保护
PPN	物理页号。
PRI	页面请求接口 - 一种 PCIe 协议 [1]，使设备能够请求操作系统内存管理器服务来驻留页面。
Process ID	用于识别进程上下文的标识号，最多 20 位。对于 PCIe 设备，这是 PASID [1]。
PSCID	进程软件上下文标识符：软件用于识别唯一地址空间的标识号。 IOMMU 可以用 PSCID 标记 IOATC 条目。
PT	页表。
PTE	页表表项。页表中的页条目或非页条目。
Reserved	为将来使用而保留的寄存器或数据结构字段。数据结构中的保留字段必须由软件设置为 0。软件必须忽略寄存器中的保留字段，并在向同一寄存器中的其他字段写入值时保留这些字段中的值。
RID / PCIe RID	PCIe 路由标识符 [1]。
RO	只读 - 寄存器位为只读，软件无法更改。在明确定义的情况下，这些位用于反映硬件状态的变化，因此在运行时可以观察到位值的变化。 如果没有实现设置这些位的可选功能，则必须将这些位硬连接为零
RW	读写器 - 寄存器位为读写器，允许软件设置或清除至所需状态。 如果与位相关的可选功能没有实现，则允许将这些位硬连接为零。
RW1C	写 1 清除状态 - 读取寄存器位时，寄存器位会显示状态。置位表示状态事件，写入 1b 则清除该事件。向 RW1C 位写入 0b 无影响。 如果设置该位的可选功能没有实现，则该位必须是只读的，并硬连线为零

Term	定义
RW1S	读写-1-设置--寄存器位在读取时指示状态。该位可通过写入 1b 设置。向 RW1S 位写入 0b 无影响。 如果没有实现引入该位的可选功能，则该位必须是只读的，并硬连接为零
SOC	片上系统，又称片上系统和片上系统。
SPA	Supervisor 物理地址：用于访问内存和内存映射资源的物理地址。
TLP	事务层数据包。
VA	虚拟地址。
VM	虚拟机：真实计算机系统的高效、隔离的复制品。在本规范中，它指的是支持 Hypervisor 扩展的 RISC-V hart 在虚拟化模式设置为 1 的情况下执行时可访问的资源和状态集合。
VMM	虚拟机监控器。也称为 Hypervisor 。
VS	虚拟监管员：虚拟化模式下的 Supervisor 权限。
WARL	写入任意值，读取合法值：寄存器字段的属性，只针对部分位编码定义，但允许写入任意值，同时保证读取时返回合法值。
WPRI	写入保留值，读取忽略值：为将来使用而保留的寄存器字段属性。

1.2. 使用模式（Usage models）

1.2.1. 非虚拟化操作系统（Non-virtualized OS）

非虚拟化操作系统可使用 IOMMU 实现以下重要的系统级功能：

1. 保护操作系统免受错误设备的不良内存访问
2. 在 64 位环境中支持 32 位设备（避免弹跳缓冲区）
3. 支持将连续虚拟地址映射到底层分片物理地址（避免使用分散/收集（scatter/gather）列表）
4. 支持共享虚拟寻址

在没有 IOMMU 的情况下，设备可以访问任何内存，如特权内存，并导致恶意或意外损坏。这可能是硬件漏洞、设备驱动程序漏洞或恶意软件/硬件造成的。

IOMMU 为操作系统提供了一种机制，通过限制设备可访问的内存来防止这种意外损坏。如图 1 所示，操作系统可通过页表配置 IOMMU，以转换 IOVA，从而将可访问的地址限制在页表允许的范围内。

传统的 32 位设备无法访问超过 4 GiB 的内存。IOMMU 通过其地址重映射功能，为设备直接访问系统中的任何地址（具有适当的访问权限）提供了一种简单的机制。如果没有 IOMMU，操作系统就必须通过分配在 4 GiB 以下内存中的缓冲区（也称为反弹缓冲区）来复制数据。在这种情况下，IOMMU 可以提高系统性能。

IOMMU 可用于执行分散/聚集（scatter/gather）DMA，因为它允许为 I/O 分配较大的内存区域，而无需所有内存都是连续的。一个连续的虚拟地址范围可以映射到这些零散的物理地址，并用虚拟地址范围对设备进行编程。

IOMMU 可用于支持共享虚拟寻址，即与设备共享进程地址空间。用于 DMA 的虚拟地址由 IOMMU 转换为 SPA。

当 IOMMU 被非虚拟化操作系统使用时，第一阶段足以提供所需的地址转换和保护功能，第二阶段可设置为 "Bare"。

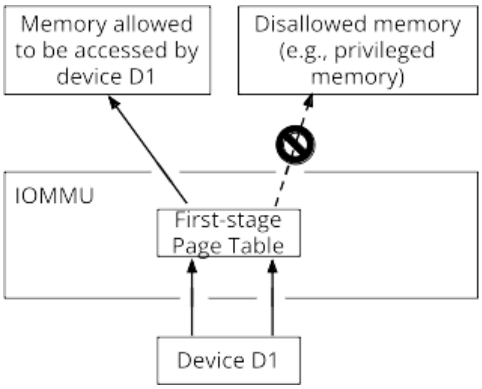


图 1. 非虚拟化操作系统中的设备隔离

1.2.2. Hypervisor

IOMMU 使虚拟机中运行的 Guest OS 能够直接控制 I/O 设备，只需 Hypervisor 进行最少的干预。

直接控制设备的 Guest OS 将使用 Guest 物理地址对设备进行编程，因为操作系统只知道这些地址。然后，当设备使用这些地址执行内存访问时，IOMMU就会引用 Hypervisor 提供的地址转换数据结构，负责将这些 Guest 物理地址转换为 Hypervisor 物理地址。

图 2 展示了这一概念。设备 D1 直接分配给 VM-1，设备 D2 直接分配给 VM-2。VMM 为每个设备配置一个第二阶段页表，并将 D1 可访问的内存限制为 VM-1 关联内存，将 D2 可访问的内存限制为 VM-2 关联内存。

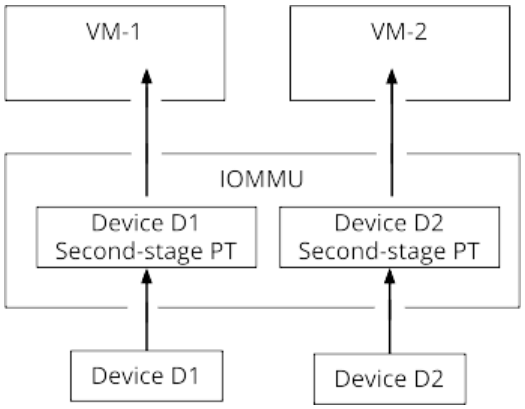


图 2. 实现直接设备分配的 DMA 转换

为了处理由 Guest OS 控制的设备发出的 MSI，Hypervisor 会配置 IOMMU，将这些 MSI 重定向到 IMSIC 中的 Guest 中断文件（见图 3）或内存驻留中断文件。IOMMU 负责使用 Hypervisor 提供的 MSI 地址转换数据结构来执行 MSI 重定向。由于每个中断文件，无论是真实的还是虚拟的，都占用一个自然对齐的 4-KiB 页面地址空间，因此所需的地址转换是从虚拟（Guest）页面地址到物理页面地址，与常规 RISC-V 基于页面的地址转换所支持的相同。

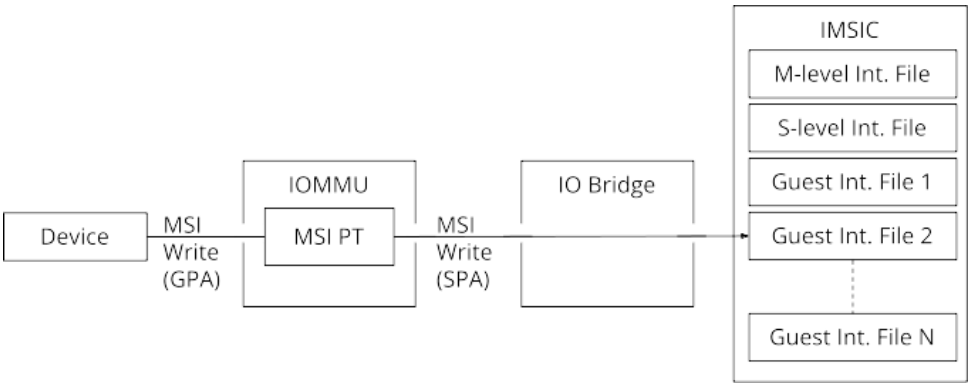


图 3.MSI 地址转换，将Guest编程的 MSI 直接转换为 IMSIC Guest 中断文件

1.2.3. Guest OS

Hypervisor 可以通过硬件仿真或让 Guest OS 使用与 Hypervisor 的软件接口（也称为准虚拟化）来提供虚拟 IOMMU 设施。Guest OS可以使用虚拟 IOMMU 提供的设施，通过使用它所控制的第一阶段页表，获得与非虚拟化操作系统相同的好处。Hypervisor 会建立一个由其控制的第二阶段页表，以虚拟化虚拟机的地址空间，并包含从传递给虚拟机的设备到与虚拟机相关的内存的内存访问。

启用两阶段地址转换后，IOVA 首先使用 Guest OS 管理的第一阶段页表转换为 GPA，然后使用 Hypervisor 管理的第二阶段页表将 GPA 转换为 SPA。

图 4 说明了这一概念。

IOMMU 配置为使用设备 D1 的第一阶段和第二阶段页表执行地址转换。第二阶段通常由 Hypervisor 用于将 GPA 转换为 SPA，并将设备 D1 限制为与 VM-1 相关的内存。第一阶段通常由 Guest OS 配置，将 VA 转换为 GPA，并将设备 D1 的访问限制在 VM-1 内存的子集上。

对于设备 D2，只有第二阶段处于激活状态，第一阶段设置为 "无"。

主机操作系统或 Hypervisor 也可以保留一个设备（如 D3）供自己使用。第一阶段足以为设备 D3 提供所需的地址转换和保护功能，第二阶段设置为 "无"。

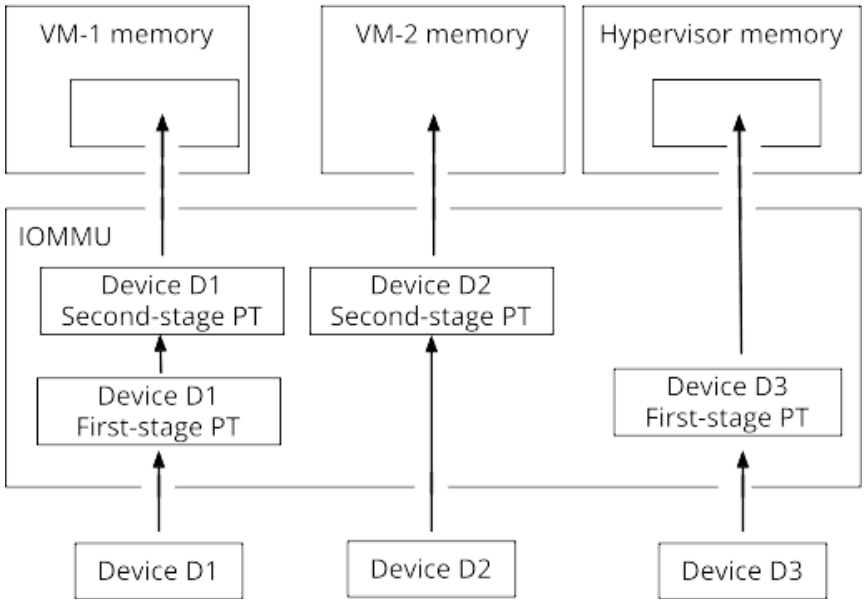


图 4.Guest OS IOMMU 中的地址转换

1.3. 位置和数据流（Placement and data flow）

图 5 显示了带有 RISC-V 硬件的典型片上系统 (SOC) 的示例。该 SOC 集成了内存控制器和多个 IO 设备。该 SOC 还集成了两个 IOMMU 实例。设备可以直接连接到 IO Bridge 和系统互连，也可以在需要将 IO 协议事务转换为系统互连事务时通过 Root Port 连接。以 PCIe [1] 为例， Root Port 是一个 PCIe 端口，它通过相关的虚拟 PCI-PCI Bridge 映射分层结构的一部分，并将 PCIe IO 协议事务映射到系统互连事务。

第一个 IOMMU 实例 IOMMU 0（与 IO Bridge 0 相关联）将一个 Root Port 连接到系统结构/互连。一个或多个终端设备通过该 Root Port 与 SoC 连接。在 PCIe 的情况下， Root Port 包含一个到 IOMMU 的 ATS 接口，用于 IOMMU 支持 PCIe ATS 协议。示例中的端点设备带有一个设备侧 ATC（DevATC），用于保存设备通过 PCIe ATS 协议 [1] 从 IOMMU 0 获得的转换。

当不需要使用 Root Port 进行这种 IO 协议到系统结构协议的转换时，设备可直接与系统结构连接。第二个 IOMMU 实例 IOMMU 1（与 IO Bridge 1 相关联）说明了在不使用 Root Port 的情况下将设备（IO 设备 A 和 B）连接到系统架构的情况。

IO Bridge 位于设备和系统互连之间，用于处理 DMA 事务。IO 设备可使用 IO 虚拟地址（VA、GVA 或 GPA）执行 DMA 事务。IO Bridge 调用相关的 IOMMU，将 IOVA 转换为监控器物理地址 (SPA)。

出站事务不调用 IOMMU。

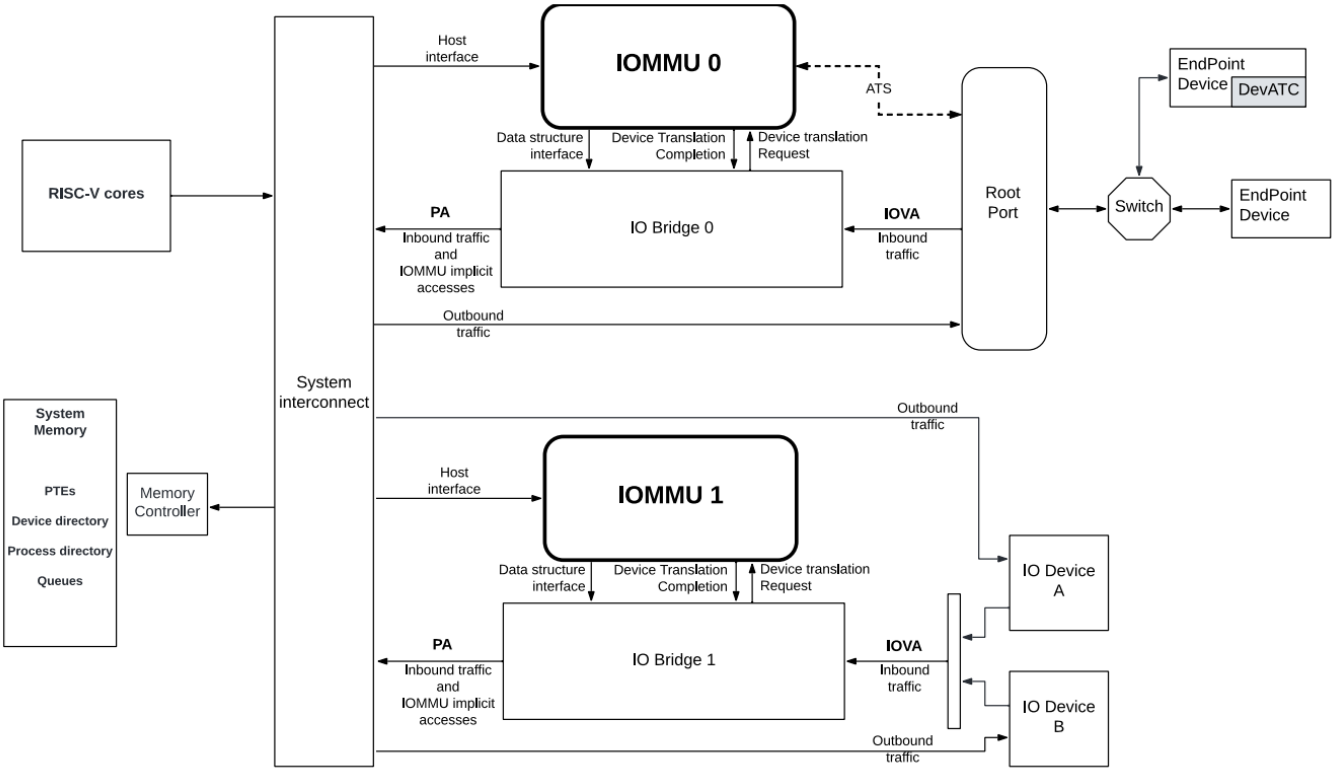


图 5.SoC 中集成 IOMMU 的示例。

IOMMU 由 IO Bridge 调用，用于地址转换和保护入站事务。IOMMU 不处理与入站事务相关的数据。IOMMU 的行为类似于 IO Bridge 的旁路 IP，有多个接口（见图 6）：

- 主机接口：它是连接 IOMMU 的接口，供 Harts 访问其内存映射寄存器，并执行全局配置和/或维护操作。
- 设备转换请求接口：这是一个从 IO Bridge 接收转换请求的接口。在此接口上，IO Bridge 提供有关请求的信息，如
  a. 与事务相关的硬件标识-- device_id 和 process_id（如适用）及其有效性。IOMMU 使用硬件标识检索上下文信息，以执行请求的地址转换。
  b. IOVA 和事务类型（转换或非转换）。
  c. 请求是读取、写入、执行还是原子操作。
     i. 请求的执行必须与请求明确关联（例如，使用 PCIe PASID）。如果没有明确请求，默认值必须为 0。
  d. 与请求相关联的权限模式。如果权限模式未与请求明确关联（如使用 PCIe PASID），则默认权限模式必须为用户模式。对于没有 process_id 的请求，权限模式必须为用户模式。
  e. 请求访问的字节数。
  f. IO Bridge 还可以提供一些额外的不透明信息（如标签），这些信息不被 IOMMU 解释，而是与 IOMMU 对 IO Bridge 的响应一起返回。由于 IOMMU 可以不按顺序完成转换请求，因此 IO Bridge

可以利用这些信息将转换完成情况与之前的请求联系起来。

- 数据结构接口：它被 IOMMU 用于隐式访问内存。它是 IO Bridge 的请求接口，用于从主内存中获取所需的数据结构。该接口用于访问
  a. 设备和进程目录，以获取上下文信息和转换规则。
  b. 用于转换 IOVA 的第一阶段和/或第二阶段页表项。
  c. 用于连接软件的内存队列（命令队列、故障队列和页面请求队列）。
- 设备转换完成接口：这是一个接口，用于提供 IOMMU 对先前请求的地址转换的完成响应。完成接口可提供以下信息
  a. 请求的状态，表示请求是成功完成还是出现故障。
  b. 如果请求已成功完成，则显示 Supervisor 物理地址 (SPA)。
  c. 与请求相关的不透明信息（如标签）（如适用）。
  d. 如果支持 Svpbmt，则从 IOMMU 地址转换页表中获取基于页面的内存类型 (PBMT)。IOMMU 提供在第一阶段和第二阶段页表表项之间解析的基于页面的内存类型。
- ATS 接口：如果 IOMMU 支持可选的 PCIe ATS 功能，则 ATS 接口用于通过 PCIe Root Port 与具有 ATS 功能的 EP 通信。该接口用于
  a. 接收来自 EP 的 ATS 转换请求，并将完成结果返回给 EP。 Root Port 可指示发出请求的 EP 是 CXL 类型 1 还是类型 2 设备。
  b. 向 EP 发送 ATS "Invalidation Request "信息，并从 EP 接收 "Invalidation Completion "信息。
  c. 接收来自 EP 的 "Page Request "和 "Stop Marker "信息，并向 EP 发送 "Page Request Group Response "信息。

与在内存驻留中断文件（MRIF）（参见 RISC-V 高级中断体系结构 [2]）中记录传入 MSI 相关的接口取决于具体的实现。IOMMU 和 IO Bridge 之间在 MRIF 中记录传入 MSI 和生成相关*通知* MSI 的责任划分是针对具体实现的。

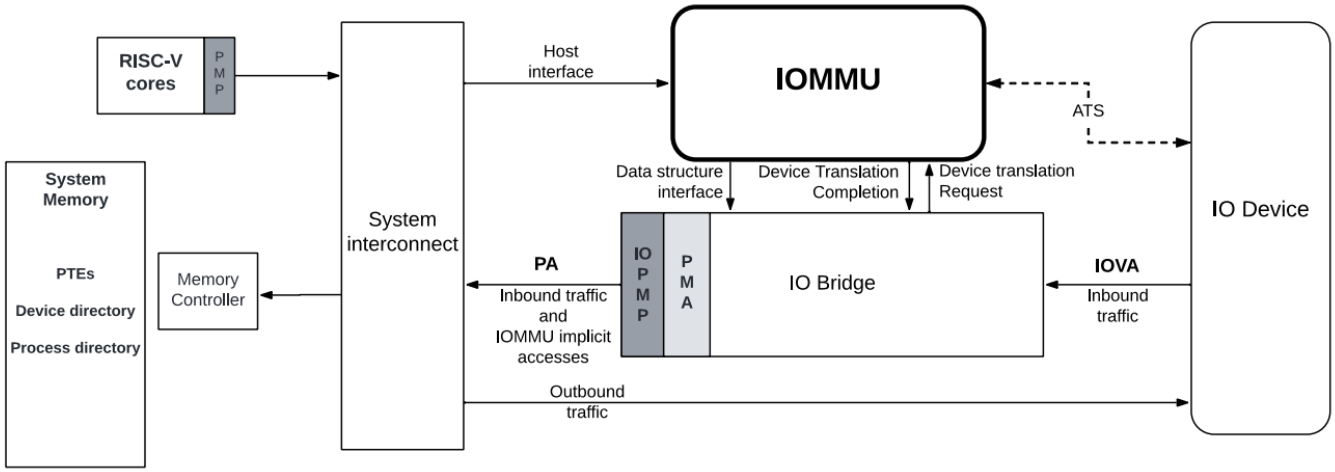


图 6.IOMMU 接口。

与 RISC-V harts 类似，即使 IOMMU 处于旁路（Bare 模式），也必须对所有入站 IO 事务完成物理内存属性 (PMA) 和物理内存保护 (PMP) 检查。PMA 和 PMP 检查器的位置和集成取决于平台选择。

PMA 和 PMP 校验器位于 IOMMU 之外。上面的示例显示它们位于 IO Bridge 中。

IOMMU 本身通过数据结构接口进行的隐式访问由 PMA 检查器进行检查。PMA 与特定物理平台的组织结构紧密相关，许多细节本质上是特定于平台的。

IOMMU 使用数据结构接口执行的内存访问一般无需与设备启动的内存访问排序。

IOMMU 可在数据结构接口上生成隐式内存访问，以访问执行地址转换所需的数据结构。此类访问不得被原始设备启动的内存访问阻塞。

IO Bridge 可在数据结构接口上执行内存访问排序，以满足 IO Bridge 和系统互连所定义的必要危险检查和其他规则。

IOMMU 将已解析的 PBMT（PMA、IO、NC）以及设备转换完成接口上的转换地址提供给 IO Bridge 。IO 桥中的 PMA 检查器可使用所提供的 PBMT 来覆盖相关内存页的 PMA。

PMP 校验器可使用总线访问启动器的硬件 ID 来确定物理内存访问权限。由于 IOMMU 本身是隐式访问的总线访问启动器，因此 PMP 检查程序可使用 IOMMU 硬件 ID 选择适当的访问控制规则。

IOMMU 不验证 IO Bridge 提供的硬件 ID 的真实性。

IO Bridge 和/或 Root Port 必须包含验证硬件 ID 的适当机制。在某些 SOC 中，由于设备已集成到 SOC 中，且其 ID 是不可变的，因此可以轻松实现这一点。例如，对于 PCIe，可使用 PCIe 定义的访问控制服务 (ACS) 源验证功能来验证硬件 ID。IO Bridge 中的其他特定实施方法也可用于执行此类验证。

1.4. IOMMU 的特点（IOMMU features）

1.0 版 RISC-V IOMMU 规范支持以下功能：

- 基于内存的设备上下文，用于定位参数和地址转换结构。设备上下文使用硬件提供的唯一 device_id 定位。支持的 device_id 宽度可达 24 位。
- 基于内存的进程上下文，使用硬件提供的唯一 process_id 定位参数和地址转换结构。支持的 process_id 最多为 20 位。
- 16 位 GSCID 和 20 位 PSCID。
- 两阶段地址转换
- RISC-V 特权规范[3]规定的基于页的虚拟内存系统，允许软件灵活地为 CPU MMU 和 IOMMU 使用通用页表，或为 IOMMU 使用单独的页表。
- 最多 57 位虚拟地址宽度、56 位系统物理地址和 59 位Guest物理地址宽度。
- 硬件更新 PTE Accessed 和 Dirty 位。
- 使用 RISC-V 高级中断架构 [2] 指定的 MSI 页表，识别虚拟中断文件的内存访问和 MSI 地址转换。
- Svnapot 和 Svpbmt 扩展。
- PCIe ATS 和 PRI 服务 [1]。支持根据转换请求将 IOVA 转换为 GPA，而不是 SPA。
- 硬件性能监控器（HPM）。
- MSI 和有线信号中断请求软件提供服务。
- 用于软件请求地址转换的寄存器接口，以支持调试。

IOMMU 支持的功能可通过第 5.3 节中的 capabilities 寄存器发现。

第 2 章 数据结构（Data Structures）

IOMMU 使用称为设备上下文（DC）的数据结构将设备与地址空间关联起来，并保存 IOMMU 用于执行地址转换的其他设备参数。使用 device_id 遍历的 radix-tree 数据结构称为设备目录表（DDT），用于定位 DC。

当设备的控制权转交给 Guest OS 时，设备使用的地址空间可能需要第二阶段的地址转换和保护。Guest OS 可以选择提供第一阶段页表，用于将 Guest OS 控制的设备使用的 IOVA 转换为 GPA。当不需要使用第一阶段时，可以通过选择第一阶段地址转换方案为 "Bare" 来有效禁用第一阶段。第二阶段用于将 GPA 转换为 SPA。

当 Hypervisor 或主机操作系统本身保留对设备的控制时，只有第一阶段足以执行必要的地址转换和保护；通过将第二阶段地址转换方案编程为 "Bare"，可以有效地禁用设备的第二阶段方案。

当第二阶段地址转换不是 Bare 时，DC 会保存 root 第二阶段页表的 PPN、Guest 软件上下文 ID (GSCID)（便于在每个虚拟机的基础上使缓存的地址转换无效）以及第二阶段地址转换方案。

某些设备支持多个进程上下文，其中每个上下文都可能与不同的进程相关联，从而与不同的虚拟地址空间相关联。此类设备中的上下文可配置一个 process_id ，用于标识地址空间。在进行内存访问时，此类设备会将 process_id 与 device_id 一起发出信号，以识别所访问的地址空间。这种设备的一个例子是支持多进程上下文的 GPU，其中每个上下文都与不同的用户进程相关联，这样 GPU 就可以使用用户进程本身提供的虚拟地址访问内存。为支持选择与 process_id 相关联的地址空间，DC 保存 root 进程目录表 (PDT) 的 PPN，这是一个 radix-tree 数据结构，使用 process_id 的字段进行索引，以定位称为进程上下文 (PC) 的数据结构。

当 PDT 处于活动状态时，第一阶段地址转换的控制权被保留在 (PC) 中。

当 PDT 未激活时，第一阶段地址转换的控制权由 DC 本身掌握。

第一阶段地址转换控制包括：第一阶段 root 页表的 PPN；进程软件上下文识别码（PSCID），该识别码有助于在每个地址空间基础上使缓存的地址转换无效；以及第一阶段地址转换方案。

要处理来自由 Guest OS 控制的设备的 MSI，IOMMU 必须能够将这些 MSI 重定向到 IMSIC 中的 Guest 中断文件。由于来自设备的 MSI 只是简单的内存写入，它们自然会受到与 IOMMU 应用于其他内存写入相同的地址转换。不过，IOMMU 架构可对指向虚拟机的 MSI 进行特殊处理，部分原因是为了简化软件，部分原因是为了对内存驻留中断文件提供可选支持。为支持这一功能，该架构在设备上下文中添加了 MSI 地址掩码和地址模式，共同用于识别 Guest 物理地址空间中作为 MSI 目的地的页面；以及 MSI 页表的实际物理地址，用于控制来自设备的 MSI 的 translation 和/或 conversion。高级中断架构规范规定了 IOMMU 对虚拟机 MSI 的支持。

DC 还对允许设备生成的事务类型进行控制。此类控制的一个例子是，是否允许设备使用 PCIe 定义的地址转换服务（ATS）[1]。

设备上下文结构支持两种格式：

- **基本格式** – 大小为 32 字节，在 IOMMU 不支持第 2.3.3 节规定的 MSI 特殊处理时使用。
- **扩展格式** - 大小为 64 字节，扩展了基本格式 DC，增加了用于转换 MSI 的字段，详见第 2.3.3 节。

如果 capabilities.MSI_FLAT 为 1，则使用扩展格式，否则使用基本格式。

用于定位 DC 的 DDT 可配置为 1 级、2 级或 3 级半径树，具体取决于支持的 device_id 最大宽度。用于获取设备目录索引（DDI）以遍历 DDT 树的 device_id 分区如下：

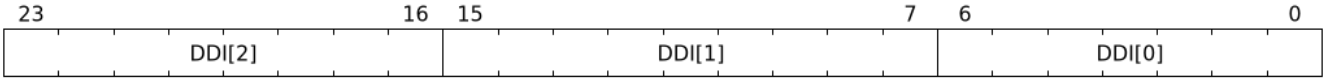


图 7. 基本格式 device_id 分区

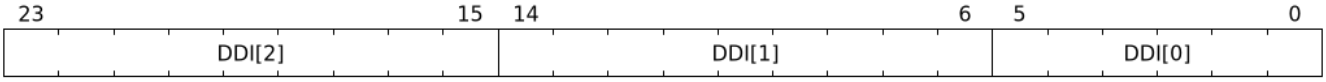


图 8. 扩展格式 device_id 分区

根据设备支持的进程标识（process_id）的最大宽度，PDT 可被配置为 1、2 或 3 级radix-tree。为获得进程目录索引（PDI）以遍历 PDT 树，对 process_id 进行的划分如下：

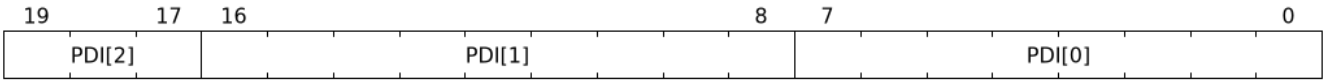


图 9. PDT 径向树遍历的 process_id 分区

process_id 分区的设计是为每个进程目录表最多占用 4 KiB（一页）内存。使用 20 位宽 process_id 时，表的根目录不会被完全填充。我们考虑过让根表占用 32 KiB 的方案，但没有采纳，因为这些表是在运行时分配的，而大于一页的连续内存分配可能会给 Guest 和 Hypervisor 内存分配器带来压力。

所有 RISC-V IOMMU 实现都必须支持位于主内存中的 DDT 和 PDT。本规范不要求但不禁止在 I/O 内存中支持数据结构。

2.1. 设备目录表 (DDT)

DDT 是一棵 1、2 或 3 级的 radix-tree，使用 device_id 的设备目录索引 (DDI) 位来定位 DC。

下图说明了 DDT radix-tree。根设备目录表的 PPN 保存在内存映射寄存器中，该寄存器称为设备目录表指针（ddtp）。

每个有效的非叶子（NL）条目大小为 8 字节，保存下一个设备目录表的 PPN。有效的叶设备目录表表项保存设备上下文（DC）。

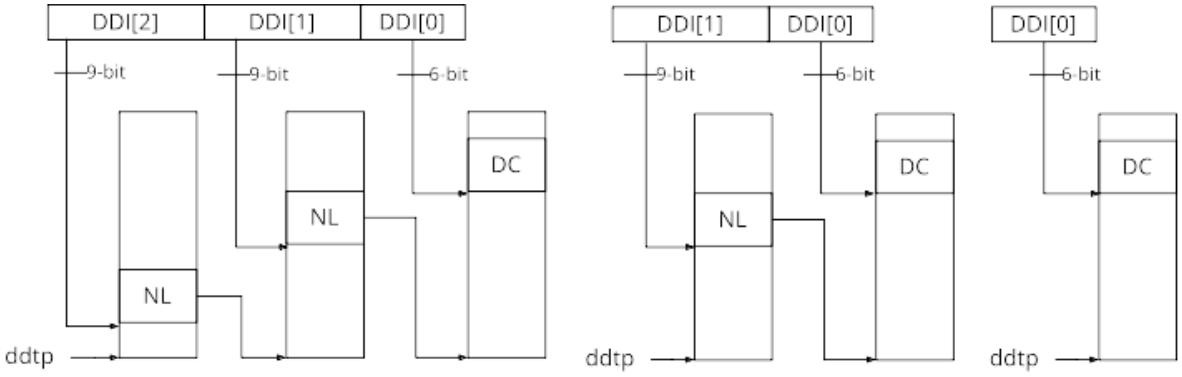


图 10. 带扩展格式 DC 的三级、二级和单级设备目录

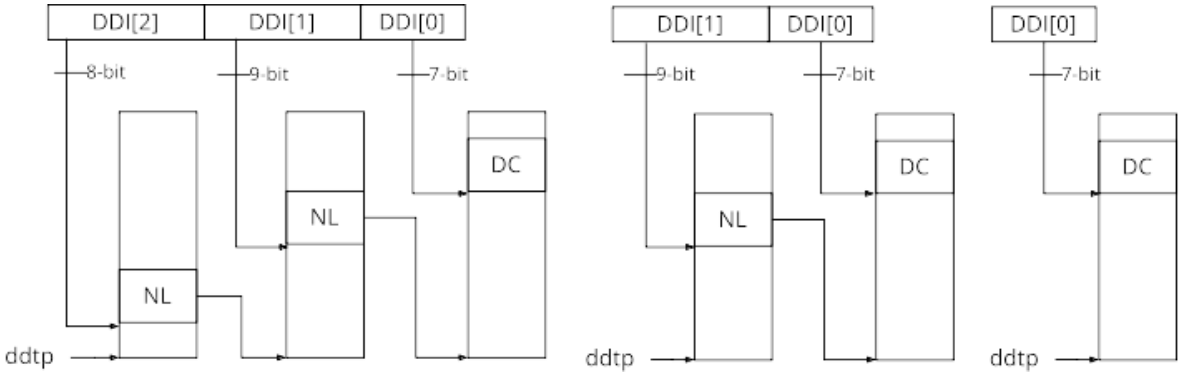


图 11. 带基本格式 DC 的三级、二级和单级设备目录

2.1.1. 非叶片 DDT 表项（Non-leaf DDT entry）

有效（V==1）的非叶 DDT 表项提供下一级 DDT 的 PPN。

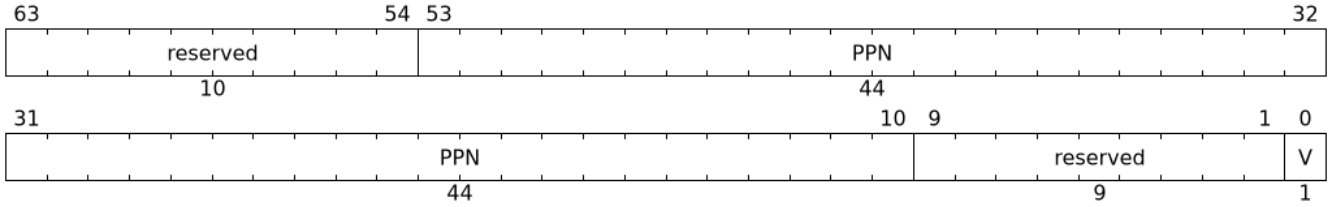


图 12. 非叶子设备目录表表项

2.1.2. 叶片 DDT 表项（Leaf DDT entry）

DDT 叶页由 DDI[0] 索引，保存设备上下文（DC）。在基本格式中，DC 为 32 字节。在扩展格式中，DC 为 64 字节。

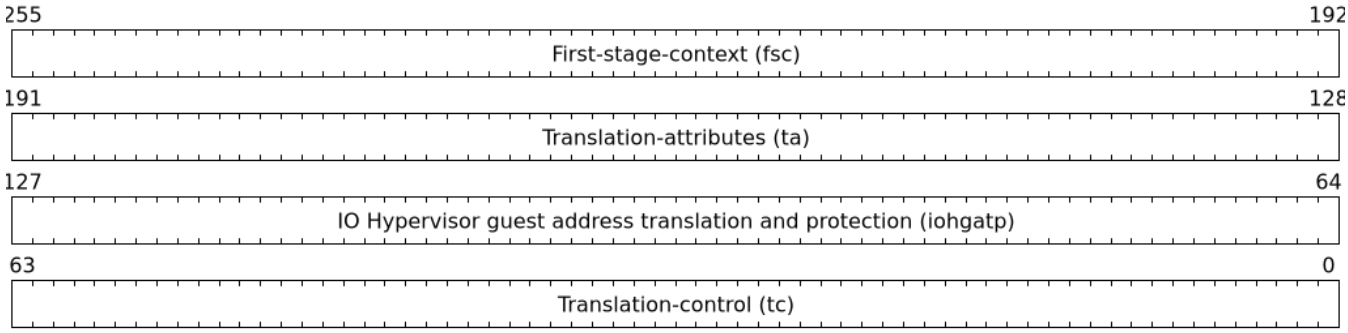


图 13. 基本格式设备上下文

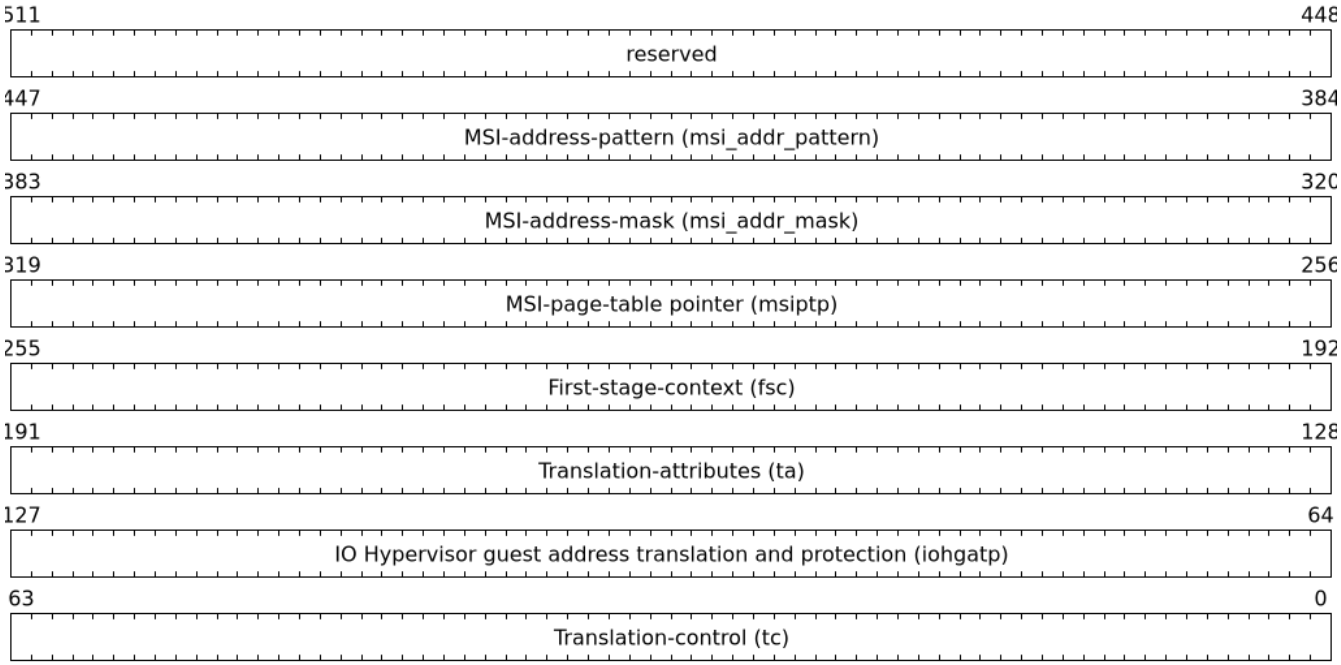


图 14. 扩展格式设备上下文

DC 在基本格式下被解释为 4 个 64 位双字，在扩展格式下被解释为 8 个 64 位双字。内存中每个双字的字节顺序（小端或大端）由 fctl.BE（第 5.4 节）决定。IOMMU 可以以任何顺序读取 DC 字段。

2.1.3. 设备上下文字段（Device-context fields）

转换控制 (tc)

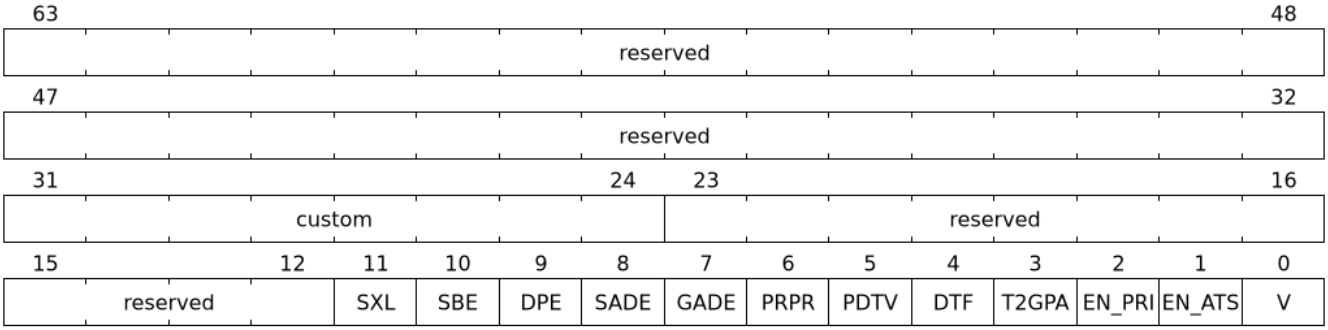


图 15. 转换控制 (tc) 字段

如果 V 位为 1，则 DC 有效；如果 V 位为 0，则 DC 中的所有其他位都无关紧要，可由软件自由使用。

如果 IOMMU 支持 PCIe ATS 规范 [1]（参见功能寄存器），则 EN_ATS 位用于启用 ATS 事务处理。如果 EN_ATS 设置为 1，则 IOMMU 支持以下入站事务；否则，这些事务将被视为不支持的请求。

- 已转换的读取执行事务
- 已转换的读取事务
- 已转换的写入/AMO 事务
- PCIe ATS 转换请求
- PCIe ATS 失效完成信息

如果 EN_ATS 位为 1 且 T2GPA 位设置为 1，IOMMU 将执行两阶段地址转换，以确定在完成来自设备的 PCIe ATS 转换请求时提供的转换权限和大小。不过，IOMMU 会返回一个 GPA，而不是 SPA，作为响应中

IOVA 的转换。在这种操作模式下，设备中的 ATC 会缓存 GPA 作为 IOVA 的转换，并在随后的已转换内存访问事务中使用 GPA 作为地址。通常，已转换请求使用 SPA，无需 IOMMU 进一步转换。但是，当 T2GPA 为 1 时，来自设备的已转换请求使用 GPA，并由 IOMMU 使用第二阶段页表转换为 SPA。T2GPA 控制使 Hypervisor 能够包含来自设备的 DMA，即使设备滥用 ATS 功能并试图访问与虚拟机无关的内存。

当启用 T2GPA 时，响应 PCIe ATS 转换请求而提供给设备的地址不能被连接设备与其他对等设备和主机的 I/O 结构（如 PCI 交换机）直接路由。当支持设备内的点对点事务（如设备功能之间的事务）时，此类地址也无法在设备内路由。

将 T2GPA 配置为 1 的虚拟机 Hypervisor 必须通过特定于协议的方式确保经转换的访问通过主机路由，以便 IOMMU 可以转换 GPA，然后根据 PA 将事务路由至内存或点对点设备。以 PCIe 为例，必须将访问控制服务 (ACS) 配置为始终将点对点 (P2P) 请求重定向到主机上游。

使用设置为 1 的 T2GPA 时，可能无法与通过响应 PCIe ATS 转换请求时返回的转换地址标记缓存的设备兼容。

作为将 T2GPA 设置为 1 的替代方案，如果设备支持身份验证协议，Hypervisor 可与设备建立信任关系。例如，对于 PCIe，PCIe 组件测量和验证（CMA）功能提供了一种机制，用于验证设备的配置、固件/可执行文件（测量）和硬件身份（验证），以建立这种信任关系。

如果 EN_PRI 位为 0，则来自设备的 PCIe "Page Request" 报文为无效请求。从设备接收到的 "Page Request" 报文会以 "Page Request Group Response" 报文的形式做出响应。通常情况下，由软件处理程序生成该响应报文。但在某些情况下，IOMMU 本身也会生成响应。对于 IOMMU 生成的 "Page Request Group Response" 报文，当 PRG-response-PASID-required (PRPR) 位设置为 1 时，表示如果相关的 "Page Request" 有 PASID，则 IOMMU 响应报文应包含 PASID。

支持 PASID 并将 "PRG Response PASID Required" 能力位设置为 1 的 Function，会期望 "Page Request Group Response" 报文包含 PASID，前提是相关的 "Page Request" 报文具有 PASID。如果功能位为 0，则函数不希望任何 "Page Request Group Response" 报文包含 PASID，如果函数收到带有 PASID 的响应，其行为也未定义。PRPR 位应配置为 "PRG Response PASID Required" 能力位中的值。

将禁用转换故障（disable-translation-fault，DTF）位设置为 1 ，将禁止报告在地址转换过程中遇到的故障。将 DTF 设置为 1 并不会禁止针对故障事务向设备生成错误响应。将 DTF 设置为 1 不会禁止 IOMMU 报告与地址转换过程无关的故障。表 11 列出了 DTF 为 1 时不报告的故障。

当 Hypervisor 识别到可能导致大量错误（如虚拟机异常终止）的情况时，可将 DTF 设为 1 以禁用故障报告。

DC.fsc 字段保存第一阶段转换的上下文。如果 PDTV 位为 1，DC.fsc 字段将保存进程目录表指针 (pdtp)。如果 PDTV 位为 0，则 DC.fsc 字段保存 iosatp。

当 DC 与支持多进程上下文的设备关联时，PDTV 位预计将被设置为 1，从而在内存访问时生成有效的 process_id 。例如，对于 PCIe，如果请求具有 PASID，则 PASID 将被用作 process_id 。

当 PDTV 为 1 时，DPE 位可设置为 1，以便在转换没有有效 process_id 的请求时，将 0 作为 process_id 的默认值。当 PDTV 为 0 时，DPE 位将保留给将来的标准扩展使用。

如果 capabilities.AMO_HWAD 为 1，则 IOMMU 支持将 GADE 和 SADE 位设置为 1。如果 capabilities.AMO_HWAD 为 0，这些位被保留。

如果 GADE 为 1，IOMMU 会原子更新第二阶段 PTE 中的 A 和 D 位。如果 GADE 为 0，则在 A 位为 0 或内存访问为存储且 D 位为 0 的情况下，IOMMU 会导致与原始访问类型相对应的 Guest 页故障。

如果 SADE 为 1，IOMMU 会原子式更新第一阶段 PTE 中的 A 和 D 位。如果 SADE 为 0，则在 A 位为 0 或内存访问为存储且 D 位为 0 的情况下，IOMMU 会导致与原始访问类型相对应的页面故障。

如果 SBE 为 0，则对 PDT 表项和第一阶段 PTE 的隐式内存访问为小端（little-endian），否则为大端（big-endian）。SBE 的支持值与 fctl.BE 字段相同。

SXL 字段控制表 3 中定义的受支持的分页虚拟内存方案。如果 fctl.GXL 为 1，则 SXL 字段必须为 1；否则，SXL 字段的合法值与 fctl.GXL 字段的合法值相同。

当 SXL 为 1 时，适用以下规则：

- 如果第一阶段不是 Bare，那么如果 IOVA 的第 31 位以外的位设置为 1，则会出现与原始访问类型相对应的页面故障。
- 如果第二阶段不是 Bare，那么如果传入的 GPA 将第 33 位以外的位设置为 1，则会出现与原始访问类型相对应的 Guest 页故障。

IO Hypervisor Guest 地址转换和保护（iohgatp）

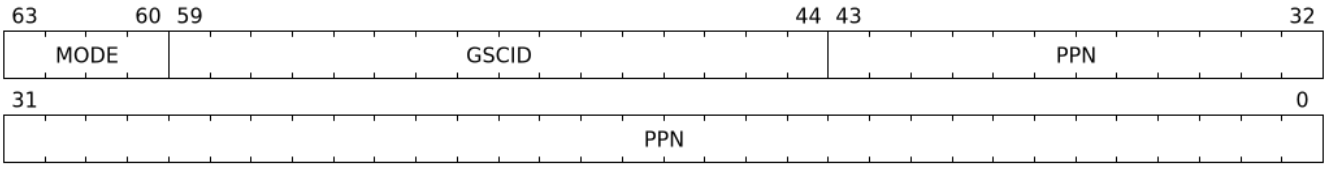


图 16.IO Hypervisor Guest地址转换和保护（iohgatp）字段

iohgatp 字段保存 root 第二阶段页表的 PPN 和由 Guest 软件上下文 ID（GSCID）标识的虚拟机，以便于在每个虚拟机基础上进行地址转换。如果多个设备与虚拟机相关联，并具有共同的第二阶段页表，则 Hypervisor 应在每个 iohgatp 中编入相同的 GSCID。MODE 字段用于选择第二阶段地址转换方案。

第二阶段页表格式由特权规范定义。fctl.GXL 字段控制表 2 中定义的 Guest 物理地址所支持的地址转换方案。

iohgatp MODE 字段用于标识分页虚拟内存方案，其编码如下：

表 2.iohgatp.MODE 字段的编码方式

fctl.GXL=0		
数值	名称	说明
0	Bare	没有转换或保护。
1-7	-	保留为标准用途。
8	Sv39x4	基于页面的 41 位虚拟寻址（Sv39 的 2 位扩展）。
9	Sv48x4	基于页面的 50 位虚拟寻址（Sv48 的 2 位扩展）。
10	Sv57x4	基于页面的 59 位虚拟寻址（Sv57 的 2 位扩展）。
11-15	-	保留为标准用途。
fctl.GXL=1		
数值	名称	说明
0	Bare	没有转换或保护。

1-7	-	保留为标准用途。
8	Sv32x4	基于页面的 34 位虚拟寻址（Sv32 的 2 位扩展）。
9-15	-	保留为标准用途。

IOMMU 不需要支持所有已定义的 iohgatp 模式设置。IOMMU 只需支持集成到系统中的 Hart MMU 也支持的模式或其子集。

由 iohgatp.PPN 确定的根页表为 16 KiB，必须与 16 KiB 边界对齐。

iohgatp 的 GSCID 字段用于标识地址空间。如果在两个 DC 中配置了相同的 GSCID，而两个 DC 所引用的第二阶段页表并不相同，那么 IOMMU 就无法预测是使用第一个页表还是第二个页表中的 PTE。这些是唯一的预期行为。

转换属性 (ta)

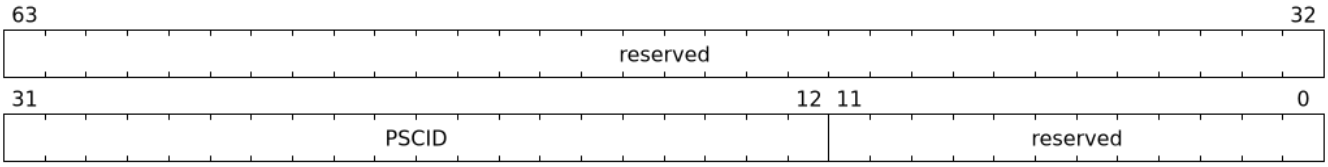


图 17. 转换属性 (ta) 字段

ta 的 PSCID 字段提供了进程软件上下文 ID，用于标识进程的地址空间。PSCID 有助于在每个地址空间基础上建立地址转换栅栏。如果 DC.tc.PDTV 为 0 且 iosatp.MODE 字段不是 Bare，ta 中的 PSCID 字段将用作地址空间 ID。如果 DC.tc.PDTV 为 1，ta 中的 PSCID 字段将被忽略。

第一阶段上下文 (fsc)

如果 DC.tc.PDTV 为 0，DC.fsc 字段将保存 iosatp，为第一阶段地址转换和保护提供控制。

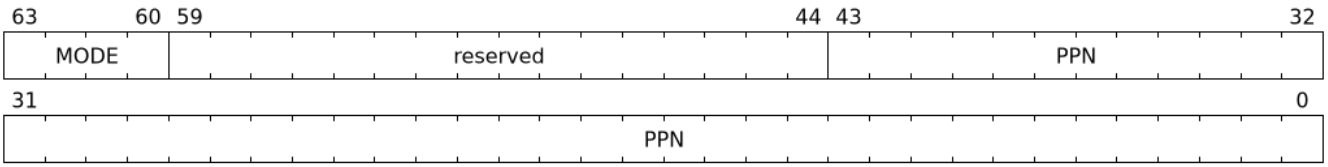


图 18. IO 监控程序地址转换和保护（iosatp）字段

第一阶段页表格式由特权规范定义。DC.tc.SXL 字段控制所支持的分页虚拟内存方案。

iosatp.MODE 标识分页虚拟内存方案，其编码如表 3 所示。iosatp.PPN 字段保存第一阶段页表根页面的 PPN。

当第二阶段地址转换不是 Bare 时，iosatp.PPN 是 guest PPN。然后，根页面的 GPA 会在 iohgatp 的控制下，由 Guest 物理地址（GPA）转换过程转换为 supervisor 物理地址（SPA）。

表 3. iosatp.MODE 字段的编码方式

DC.tc.SXL=0		
数值	名称	说明
0	Bare	没有转换或保护。
1-7	-	保留为标准用途。
8	Sv39	基于页面的 39 位虚拟寻址
9	Sv48	基于页面的 48 位虚拟寻址
10	Sv57	基于页面的 57 位虚拟寻址

11-13	-	保留为标准用途。
14-15	-	指定用于定制用途。
DC.tc.SXL=1		
数值	名称	说明
0	Bare	没有转换或保护。
1-7	-	保留为标准用途。
8	Sv32	基于页面的 32 位虚拟寻址
9-15	-	保留为标准用途。

当 DC.tc.PDTV 为 1 时，DC.fsc 字段保存进程目录表指针 (pdtp)。当设备支持由 process_id 选择的多个进程上下文时，PDT 用于确定第一阶段页表和相关的 PSCID，以进行虚拟地址转换和保护。

pdtp 字段包含根 PDT 的 PPN 和决定 PDT 级别数的 MODE 字段。

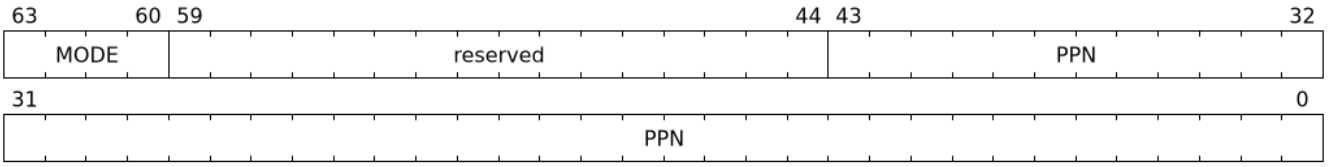


图 19.进程目录表指针 (pdtp) 字段

当第二阶段地址转换不是 Bare 时，pdtp.PPN 字段将保存一个 Guest PPN。然后，根 PDT 的 GPA 会被 Guest 物理地址转换过程（由 iohgatp 控制）转换为 supervisor 物理地址。使用第二阶段页表转换 PDT 的地址，允许 PDT 保留在 Guest OS 分配的内存中，并允许 Guest OS 直接编辑 PDT，将第一阶段页表确定的虚拟地址空间与 process_id 关联起来。

表 4.pdtp.MODE 字段的编码

数值	名称	说明
0	Bare	无一级地址转换或保护。
1	PD8	启用 8 位 process_id 。目录有 1 层，256 个条目。 process_id 的 19:8 位必须为 0。
2	PD17	启用 17 位 process_id 。目录分为两层。根 PDT 页有 512 个条目，叶级有 256 个条目。 process_id 的 19:17 位必须为 0。
3	PD20	启用 20 位 process_id 。目录有 3 层。根 PDT 有 8 个条目，下一级非叶子级有 512 个条目。叶级有 256 个条目。
4-13	-	保留为标准用途。
14-15	-	指定用于定制用途。

MSI 页表指针 (msiptp)

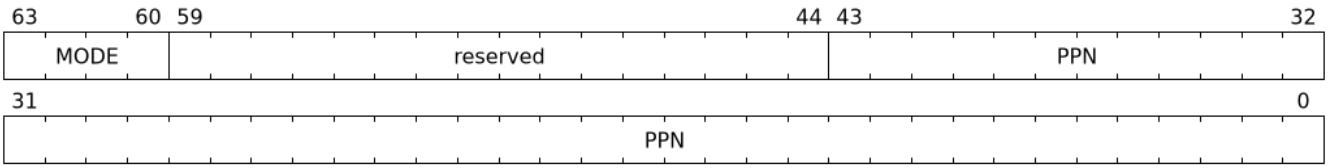


图 20.MSI 页表指针 (msiptp) 字段

msiptp.PPN 字段保存 root MSI 页表的 PPN，用于将 MSI 指向 IMSIC 中的 Guest 中断文件。MSI 页表格式由高级中断架构规范（AIA）定义。

msiptp.MODE 字段用于选择 MSI 地址转换方案。

表 5.msiptp.MODE 字段的编码

数值	名称	说明
0	关闭	无法识别使用 MSI 地址掩码和模式对虚拟中断文件的访问。
1	Flat	Flat MSI 页表
2-13		保留为标准用途。
14-15		指定用于定制用途。

MSI 地址掩码 (msi_addr_mask) 和 pattern (msi_addr_pattern)

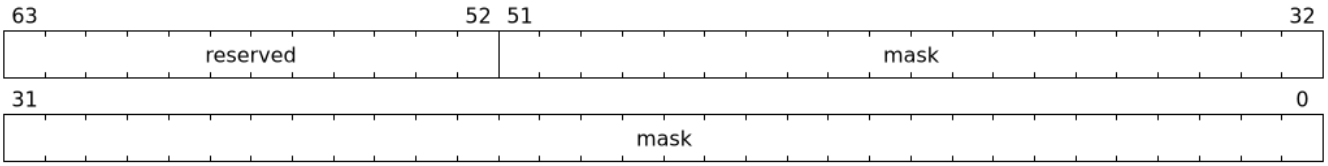


图 21.MSI 地址掩码（msi_addr_mask）字段

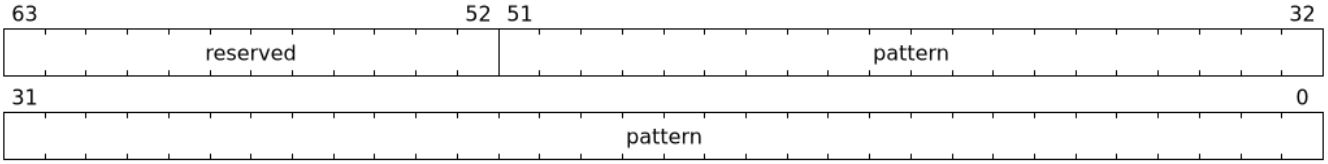


图 22.MSI 地址 pattern (msi_addr_pattern) 字段

MSI 地址掩码（msi_addr_mask）和 pattern （msi_addr_pattern）字段用于识别相关虚拟机 Guest 物理地址空间中的 4-KiB 虚拟中断文件页。如果目标 Guest 物理页面在所提供的地址掩码中所有为 0 的位位置上都与所提供的地址 pattern 相匹配，则设备进行的 incoming 内存访问将被视为对虚拟中断文件的访问。具体来说，在以下情况下，对 Guest 物理地址 A 的内存访问会被识别为对虚拟中断文件内存映射页的访问：

(A >> 12) & ~msi_addr_mask = (msi_addr_pattern & ~msi_addr_mask)

其中，>> 12 表示向右移动 12 位，& 表示按位逻辑 AND， ~msi_addr_mask 是地址掩码的按位逻辑补码。

2.1.4. 设备上下文配置检查（Device-context configuration checks）

如果以下任一条件为真，DC.tc.V=1 的 DC 将被视为配置错误。如果配置错误，则停止并报告 "DDT 表项配置错误"（原因 = 259）。

1. 如果设置了预留给未来标准使用的任何位或编码，则该位或编码将被删除。
2. capabilities.ATS 为 0 且 DC.tc.EN_ATS 或 DC.tc.EN_PRI 或 DC.tc.PRPR 为 1
3. DC.tc.EN_ATS 为 0，DC.tc.T2GPA 为 1
4. DC.tc.EN_ATS 为 0，DC.tc.EN_PRI 为 1
5. DC.tc.EN_PRI 为 0，DC.tc.PRPR 为 1
6. capabilities.T2GPA 为 0，DC.tc.T2GPA 为 1
7. DC.tc.T2GPA 为 1，DC.iohgatp.MODE 为 Bare
8. DC.tc.PDTV 为 1，且 DC.fsc.pdtp.MODE 不是受支持的模式
    a. capabilities.PD20 为 0 且 DC.fsc.pdtp.MODE 为 PD20
    b. capabilities.PD17 为 0 且 DC.fsc.pdtp.MODE 为 PD17
    c. capabilities.PD8 为 0 且 DC.fsc.pdtp.MODE 为 PD8
9. DC.tc.PDTV 为 0，且 DC.fsc.iosatp.MODE 编码不是由表 3 确定的有效编码

10. DC.tc.PDTV 为 0，DC.tc.SXL 为 0 DC.fsc.iosatp.MODE 不是支持的模式之一
    a. capabilities.Sv39 为 0，DC.fsc.iosatp.MODE 为 Sv39
    b. capabilities.Sv48 为 0，DC.fsc.iosatp.MODE 为 Sv48
    c. capabilities.Sv57 为 0，DC.fsc.iosatp.MODE 为 Sv57
11. DC.tc.PDTV 为 0，DC.tc.SXL 为 1 DC.fsc.iosatp.MODE 不是支持的模式之一
    a. capabilities.Sv32 为 0，DC.fsc.iosatp.MODE 为 Sv32
12. DC.tc.PDTV 为 0，DC.tc.DPE 为 1
13. DC.iohgatp.MODE 编码不是由表 2 确定的有效编码
14. fctl.GXL 为 0，且 DC.iohgatp.MODE 不是受支持的模式
    a. capabilities.Sv39x4 为 0，DC.iohgatp.MODE 为 Sv39x4
    b. capabilities.Sv48x4 为 0，DC.iohgatp.MODE 为 Sv48x4
    c. capabilities.Sv57x4 为 0，DC.iohgatp.MODE 为 Sv57x4
15. fctl.GXL 为 1，且 DC.iohgatp.MODE 不是受支持的模式
    a. capabilities.Sv32x4 为 0，DC.iohgatp.MODE 为 Sv32x4
16. capabilities.MSI_FLAT 为 1，DC.msiptp.MODE 不是关闭，也不是扁平模式
17. DC.iohgatp.MODE 不为 Bare ，且 DC.iohgatp.PPN 确定的根页表未对齐至 16-KiB 边界。
18. capabilities.AMO_HWAD 为 0 且 DC.tc.SADE 或 DC.tc.GADE 为 1
19. capabilities.END 为 0 且 fctl.BE != DC.tc.SBE
20. DC.tc.SXL 值不合法。如果 fctl.GXL 为 1，则 DC.tc.SXL 必须为 1。如果 fctl.GXL 为 0 且可写，则 DC.tc.SXL 可以为 0 或 1。如果 fctl.GXL 为 0 且不可写，则 DC.tc.SXL 必须为 0。
21. DC.tc.SBE 值不合法。如果 fctl.BE 可写，则 DC.tc.SBE 可能为 0 或 1。如果 fctl.BE 不可写，则 DC.tc.SBE 必须与 fctl.BE 相同。

某些 DC 字段保存有监管物理地址或 Guest 物理地址。某些实现可能会验证地址的有效性，例如，在定位 DC 时，SPA 的宽度不超过 capabilities.PAS 等确定的支持范围。此类实现可能会导致 "DDT 表项配置错误"（原因 = 259）故障。
其他实现方式只有在需要访问这些字段引用的数据结构时才会检测到这些地址无效。这类实现可能会在访问过程中检测到访问违规故障。

2.2. 进程目录表 (PDT)

PDT 是一棵 1 级、2 级或 3 级 radix-tree，使用 process_id 的进程目录索引（PDI）位进行索引。

以下图表说明了 PDT radix-tree。根进程目录页码通过设备上下文的进程目录表指针（pdtp）字段定位。每个非叶片（NL）条目提供下一级进程目录表的 PPN。叶进程目录表表项包含进程上下文（PC）。

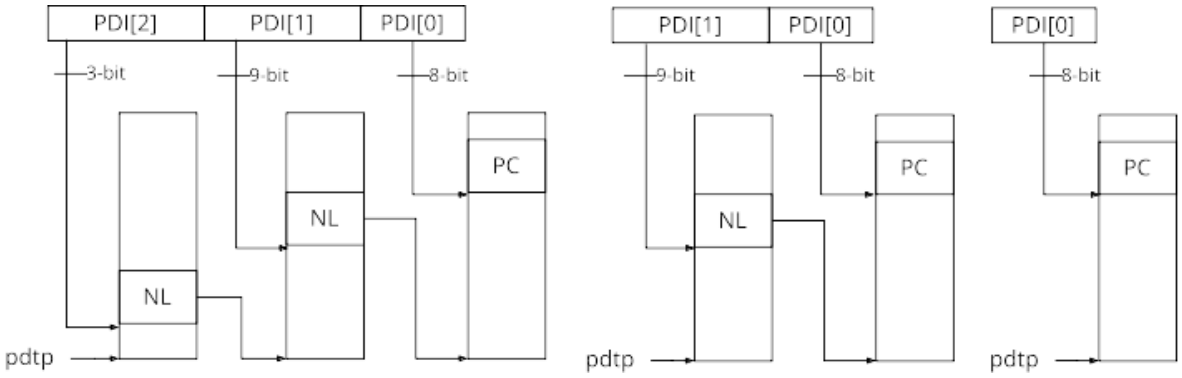


图 23. 三级、二级和单级进程目录

2.2.1. 非叶片 PDT 表项（Non-leaf PDT entry）

有效（V==1）的非叶片 PDT 表项保存下一级 PDT 的 PPN。

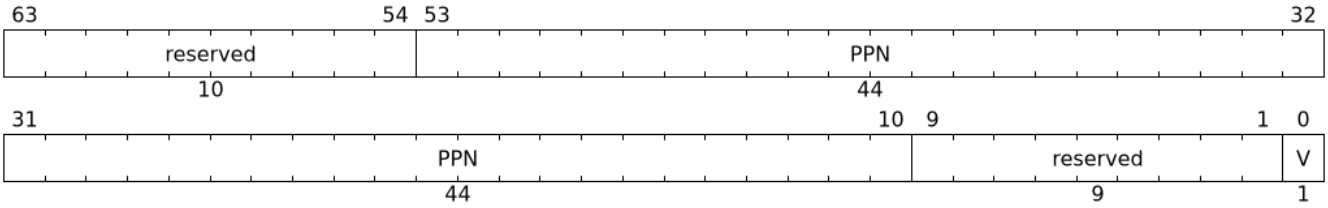


图 24. 非叶片进程目录表表项

2.2.2. 叶片 PDT 表项（Leaf PDT entry）

叶 PDT 页由 PDI[0] 索引，保存 16 字节的进程上下文（PC）。

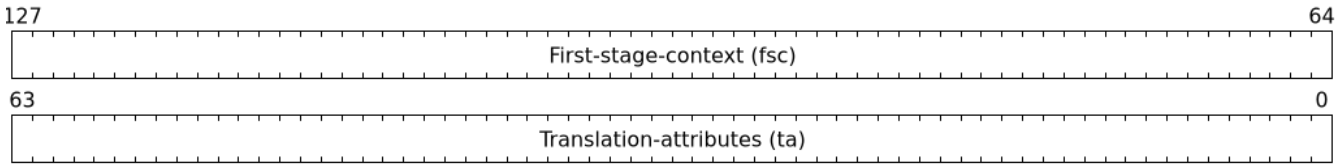


图 25. 进程上下文

PC 被解释为两个 64 位双字。内存中每个双字的字节顺序（小端或大端）由 DC.tc.SBE 确定。IOMMU 可以以任何顺序读取 PC 字段。

2.2.3. 进程上下文字段（Process-context fields）

转换属性 (ta)

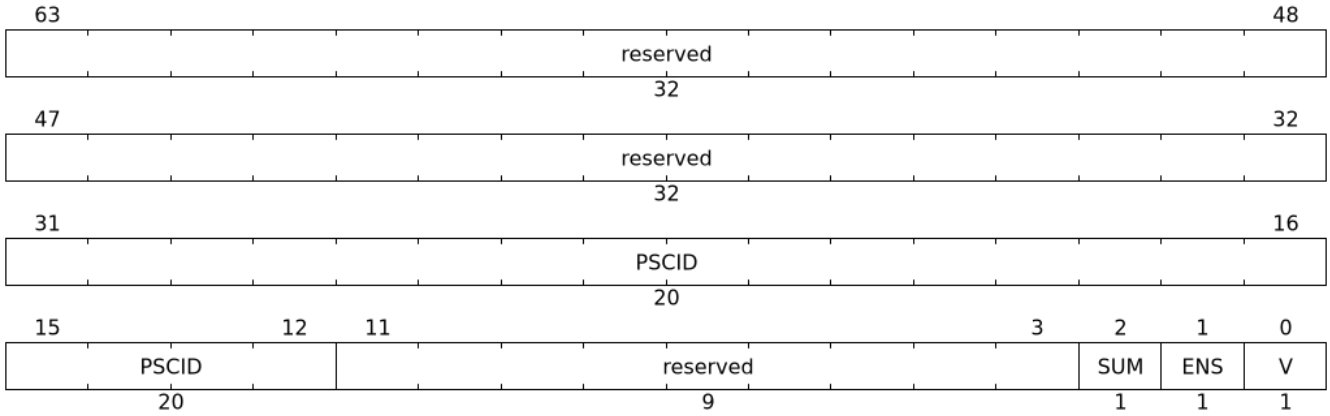


图 26. 转换属性 (ta) 字段

如果 V 位为 1，则 PC 有效；如果 V 位为 0，则 PC 中的所有其他位都无关紧要，可由软件自由使用。

当启用 Supervisory 访问（Enable-Supervisory-access，ENS）为 1 时，允许使用此 process_id 请求 supervisory 权限，否则该事务将被视为不支持的请求。

当 ENS 为 1 时，SUM（允许 Supervisor User Memory 访问）位会修改 supervisor 权限事务访问虚拟内存的权限。当 SUM 为 0 时，不允许对 PTE 中 U 位设置为 1 的映射页进行 supervisor 权限事务。

当 ENS 为 1 时，无论 SUM 的值如何，都不允许以执行意图读取PTE 中的 U 位设置为 1的映射页的 supervisor 权限事务。

当第一阶段地址转换不是 Bare 时，软件分配的进程软件上下文 ID (PSCID) 将用作第一阶段页表标识的进程的地址空间 ID。

第一阶段上下文 (fsc)

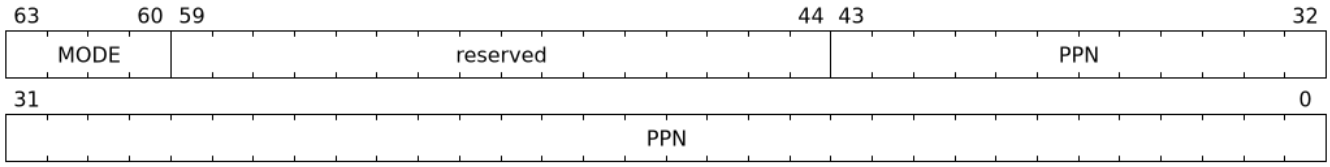


图 27. 进程第一阶段上下文

PC.fsc 字段提供第一阶段地址转换和保护的控制。

PC.fsc.MODE 用于确定第一阶段分页虚拟内存方案，其编码如表 3 所定义。DC.tc.SXL 字段控制所支持的分页虚拟内存方案。当 PC.fsc.MODE 不是 Bare 时，PC.fsc.PPN 字段保存第一阶段页表根页面的 PPN。

当第二阶段地址转换不是 Bare 时，PC.fsc.PPN 字段将保存第一阶段页表根目录的 Guest PPN。然后，第一阶段页表项的地址将通过 guest 物理地址转换过程（由 DC.iohgatp 控制）转换为 supervisor 物理地址。因此，Guest OS可以直接编辑第一阶段页表，限制设备对其内存子集的访问，并为设备访问指定权限。

PC.ta.PSCID 标识一个地址空间。如果在两个 PC 中配置了相同的 PSCID，而两个 PC 所引用的页表并不相同，那么 IOMMU 将无法预测是使用第一张页表中的 PTE 还是第二张页表中的 PTE。这些是唯一的预期行为。

2.2.4. 进程上下文配置检查（Process-context configuration checks）

如果以下任何条件为真，PC.ta.V=1 的 PC 将被视为配置错误。如果配置错误，则停止并报告 "PDT 表项配置错误"（原因 = 267）。

1. 如果设置了保留给未来标准使用的任何位或编码
2. PC.fsc.MODE 编码无效，如表 3 所示
3. DC.tc.SXL 为 0，且 PC.fsc.MODE 不是支持的模式之一
   a. capabilities.Sv39 为 0，PC.fsc.MODE 为 Sv39
   b. capabilities.Sv48 为 0，PC.fsc.MODE 为 Sv48
   c. capabilities.Sv57 为 0，PC.fsc.MODE 为 Sv57
4. DC.tc.SXL 为 1，且 PC.fsc.MODE 不是支持的模式之一
   a. capabilities.Sv32 为 0，PC.fsc.MODE 为 Sv32

某些 PC 字段包含 supervisor 物理地址或 guest 物理地址。某些实现可能会验证地址的有效性，例如，在定位 PC 时，supervisor 物理地址的宽度不超过 capabilities.PAS 等确定的支持范围。此类实现可能会导致 "PDT 表项配置错误"（原因 = 267）故障。

其他实现方式只有在需要访问这些字段引用的数据结构时才会检测到这些地址无效。这类实现可能会在访问过程中检测到访问违规故障。

2.3. 转换 IOVA 的过程（Process to translate an IOVA）

转换 IOVA 的过程使用硬件 ID（device_id 和 process_id）来定位 "设备上下文"（Device-Context）和 "进程上下文"（Process-Context）。设备上下文（Device-context）和进程上下文（Process-context）提供了页表的根 PPN、PSCID、GSCID 以及其他影响地址转换和保护进程的控制参数。当实施地址转换缓存（ATC）（第 2.8 节）时，转换过程可使用 GSCID 和 PSCID 将缓存的转换与其地址空间相关联。

IOVA 的转换过程如下：

1. 如果 ddtp.iommu_mode == Off，则停止并报告 "禁止所有入站事务"（原因 = 256）。
2. 如果 ddtp.iommu_mode == Bare 且以下任一条件成立，则停止并报告 "事务类型不允许"（原因 = 260）；否则转到步骤 20，转换地址与 IOVA 相同。
   a. 事务类型为已转换请求（读、写/AMO、读取执行）或 PCIe ATS 转换请求。
3. 如果 capabilities.MSI_FLAT 为 0，则 IOMMU 使用基本格式设备上下文。让 DDI[0] 为 device_id[6:0]，DDI[1] 为 device_id[15:7]，DDI[2] 为 device_id[23:16]。
4. 如果 capabilities.MSI_FLAT 为 1，则 IOMMU 使用扩展格式设备上下文。让 DDI[0] 为 device_id[5:0]，DDI[1] 为 device_id[14:6]，DDI[2] 为 device_id[23:15]。
5. 如果 device_id 的范围大于 IOMMU 模式支持的范围（由以下检查确定），则停止并报告 "事务类型不允许"（cause = 260）。
   a. ddtp.iommu_mode 为 2LVL，且 DDI[2] 不为 0
   b. ddtp.iommu_mode 为 1LVL，且 DDI[2] 不为 0 或 DDI[1] 不为 0
6. 然后使用 device_id 查找第 2.3.1 节规定的设备上下文 (DC)。
7. 如果以下任何条件成立，则停止并报告 "事务类型不允许"（原因 = 260）。
   a. 事务类型为已转换请求（读、写/AMO、读取执行）或 PCIe ATS 转换请求，且 DC.tc.EN_ATS 为 0。
   b. 事务具有有效的 process_id ，且 DC.tc.PDTV 为 0。

c. 事务具有有效的 process_id ，且 DC.tc.PDTV 为 1，且 process_id 宽于 pdtp.MODE 支持的范围。

d. IOMMU 不支持的事务类型。

8. 如果请求是已转换请求，且 DC.tc.T2GPA 为 0，则转换过程完成。转至步骤 20。
9. 如果请求为已转换请求，且 DC.tc.T2GPA 为 1，则 IOVA 为 GPA。根据下页表格信息转至步骤 17：
   a. 让 A 成为 IOVA（IOVA 是一个 GPA）。
   b. 将 iosatp.MODE 设为 "Bare"。
      i. 第一阶段为 Bare 时，不使用 PSCID 值。
   c. 让 iohgatp 成为 DC.iohgatp 字段中的值
10. 如果 DC.tc.PDTV 设置为 0，则转至步骤 17，并输入下面的页表信息：
    a. 设置 DC.fsc.MODE 字段中的值为 iosatp.MODE
    b. 设置 DC.fsc.PPN 字段中的值为 iosatp.PPN
    c. 设置 DC.ta.PSCID 字段中的值为 PSCID
    d. 设置 DC.iohgatp 字段中的值为 iohgatp
11. 如果 DPE 为 1，且没有与事务相关联的 process_id，那么 process_id 的默认值为 0。
12. 如果 DPE 为 0，且没有与事务相关联的 process_id，则转到步骤 17，并提供以下页表信息：
    a. 将 iosatp.MODE 设为 "Bare"。
       i. 第一阶段为 Bare 时，不使用 PSCID 值。
    b. 设置 DC.iohgatp 字段中的值为 iohgatp
13. 如果 DC.fsc.pdtp.MODE = Bare，则转至步骤 17，并输入以下页表信息：
    a. 将 iosatp.MODE 设为 "Bare"。
       i. 第一阶段为 Bare 时，不使用 PSCID 值。
    b. 设置 DC.iohgatp 字段中的值为 iohgatp
14. 按照第 2.3.2 节的规定找到进程上下文 (PC)。
15. 如果以下任何条件成立，则停止并报告 "事务类型不允许"（原因 = 260）。
    a. 事务请求 supervisor 权限，但未设置 PC.ta.ENS。
16. 根据下表信息转到步骤 17：
    a. 设置 PC.fsc.MODE 字段中的值为 iosatp.MODE
    b. 设置 PC.fsc.PPN 字段中的值为 iosatp.PPN
    c. 设置 PC.ta.PSCID 字段中的值为 PSCID
    d. 设置 DC.iohgatp 字段中的值为 iohgatp
17. 使用 RISC-V 特权规范[3]中 "两阶段地址转换 "一节规定的流程来确定事务访问的 GPA。如果第一阶段地址转换过程检测到故障，则停止并报告故障。如果地址转换过程成功完成，则 A 为转换后的 GPA。
18. 如果启用了使用 MSI 页表的 MSI 地址转换（即 DC.msiptp.MODE != Off），则将调用第 2.3.3 节规定的 MSI 地址转换过程。如果 GPA A 未被确定为虚拟中断文件的地址，则继续执行步骤 19。如果 MSI 地址转换过程检测到故障，则停止并报告故障，否则继续执行步骤 20。
19. 使用 RISC-V 特级规范[3]中 "两阶段地址转换 "一节规定的第二阶段地址转换过程来转换 GPA A，以确定事务访问的 SPA。如果地址转换过程检测到故障，则停止并报告故障。

20. 转换过程已完成

当检查第二阶段 PTE 中的 U 位时，该事务将被视为未请求 supervisor 权限。

当转换过程报告故障，且请求为未转换请求或已转换请求时，IOMMU 会请求 IO Bridge 中止事务。IO Bridge 处理故障事务的指导原则见第 7.3 节。故障可能使用第 3.2 节规定的故障/事件报告机制和故障记录格式进行报告。

如果故障是由 PCIe ATS 转换请求检测到的，那么 IOMMU 可以提供 PCIe 协议定义的响应，而不是向软件报告故障或导致中止。第 2.6 节规定了对 PCIe ATS 转换请求故障的处理方法。

2.3.1. 定位设备上下文的过程（Process to locate the Device-context）

使用 device_id 查找事务的设备上下文的过程如下：

1. 设 a 为 ddtp.PPN x 2^{12}，设 i = LEVELS - 1。当 ddtp.iommu_mode 为 3LVL 时，LEVELS 为 3。当 ddtp.iommu_mode 为 2LVL 时，LEVELS 为 2。当 ddtp.iommu_mode 为 1LVL 时，LEVELS 为 1。
2. 如果 i == 0，转至步骤 8。
3. 如果访问 ddte 时违反了 PMA 或 PMP 检查，则停止并报告 "DDT 表项加载访问故障"（原因 = 257）。
4. 如果 ddte 访问检测到数据损坏（又称中毒数据），则停止并报告 "DDT 数据损坏"（原因 = 268）。
5. 如果 ddte.V == 0，则停止并报告 "DDT 表项无效"（原因 = 258）。
6. 如果在 ddte 中设置了保留给未来标准使用的任何位或编码，则停止并报告 "DDT 表项配置错误"（原因 = 259）。
7. 让 i = i - 1，让 a = ddte.PPN x 2^{12} 。 转到步骤 2。
8. 地址 a + DDI[0] * DC_SIZE 处的字节数量为 DC_SIZE 的值就是 DC。如果 capabilities.MSI_FLAT 为 1，则 DC_SIZE 为 64 字节，否则为 32 字节。如果访问 DC 时违反了 PMA 或 PMP 检查，则停止并报告 "DDT 入口加载访问故障"（原因 = 257）。如果 DC 访问检测到数据损坏（又称中毒数据），则停止并报告 "DDT 数据损坏"（原因 = 268）。
9. 如果 DC.tc.V == 0，则停止并报告 "DDT 表项无效"（原因 = 258）。
10. 如果根据第 2.1.4 节所述规则确定 DC 配置错误，则停止并报告 "DDT 表项配置错误"（原因 = 259）。
11. 设备上下文已成功定位。

2.3.2. 定位进程上下文的过程（Process to locate the Process-context）

设备上下文提供 PDT 根页面的 PPN (pdtp.ppn)。当 DC.iohgatp.mode 不为 Bare 时，pdtp.PPN 和 pdte.PPN 均为 Guest 物理地址 (GPA)，必须使用 DC.iohgatp 指向的第二阶段页表将其转换为 Supervisor 物理地址 (SPA)。对 PDT 的内存访问被第二阶段视为隐式读取内存访问。

使用事务的 process_id 查找事务的 Process-context 的过程如下：

1. 设 a 为 pdtp.PPN x 2^{12}，设 i = LEVELS - 1。当 pdtp.MODE 为 PD20 时，LEVELS 为 3。当 当 pdtp.MODE 为 PD17 时，LEVELS 为 2。当 pdtp.MODE 为 PD8 时，LEVELS 为 1。
2. 如果 DC.iohgatp.mode != Bare，则 a 是一个 GPA。调用进程将 a 转换为 SPA，作为隐式内存访问。如果在 a 的第二阶段地址转换过程中出现故障，则停止并报告第二阶段地址转换进程检测到的故障。转换后的 a 在后续步骤中使用。
3. 如果 i == 0，转到步骤 9。
4. 如果 pdte 的访问违反了 PMA 或 PMP 检查，则停止并报告 "PDT 表项加载访问故障"（原因 = 265）。

5. 如果 pdte 访问检测到数据损坏（又称中毒数据），则停止并报告 "PDT 数据损坏"（原因 = 269）。
6. 如果 pdte.V == 0，则停止并报告 "PDT 表项无效"（原因 = 266）。
7. 如果在 pdte 中设置了保留给未来标准使用的任何位或编码，则停止并报告 "PDT 表项配置错误"（原因 = 267）。
8. 让 i = i - 1，让 a = pdte.PPN x 2^{12} 。 转到第 2 步。
9. 位于 a + PDI[0] x 16 地址处的 16 字节值就是 PC。如果访问 PC 时违反了 PMA 或 PMP 检查，则停止并报告 "PDT 入口加载访问故障"（原因 = 265）。如果 PC 访问检测到数据损坏（又称中毒数据），则停止并报告 "PDT 数据损坏"（原因 = 269）。
10. 如果 PC.ta.V == 0，则停止并报告 "PDT 表项无效"（原因 = 266）。
11. 如果根据第 2.2.4 节所述规则确定 PC 配置错误，则停止并报告 "PDT 表项配置错误"（原因 = 267）。
12. 进程上下文已成功定位。

2.3.3. 转换 MSI 地址的过程（Process to translate addresses of MSIs）

当 I/O 设备由 Guest OS 直接配置时，来自设备的 MSI 预计会针对 Guest OS 虚拟机中的虚拟 IMSIC，使用对真实机器不合适和不安全的 Guest 物理地址。IOMMU 必须将此类设备的某些传入写入识别为 MSI，并根据实际机器的需要进行转换。

来自单个设备、需要转换的 MSI 预计已由运行在一个 RISC-V 虚拟机中的单个 Guest OS 在该设备上进行了配置。假设虚拟机本身符合 RISC-V 高级中断架构[2]，MSI 将通过写入虚拟 IMSIC 中断文件的内存映射寄存器发送到虚拟机内的虚拟 Harts。每个虚拟中断文件都占用虚拟机 Guest 物理地址空间中一个单独的 4-KiB 页面，这与真实中断文件在真实机器物理地址空间中的占用情况相同。因此，如果写入的是虚拟机中虚拟 IMSIC 的中断文件所占用的页面，则对 Guest 物理地址的写入可被识别为对虚拟机的 MSI。

当支持 MSI 地址转换（第 5.3 节 capabilities.MSI_FLAT）时，将传入的 IOVA 识别为虚拟中断文件地址并使用 MSI 页表转换地址的过程如下：

1. 让 A 成为 GPA
2. 让 DC 成为设备上下文，使用第 2.3.1 节中概述的流程，使用设备的 device_id 进行定位。
3. 确定地址 A 是否是第 2.1.3.6 节规定的虚拟中断文件访问。
4. 如果地址未被确定为虚拟中断文件的地址，则停止此过程，转而使用常规转换数据结构进行地址转换。
5. 从 A 中提取中断文件编号 I，即 I = extract(A >> 12, DC.msi_addr_mask)。位提取函数 extract(x, y) 丢弃 x 中与掩码 y 中相同位置的匹配位为零的所有位，并将 x 中的剩余位连续打包到结果的最末端，保持与 x 相同的位顺序，并将结果最末端的其他位填充为零。例如，如果 x 和 y 的位数是
   - x = a b c d e f g h
   - y = 1 0 1 0 0 1 1 0
   - 那么 extract(x, y) 的值的位数为 0 0 0 0 a c f g。
6. 设 m 为（DC.msiptp.PPN x 2^{12} ）。
7. 设 msipte 为地址 (m | (I x 16)) 处 16 个字节的值。如果访问 msipte 时违反了 PMA 或 PMP 检查，则停止并报告 "MSI PTE 加载访问故障"（原因 = 261）。
8. 如果 msipte 访问检测到数据损坏（又称中毒数据），则停止并报告 "MSI PT 数据损坏"（原因 = 270）。
9. 如果 msipte.V == 0，则停止并报告 "MSI PTE 无效"（原因 = 262）。
10. 如果 msipte.C == 1，那么解释 PTE 的进一步处理由实现定义。
11. 如果 msipte.C == 0，则后续步骤将概述该过程。

12. 如果 msipte.M == 0 或 msipte.M == 2，则停止并报告 "MSI PTE 配置错误"（原因 = 263）。
13. 如果 msipte.M == 3，则 PTE 处于基本转换模式，转换过程如下：
    a. 如果 msipte 中设置了保留给未来标准使用的任何位或编码，则停止并报告 "MSI PTE 配置错误"（原因 = 263）。
    b. 计算转换地址为 msipte.PPN << 12 | A[11:0]。
14. 如果 msipte.M == 1，则 PTE 处于 MRIF 模式，转换过程如下：
    a. 如果 capabilities.MSI_MRIF == 0，则停止并报告 "MSI PTE 配置错误"（原因 = 263）。
    b. 如果 msipte 中设置了保留给未来标准使用的任何位或编码，则停止并报告 "MSI PTE 配置错误"（原因 = 263）。
    c. 目标 MRIF 的地址为 msipte.MRIF_Address[55:9] * 512。
    d. 通知 MSI 的目标地址是 msipte.NPPN << 12。
    e. 设 NID 为 (msipte.N10 << 10) | msipte.N[9:0]。通知 MSI 的数据值是 11 位 NID 零扩展到 32 位的值。
15. 通过此过程确定的与转换相关的访问权限，与 R=W=U=1 和 X=0 的常规 RISC-V 第二阶段 PTE 的访问权限相当。
16. 如果事务是 "未转换 "或 "转换后读取执行"，则停止并报告 "指令访问故障"（原因 = 1）。
17. MSI 地址转换过程已完成。

在 MRIF 模式下，高级中断架构规范定义了将输入的 MSI 存储到目标 MRIF 和生成通知 MSI 的操作。这些操作可由 IOMMU 本身执行，或者 IOMMU 可根据转换请求向 I/O Bridge 提供目标 MRIF 地址、通知 MSI 地址和通知 MSI 数值，这些操作可由 I/O Bridge 执行。

2.4. IOMMU 更新 PTE 访问 (A) 和 dirty (D) 最新信息（IOMMU updating of PTE accessed (A) and dirty (D) updates）

当 capabilities.AMO_HWAD 为 1 时，IOMMU 支持原子更新 PTE 中的 A 和 D 位。启用更新第二阶段 PTE 中的 A 和 D 位（DC.tc.GADE=1）和/或启用更新第一阶段 PTE 中的 A 和 D 位（DC.tc.SADE=1）时，适用以下规则：

1. IOMMU 的 A 和/或 D 位更新必须遵循特权规范规定的有效性、权限检查和原子性规则。
2. PTE 更新必须在使用 IOMMU 提供的已转换地址的内存访问变成全局可见之前全局可见。具体来说，当在 ATS 转换完成中向设备提供已转换地址时，PTE 更新必须在设备使用已转换地址进行内存访问变得全局可见之前全局可见。

IOMMU 永远不会清除 A 和 D 位。如果上位机软件不依赖已访问位和/或脏位，例如不将内存页交换到二级存储空间，或者内存页被用于映射 I/O 空间，则应在 PTE 中将其设为 1，以提高性能。

2.5. 虚拟地址转换过程中的故障（Faults from virtual address translation process）

在 RISC-V 特权规范[3]规定的两阶段地址转换过程中检测到的故障会导致 IOVA 转换过程停止并报告检测到的

故障。

2.6. PCIe ATS 转换请求处理（PCIe ATS translation request handling）

ATS [1] 转换请求如果遇到配置错误，请求者将收到 "完成者中止"（CA）响应。以下原因代码属于此类：

- 指令访问故障（原因 = 1）
- 读取访问故障（原因 = 5）
- 写入/AMO 访问故障（原因 = 7）
- MSI PTE 负载访问故障（原因 = 261）
- MSI PTE 配置错误（原因 = 263）
- PDT 入口负载访问故障（原因 = 265）
- PDT 表项配置错误（原因 = 267）

如果出现永久性错误或禁用了 ATS 事务，则会生成不支持的请求 (UR) 响应。以下原因代码属于此类：

- 禁止所有入站事务（原因 = 256）
- DDT 入口加载访问故障（原因 = 257）
- DDT 表项无效（原因 = 258）
- DDT 表项配置错误（原因 = 259）
- 事务类型不允许（原因 = 260）

由于以下原因导致转换无法完成时，将生成 R 和 W 位设置为 0 的 "成功响应"。这些错误不会记录在故障队列中。在这种情况下返回的转换地址为 UNSPECIFIED。

- 指令页面故障（原因 = 12）
- 读取页面故障（原因 = 13）
- 写入/AMO 页故障（原因 = 15）
- 指令Guest页故障（原因 = 20）
- 读取Guest页故障（原因 = 21）
- 写入/AMO Guest-page 故障（原因 = 23）
- PDT 表项无效（原因 = 266）
- MSI PTE 无效（原因 = 262）

如果转换请求具有 PASID，且 "请求特权模式 "字段设置为 0，或者请求不具有 PASID，则该请求不以特权内存为目标。如果表示内存是否可访问用户模式的 U 位为 0，则会生成 R 和 W 位均设为 0 的成功响应。

如果转换请求的 PASID 的 "特权模式请求 "字段设置为 1，则该请求以特权内存为目标。如果表示页面是否可访问用户模式的 U 位为 1，且进程上下文 ta 字段中的 SUM 位为 0，则会生成 R 和 W 位均设为 0 的成功响应。

如果转换可以成功完成，但所请求的权限不存在（请求执行但无执行权限；未请求不写且无写权限；无读权限），则返回成功响应，同时将被拒绝的权限（R、W 或 X）设置为 0，并将其他权限位设置为根据页表确

定的值。只有同时授予 R 权限时，才会授予 X 权限。“仅执行”转换与 PCIe ATS 不兼容，因为 PCIe 要求在授予执行权限的情况下授予读取权限。

当 ATS 转换请求生成成功响应时，即使响应显示未授予访问权限或某些权限被拒绝，也不会通过故障/事件报告机制向软件报告故障记录。

如果使用第 2.1.3.6 节定义的规则将转换请求的地址确定为 MSI 地址，但 MSI PTE 配置为 MRIF 模式，则响应 "成功"。

响应中设置为 1 的 U 位指示设备只能使用未转换请求访问隐含的 4 KiB 内存范围。

当 MSI PTE 配置为 MRIF 模式时，带数据值 D 的 MSI 写入要求 IOMMU 设置 MRIF 中中断标识 D 的中断等待位。从设备向通过 MRIF 模式 MSI PTE 映射的 GPA 发出的转换请求不符合接收转换地址的条件。这可通过将返回响应的 "仅限未转换访问"（U）字段设置为 1 来实现。

为 ATS 转换请求生成成功响应时，Priv、N、CXL.io、Global 和 AMA 字段的设置如下：

- 如果请求没有 PASID，则 ATS 转换完成的 Priv 字段总是设置为 0。如果存在 PASID，则 Priv 字段将设置为 "请求的特权模式 "字段中的值，因为所提供的权限与请求中指明的特权模式一致。
- ATS 转换完成的 N 字段始终设置为 0。 设备可使用其他方法来确定是否应在转换请求中设置 No-snoop 标志。
- 如果转换可以成功完成，且请求中存在 PASID，则 Global 字段将被设置为从第一阶段页表中确定的值。在所有其他情况下，包括 MSI 地址转换，该字段将设置为 0。
- 如果请求设备不是 CXL 设备，则 CXL.io 设置为 0。
- 如果申请设备是 type 1 或 type 2 CXL 设备
  - 如果地址被确定为 MSI，那么 CXL.io 位将被置 1。
  - 否则，如果设备上下文中 T2GPA 为 1，则 CXL.io 位将被设置为 1。
  - 否则，如果由 Svpbmt 扩展确定的内存类型为 NC 或 IO，则 CXL.io 位将设为 1。如果内存类型为 PMA，则该位的设置将由 UNSPECIFIED 决定。如果不支持 Svpbmt 扩展，则该位的设置为 UNSPECIFIED。
  - 在所有其他情况下，该位的设置为 UNSPECIFIED。
- AMA 字段默认设置为 000b。IOMMU 可以支持特定的实现方法来提供其他编码。

IO Bridge 可根据转换地址的 PMA 覆盖 ATS 转换完成中的 CXL.io 位。其他实现可提供一种实现定义的方法，用于确定转换地址的 PMA，以设置 CXL.io 位。

将 T2GPA 设置为 1 可能与 CXL type 1 或 type 2 设备不兼容，因为它们使用 CXL.cache 协议实施缓存，缓存由响应 PCIe ATS 转换请求时返回的已转换地址标记。在 CXL.cache 事务中，不得调用 IOMMU 来转换地址。

2.7. PCIe ATS 页面请求处理（PCIe ATS Page Request handling）

要处理 "页面请求 "或 "停止标记 "报文[1]，IOMMU 首先要定位设备上下文，以确定请求者是否启用了 ATS 和 PRI。如果启用了 ATS 和 PRI，即 EN_ATS 和 EN_PRI 均设为 1，IOMMU 就会将报文排入内存队列，即页面请求队列（PQ）（见第 3.3 节）。在对 "页面请求 "进行适当处理后，软件处理程序可向设备生成 "Page Request Group Response "报文。

设备启用 PRI 后，IOMMU 仍可能无法通过 PQ 报告 "页面请求 "或 "停止标记 "报文，原因是队列被禁用、队列已满或 IOMMU 在尝试访问队列内存时遇到访问故障等错误条件。这些错误条件在第 3.3 节中有具体说明。

如果 ddtp.iommu_mode 为 Bare 或 Off，则 IOMMU 无法为请求者找到设备上下文。

如果 EN_PRI 设置为 0，或 EN_ATS 设置为 0，或 IOMMU 无法找到 DC 以确定 EN_PRI 配置，或请求无法排队进入 PQ，则 IOMMU 的行为取决于 "页面请求" 的类型。

- 如果 "页面请求 "不需要响应，即报文的 "Last Request in PRG" 字段设置为 0，则此类报文将被静默丢弃。"Stop Marker"（停止标记）报文不需要响应，在出现此类错误时总是被静默丢弃。
- 如果 "页面请求 "需要响应，那么 IOMMU 本身就会向设备发送 "Page Request Group Response "信息。

当 IOMMU 生成响应时，响应的状态字段取决于错误原因。

如果遇到以下故障，状态将设置为响应失败（Response Failure）：

- ddtp.iommu_mode 是 Off
- DDT 入口加载访问故障（原因 = 257）
- DDT 表项配置错误（原因 = 259）
- DDT 表项无效（原因 = 258）
- 页面请求队列未启用（pqcsr.pqen == 0 或 pqcsr.pqon == 0）
- 页面请求队列遇到内存访问故障 (pqcsr.pqmf == 1)

如果遇到以下故障，状态将设置为无效请求：

- ddtp.iommu_mode 为 Bare
- EN_PRI 设置为 0

如果没有遇到其他故障，但由于页面请求队列已满（pqt == pqh - 1）或溢出（pqcsr.pqof == 1），"页面请求 "无法排队，则状态设为成功。

SR-IOV VF 用作分配单元时，Hypervisor 可通过将 EN_PRI 设置为 0 来禁用其中一个虚拟功能的页面请求。IOMMU 协议特定逻辑在其响应中将这种情况（原因 = 260）归类为非灾难性故障（无效请求），以避免设备中的共享 PRI 被所有 PF/VF 禁用。

“停止标记”编码为带有 PASID 的”Page Request”，但 L、W 和 R 字段分别设置为 1、0 和 0。

对于 IOMMU 生成的状态为无效请求或成功的 "Page Request Group Response "报文，PRG-response-PASID-required (PRPR) 位设置为 1 时表示，如果相关的 "页面请求 "有 PASID，则 IOMMU 响应报文应包含 PASID。

对于 IOMMU 生成的响应代码设置为响应失败的 "Page Request Group Response"，如果 "页面请求 "有 PASID，则生成带有 PASID 的响应。

在下列情况下，PCIe ATS "页面请求 "信息的故障队列中不会记录故障：

- 页面请求队列未启用（pqcsr.pqen == 0 或 pqcsr.pqon == 0）
- 页面请求队列遇到内存访问故障 (pqcsr.pqmf == 1)
- 由于页面请求队列已满（pqt == pqh - 1）或溢出（pqcsr.pqof == 1），"页面请求 "无法排队。

2.8. 缓存内存数据结构（Caching in-memory data structures）

为加快直接内存访问（DMA）转换速度，IOMMU 可使用转换缓存来保存设备目录表、进程目录表、第一阶段和第二阶段转换表以及 MSI 页表中的条目。这些缓存统称为 IOMMU 地址转换缓存（IOATC）。

本规范不允许缓存 V（有效）位清零的第一/第二阶段 PTE、V（有效）位清零的非叶 DDT 表项、V（有效）位清零的设备上下文、V（有效）位清零的非叶 PDT 表项、V（有效）位清零的流程上下文或 V 位清零的 MSI PTE。

这些 IOATC 无法观察到 RISC-V harts 或设备 DMA 使用显式加载和存储对内存数据结构的修改。软件必须使用 IOMMU 命令使缓存的数据结构条目失效，使用 IOMMU 命令同步 IOMMU 操作以观察内存数据结构的更新。较简单的实现可能不会对某些或任何内存数据结构实施 IOATC。IOMMU 命令可使用一个或多个 ID 标记缓存条目，以识别特定条目或条目组。

表 6.用于标记 IOATC 条目的标识符

缓存的数据结构	用于标记条目的 ID	使无效的命令
设备目录表	device_id	iodir.inval_ddt
进程目录表	device_id，process_id	iodir.inval_pdt
第一阶段页表（当第二阶段页表不为 Bare 时）	GSCID、PSCID 和 IOVA	IOTINVAL.VMA
第一阶段页表（当第二阶段为 Bare 时）	PSCID 和 IOVA	IOTINVAL.VMA
第二阶段页表	GSCID, GPA	IOTINVAL.GVMA
MSI 页表	GSCID, GPA	IOTINVAL.GVMA

2.9. 更新内存数据结构条目（Updating in-memory data structure entries）

RISC-V 内存模型要求从 Hart 进行的内存访问必须是单拷贝原子访问。实现 RV32 时，单拷贝原子内存访问的大小最多为 32 位。实现 RV64 时，单拷贝原子内存访问的大小最多为 64 位。IOMMU 实现的单拷贝原子内存访问大小是 UNSPECIFIED，但如果系统中的所有 hart 都实现了 RV32，则要求至少为 32 位；如果系统中的任何 hart 都实现了 RV64，则要求至少为 64 位。

IOMMU 数据结构条目有一个 V 位，设置为 1 时表示条目有效。

允许软件对 V 位设置为 1 的数据结构条目进行更新，但必须遵守以下规则。

- 一般来说，软件使用一组宽度小于 IOMMU 支持的最小单拷贝原子内存访问的存储来更新有效数据结构条目的字段是不安全的，因为 IOMMU 在任何时候读取条目都是合法的，包括只有部分存储生效的时候。
- 要使 IOMMU 数据结构条目的更新具有原子性，软件必须使用宽度等于 IOMMU 支持的最小单拷贝原子内存访问的单次存储。
- 如果更新某个字段会使该字段与条目的另一个字段不一致，那么软件必须首先将 V 字段设为 0，并在更新该条目的其他字段之前，使用第 2.8 节中概述的命令使 IOMMU 缓存中可能存在的该条目以前的副本失效。
- IOMMU 无需立即观察软件对条目进行的更新。软件必须使用第 2.8 节中概述的命令，使 IOMMU 缓存中可能存在的该条目以前的副本失效，从而使条目更新与 IOMMU 的运行同步。

如果数据结构条目发生变化，IOMMU 可能会使用条目的旧值或新值，在软件使用第 2.8

节中概述的命令使 IOMMU 缓存中可能存在的该条目以前的副本失效，从而使条目更新与 IOMMU 运行同步之前，IOMMU 的选择是不可预测的。这些是唯一的预期行为。

2.10. 内存数据结构的大小端（Endianness of in-memory data structures）

RISC-V 内存模型规定了整个地址空间的字节不变性。当支持混合大小端工作模式时，IO Bridge 和 IOMMU 必须实现字节不变寻址，这样，在小端和大端工作模式下，对给定地址的字节访问都能访问相同的内存位置。

对内存数据结构的隐式内存访问的字节序由 fctl.BE 或 DC.tc.SBE决定，如下：

表 7.数据结构内存访问的大小端

数据结构	Controlled by
设备目录表	fctl.BE
第二阶段页表	fctl.BE
MSI 页表	fctl.BE
进程目录表	DC.tc.SBE
第一阶段页表	DC.tc.SBE

第一阶段上下文的 PSCID 字段与 GSCID（当两阶段地址转换激活时）一起标识一个地址空间。在两个具有不同 SBE 的 DC 中配置相同的 GSCID 和 PSCID 是不可取的，这样做可能导致 IOMMU 将第一阶段 PTE 解释为 big-endian 或 little-endian。这些是唯一的预期行为。

在与不共享内码的 IO 代理共享数据时，软件必须使用适当的软件序列交换字节，以创建双方同意的数据表示。当必须以原子方式访问与此类 IO 代理共享的数据时，软件必须使用 LR/SC 序列，以非本地 endian 格式执行原子操作。

第 3 章 内存队列接口（In-memory queue interface）

软件和 IOMMU 使用 3 种内存队列数据结构进行交互。

- 命令队列（CQ），用于软件向 IOMMU 发送队列命令。
- 故障/事件队列 (FQ)，由 IOMMU 使用，用于提请软件注意故障和事件。
- IOMMU 用来报告从 PCIe 设备接收到的 "页面请求 "信息的页面请求队列（PQ）。如果 IOMMU 支持 PCIe [1] 定义的页面请求接口，则支持该队列。

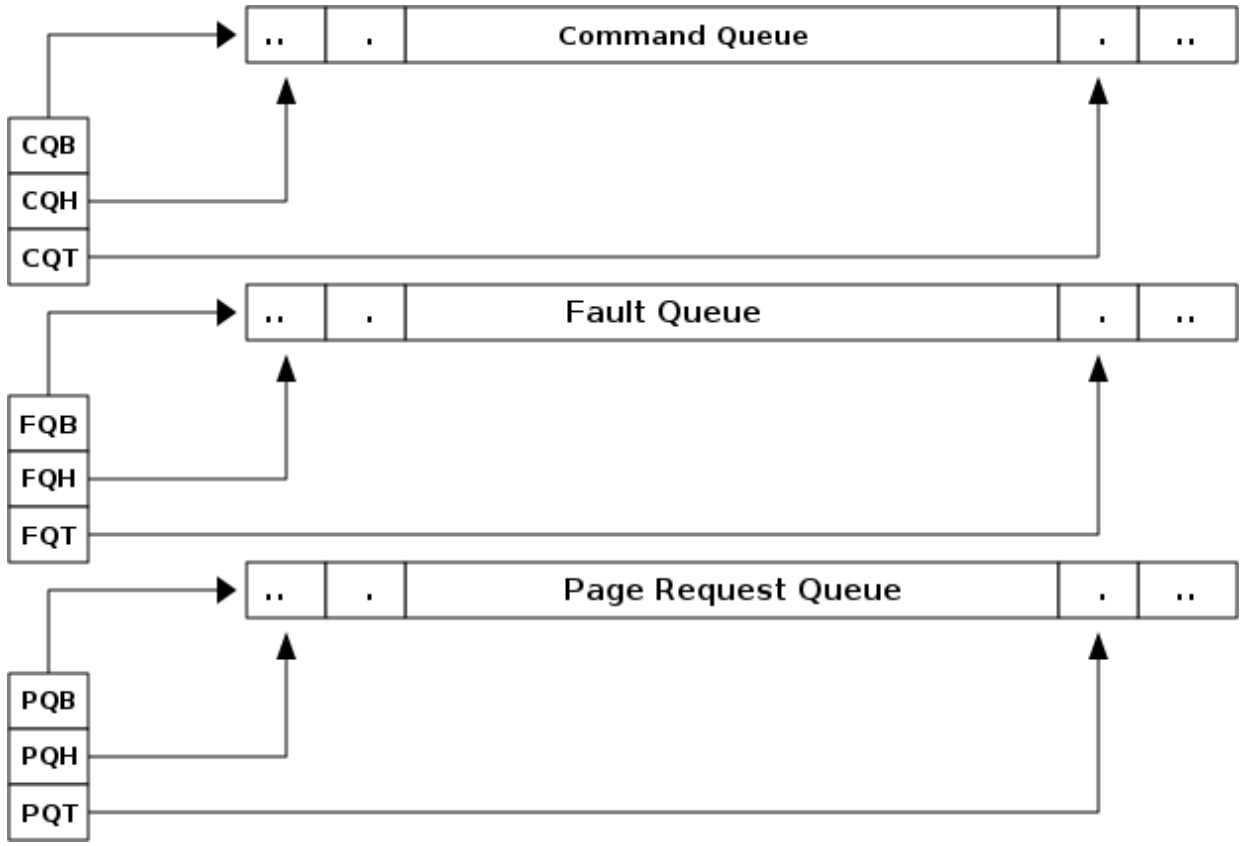


图 28.IOMMU 内存队列

每个队列都是一个循环缓冲区，头部由队列中的数据消费者控制，尾部由队列中的数据生产者控制。IOMMU 是 PQ 和 FQ 的记录生产者，控制着尾寄存器。IOMMU 是将软件生成的命令输入 CQ 的消费者，并控制头部寄存器。尾寄存器保存队列中的索引，生产者将在队列中写入下一条记录。头部寄存器保存队列中的索引，消费者将从这里读取下一个要处理的条目。

如果队列首等于队列尾，则队列为空。如果尾部是头部减一，则队列为满。当队列头和队列尾到达循环缓冲区的末端时，他们会回到缓冲区的起始端。

数据生产者必须确保写入队列的数据和尾部更新是有序的，以便观察尾部寄存器更新的消费者也能观察到在头部和尾部确定的偏移量之间产生的所有数据。

所有 RISC-V IOMMU 实现都必须支持位于主内存中的内存队列。本规范不要求但不禁止在 I/O 内存中支持内存队列。

3.1. 命令队列（Command-Queue，CQ）

命令队列用于软件排队等待 IOMMU 处理的命令。每条命令有 16 个字节。

该内存队列基地址的 PPN 和队列的大小被配置到一个名为命令队列基地址（cqb）的内存映射寄存器中。

命令队列尾部位于一个由软件控制的读/写内存映射寄存器中，该寄存器称为命令队列尾部（cqt）。cqt 是软件将写入的下一个命令队列条目的索引。在写入命令后，软件会按照写入命令的数量将 cqt 向前推进。

命令队列的首部位于一个名为命令队列首部（cqh）的只读内存映射 IOMMU 控制的寄存器中。cqh 是 IOMMU 下一步要处理的命令队列的索引。读取每条命令后，IOMMU 可以将 cqh 推进 1。如果 cqt == (cqh - 1)，则命令队列已满。

当错误位或 cqcsr 中的 fence_w_ip 位为 1 时，如果启用了命令队列中断（即 cqcsr.cie 为 1），则在 ipsr 中设置命令队列中断待处理 (cip) 位。

IOMMU 命令由操作码决定归入一个主要命令组，每个命令组中的 func3 字段指定该命令调用的功能。操作码定义了操作数字段的格式。其中一个或多个字段可能会被调用的特定函数使用。操作码编码 64 至 127 指定为自定义使用。

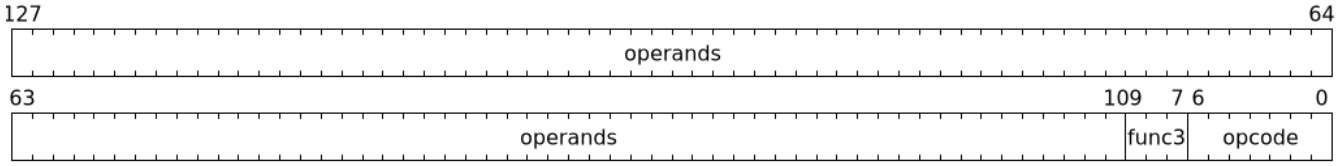


图 29.IOMMU 命令格式

命令被解释为两个 64 位双字。内存中每个双字的字节顺序（小端或大端）由 fctl.BE 确定（第 5.4 节）。

定义了以下命令操作码：

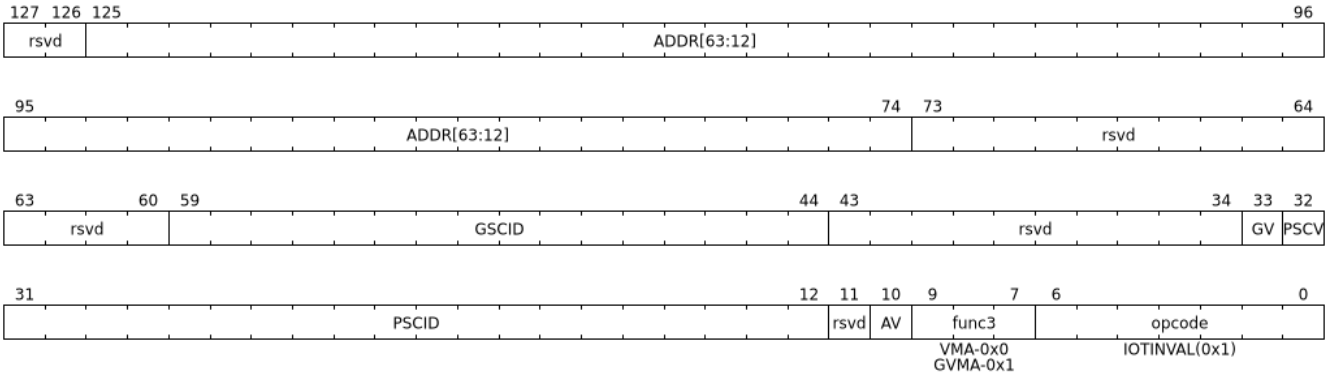
表 8.IOMMU 命令操作码

操作码	编码	说明
IOTINVAL	1	IOMMU 页表缓存失效命令。
IOFENCE	2	IOMMU 命令队列围栏命令。
IOTDIR	3	IOMMU 目录缓存失效命令。
ATS	4	IOMMU PCIe [1] ATS 命令。
保留	5-63	预留给未来标准使用。
定制	64-127	指定用于定制用途。

命令操作码 0 至 63 的所有未定义功能均保留供未来标准使用。

如果一条命令使用了保留编码或保留位被置 1，则该命令为非法命令。根据 IOMMU 能力（capabilities）寄存器的判断，如果命令已定义但未执行，则为不支持命令。如果命令队列获取并解码了非法或不支持的命令，则命令队列会设置 cqcsr.cmd_ill 位，并停止处理来自命令队列的命令。要重新启用命令处理功能，软件应将 cmd_ill 位写入 1，以清除该位。

3.1.1. IOMMU 页表缓存失效命令（IOMMU Page-Table cache invalidation commands）



IOMMU 操作会导致对 PDT、第一阶段和第二阶段页表的隐式读取。为减少此类读取的延迟，IOMMU 可能会在 IOMMU 地址转换缓存（IOATC）中缓存来自第一阶段和/或第二阶段页表的条目。这些缓存可能无法观察到软件对内存中这些数据结构所做的修改。

IOMMU 转换表缓存失效命令 IOTINVAL.VMA 和 IOTINVAL.GVMA 分别将内存中第一阶段和第二阶段页表数据结构的更新与 IOMMU 的操作同步，并使匹配的 IOATC 条目失效。

GV 操作数表示 Guest 软件上下文 ID (GSCID) 操作数是否有效。PSCV 操作数表示进程软件上下文 ID (PSCID) 操作数是否有效。只有 IOTINVAL.VMA 允许将 PSCV 设为 1。AV 操作数指示地址 (ADDR) 操作数是否有效。当 GV 为 0 时，将对与主机相关的译码（即第二阶段为 Bare 的译码）进行操作。GV 为 0 时，GSCID 操作数被忽略。当 AV 为 0 时，ADDR 操作数被忽略。当 PSCV 操作数为 0 时，PSCID 操作数被忽略。

IOTINVAL.VMA 可确保 IOMMU 在从 IOMMU 向相应的第一阶段页表进行所有后续隐式读取之前，观察到在这之前 hart 对第一阶段页表的存储。

表 9.IOTINVAL.VMA 操作数和操作

GV	AV	PSCV	运行
0	0	0	使所有主机地址空间的所有地址转换缓存项无效，包括包含全局映射的缓存项。
0	0	1	使 PSCID 操作符标识的主机地址空间的所有地址转换缓存条目失效，但包含全局映射的条目除外。
0	1	0	使包含第一阶段叶子页表项的所有地址转换缓存项无效，包括包含全局映射的缓存项，与所有主机地址空间的 ADDR 操作数中的 IOVA 相对应。
0	1	1	使包含与 ADDR 操作符中 IOVA 对应的第一阶段叶子页表项且与 PSCID 操作符标识的主机地址空间相匹配的所有地址转换缓存项无效，包含全局映射的项除外。
1	0	0	使与 GSCID 操作符相关的所有虚拟机地址空间的所有地址转换缓存项（包括包含全局映射的缓存项）失效。

GV	AV	PSCV	运行
1	0	1	使 PSCID 和 GSCID 操作数标识的虚拟机地址空间的所有地址转换缓存条目失效，但包含全局映射的条目除外。
1	1	0	对于与 GSCID 操作数关联的所有虚拟机地址空间，使包含第一级叶页表表项的所有地址转换缓存条目无效，包括包含全局映射的条目，这些条目对应于 ADDR 操作数中的 IOVA。
1	1	1	针对 PSCID 和 GSCID 操作数标识的虚拟机地址空间，使包含与 ADDR 操作数中 IOVA 相对应的第一阶段叶子页表项的所有地址转换缓存项无效，包含全局映射的项除外。

IOTINVAL.GVMA 可确保在从 IOMMU 向相应的第二阶段页表进行所有后续隐式读取之前，观察到先前对第二阶段页表的存储。使用 IOTINVAL.GVMA 将 PSCV 设置为 1 是非法的。

表 10.IOTINVAL.GVMA 操作数和操作

GV	AV	运行
0	忽略	使所有虚拟机地址空间的第二阶段页表中任何一级缓存的信息失效。
1	0	使从第二阶段页表任何级别缓存的信息失效，但仅限于 GSCID 操作符标识的虚拟机地址空间。
1	1	仅针对 GSCID 操作数中标识的虚拟机地址空间，使 ADDR 操作数中与 Guest 物理地址相对应的第二阶段页表表项中缓存的信息失效。

从概念上讲，IOMMU的具体实现可能包含两个地址转换缓存：一个是将 guest 虚拟地址映射到 guest 物理地址，另一个是将 guest 物理地址映射到 supervisor 物理地址。IOTINVAL.GVMA 不需要使前一个缓存失效，但必须使后一个缓存中与 IOTINVAL.GVMA 地址和 GSCID 操作数相匹配的条目失效。

更常见的情况是，实现包含地址转换缓存，可将 guest 虚拟地址直接映射到 supervisor 物理地址，从而消除一层间接性。对于此类实现，其 guest 虚拟地址映射到 guest 物理地址的任何条目，如果与 IOTINVAL.GVMA 地址和 GSCID 参数相匹配，都必须作废。要以这种方式有选择地使条目失效，需要用 guest 物理地址对它们进行标记，而这样做的成本很高，因此一种常用的技术是使所有与 GSCID 参数相匹配的条目失效，而不管地址参数是什么。

较简单的实现可以忽略 IOTINVAL.VMA 和/或 IOTINVAL.GVMA 的操作数，并始终对所有地址转换条目执行全局失效。

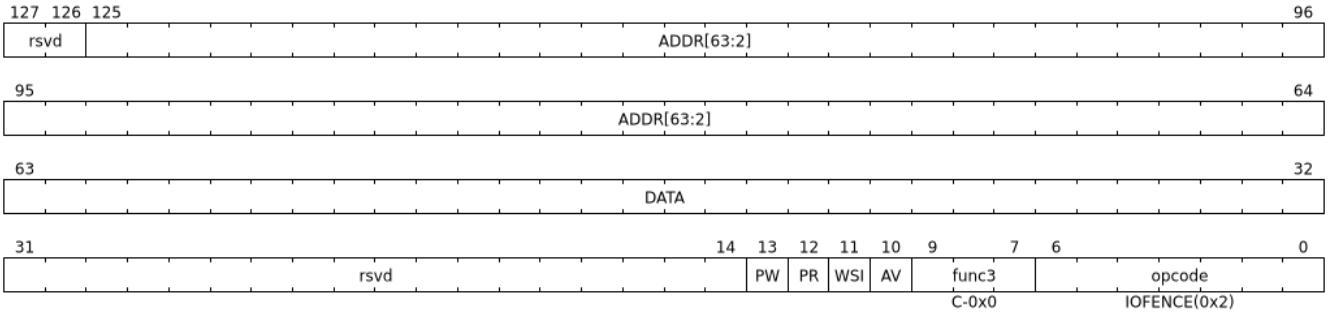
本规范的一个结果是，实现者可以使用自最近一次 IOTINVAL 以来任何时间有效的任何地址转换，而该地址的 IOTINVAL 是包含该地址的。特别是，如果修改了叶子 PTE，但未执行包含该地址的 IOTINVAL，则将使用旧的转换或新的转换，但选择是不可预测的。在其他方面，该行为是明确定义的。

在传统的 TLB 设计中，多个条目有可能与单个地址相匹配，例如，如果页面升级到更大的页面，而没有首先清除原始非叶 PTE 的有效位，并在 AV=0 的情况下执行 IOTINVAL.VMA 或 IOTINVAL.GVMA（如适用）。

该规范的另一个结果是，使用一组宽度小于 PTE 宽度的存储空间来更新 PTE 通常是不安

全的，因为实现可以在任何时候读取 PTE，包括只有部分存储空间生效的时候。

3.1.2. IOMMU 命令队列栅栏命令（IOMMU Command-queue Fence commands）



IOMMU 按顺序从 CQ 获取命令，但 IOMMU 可能会不按顺序执行所获取的命令。IOMMU 推进 cqh 并不保证 IOMMU 抓取的命令已被执行或提交。

IOFENCE.C 命令的完成（由 cqh 经过 CQ 中 IOFENCE.C 命令的索引确定）保证了从 CQ 获取的所有先前命令都已完成并提交。

如果 IOFENCE.C 等待指定超时的前一条命令完成时超时，则设置 cqcsr 第 5.15 节中的 cmd_to 位，以提示此情况。cqh 保存超时的 IOFENCE.C 的索引，所有未指定超时的先前命令均已完成并提交。

在此版本的规范中，只有 ATS.INVAL 命令规定了超时时间。

这些命令可用于对连接到 IOMMU 的 I/O 设备、其他 RISC-V Harts 以及外部设备或协处理器的内存访问进行排序。

PR 位设置为 1 时，可用于请求 IOMMU 确保将已由 IOMMU 处理过的设备的所有先前读取请求提交到全局排序点，以便系统中的所有 RISC-V Harts 和 IOMMU 都能观察到这些请求。

PW 位设置为 1 时，可用于请求 IOMMU 确保将已由 IOMMU 处理过的来自设备的所有先前写入请求提交到全局排序点，以便系统中的所有 RISC-V Harts 和 IOMMU 都能观察到它们。

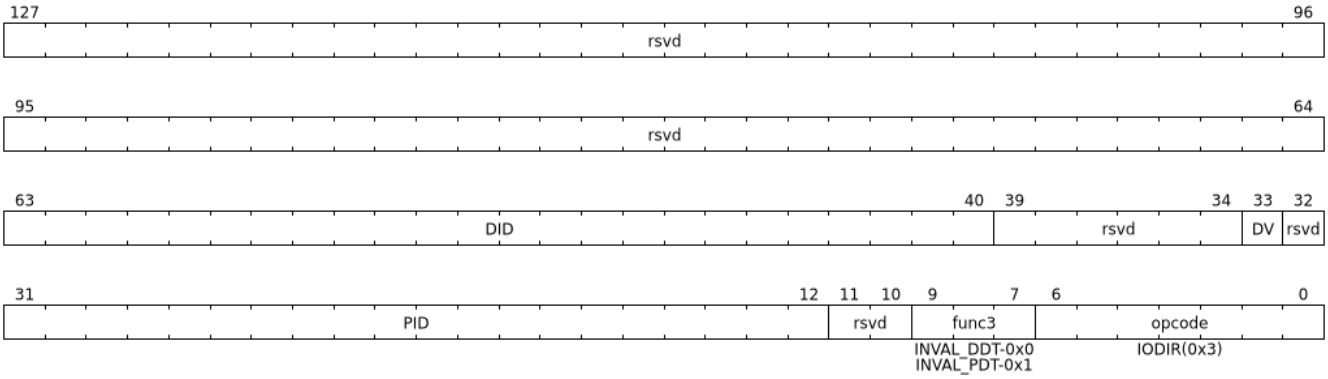
将有线信号中断（WSI）位设为 1 时，将在 IOFENCE.C 完成时从命令队列（通过设置 cqcsr.fence_w_ip - 第 5.15 节）产生一个有线中断。如果 IOMMU 不支持有线中断或未启用有线中断（即 fctl.WSI ==0），则保留该位。

> 软件应确保 IOMMU 先前处理的所有读写操作都已提交到全局排序点，然后再回收先前允许设备访问的内存。回收内存的安全顺序是首先更新页表，禁止设备访问内存，然后适当使用 IOTINVAL.VMA 或 IOTINVAL.GVMA 使 IOMMU 与页表更新同步。作为同步工作的一部分，如果回收的内存先前可被设备读取，则要求对所有先前的读取进行排序；否则，如果回收的内存先前可被设备写入，则要求对所有先前的读取和写入进行排序。如果回收的内存将用于保存设备不可见的数据，则可能需要对之前的读取进行排序。
>
> PR 和/或 PW 设置为 1 的 IOFENCE.C 只能确保 IOMMU 已处理的请求提交到全局排序点。如果需要确保 IOMMU 尚未处理的设备发出的所有飞行中请求得到遵守，软件必须执行特定于互连的栅栏操作。例如，对于 PCIe，从设备内存读取数据时，设备发出的完成指令具有确保 IOMMU 观察到之前发布的写操作的特性，因为完成指令可能不会通过之前发布的写操作。
>
> 排序保证是针对主内存的访问做出的。对于 I/O 内存的访问，排序保证由具体实现和 I/O 协议定义。
>
> 较简单的实现方式可能会无条件地对之前的所有内存访问进行全局排序。

AV 命令操作数表示 ADDR[63:2] 操作数和 DATA 操作数是否有效。如果 AV=1 ，则 IOMMU 在命令完成时将 DATA 以 4 字节对齐地址 ADDR[63:2] * 4 作为 4 字节存储写入内存。当 AV 为 0 时，ADDR[63:2] 和 DATA 操作数被忽略。

软件可将 ADDR[63:2] 命令操作数配置为指定 IMSIC 中 seteipnum_le/seteipnum_be 寄存器的地址，以便在 IOFENCE.C 完成时发出外部中断通知。或者，软件可将 ADDR[63:2] 编程到内存位置，并使用 IOFENCE.C 在内存中设置一个标志，指示命令完成。

3.1.3. IOMMU 目录缓存失效命令（IOMMU directory cache invalidation commands）



IOMMU 操作会导致对 DDT 和/或 PDT 的隐式读取。为减少此类读取的延迟，IOMMU 可能会在 IOMMU 目录缓存中缓存 DDT 和/或 PDT 的条目。这些缓存可能无法观察到软件对内存中这些数据结构所做的修改。

IOMMU DDT 缓存失效命令 IODIR.INVAL_DDT 将 DDT 的更新与 IOMMU 的操作同步，并刷新匹配的缓存条目。

IOMMU PDT 缓存失效命令 IODIR.INVAL_PDT 将 PDT 的更新与 IOMMU 的操作同步，并刷新匹配的缓存条目。

DV 操作数表示设备 ID (DID) 操作数是否有效。对于 IODIR.INVAL_PDT，DV 操作数必须为 1，否则该命令是非法的。当 DV 操作数为 1 时，DID 操作数的值不得大于 ddtp.iommu_mode 支持的值。

IODIR.INVAL_DDT 保证在从 IOMMU 到 DDT 的所有后续隐式读取之前，RISC-V hart 先前对 DDT 的任何存储都会被观察到。如果 DV 为 0，则该命令会使所有设备缓存的所有 DDT 和 PDT 表项无效；DID 操作数被忽略。如果 DV 为 1，则命令会使 DID 操作数标识的设备的缓存叶级 DDT 表项和所有相关 PDT 表项无效。在 IODIR.INVAL_DDT 命令中，PID 操作数是保留位。

IODIR.INVAL_PDT 保证，在从 IOMMU 向 PDT 进行所有后续隐式读取之前，RISC-V hart 先前向 PDT 进行的任何存储都会被观察到。该命令使指定 PID 和 DID 的缓存页 PDT 表项无效。IODIR.INVAL_PDT 的 PID 操作数不得大于 IOMMU 支持的宽度（参见第 5.3 节）。

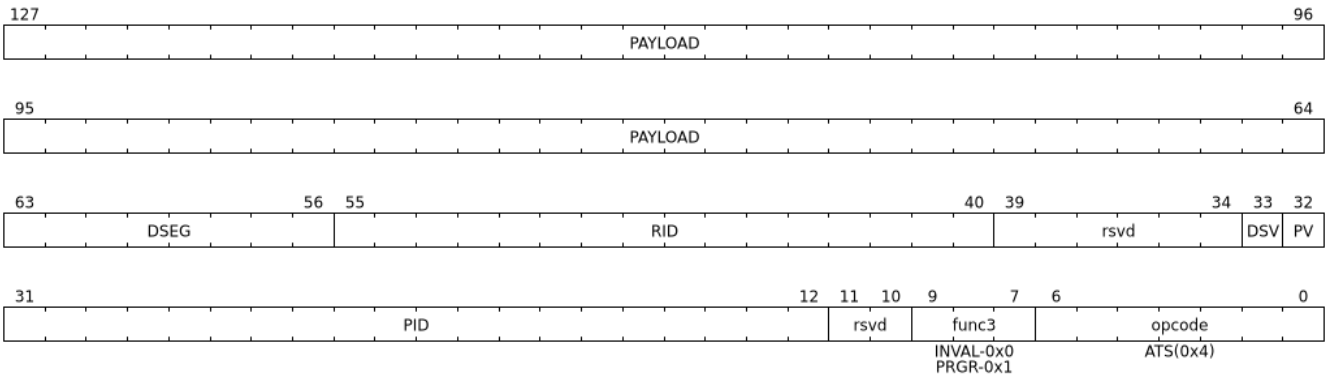
设备上下文（Device-context）或进程上下文（Process-context）中的某些字段可能是 guest 物理地址。缓存 "设备上下文 "或 "进程上下文 "时，可以先缓存这些字段，然后再将其转换为 supervisor 物理地址。其他实现可以将这些字段作为 guest 物理地址缓存，并在访问这些地址引用的内存之前，使用第二阶段页表将其转换为 supervisor 物理地址。

如果修改了用于这些转换的第二阶段页表，软件必须发出相应的 IODIR 命令，因为某些实现可能会选择在 IOMMU 目录缓存中缓存转换后的 supervisor 物理地址指针。

IOTINVAL 命令对 IOMMU 目录缓存没有影响。

3.1.4. IOMMU PCIe ATS 命令（IOMMU PCIe ATS commands）

如果 capabilities.ATS 设置为 1，则支持此命令。



ATS.INVAL 命令指示 IOMMU 向 RID 标识的 PCIe 设备 function 发送 “Invalidation Request”信息。“Invalidation Request”消息用于从设备 function 的地址转换缓存中清除地址范围的特定子集。当收到设备发送的“Invalidation Completion”响应信息或在等待响应时出现协议定义的超时，ATS.INVAL 命令完成。在等待响应期间，IOMMU 可将 cqh 推进并从 CQ 获取更多命令。如果发生超时，则在执行后续 IOFENCE.C 命令时报告。

需要知道设备上的失效操作是否完成的软件可使用 IOMMU 命令队列栅栏命令 (IOFENCE.C) 等待所有先前“Invalidation Request”消息的响应。IOFENCE.C 保证不会在所有先前获取的命令执行并完成之前完成。在请求超时或收到来自设备的有效响应之前，先前获取的使设备 ATC 失效的 ATS 命令不会完成。

如果 IOFENCE.C 之前的一条或多条 ATS 失效命令超时，则软件可使 CQ 重新运行，并重新提交可能超时的失效命令。如果在 IOFENCE.C 之前排队的 ATS.INVAL 命令针对多个设备，那么软件可以将这些命令作为 ATS.INVAL 和 IOFENCE.C 对重新提交，以识别导致超时的设备。

ATS.PRGR 命令指示 IOMMU 向 RID 标识的 PCIe 设备 function 发送“Page Request Group Response”信息。系统硬件和/或软件使用“Page Request Group Response”消息与设备功能的页面请求接口通信，以发出“Page Request”完成或接口发生灾难性故障的信号。

如果 PV 操作数设置为 1，则生成带有 PASID 的报文，PASID 字段设置为 PID 操作数。如果 PV 操作数设置为 0，则忽略 PID 操作数，生成不带 PASID 的报文。

命令的 PAYLOAD 操作数用于构成报文主体，其字段由 PCIe 规范 [1] 规定。PAYLOAD 字段格式如下：

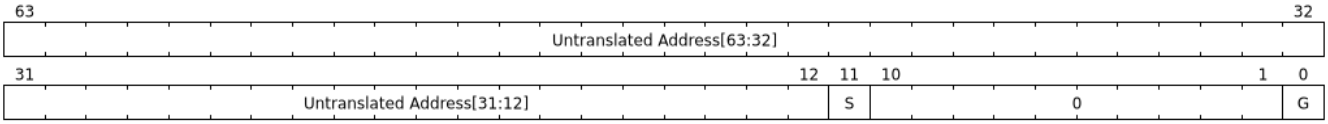


图 30.ATS.INVAL 命令的 PAYLOAD

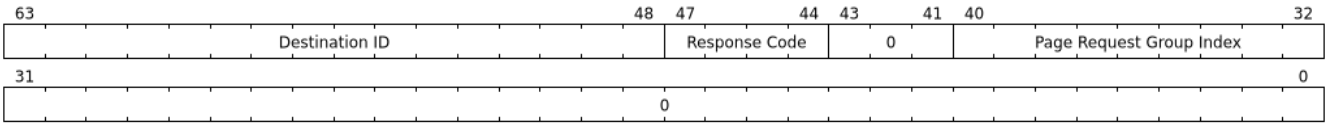


图 31.ATS.PRGR 命令的 PAYLOAD

如果 DSV 操作数为 1，则通过 DSEG 操作数指定有效的目标地址段编号。如果 DSV 操作数为 0，则忽略 DSEG 操作数。

层次结构是一种 PCI Express I/O 互连拓扑结构，其中的配置空间地址（总线/设备/功能编号的元组）是唯一的。在某些情况下，层次结构也被称为 "段"（Segment），在 Flit 模式

下，分段编号有时包含在 Function 的 ID 中。

3.2. 故障/事件队列（Fault/Event-Queue，FQ）

故障/事件队列是一种内存队列数据结构，用于报告处理事务时发生的事件和故障。每条故障记录为 32 字节。

该内存队列基地址的 PPN 和队列的大小被配置到一个名为故障队列基地址（fqb）的内存映射寄存器中。

故障队列的尾部位于 IOMMU 控制的只读内存映射寄存器 fqt 中，fqt 是 IOMMU 将写入故障队列的下一条故障记录的索引。写入记录后，IOMMU 会将 fqt 推进 1。 故障队列的头部位于一个名为 fqh 的读/写内存映射软件控制寄存器中。fqh 是 SW 下一步要处理的故障记录的索引。在处理完故障记录后，软件会按照已处理故障记录的数量推进 fqh。如果 fqh == fqt，则故障队列为空。如果 fqt == (fqh - 1)，则故障队列已满。

故障记录被解释为四个 64 位双字。内存中每个双字的字节顺序（小端或大端）由 fctl.BE 确定（第 5.4 节）。

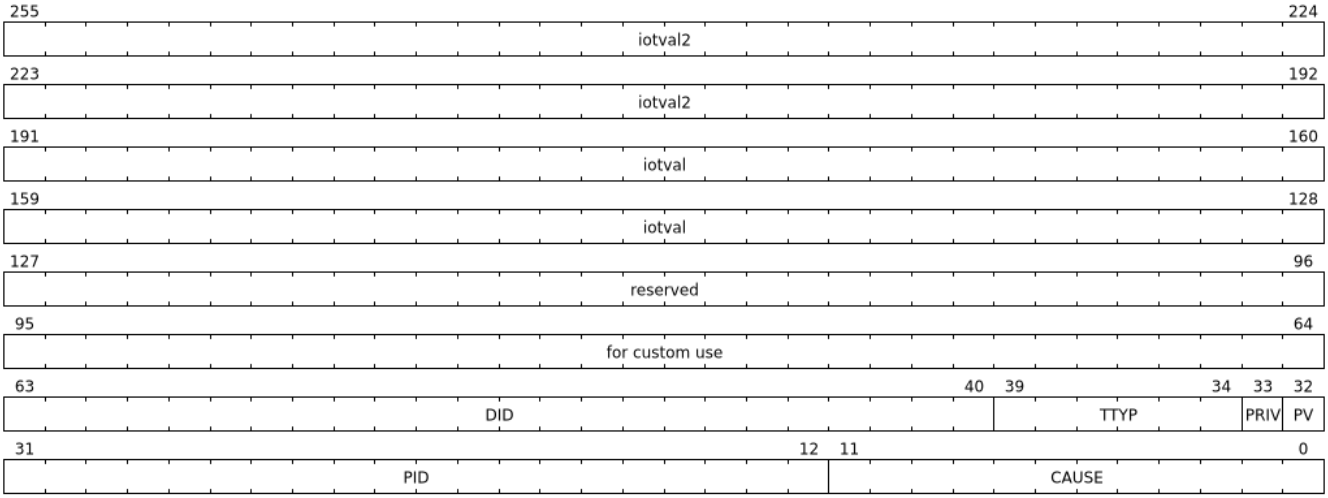


图 32. 故障队列记录

CAUSE 是表示故障/事件原因的代码。

表 11. 故障记录 CAUSE 字段编码

原因	说明	如果 DTF 是 1，是否报告？
1	指令访问故障（Instruction access fault）	No
4	读取地址非对齐（Read address misaligned）	No
5	读取访问故障（Read access fault）	No
6	写入/AMO 地址非对齐（Write/AMO address misaligned）	No
7	写入/AMO 访问故障（Write/AMO access fault）	No
12	指令页面故障（Instruction page fault）	No
13	读取页面故障（Read page fault）	No
15	写入/AMO 页故障（Write/AMO page fault）	No
20	指令 guest 页故障（Instruction guest page fault）	No
21	读取 guest 页故障（Read guest-page fault）	No
23	写入/AMO guest页面故障（Write/AMO guest-page fault）	No

原因	说明	如果 DTF 是 1，是否报告？
256	禁止所有入站事务（All inbound transactions disallowed）	Yes
257	DDT 表项加载访问故障（DDT entry load access fault）	Yes
258	DDT 表项无效（DDT entry not valid）	Yes
259	DDT 表项配置错误（DDT entry misconfigured）	Yes
260	不允许的事务类型（Transaction type disallowed）	No
261	MSI PTE 加载访问故障（MSI PTE load access fault）	No
262	MSI PTE 无效（MSI PTE not valid）	No
263	MSI PTE 配置错误（MSI PTE misconfigured）	No
264	MRIF 访问故障（MRIF access fault）	No
265	PDT 表项加载访问故障（PDT entry load access fault）	No
266	PDT 表项无效（PDT entry not valid）	No
267	PDT 表项配置错误（PDT entry misconfigured）	No
268	DDT 数据损坏（DDT data corruption）	Yes
269	PDT 数据损坏（PDT data corruption）	No
270	MSI PT 数据损坏（MSI PT data corruption）	No
271	MSI MRIF 数据损坏（MSI MRIF data corruption）	No
272	内部数据路径错误（Internal data path error）	Yes
273	IOMMU MSI 写访问故障（IOMMU MSI write access fault）	Yes
274	第一/第二阶段 PT 数据损坏（First/second-stage PT data corruption）	No

CAUSE 编码 275 至 2047 保留供未来标准使用，编码 2048 至 4095 供定制使用。表 11 未指定的 0 至 275 之间的编码保留给未来标准使用。

如果故障情况导致无法定位有效的设备上下文，则报告此类故障时假设的 DTF 值为 0。

TTYP 字段报告入站事务类型。

表 12. 故障记录 TTYP 字段编码

TTYP	说明
0	无。故障并非由入站事务引起。
1	非转换的读取执行事务
2	未转换的读取事务
3	未转换的写入/AMO 事务
4	保留
5	已转换的读取执行事务
6	已转换的读取事务
7	已转换的写入/AMO 事务
8	PCIe ATS 转换请求
9	PCIe Message 请求
10 - 31	保留
31 - 63	指定用于定制用途

如果 TTYP 是带有 IOVA 的事务，则在 iotval 中报告。如果 TTYP 是 PCIe Message 请求，则在 iotval 中报告消息代码。如果 TTYP 为 0，则 iotval 和 iotval2 字段中报告的值由 CAUSE 定义。

IOVA 被划分为虚拟页码（VPN）和页偏移。虚拟页码由地址转换过程转换成物理页码（PPN），而页面偏移则不是这个过程所需要的。在某些实现中，IO Bridge 可能不会向 IOMMU 提供 IOVA 的页面偏移量部分，IOMMU 可能会将 iotval 中的页面偏移量报告为 0，同样，IOMMU 可能会将 iotval2 中 GPA 的页面偏移量报告为 0。

DID 保存的是事务的 device_id。如果 PV 为 0，则 PID 和 PRIV 均为 0。 如果 PV 为 1，则 PID 保存事务的 process_id ，如果事务的特权等级为 Supervisor，则 PRIV 位为 1，否则为 0。 如果 TTYP 为 0，则 DID、PV、PID 和 PRIV 字段均为 0。

如果 CAUSE 是 guest 页故障，那么零扩展 guest 物理地址的第 63:2 位将在 iotval2[63:2] 中报告。如果 iotval2 的第 0 位为 1，则 guest 页故障是由第一阶段地址转换的隐式内存访问引起的。如果 iotval2 的第 0 位为 1，且隐式访问是写入，则 iotval2 的第 1 位被设置为 1，否则被设置为 0。

如果实现支持 A/D 位的硬件更新，且隐式内存访问试图自动更新第一阶段页表中的 A 和/或 D，则 iotval2 的位 1 将被设置。所有其他用于第一阶段地址转换的隐式内存访问都将被读取。如果未实现 A/D 位的硬件更新，则永远不会出现写入情况。

当第二阶段不是 Bare 阶段时，用于读取 PDT 表项以定位进程上下文的内存访问是用于第一阶段地址转换的隐式内存访问。如果读取 PDT 的隐式内存访问导致 guest 页故障条目，则 iotval2 的第 0 位报告为 1，第 1 位报告为 0。

由于故障队列已满或 IOMMU 在尝试访问队列内存时遇到访问故障等错误条件，IOMMU 可能无法通过故障队列报告故障。内存映射的故障控制和状态寄存器（fqcsr）保存有关此类故障的信息。如果检测到故障队列已满，IOMMU 将设置 fqcsr 中的故障队列溢出（fqof）位。如果 IOMMU 在访问故障队列内存时遇到故障，则 IOMMU 会设置 fqcsr 中的故障队列内存访问故障 (fqmf) 位。当 fqcsr 中的任一错误位被设置时，IOMMU 会丢弃导致故障的记录和所有其他故障记录。当 fqcsr 中的错误位为 1 或故障队列中产生新的故障记录时，如果启用了故障队列中断，即 fqcsr.fie 为 1，则在 ipsr 中设置故障中断待处理（fip）位。

IOMMU 可将多个请求识别为检测到相同故障。在这种情况下，IOMMU 可以单独报告每个故障，或报告一个子集（包括一个）请求的故障。

3.3. 页面请求队列（Page-Request-Queue，PQ）

页面请求队列是一种内存队列数据结构，用于向软件报告 PCIe ATS "页面请求 "和 "停止标记 "信息[1]。该内存队列的基准 PPN 和队列大小配置在一个名为页面请求队列基地址 (pqb) 的内存映射寄存器中。每个页面请求记录为 16 字节。

队列的尾部位于 IOMMU 控制的一个名为 pqt 的只读内存映射寄存器中，pqt 保存着队列的索引，IOMMU 将在该寄存器中写入下一条页面请求信息。写入信息后，IOMMU 会将 pqt 推进 1。

队列的首部位于一个名为 pqh 的软件控制读/写内存映射寄存器中。pqh 保存着队列的索引，软件将在队列中接收下一条页面请求信息。在处理信息后，软件会根据处理信息的数量将 pqh 推进。

如果 pqh == pqt，则页面请求队列为空。如果 pqt == (pqh - 1)，则页面请求队列已满。

由于队列被禁用、队列已满或 IOMMU 在尝试访问队列内存时遇到访问故障等错误条件，IOMMU 可能无法通过队列报告 "页面请求"信息。内存映射的页面请求队列控制和状态寄存器（pqcsr）用于保存有关此类故障的信息。当页面队列满时，pqcsr 中的页面请求队列溢出（pqof）位被置位。如果 IOMMU 在访问队列内存时遇到故障，则在 pqcsr 中设置页面请求-队列内存访问故障（pqmf）位。当 pqcsr 中的任一错误位被设置时，IOMMU 会丢弃所有后续的 "页面请求"报文，包括导致错误位被设置的报文。不需要响应的 "页面请求"报文（即 "Last Request in PRG "字段为 0 的报文）将被静默丢弃。需要响应的 "页面请求 "报文，即 "Last Request in PRG"（"PRG 中的最后请求"）字段设为 1 且不属于 "Stop Marker"（停止标记）报文的报文，可由 IOMMU 生成 "Page Request Group Response"（"页面请求组响应"）报文自动完成，详见第 2.7 节。

当 pqcsr 中的错误位为 1 或队列中产生新报文时，如果启用了来自页面请求队列的中断，即 pqcsr.pie 为 1，则在 ipsr 中设置页面请求队列中断待处理 (pip) 位。

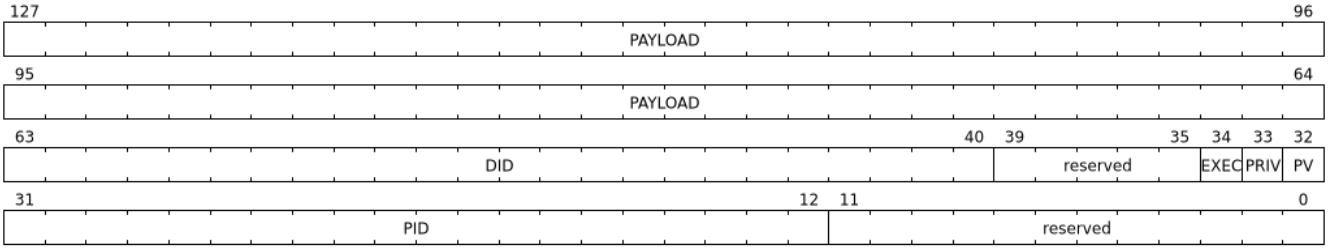


图 33. 页面-请求-队列记录

DID 字段包含报文中的请求者 ID。PID 字段在 PV 为 1 时有效，并报告报文中的 PASID。如果报文没有 PASID，则 PRIV 设为 0，否则将保留 TLP 中的 "请求的特权模式 "位。如果报文没有 PASID，EXEC 位被设为 0，否则它将报告 TLP 中的 "请求执行 "位。"页面请求"报文的有效负载（报文的 0x08 至 0x0F 字节）保存在 PAYLOAD 字段中。如果 R 和 W 均为 0，L 为 1，则报文为 "停止标记"。

页面请求队列记录被解释为两个 64 位双字。内存中每个双字的字节顺序是小端或大端，由 fctl.BE 确定（第 5.4 节）。

PAYLOAD 包含信息主体，其字段由 PCIe 规范 [1] 规定。PAYLOAD 字段格式如下：

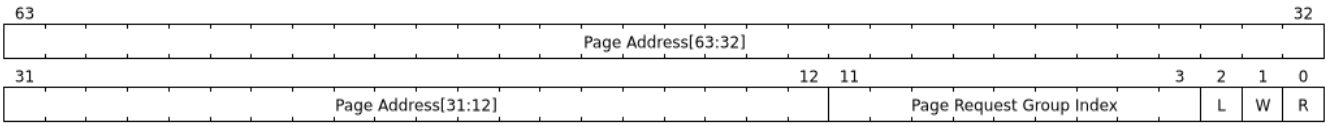


图 34.“ 页面请求”报文的 PAYLOAD

第 4 章 调试支持

为支持软件调试，IOMMU 可提供一个可选的寄存器接口，软件可使用该接口请求 IOMMU 执行地址转换。当 capabilities.DBG 为 1 时，IOMMU 支持该功能。该接口由两组寄存器组成：转换请求寄存器，用于软件编程 IOVA，以及将 IOVA（第 2.3 节）作为未转换请求转换所需的其他输入。如果进程成功完成，则通过转换响应寄存器报告转换结果。如果进程因故障而停止，则会在故障队列中正常报告故障，并用故障指示符更新转换响应寄存器。如果 IOVA 被确定为虚拟中断文件的 IOVA（第 2.1.3.6 节），且相应的 MSI PTE 处于 MRIF 模式，则进程停止并报告 "事务类型不允许"（原因 = 260）故障。

当为此目的调用 IOVA 转换过程时，IOMMU 可以缓存或不缓存 IOATC 中为转换过程访问的第一阶段 PTE、第二阶段 PTE、DDT 表项、PDT 表项或 MSI PTE。IOMMU 可使用 IOATC 中已缓存的任何 PTE 或目录结构条目。如果 IOMMU 支持，IOMMU 可更新用于转换过程的 PTE 中的 Accessed（A）和/或 Dirty（D）位。当 IOMMU 实现 HPM 时，HPM 计数器可由 IOMMU 正常更新。就 HPM 计数而言，这些请求被视为未转换请求。

转换请求接口由以下 64 位 WARL 寄存器组成：

- tr_req_iova（第 5.24 节）
- tr_req_ctl（第 5.25 节）

转换响应接口由一个 64 位 RO 寄存器 tr_response（第 5.26 节）组成。

要请求转换，首先要向寄存器 tr_req_iova 写入所需的 IOVA 值。接下来写入 tr_req_ctl 寄存器。tr_req_ctl 中的 "Go/Busy "位被置位，表示寄存器中的请求有效。Go/Busy "位是一个读写延迟（RWS）位，一旦被设置，将无法通过写入寄存器来清除。当进程完成（成功或遇到故障）时，IOMMU 将把 Go/Busy 位清零。当 Go/Busy 位从 1 变为 0 时，tr_response 寄存器中的响应有效。

如果出现以下情况，则 IOMMU 行为为 UNSPECIFIED：

- 当 Go/Busy 位为 1 时，tr_req_iova 或 tr_req_ctl 被修改。
- 修改 IOMMU 配置，如 ddtp.iommu_mode 等。

通过该调试接口完成转换请求的时间 UNSPECIFIED，但要求是有限的。如果通过该寄存器接口发出请求时，IOMMU 正在处理来自 IO Bridge 的转换请求，那么完成请求的时间可能会比 IOMMU 空闲时长。

调试接口是可选的，但建议实施调试接口，以帮助软件调试和实施架构符合性测试。

第 5 章 内存映射寄存器接口

IOMMU 提供内存映射编程接口。每个 IOMMU 的内存映射寄存器都位于物理地址空间中自然对齐的 4-KiB 区域（页）内。

如果寄存器地址与访问大小不对齐，或访问跨越多个寄存器，或访问大小不是 4 字节或 8 字节，则 IOMMU 的行为是 UNSPECIFIED。对 IOMMU 寄存器的 4 字节访问必须是单拷贝原子访问。对 IOMMU 寄存器的 8 字节访问是否为单次原子拷贝，则UNSPECIFIED，在 IOMMU 内部看来，这种访问可能是两次独立的 4 字节访问。

8 字节 IOMMU 寄存器的定义方式是，只要在两次软件访问或两次硬件事务处理之间遵守寄存器语义，不产生副作用，软件就可以对寄存器的高半部和低半部分别执行两次单独的 4 字节访问，或硬件就可以对 8 字节访问产生的两个独立的 4 字节事务处理。

IOMMU 寄存器采用小端字节序（即使在所有寄存器都是大端字节序的系统中也是如此）。

使用 IOMMU 的 Big-endian 配置的 Harts 应执行 Zbb 扩展定义的 REV8 字节反转指令。如果没有执行 REV8，则可以使用一系列指令来实现字节序转换。

如果某个寄存器为可选寄存器（由相应的 capabilities 寄存器位为 0 决定），那么从该寄存器的内存映射寄存器偏移读取将返回 0，而对该偏移的写入将被忽略。

5.1. 寄存器布局

表 13.IOMMU 内存映射寄存器布局

偏移	名称	尺寸	说明	是否可选？
0	capabilities	8	IOMMU 的能力	No
8	fctl	4	功能控制	No
12	*定制*	4	*指定 专用*	
16	ddtp	8	设备目录表指针	No
24	cqb	8	命令队列基地址	No
32	cqh	4	命令队列头	No
36	cqt	4	命令队列尾	No
40	fqb	8	故障队列基地址	No
48	fqh	4	故障队列头	No
52	fqt	4	故障队列尾	No
56	pqb	8	页面请求队列基地址	if capabilities.ATS==0
64	pqh	4	页面请求队列头	if capabilities.ATS==0
68	pqt	4	页面请求队列尾	if capabilities.ATS==0
72	cqcsr	4	命令队列 CSR	No
76	fqcsr	4	故障队列 CSR	No

偏移	名称	尺寸	说明	是否可选？
80	pqcsr	4	页面请求队列 CSR	if capabilities.ATS==0
84	ipsr	4	中断待处理状态寄存器	No
88	iocntovf	4	HPM 计数器溢出	if capabilities.HPM==0
92	iocntinh	4	HPM 计数器禁止	if capabilities.HPM==0
96	iohpmcycles	8	HPM 循环计数器	if capabilities.HPM==0
104	iohpmctr1-31	248	HPM 事件计数器	if capabilities.HPM==0
352	iohpmevt1-31	248	HPM 事件选择器	if capabilities.HPM==0
600	tr_req_iova	8	转换请求 IOVA	if capabilities.DBG==0
608	tr_req_ctl	8	转换请求控制	if capabilities.DBG==0
616	tr_response	8	转换请求响应	if capabilities.DBG==0
624	保留	64	留待今后使用（WPRI）	
688	*定制*	72	*指定为定制用途 (*WARL*)*	
760	icvec	8	矢量寄存器的中断原因	No
768	msi_cfg_tbl	256	MSI 配置表	if capabilities.IGS==WSI
1024	保留	3072	保留为标准用途	

5.2. 复位行为

以下寄存器字段的复位值为 0。

- cqcsr – cqen，cqie，cqon 和 busy
- fqcsr – fqen，fqie，fqon 和 busy
- pqcsr – pqen，pqie，pqon 和 busy
- tr_req_ctl.Go/Busy
- ddtp.busy

以下寄存器的复位值为 0。

- ipsr

ddtp.iommu_mode 字段的重置值必须为 Off 或 Bare。重置后，缓存（第 2.8 节）必须没有有效条目。

iommu_mode 的重置值建议为 Off。

所有其他寄存器和/或字段的重置值均为 UNSPECIFIED。

5.3. IOMMU 能力（capabilities）

Capabilities 寄存器是一个只读寄存器，报告 IOMMU 支持的功能。每个字段如果未清零，则表示 IOMMU 中存在该功能。复位时，寄存器将包含 IOMMU 支持的功能。

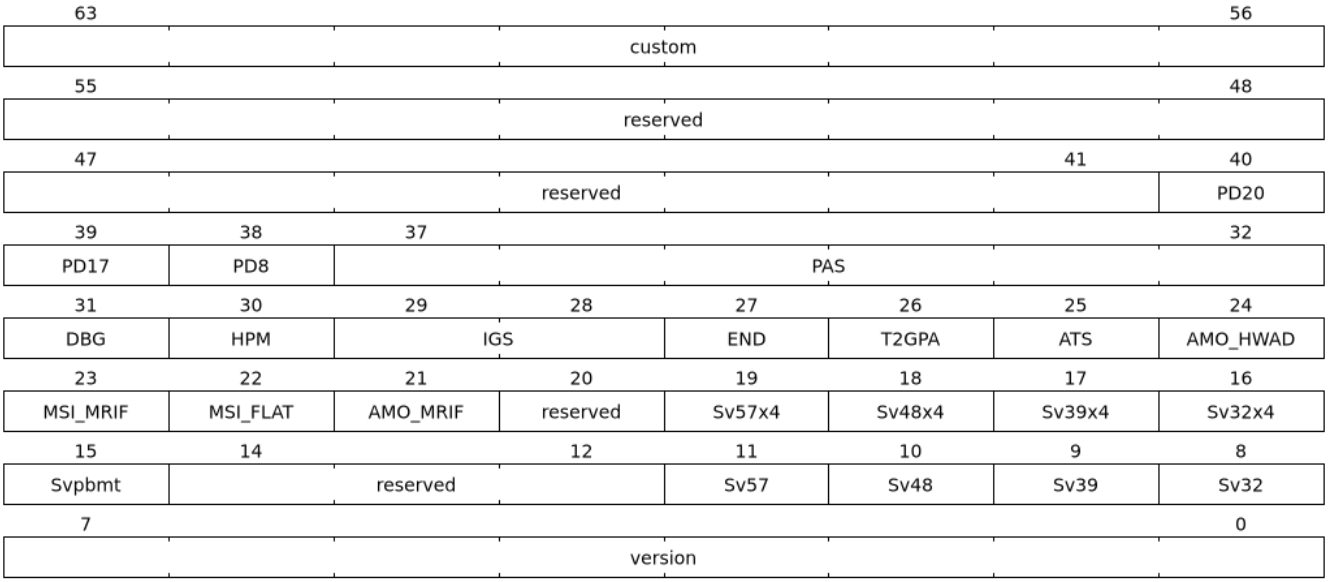


图 35.IOMMU 能力寄存器字段

位	名称	属性	说明
7:0	version	RO	保存 IOMMU 执行的规范版本。低位小数用于保存规范的次要版本，高位小数用于保存规范的主要版本。例如，支持 1.0 版规范的执行报告为 0x10。
8	Sv32	RO	支持基于页面的 32 位虚拟寻址。

<table>
<tr><th>位</th><th>名称</th><th>属性</th><th>说明</th></tr>
<tr><td>9</td><td>Sv39</td><td>RO</td><td>支持基于页面的 39 位虚拟寻址。</td></tr>
<tr><td>10</td><td>Sv48</td><td>RO</td><td>支持基于页面的 48 位虚拟寻址。设置 Sv48 时，必须设置 Sv39。</td></tr>
<tr><td>11</td><td>Sv57</td><td>RO</td><td>支持基于页面的 57 位虚拟寻址 设置 Sv57 时，必须设置 Sv48。</td></tr>
<tr><td>14:12</td><td>保留</td><td>RO</td><td>保留为标准用途。</td></tr>
<tr><td>15</td><td>Svpbmt</td><td>RO</td><td>基于页面的内存类型</td></tr>
<tr><td>16</td><td>Sv32x4</td><td>RO</td><td>支持基于页面的 34 位虚拟寻址，用于第二阶段地址转换。</td></tr>
<tr><td>17</td><td>Sv39x4</td><td>RO</td><td>支持基于页面的 41 位虚拟寻址，用于第二阶段地址转换。</td></tr>
<tr><td>18</td><td>Sv48x4</td><td>RO</td><td>支持基于页面的 50 位虚拟寻址，用于第二阶段地址转换。</td></tr>
<tr><td>19</td><td>Sv57x4</td><td>RO</td><td>支持用于第二阶段地址转换的基于页面的 59 位虚拟寻址。</td></tr>
<tr><td>20</td><td>保留</td><td>RO</td><td>保留为标准用途。</td></tr>
<tr><td>21</td><td>AMO_MRIF</td><td>RO</td><td>支持对 MRIF 进行原子更新。</td></tr>
<tr><td>22</td><td>MSI_FLAT</td><td>RO</td><td>支持使用 Pass-through 模式 MSI PTE 的 MSI 地址转换。</td></tr>
<tr><td>23</td><td>MSI_MRIF</td><td>RO</td><td>支持使用 MRIF 模式 MSI PTE 进行 MSI 地址转换。</td></tr>
<tr><td>24</td><td>AMO_HWAD</td><td>RO</td><td>支持对已访问的 PTE (A) 和 dirty (D) 位进行原子更新。</td></tr>
<tr><td>25</td><td>ATS</td><td>RO</td><td>支持 PCIe 地址转换服务（ATS）和页面请求接口（PRI）[1]。</td></tr>
<tr><td>26</td><td>T2GPA</td><td>RO</td><td>支持在 ATS 转换完成中返回 Guest 物理地址。</td></tr>
<tr><td>27</td><td>END</td><td>RO</td><td>0 时，IOMMU 支持一种字节序（小字节序或大字节序）。当为 1 时，IOMMU 支持两种字节序。
endianness 在 fctl 寄存器中定义。</td></tr>
<tr><td>29:28</td><td>IGS</td><td>RO</td><td>支持 IOMMU 中断生成
<table>
<tr><th>数值</th><th>名称</th><th>说明</th></tr>
<tr><td>0</td><td>MSI</td><td>IOMMU 仅支持消息信号中断生成。</td></tr>
<tr><td>1</td><td>WSI</td><td>IOMMU 仅支持有线信号中断生成。</td></tr>
<tr><td>2</td><td>BOTH</td><td>IOMMU 支持 MSI 和 WSI 生成。中断发生方式必须在 fctl 寄存器中定义。</td></tr>
<tr><td>3</td><td>0</td><td>保留为标准用途</td></tr>
</table></td></tr>
<tr><td>30</td><td>HPM</td><td>RO</td><td>IOMMU 实现了硬件性能监控器。</td></tr>
<tr><td>31</td><td>DBG</td><td>RO</td><td>IOMMU 支持转换请求接口</td></tr>
<tr><td>37:32</td><td>PAS</td><td>RO</td><td>IOMMU 支持的物理地址大小。</td></tr>
<tr><td>38</td><td>PD8</td><td>RO</td><td>单级 PDT，支持 8 位 process_id 。</td></tr>
<tr><td>39</td><td>PD17</td><td>RO</td><td>两级 PDT，支持 17 位 process_id 。</td></tr>
<tr><td>40</td><td>PD20</td><td>RO</td><td>三级 PDT，支持 20 位 process_id 。</td></tr>
<tr><td>55:41</td><td>保留</td><td>RO</td><td>保留为标准用途</td></tr>
<tr><td>63:56</td><td>定制</td><td>RO</td><td>指定用于定制用途</td></tr>
</table>

当 HPM 为 1 时，必须有至少 32 位宽的 iohpmcycles 和 iohpmctr1 寄存器。

必须至少支持一种从 IOMMU 产生中断的方法（MSI 或 WSI）。

IOMMU 实现必须支持 NAPOT 转换连续性的 Svnapot 标准扩展。

虚拟机 Hypervisor 可提供一个 SW 仿真 IOMMU，允许 Guest 管理第一阶段页表，以便对 Guest 控制设备访问的内存进行精细控制。

为 Guest 提供这种仿真 IOMMU 的 Hypervisor 可以保留对第二阶段地址转换的控制，并清除仿真 capabilities 寄存器的 SvNx4 字段。

为 Guest 提供这种仿真 IOMMU 的 Hypervisor 可以保留 MSI 页表的控制权，用于将 MSI 引导到 IMSIC 中的 Guest 中断文件或内存驻留中断文件，并清除仿真 capabilities 寄存器的 MSI_FLAT 和 MSI_MRIF 字段。

AMO_HWAD/AMO_MRIF 位不表示支持设备启动的原子内存操作。必须通过其他方式才能发现对设备启动的原子内存操作的支持。

IOMMU 的设计目的是提供一套高度模块化和可扩展的功能，允许实现者只包含应用所需的确切功能。此外，实现者还可以向 IOMMU 添加自己的自定义扩展。

IOMMU 必须支持系统中任何硬盘所支持的所有虚拟内存扩展。

RISC-V 平台规范可能会强制要求实施方案必须提供一组 IOMMU 功能，以符合这些规范的要求。

5.4. 功能控制寄存器（fctl）

该寄存器必须是可读的。实现过程中可以允许寄存器中的一个或多个字段可写，以支持启用或禁用该字段控制的功能。

如果软件在 IOMMU 未关闭（即 ddtp.iommu_mode != Off）时启用或禁用了某项功能，则 IOMMU 的行为将是 UNSPECIFIED。

如果软件在启用 IOMMU 内存队列时启用或禁用了某项功能（即 cqcsr.cqon/cqen == 1、fqcsr.fqon/cqen == 1 或 pqcsr.pqon/pqen == 1），则 IOMMU 行为为 UNSPECIFIED。

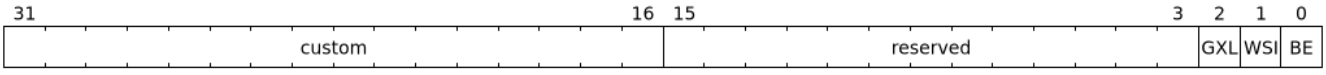


图 36. 功能控制寄存器字段

位	名称	属性	说明
0	BE	WARL	当为 0 时，IOMMU 对内存驻留数据结构的访问（如表 7 所示）以及对内存队列的访问均以小端访问方式进行，而当为 1 时，则以大端访问方式进行。
1	WSI	WARL	当为 1 时，IOMMU 中断作为有线信号中断发出信号，否则作为报文信号中断发出信号。

2	GXL	WARL	控制可用于 Guest 物理地址的地址转换方案，如表 2 所定义。
15:3	reserved	WPRI	保留为标准用途。
31:16	*custom*	WPRI	*指定用于定制用途。*

5.5. 设备目录表指针 (ddtp)

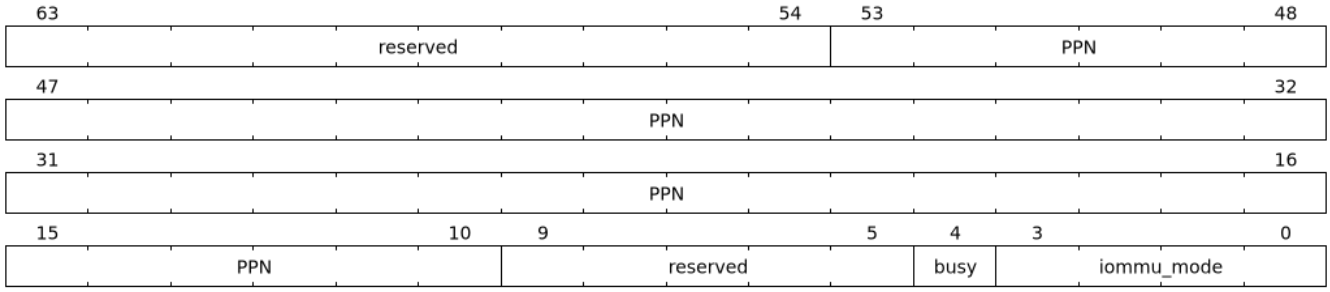


图 37. 设备目录表指针寄存器字段

位	名称	属性	说明
3:0	iommu_mode	WARL	IOMMU 可配置为以下模式： 数值 / 名称 / 说明 0 / Off / IOMMU 不允许进行入站内存事务。 1 / Bare / 没有翻译或保护。所有入站内存访问都会通过。 2 / 1LVL / 单级设备目录表 3 / 2LVL / 两级设备目录表 4 / 3LVL / 三级设备目录表 5-13 / 保留 / 保留为标准用途。 14-15 / 定制 / 指定用于定制用途。
4	busy	RO	写入 ddtp 可能需要 IOMMU 执行许多操作，这些操作可能不会与写入同步进行。当 ddtp 观察到一次写入时，忙位被置 1。当忙位为 1 时，对 ddtp 的其他写入行为是未知的。某些实现可能会忽略第二次写入，而其他实现可能会执行第二次写入所决定的操作。在写入 ddtp 之前，软件必须验证忙位是否为 0。 如果忙位读数为 0，则 IOMMU 已完成与上次向 ddtp 写入相关的操作。 可同步完成这些操作的 IOMMU 可将该位硬连线为 0。
9:5	保留	WPRI	保留为标准用途
53:10	PPN	WARL	保存设备目录表根页面的 PPN。
63:54	保留	WPRI	保留为标准用途

如果 capabilities.MSI_FLAT 为 1，设备上下文大小为 64 字节，否则为 32 字节。

当 iommu_mode 为 Bare 或 Off 时，PPN 字段为 don't-care 字段。在 "Bare" 模式下，只有未转换请求是允许

的。不支持已转换请求、转换请求和 PCIe 消息事务。

所有 IOMMU 必须支持 Off 和 Bare 模式。允许 IOMMU 支持目录表级别和设备上下文宽度的子集。至少必须支持其中一种模式。

当 IOMMU 的 iommu_mode 字段值更改为 Off 时，IOMMU 保证连接到 IOMMU 的设备的处理中的（in-flight）事务将使用适用于 iommu_mode 字段旧值的配置进行处理，并且 IOMMU 已经处理过的所有事务和来自设备的先前请求将提交到全局排序点，以便平台中的所有 RISC-V 硬件、设备和 IOMMU 都能观察到它们。

当 IOMMU 的 iommu_mode 先前值不是 Off 或 Bare 时，IOMMU 将 iommu_mode 写入 1LVL、2LVL 或 3LVL 的行为是 UNSPECIFIED。要更改 DDT 级别，必须先将 IOMMU 转换到 Bare 状态或 Off 状态。

当 IOMMU 过渡到 "Off" 状态时，IOMMU 可能会保留从内存数据结构（如页表、DDT、PDT 等）缓存的信息。软件必须使用适当的失效命令来失效缓存条目。

在 RV32 中，只需写入寄存器的低阶 32 位（22 位 PPN 和 4 位 iommu_mode）。

5.6. 命令队列基地址（cqb）

这个 64 位寄存器（RW）保存命令队列根页面的 PPN 和队列中的条目数。每个命令为 16 字节。

当 cqcsr.busy 或 cqon 位为 1 时，IOMMU 写入 cqb 的行为是未知的。软件建议的更改 cqb 顺序是，首先通过清除 cqen 禁用命令队列，并等待 cqcsr.busy 和 cqon 均为 0 后再更改 cqb。写入 cqb 后，cqt 中的第 31:cqb.LOG2SZ 位状态为 0，cqt 中的第 cqb.LOG2SZ-1:0 位假设为有效值，否则为 UNSPECIFIED 值。

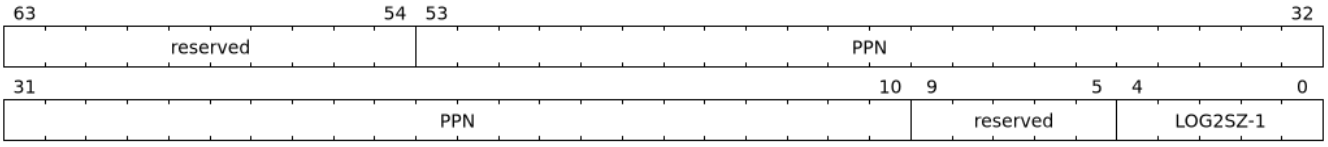


图 38. 命令队列基础寄存器字段

位	名称	属性	说明
4:0	LOG2SZ-1	WARL	LOG2SZ-1 字段表示命令队列中以 2 为底减去 1 的对数的条目数。数值为 0 表示队列中有 2 个条目。每条 IOMMU 命令为 16 字节。如果命令队列的条目数为 256 或更少，则队列的基地址始终对齐为 4-KiB。如果命令队列的条目数超过 256 个，则命令队列的基地址必须自然对齐为 2^{LOG2SZ} x 16。
9:5	保留	WPRI	保留为标准用途
53:10	PPN	WARL	保存内存中命令队列根页面的 PPN，用于软件向 IOMMU 发送队列命令。如果由 PPN 确定的基地址未按要求对齐，则队列中的所有条目在 IOMMU 看来都是 UNSPECIFIED，IOMMU 计算并用于访问队列中的条目的任何地址也是 UNSPECIFIED。
63:54	保留	WPRI	保留为标准用途

在 RV32 中，只需写入寄存器的低阶 32 位（22 位 PPN 和 5 位 LOG2SZ-1）。

5.7. 命令队列头（cqh）

这个 32 位寄存器（RO）保存着 IOMMU 将获取下一条命令的命令队列索引。

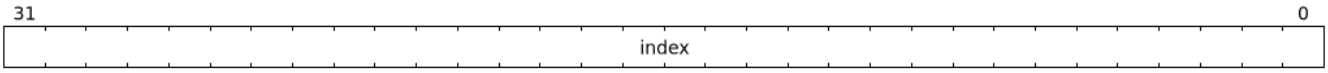


图 39. 命令队列头寄存器字段

位	名称	属性	说明
31:0	index	RO	保存 IOMMU 获取下一条命令的命令队列索引。

5.8. 命令队列尾（cqt）

这个 32 位寄存器（RW）保存命令队列的索引，软件在此排列 IOMMU 的下一条命令。

图 40. 命令队列尾寄存器字段

位	名称	属性	说明
31:0	index	WARL	保存命令队列中的索引，软件在此队列中为 IOMMU 保存下一条命令。只有 LOG2SZ-1:0 位可写。

5.9. 故障队列基地址 (fqb)

这个 64 位寄存器（RW）保存故障队列根页面的 PPN 和队列中的条目数。每条故障记录为 32 字节。

当 fqcsr.busy 或 fqon 位为 1 时，IOMMU 在写入 fqb 时的行为是未知的。软件建议的更改 fqb 顺序是，首先通过清除 fqen 禁用故障队列，然后等待 fqcsr.busy 和 fqon 均为 0 后再更改 fqb。写入 fqb 后，fqh 中 31:fqb.LOG2SZ 位的状态为 0，fqh 中 fqb.LOG2SZ-1:0 位的值为有效值，否则为UNSPECIFIED。

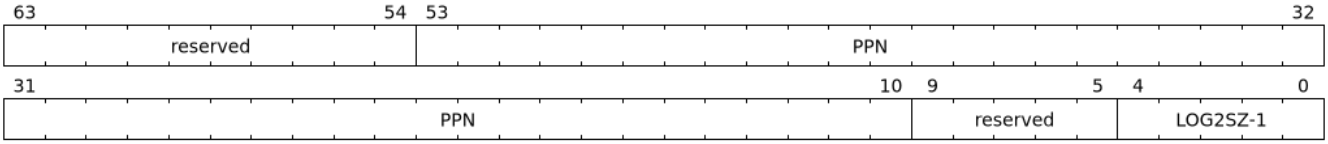


图 41. 故障队列基础寄存器字段

位	名称	属性	说明
4:0	LOG2SZ-1	WARL	LOG2SZ-1 字段表示故障队列中的条目数，即log-to-base-2 减 1。值为 0 表示队列中有 2 个条目。每条故障记录为 32 字节。如果故障队列的条目数为 128 或更少，则队列的基地址始终对齐为 4-KiB。如果故障队列的条目数超过 128 个，则故障队列的基地址必须自然对齐为 $2^{LOG2SZ} \times 32$。
9:5	保留	WPRI	保留为标准用途
53:10	PPN	WARL	保存 IOMMU 用来排列故障记录的内存故障队列根页面的 PPN。如果由 PPN 确定的基地址未按要求对齐，则队列中的所有条目在 IOMMU 看来都是 UNSPECIFIED，而且 IOMMU 可能计算并用于访问队列中的条目的任何地址也是 UNSPECIFIED。
63:54	保留	WPRI	保留为标准用途

在 RV32 中，只有寄存器的低阶 32 位（22 位 PPN 和 5 位 LOG2SZ-1）需要编写。

5.10. 故障队列头 (fqh)

这个 32 位寄存器 (RW) 保存着故障队列的索引，软件将从该索引中获取下一条故障记录。

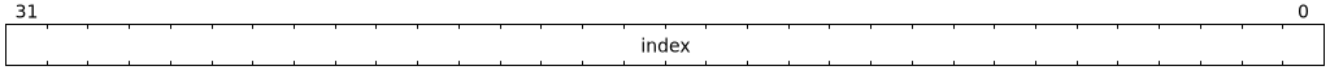


图 42. 故障队列头部寄存器字段

位	名称	属性	说明
31:0	index	WARL	保存故障队列的索引，软件从中读取下一条故障记录。只有 LOG2SZ-1:0 位可写。

5.11. 故障队列尾（fqt）

这个 32 位寄存器（RO）保存故障队列的索引，IOMMU 在此队列中排列下一条故障记录。

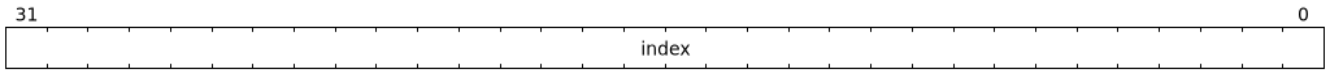


图 43. 故障队列尾寄存器字段

位	名称	属性	说明
31:0	索引	RO	保存 IOMMU 写入下一条故障记录的故障队列索引。

5.12. 页面请求队列基地址（pqb）

这个 64 位寄存器 (WARL) 保存着页面请求队列根页面的 PPN 和队列中的条目数。每个 "页面请求 "信息为 16 个字节。

当 pqcsr.busy 或 pqon 位为 1 时，IOMMU 写 pqb 的行为是未知的。软件建议的 pqb 更改顺序是，首先通过清除 pqen 禁用页面请求队列，然后等待 pqcsr.busy 和 pqon 均为 0 后再更改 pqb。写入 pqb 后，pqh 中的第 31:pqb.LOG2SZ 位状态为 0，pqh 中的第 pqb.LOG2SZ-1:0 位假设为有效值，否则为未知值。

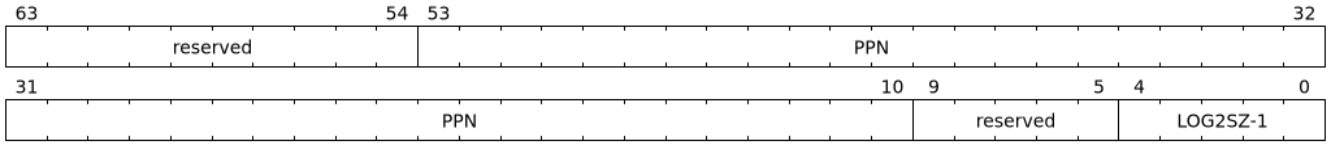


图 44. 页面请求队列基础寄存器字段

位	名称	属性	说明
4:0	LOG2SZ-1	WARL	LOG2SZ-1 字段表示页面请求队列中的条目数，即 log-to-base-2 减 1。值为 0 表示队列中有 2 个条目。每个页面请求为 16 字节。如果页面请求队列的条目数为 256 或更少，则队列的基地址始终对齐为 4-KiB。如果页面请求队列的条目数超过 256 个，则页面请求队列的基地址必须自然对齐为 2^{LOG2SZ} x 16。
9:5	保留	WPRI	保留为标准用途
53:10	PPN	WARL	保存内存中页面请求队列（page-request-queue）根页面的 PPN，该队列被 IOMMU 用来排列 "页面请求"（Page Request）信息。如果由 PPN 确定的基地址未按要求对齐，则队列中的所有条目在 IOMMU 看

位	名称	属性	说明
			来都是 UNSPECIFIED（未定义），而且 IOMMU 计算并用于访问队列中的条目的任何地址也是 UNSPECIFIED（未定义）。
63:54	保留	WPRI	保留为标准用途

在 RV32 中，只需写入寄存器的低阶 32 位（22 位 PPN 和 5 位 LOG2SZ-1）。

5.13. 页面请求队列头（pqh）

这个 32 位寄存器（RW）保存着页面请求队列的索引，软件将在该队列中获取下一个页面请求。

图 45. 页面请求队列头部寄存器字段

位	名称	属性	说明
31:0	index	WARL	保存页面请求队列的索引，软件可从中读取下一条 "页面请求 "信息。只有 LOG2SZ-1:0 位可写。

5.14. 页面请求队列尾（pqt）

这个 32 位寄存器（RO）保存着 IOMMU 写入下一个页面请求的页面请求队列索引。

图 46. 页面请求队列尾寄存器字段

位	名称	属性	说明
31:0	index	RO	保存 IOMMU 写入下一条 "页面请求 "报文的页面请求队列索引。

5.15. 命令队列 CSR (cqcsr)

这个 32 位寄存器（RW）用于控制操作和报告命令队列的状态。

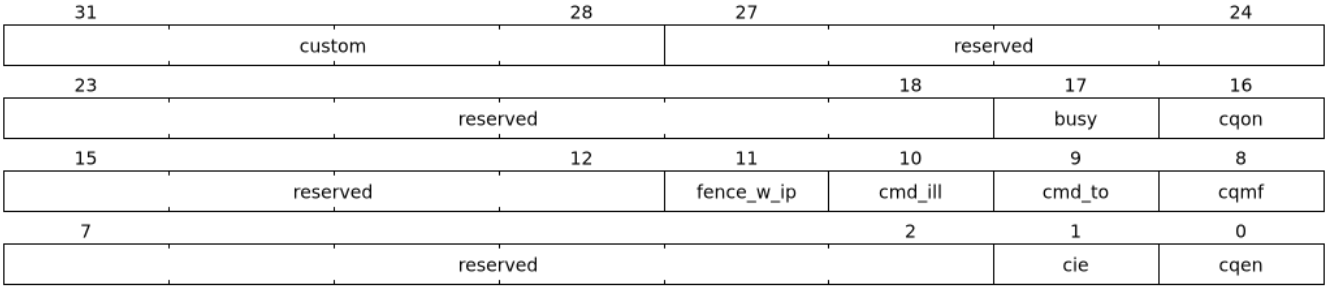


图 47. 命令队列 CSR 寄存器字段

位	名称	属性	说明

0	cqen	RW	当命令队列使能位设置为 1 时，将启用命令队列。 将 cqen 从 0 变为 1 后，cqh 寄存器和 cqcsr 位 cmd_ill、cmd_to、cqmf、fence_w_ip 将置 0。 将 cqen 设置为 1 后，命令队列可能需要一段时间才能激活。当命令队列处于活动状态时，cqon 位读数为 1。 当 cqen 从 1 变为 0 时，命令队列可能会保持激活状态（busy 为 1），直到已从命令队列获取的命令被处理完毕和/或命令队列有未完成的隐式加载。当命令队列关闭时，cqon 位的读数为 0。 当 cqon 位读数为 0 时，IOMMU 保证没有对命令队列的隐式内存访问，命令队列不会对队列内存产生新的隐式加载。
1	cie	RW	命令队列中断启用位设置为 1 时，可从命令队列产生中断。
7:2	保留	WPRI	保留为标准用途
8	cqmf	RW1C	如果命令队列访问导致内存故障，则命令队列内存故障位将被置 1，命令队列停滞，直至该位被清除。要重新启用命令处理功能，软件应通过写 1 来清除该位。
9	cmd_to	RW1C	如果命令的执行导致超时（例如，一条使设备 ATC 失效的命令在等待完成时可能会超时），则命令队列会设置 cmd_to 位并停止命令队列的处理。要重新启用命令处理，软件应通过写 1 来清除该位。
10	cmd_ill	RW1C	如果命令队列获取并解码了非法或不支持的命令，则命令队列会设置 cmd_ill 位并停止命令队列的处理。要重新启用命令处理功能，软件应通过写 1 来清除该位。
11	fence_w_ip	RW1C	仅支持有线信号中断的 IOMMU 会设置 fence_w_ip 位，以指示 IOFENCE.C 命令的完成。要在 IOFENCE.C 完成时重新启用中断，软件应通过写 1 来清除该位。如果 IOMMU 不支持有线信号中断或未启用有线信号中断（即 fctl.WSI ==0），则保留该位。
15:12	保留	WPRI	保留为标准用途
16	cqon	RO	如果 cqon 为 1，则命令队列处于活动状态。
17	busy	RO	写入 cqcsr 可能需要 IOMMU 执行许多操作，这些操作可能不会与写入同步进行。当 cqcsr 观察到写入操作时，busy 位被置 1。 当 busy 位为 1 时，对 cqcsr 的额外写入行为不明。某些实现可能会忽略第二次写入，而另一些实现可能会执行第二次写入所决定的操作。 软件在写入 cqcsr 前必须验证 busy 位是否为 0。 可同步完成这些操作的 IOMMU 可将该位硬连线为 0。
27:18	保留	WPRI	保留为标准用途
31:28	*定制*	WPRI	*指定用于定制用途。*

当 cqcsr 中的 cmd_ill 或 cqmf 为 1 时，cqh 将引用 CQ 中导致错误的命令。之前的命令可能已经完成、超时或被 IOMMU 中止执行。

如果软件在 cmd_ill 或 cqmf 出错后使 CQ 重新运行，则软件应重新提交自上次成功完成 IOFENCE.C 以来提交的命令。

当 IOFENCE.C 命令检测到前面一条或多条指定超时的命令超时，但 IOFENCE.C 之前的所有其他命令都已完成时，cmd_to 位被置位。cmd_to 为 1 时，cqh 将引用检测到超时的 IOFENCE.C 命令。

命令队列为空并不意味着从命令队列获取的所有命令都已执行完毕。当要求禁用命令队列时，执行程序可以完成已获取的命令，也可以放弃执行这些命令。在关闭命令队列之前，软件必须使用 IOFENCE.C 命令等待所有先前的命令提交（如果需要）。

5.16. 故障队列 CSR (fqcsr)

这个 32 位寄存器（RW）用于控制操作和报告故障队列的状态。

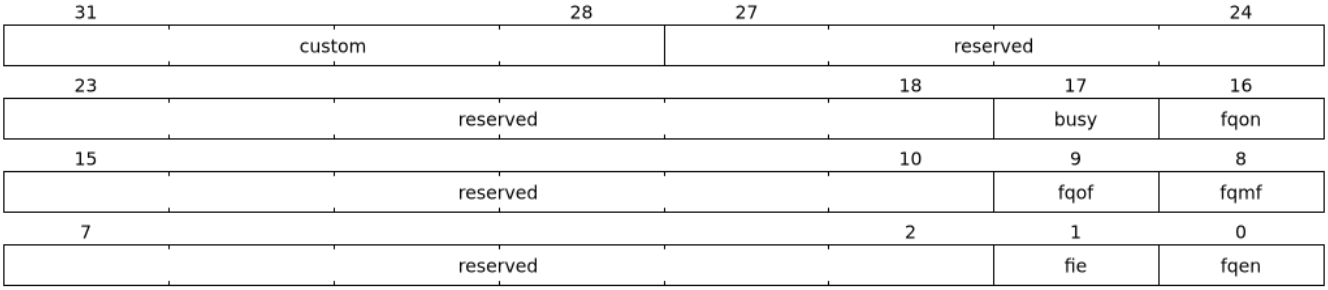


图 48. 故障队列 CSR 寄存器字段

位	名称	属性	说明
0	fqen	RW	当故障队列使能位设置为 1 时，将启用故障队列。 将 fqen 从 0 变为 1 后，fqt 寄存器以及 fqcsr 位 fqof 和 fqmf 将置 0。将 fqen 设置为 1 后，故障队列可能需要一段时间才能激活。当故障队列激活时，fqon 位读数为 1。 当 fqen 从 1 变为 0 时，故障队列可能保持活动状态（busy 为 1），直到处理中的故障记录完成。当故障队列关闭时，fqon 位读数为 0。当 fqon 位读数为 0 时，IOMMU 保证处理中没有对故障队列的隐式写入，也不会向故障队列写入新的故障记录。
1	fie	RW	故障队列中断使能位设置为 1 时，可从故障队列产生中断。
7:2	保留	WPRI	保留为标准用途
8	fqmf	RW1C	如果 IOMMU 在向故障队列存储故障记录时遇到访问故障，则 fqmf 位被置 1。试图写入的故障记录将被丢弃，不再生成故障记录，直到软件将 fqmf 位写 1 清除为止。
9	fqof	RW1C	如果 IOMMU 需要将故障记录放进队列，但故障队列已满（即 fqt == fqh -1），则故障队列溢出位设为 1。 该故障记录将被丢弃，不再生成故障记录，直到软件通过向该位写 1 来清除 fqof。
15:10	保留	WPRI	保留为标准用途

位	名称	属性	说明
16	fqon	RO	如果 fqon 读数为 1，则故障队列处于活动状态。
17	busy	RO	写入 fqcsr 可能要求 IOMMU 执行许多与写入不同步的操作。当 fqcsr 观察到一次写入时，忙位被置 1。当 busy 位为 1 时，对 fqcsr 的其他写入行为不明。某些实现可能会忽略第二次写入，而其他实现可能会执行第二次写入所决定的操作。 软件在写入 fqcsr 前应确保 busy 位为 0。 可同步完成控制的 IOMMU 可将该位硬接为 0。
27:18	保留	WPRI	保留为标准用途
31:28	*定制*	WPRI	*指定用于定制用途。*

5.17. 页面请求队列 CSR (pqcsr)

这个 32 位寄存器（RW）用于控制操作和报告页面请求队列的状态。

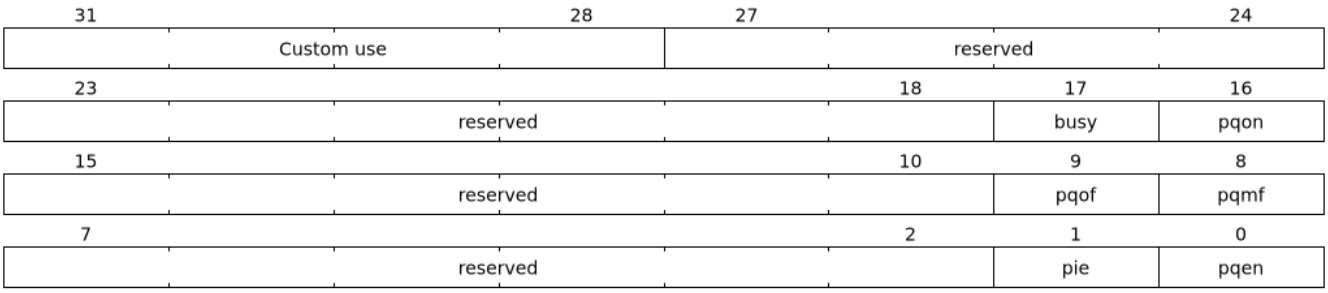


图 49. 页面请求队列 CSR 寄存器字段

位	名称	属性	说明
0	pqen	RW	页面请求启用位设置为 1 时，将启用页面请求队列。 将 pqen 从 0 变为 1 后，pqh 寄存器以及 pqcsr 位 pqmf 和 pqof 将置 0。当页面请求队列处于活动状态时，pqon 位读数为 1。 当 pqen 从 1 变为 0 时，页面请求队列可能保持激活状态（busy 为 1），直到完成处理中的页面请求写入。当页面请求队列关闭时，pqon 位的读数为 0。 当 pqon 读取为 0 时，IOMMU 保证队列内存中没有旧的处理中的隐式写入，也不会再向队列内存产生隐式写入。 如第 2.7 节所述，当页面请求队列关闭或正在关闭时，IOMMU 可对收到的 "页面请求 "信息做出响应。
1	pie	RW	页面请求队列中断启用位设置为 1 时，可从页面请求队列产生中断。
7:2	保留	WPRI	保留为标准用途
8	pqmf	RW1C	如果 IOMMU 在向页面请求队列存储 "页面请求 "报文时遇到访问故障，则 pqmf 位被置 1。 导致 pqmf 或 pqof 错误的 "页面请求 "报文以及所有后续的 "页面请求

位	名称	属性	说明
			"报文将被丢弃，直到软件将 pqof 和/或 pqmf 位写 1 清除为止。 IOMMU 可响应导致 pqof 或 pqmf 位被设置的 "页面请求 "信息，以及在这些位为 1 时收到的所有后续 "页面请求 "信息，如第 2.7 节所述。
9	pqof	RW1C	如果页面请求队列溢出，即 IOMMU 需要将 "页面请求 "报文放入队列，但页面请求队列已满（即 pqt == pqh - 1），则页面请求队列溢出位被置 1。 导致 pqmf 或 pqof 错误的 "页面请求 "报文以及所有后续 "页面请求 " 报文都将被丢弃，直到软件将 pqof 和/或 pqmf 位写 1 清除为止。 IOMMU 可响应导致 pqof 或 pqmf 位被设置的 "页面请求 "信息，以及在这些位为 1 时收到的所有后续 "页面请求 "信息，如第 2.7 节所述。
15:10	保留	WPRI	保留为标准用途
16	pqon	RO	当 pqon 读数为 1 时，页面请求处于活动状态。
17	busy	RO	写入 pqcsr 可能需要 IOMMU 执行许多操作，这些操作可能不会与写入同步进行。当 pqcsr 观察到写入操作时，busy 位被置 1。 当 busy 位为 1 时，对 pqcsr 的额外写入行为不明。某些实现可能会忽略第二次写入，而另一些实现可能会执行第二次写入所决定的操作。软件在写入 pqcsr 前应确保 busy 位为 0。 可同步完成控制的 IOMMU 可将该位硬连接为 0
27:18	保留	WPRI	保留为标准用途
31:28	*定制*	WPRI	*指定用于定制用途。*

5.18. 中断待处理状态寄存器 (ipsr)

这个 32 位寄存器（RW1C）报告需要软件服务的待处理中断。寄存器中的每个中断等待位都对应 IOMMU 中的一个中断源。寄存器中的中断等待位一旦置 1 就会保持 1，直到软件通过写 1 来清除该中断等待位。

当 fctl.WSI 为 1 时，中断等待位驱动相应 icvec 字段选择的信号线来发出中断信号。

当 fctl.WSI 为 0 时，IOMMU 使用报文发出中断信号。消息的地址和数据来自与中断等待位相对应的 icvec 字段所选择的 msi_cfg_tbl 条目。

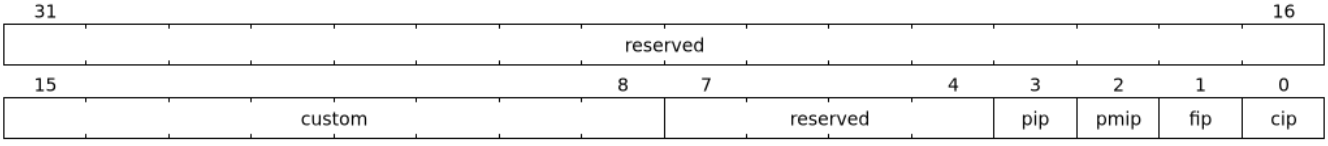


图 50. 中断待处理状态寄存器字段中断待处理状态寄存器

位	名称	属性	说明
0	cip	RW1C	如果 cqcsr.cie 设置为 "1"，且以下任一条件为真，则命令队列中断等待位设置为 "1"： • cqcsr.fence_w_ip 为 1。 • cqcsr.cmd_ill 为 1。

位	名称	属性	说明
			• cqcsr.cmd_to 为 1。 • cqcsr.cqmf 是 1。
1	fip	RW1C	如果 fqcsr.fie 为 1，且以下任一条件为真，则故障队列中断等待位设置为 1： • fqcsr.fqof 是 1。 • fqcsr.fqmf 是 1。 • FQ 中产生了一个新记录。
2	pmip	RW1C	当 iohpmcycles 或任何 iohpmctr1-31 寄存器中的 OF 位从 0 变为 1 时，性能监控中断等待设置为 1。
3	pip	RW1C	如果 pqcsr.pie 设置为 "page-request-queue-interrupt-pending"，且以下任一条件为真，则页面请求队列中断等待设置为 1： • pqcsr.pqof 是 1。 • pqcsr.pqmf 是 1。 • 在 PQ 中生成一条新信息。
7:4	保留	WPRI	保留为标准用途
15:8	*定制*	WPRI	*指定用于定制用途。*
31:16	保留	WPRI	保留为标准用途

如果 ipsr 中的某个位为 1，那么向该位写入 1 将使该位从 1→0 转换。如果设置该位的条件仍然存在（参见 [IPSR_FIELDS]），或在清除该位后出现这些条件，则该位将再次从 0→1 转换。

5.19. 性能监测计数器溢出状态（iocountovf）

性能监控计数器溢出状态是一个 32 位只读寄存器，其中包含 iohpmevt1-31 寄存器中 OF 位的影子副本，其中 iocntovf 位 X 对应于iohpmevtX，位 0 对应 iohpmcycles 的 OF 位。

该寄存器使溢出中断处理程序软件能够快速、轻松地确定哪个（些）计数器溢出。

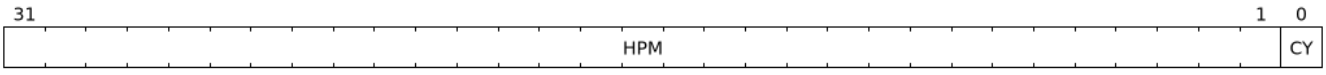


图 51. 性能监测计数器溢出状态寄存器字段

位	名称	属性	说明
0	CY	RO	iohpmcycles.OF 的影子
31:1	HPM	RO	iohpmevt[1-31].OF的影子

5.20. 性能监测计数器禁止 (iocountinh)

性能监测计数器禁止是一个 32 位 WARL 寄存器，其中包含禁止相应计数器计数的位。位 X 设置后会禁止 iohpmctrX 的计数，位 0 则会抑制 iohpmcycles 的计数。

图 52. 性能监测计数器抑制寄存器字段

位	名称	属性	说明
0	CY	RW	设置后，iohpmcycles 计数器将被禁止计数。
31:1	HPM	WARL	当 X 位被设置时，iohpmctrX 中的事件计数将被禁止。

当不需要 iohpmcycles 计数器时，最好有条件地禁止它，以降低能耗。通过提供一个寄存器来禁止所有计数器，可以 a) 以原子方式对一个或多个计数器进行事件编程，以便计数 b) 一个或多个原子采样计数器。

5.21. 性能监测循环计数器（iohpmcycles）

这个 64 位寄存器是一个自由运行的时钟周期计数器。没有相关的 iohpmevt0。

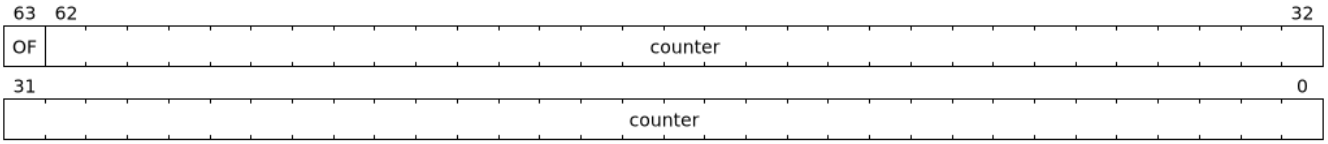


图 53. 性能监测周期计数寄存器字段

位	名称	属性	说明
62:0	counter	WARL	循环计数器值。
63	OF	RW	溢出

当 iohpmcycles 计数器溢出时，OF 位被置位，并保持置位直到软件清除。由于 iohpmcycles 值是无符号值，因此溢出被定义为无符号溢出。需要注意的是，溢出后不会丢失信息，因为计数器会绕一圈继续计数，而粘滞的 OF 位仍然置位。

如果在 OF 位为 0 时 iohpmcycles 计数器溢出，则通过将 ipsr.pmip 位设置为 1 来产生 HPM 计数器溢出中断。如果 OF 位已为 1，则不会产生中断请求。因此，OF 位也可用作 iohpmcycles 的计数溢出中断禁用位。

5.22. 性能监控事件计数器（iohpmctr1-31）

这些寄存器是 64 位 WARL 计数寄存器。

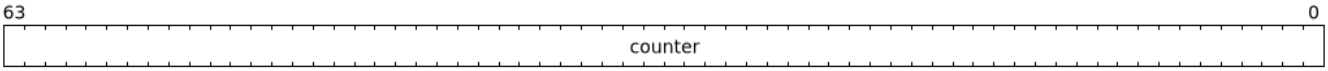


图 54. 性能监控事件计数器寄存器字段

位	名称	属性	说明
63:0	counter	WARL	事件计数器值。

5.23. 性能监控事件选择器（iohpmevt1-31）

这些性能监控事件寄存器是 64 位 RW 寄存器。当 IOMMU 处理的事务导致计数器计数的事件发生时，计数器就会递增。除了匹配事件外，事件选择器还可根据 device_id 、 process_id 、GSCID 和 PSCID 设置附加过滤器，这样计数器就会根据匹配这些附加过滤器的事务有条件地递增。使用这种基于 device_id 的过滤时，可将匹配配置为精确匹配或部分匹配。部分匹配允许计数器计算一系列 ID 的事务。

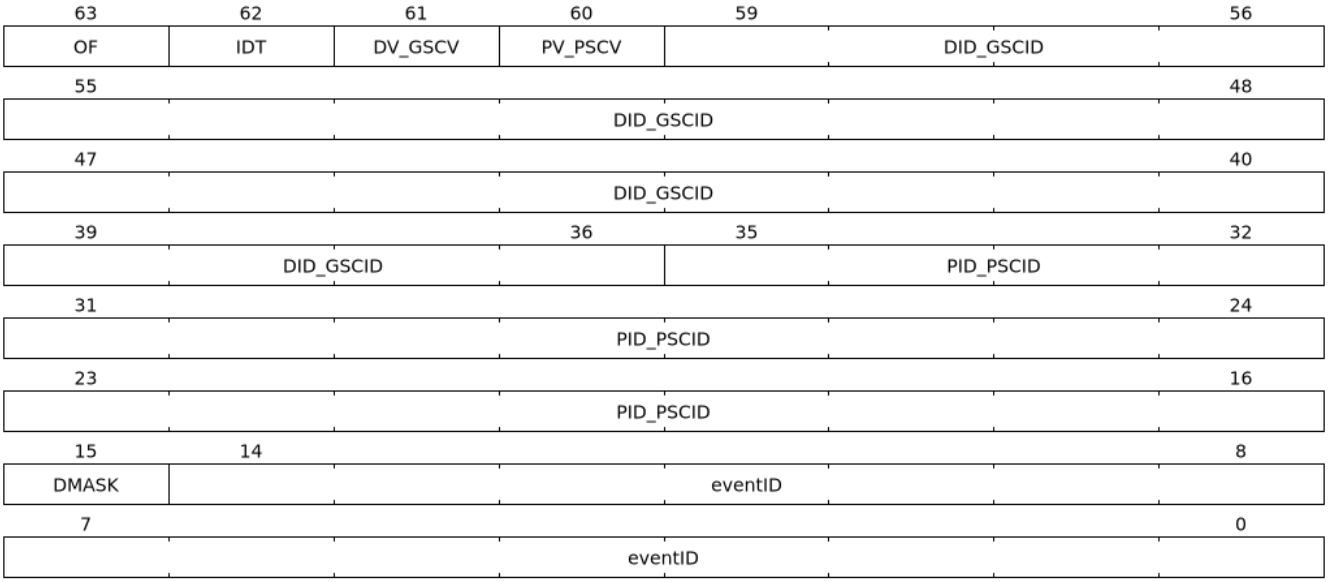


图 55. 性能监控事件选择器寄存器字段

位	名称	属性	说明
14:0	eventID	WARL	表示要计数的事件。值为 0 表示不计算任何事件。 编码 1 至 16383 保留给表 17 中定义的标准事件。 编码 16384 至 32767 指定用于自定义用途。 当 eventID 发生变化（包括变为 0）时，计数器会保留其值。
15	DMASK	RW	当设置为 1 时，将对事务执行 DID_GSCID 的部分匹配。DID_GSCID 的低位一直到第一个低阶 0 位（包括 0 位位置本身）都会被屏蔽。
35:16	PID_PSCID	RW	IDT 为 0 时为 process_id，IDT 为 1 时为 PSCID。
59:36	DID_GSCID	RW	IDT 为 0 时为 device_id；IDT 为 1 时为 GSCID。
60	PV_PSCV	RW	如果设置，则只有与 process_id 或 PSCID（基于过滤器 ID 类型）匹配的事务被计算在内。
61	DV_GSCV	RW	如果设置，则只有与 device_id 或 GSCID（基于过滤器 ID 类型）匹配的事务被计算在内。
62	IDT	RW	筛选 ID 类型：该字段表示要过滤的 ID 类型。当为 0 时，DID_GSCID 字段保存 device_id ，PID_PSCID 字段保存 process_id 。当为 1 时，DID_GSCID 字段保存 GSCID，PID_PSCID 字段保存 PSCID。
63	OF	RW	溢出状态或中断禁用

下表概述了支持按 ID 过滤的事件的过滤选项。

表 15. 过滤选项

IDT	DV_GSCV	PV_PSCV	运行
0/1	0	0	计数器递增。无基于 ID 的过滤功能。
0	0	1	如果事务有一个有效的 process_id ，则在 process_id 与 PID_PSCID 匹配时计数器递增。
0	1	0	如果 device_id 与 DID_GSCID 匹配，计数器将递增。
0	1	1	如果事务有一个有效的 process_id，且 device_id 与 DID_GSCID 匹配，process_id 与 PID_PSCID 匹配，则计数器递增。

IDT	DV_GSCV	PV_PSCV	运行
1	0	1	如果事务具有有效的 PSCID，则在该进程的 PSCID 与 PID_PSCID 一致时，计数器递增
1	1	0	如果 GSCID 有效且与 DID_GSCID 匹配，计数器将递增。
1	1	1	如果 GSCID 有效且与 DID_GSCID 匹配，且 PSCID 有效并与 PID_PSCID 匹配，计数器递增。

当选择按 device_id 或 GSCID 过滤且事件支持基于 ID 的过滤时，DMASK 字段可用于配置部分匹配。当 DMASK 设置为 1 时，将对事务执行 DID_GSCID 的部分匹配。DID_GSCID 的低位一直到第一个低阶 0 位（包括 0 位位置本身）都会被屏蔽。

下面的示例说明了 DMASK 和按 device_id 过滤的使用。

表 16. 将 IDT 设置为基于 device_id 的过滤时的 DMASK

DMASK	DID_GSCID	评论
0	yyyyyyyy yyyyyyyy yyyyyyyy	一个特定的 seg:bus:dev:func
1	yyyyyyyy yyyyyyyy yyyyy011	SEG:BUS:DEV - 任何功能
1	yyyyyyyy yyyyyyyy 01111111	SEG:BUS - 任何 DEV:FUNC
1	yyyyyyyy 01111111 11111111	seg - 任何 bus:dev:func

下表列出了可计算的标准事件：

表 17. 标准事件列表

事件 ID	已统计事件	支持 IDT 设置
0	不要计算	
1	未转换的请求	0
2	转换请求	0
3	ATS 转换请求	0
4	TLB 未命中	0/1
5	设备目录漫步	0
6	进程目录漫步	0
7	第一阶段页表漫步	0/1
8	第二阶段页表漫步	0/1
9 - 16383	保留为未来标准	-

如果某个事件不支持已编程的 IDT 设置，则相关计数器不会递增。

当相应的 iohpmctr1-31 计数器溢出时，OF 位被置位，并保持置位直到软件清除。由于 iohpmctr1-31 值是无符号值，因此溢出定义为无符号溢出。需要注意的是，溢出后不会丢失信息，因为计数器会绕一圈继续计数，同时粘滞的 OF 位会保持置位。

如果 iohpmctr1-31 计数器在相关 OF 位为 0 时溢出，则通过将 ipsr.pmip 位设置为 1 来产生 HPM 计数器溢出中断。因此，OF 位也是相关 iohpmctr1-31 的计数溢出中断禁用位。

没有单独的溢出状态位和溢出中断启用位。实际上，启用溢出中断（通过清除 OF 位）是与将计数器初始化为起始值同时进行的。计数器溢出后，必须重新初始化计数器和 OF 位，才能产生另一个溢出中断。

在 RV32 中，内存映射写入 iohpmevt1-31 只修改寄存器的一个 32 位部分。可以使用以下序列更新寄存器，而不会因为寄存器的中间值而错误地计算事件：

- 写入低阶 32 位，将 eventID 设置为 0。
- 在高阶 32 位写入新的预期值。
- 向低阶 32 位写入新的预期值，包括 eventID 字段。

或者，可以先禁止计数器，使其在更新过程中不计数任何事件，然后在寄存器编程为所需值后，再取消禁止。

如果 capabilities.HPM 为 1，则除周期计数器外，还需要至少一个可编程事件计数器才能符合本规范。可使用一个计数器以时间多路复用的方式对事件进行采样，但这种分析可能需要更长的时间才能完成。IOMMU 与 CPU MMU 不同，可为多个 IO 流提供服务，性能分析师可使用 HPM 同时分析一个或多个 IO 流。通常，性能分析人员可能需要四个可编程计数器来计算 IO 流的事件。为支持至少两个 IO 流的并发分析，建议支持七个可编程计数器。

5.24. 转换请求 IOVA (tr_req_iova)

tr_req_iova 是一个 64 位寄存器，用于实现调试的转换请求接口。当 capabilities.DBG == 1 时，该寄存器存在。

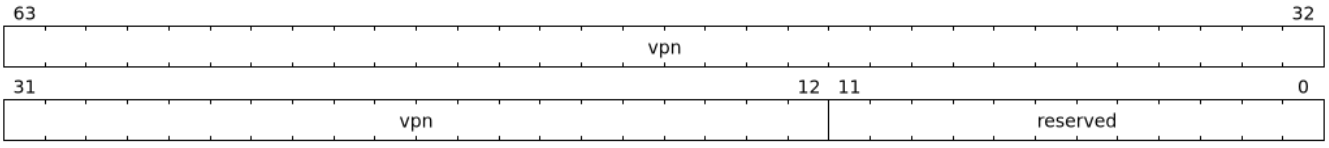


图 56. 转换请求 IOVA 寄存器字段

位	名称	属性	说明
11:0	保留	WPRI	保留为标准用途
63:12	vpn	WARL	IOVA 虚拟页号

5.25. 转换请求控制 (tr_req_ctl)

tr_req_ctl 是一个 64 位 WARL 寄存器，用于实现调试的转换请求接口。当 capabilities.DBG == 1 时，该寄存器存在。

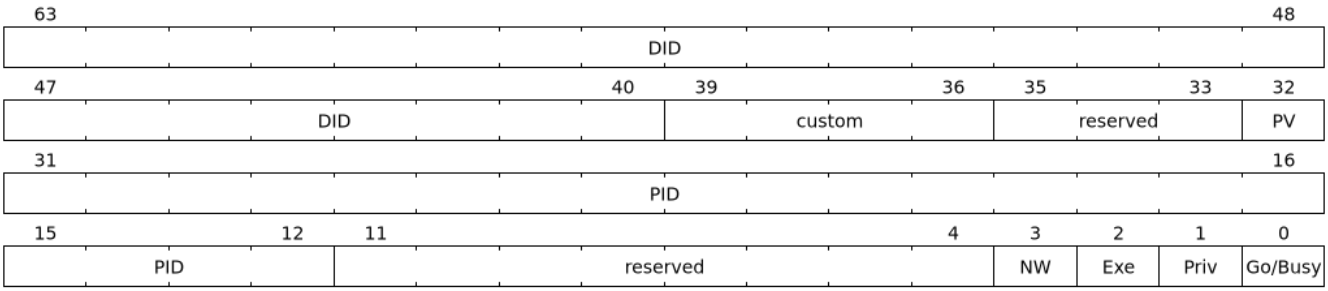


图 57. 转换请求控制寄存器字段

位	名称	属性	说明
0	Go/Busy	RW1S	该位被置位，表示已在tr_req_iova/tr_req_ctl 寄存器中设置了有效请求，供 IOMMU 转换。 IOMMU 将该位清零为 0，表示转换请求已完成。 转换完成后，转换结果将存入 tr_response 寄存器。
1	Priv	WARL	如果设置为 1，则请求特权模式访问，否则不请求特权模式访问。
2	Exe	WARL	如果设置为 1，则请求执行权限，否则不请求执行权限。
3	NW	WARL	如果设置为 1，则请求读取权限。如果设置为 0，则同时申请读取和写入权限。
11:4	保留	WPRI	保留为标准用途
31:12	PID	WARL	如果 PV 为 1，则该字段将为该转换请求提供 process_id 输入。如果 PV 为 0，则不使用该字段。
32	PV	WARL	如果设置为 1，则寄存器的 PID 字段有效，并为该转换请求提供 process_id。如果设置为 0，则不使用 PID 字段，该转换请求的 process_id 无效。
35:33	保留	WPRI	保留为标准用途
39:36	*定制*	WPRI	*指定用于定制用途*
63:40	DID	WARL	该字段提供了该转换请求的 device_id。

5.26. 转换应答 (tr_response)

tr_response 是一个 64 位 RO 寄存器，用于保存使用转换请求接口请求的转换结果。当 capabilities.DBG == 1 时，该寄存器存在。

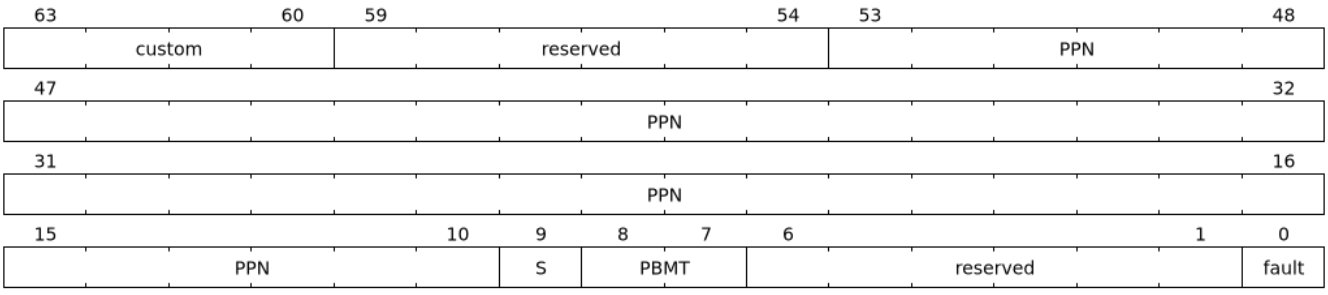


图 58. 转换响应寄存器字段

位	名称	属性	说明
0	fault	RO	如果转换 IOVA 的过程检测到故障，则 fault 字段将设为 1。
6:1	保留	RO	保留为标准用途

<table>
<tr><th>位</th><th>名称</th><th>属性</th><th>说明</th></tr>
<tr><td>8:7</td><td>PBMT</td><td>RO</td><td>使用用于转换的第一阶段和/或第二阶段页表中的 PBMT 字段为转换确定的内存类型。如果 fault 字段为 1，则该字段的值为 UNSPECIFIED。</td></tr>
<tr><td>9</td><td>S</td><td>RO</td><td>转换范围大小字段，设置为 1 时表示转换适用于大于 4 KiB 的范围，转换范围的大小在 PPN 字段中编码。如果 fault 字段为 1，则该字段的值为 UNSPECIFIED。</td></tr>
<tr><td>53:10</td><td>PPN</td><td>RO</td><td>如果 fault 位为 0，则该字段提供的 PPN 是转换 tr_req_iova 中的 vpn 后确定的。
如果 fault 位为 1，则该字段的值为 UNSPECIFIED。
如果 S 位为 0，则转换的大小为 4 KiB（一页）。
如果 S 位为 1，则转换结果为超级页，超级页的大小编码在 PPN 本身中。如果从第 0 位扫描到第 43 位，在第 X 位的第一个位值为 0，则超级页的大小为 $2^{X+1} * 4$ KiB。
如果 X 不是 0，那么位置 0 到 X-1 的所有比特的编码值均为 1。
表 18.PPN 中超级页面大小的编码示例
<table><tr><th>PPN</th><th>S</th><th>尺寸</th></tr><tr><td>yyyy....yyy yyyy yyyy</td><td>0</td><td>4 KiB</td></tr><tr><td>yyyy....yyyy yyyy 0111</td><td>1</td><td>64 KiB</td></tr><tr><td>yyyy....yyy0 1111 1111</td><td>1</td><td>2 MiB</td></tr><tr><td>yyyy....yy01 1111 1111</td><td>1</td><td>4 MiB</td></tr></table></td></tr>
<tr><td>59:54</td><td>保留</td><td>RO</td><td>保留为标准用途</td></tr>
<tr><td>63:60</td><td>定制</td><td>RO</td><td>指定用于定制用途</td></tr>
</table>

IOMMU 实现无需报告超级页转换或支持报告所有可能的超级页大小。允许实现报告与请求的 vpn 相对应的 4 KiB 转换，或报告小于页面表中配置的超级页大小的转换大小。

5.27. 中断原因向量寄存器 (icvec)

中断原因向量寄存器将一个原因映射到一个向量。所有原因可以映射到同一个向量，也可以给一个原因一个唯一的向量。

向量用于：

1. 以 MSI 方式产生中断的 IOMMU，要对 MSI 配置表（msi_cfg_tbl）进行索引，以确定要产生的 MSI。如果 capabilities.IGS==MSI 或 capabilities.IGS==BOTH，则 IOMMU 能够以 MSI 的形式生成中断。当 capabilities.IGS==BOTH 时，IOMMU 可以通过将 fctl.WSI 设置为 0 来配置为以 MSI 方式产生中断。
2. 以 WSI 方式产生中断的 IOMMU，确定发出中断的具体信号。如果 capabilities.IGS==WSI 或 capabilities.IGS === BOTH，则IOMMU 能够以 WSI 的形式生成中断。当 capabilities.IGS === BOTH 时，可通过将 fctl.WSI 设置为 1 来配置 IOMMU，使其产生有线信号中断。

如果实现只支持一个向量，那么寄存器的所有位都可以硬连接为 0 (WARL)。同样，如果只支持两个向量，那么每个原因只有第 0 位可以写入。

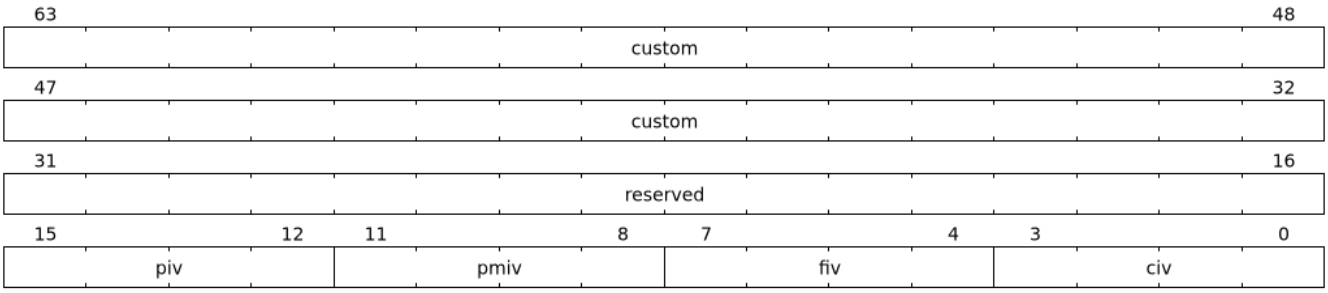


图 59. 中断原因到向量寄存器字段

位	名称	属性	说明
3:0	civ	WARL	命令队列中断向量（civ）是分配给命令队列中断的向量编号。
7:4	fiv	WARL	故障队列中断向量 (fiv) 是分配给故障队列中断的向量编号。
11:8	pmiv	WARL	性能监控中断向量 (pmiv) 是分配给性能监控中断的向量编号。
15:12	piv	WARL	页面请求队列中断向量 (piv) 是分配给页面请求队列中断的向量编号。
31:16	保留	WPRI	保留为标准用途
63:32	*定制*	WPRI	*指定用于定制用途*

5.28. MSI 配置表 (msi_cfg_tbl)

支持将 IOMMU 引发的中断（即 capabilities.IGS == MSI 或 capabilities.IGS == BOTH）作为 MSI 生成的 IOMMU 实现了一个 MSI 配置表，该表通过 icvec 中的向量进行索引，以确定 MSI 表项。中断向量 x 的每个 MSI 表项有三个寄存器 msi_addr_x、msi_data_x 和 msi_vec_ctl_x。如果 capabilities.IGS == WSI，这些寄存器将被硬连接为 0。

如果在使用 msi_addr_x 进行 MSI 写入时检测到访问故障，则 IOMMU 将报告 "IOMMU MSI 写入访问故障"（原因 273），TTYP 设置为 0，iotval 设置为 msi_addr_x 的值。

表 19.MSI 配置表结构

位 63	**位 0**	**字节偏移**
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

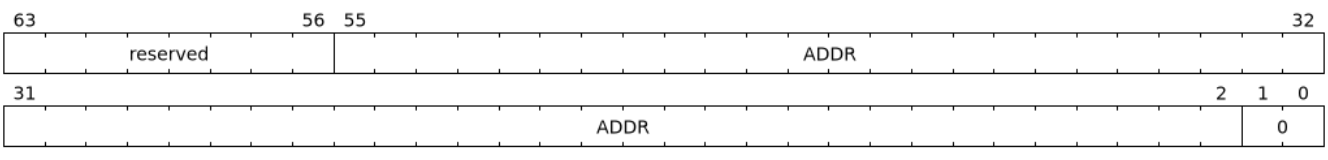


图 60. Message address 寄存器字段

位	名称	属性	说明
1:0	0	RO	固定为 0
55:2	ADDR	WARL	保存 4 字节对齐的 MSI 地址。
63:56	保留	WPRI	保留为标准用途。

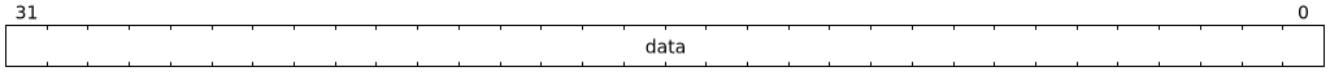


图 61. Message data 寄存器字段

位	名称	属性	说明
31:0	data	WARL	保存 MSI 数据

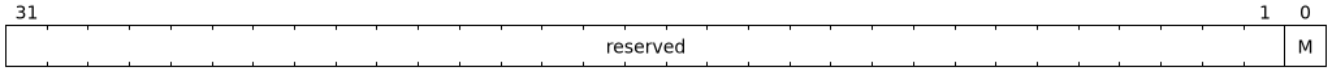


图 62. Vector control 寄存器字段

位	名称	属性	说明
0	M	RW	当屏蔽位 M 为 1 时，相应的中断向量被屏蔽，IOMMU 被禁止发送相关报文。如果相应的屏蔽位清零为 0，则随后会生成该向量的待发报文。
31:1	保留	WPRI	保留为标准用途。

第 6 章 软件指南（Software guidelines）

本节就 IOMMU 接口的正确和预期使用顺序为软件开发人员提供指导。如果不遵守这些指导原则，IOMMU 的行为将由具体实现决定。

6.1. 读写 IOMMU 寄存器（Reading and writing IOMMU registers）

读取或写入 IOMMU 寄存器必须遵循以下规则：

- 访问地址必须与访问大小对齐。
- 访问不得跨越多个寄存器。
- 可以使用 32 位或 64 位访问方式访问 64 位宽的寄存器。
- 32 位宽的寄存器只能使用 32 位访问。

6.2. 初始化指南（Guidelines for initialization）

初始化 IOMMU 的指导原则如下：

1. 读取 capabilities 寄存器，了解 IOMMU 的能力。
2. 如果不支持 capabilities.version，则停止并报告失败。
3. 读取功能控制寄存器（fctl）。
4. 如果需要访问大端内存，且 capabilities.END 字段为 0（即只有一种端位）且 fctl.BE 为 0（即小端位），则停止并报告失败。
5. 如果需要大端内存访问，且 capabilities.END 字段为 1（即支持两种端位），则将 fctl.BE 设置为 1（即大端）（如果该字段尚未为 1）。
6. 如果 IOMMU 初始化中断时需要有线信号中断，并且capabilities.IGS 不是 WSI，则停止并报告故障。
7. 如果 IOMMU 初始化中断时需要有线信号中断，并且 capabilities.IGS 是 BOTH，则将 fctl.WSI 设为 1（如果该字段尚未设为 1）。
8. 如果不支持其他所需的功能（如虚拟地址模式、MSI 转换等），则停止运行并报告故障。
9. icvec 寄存器用于为每个中断原因编程中断向量。向每个字段写入 0xF，然后读回可写位数，即可确定 IOMMU 支持的向量数。如果可写入位数为 N，则支持的向量数为 2^N 。对于每个原因 C，将向量 V 与该原因相关联。V 是介于 0 和 $(2^N - 1)$ 之间的数字。
10. 如果 IOMMU 被配置为使用有线中断，则每个向量 V 对应一条连接到平台级中断控制器（如 APLIC）的中断线。利用配置机制（如设备树）提供的配置信息，为每条中断线确定需要编程的中断控制器配置寄存器，并对中断控制器进行编程。
11. 如果 IOMMU 配置为使用 MSI，则每个向量 V 都是 msi_cfg_tbl 的索引。msi_addr_V 寄存器的值为 A，msi_data_V 寄存器的值为 D。
12. 要对命令队列进行编程，首先要确定命令队列所需的条目数 N。命令队列中的条目数必须是 2 的幂次。分配一个 N x 16 字节大小的内存缓冲区，自然对齐为 4-KiB 或 N x 16 字节中的较大值。假设 $k=\log_2(N)$，B 为分配的内存缓冲区的物理页码（PPN）。命令队列寄存器的编程如下：
    - temp_cqb_var.PPN = B
    - temp_cqb_var.LOG2SZ-1 = (k - 1)

- cqb = temp_cqb_var
- cqt = 0
- cqcsr.cqen = 1
- 轮询 cqcsr.cqon 直到读数为 1

13. 要对故障队列进行编程，首先要确定故障队列所需的条目数 N。故障队列中的条目数总是 2 的幂次。分配一个 N x 32 字节大小的内存缓冲区，该缓冲区自然对齐为 4-KiB 或 N x 32 字节中的较大值。假设 $k=\log_2(N)$，B 为分配的内存缓冲区的 PPN。对故障队列寄存器编程如下：
    - temp_fqb_var.PPN = B
    - temp_fqb_var.LOG2SZ-1 = (k - 1)
    - fqb = temp_fqb_var
    - fqh = 0
    - fqcsr.fqen = 1
    - 轮询 fqcsr.fqon 直到读数为 1
14. 要对页面请求队列进行编程，首先要确定页面请求队列所需的条目数 N。页面请求队列中的条目数总是 2 的幂次。分配一个 N x 16 字节大小的缓冲区，该缓冲区自然对齐为 4-KiB 或 N x 16 字节中的较大值。假设 $k=\log_2(N)$，B 为分配的内存缓冲区的 PPN。对页面请求队列寄存器编程如下：
    - temp_pqb_var.PPN = B
    - temp_pqb_var.LOG2SZ-1 = (k - 1)
    - pqb = temp_pqb_var
    - pqh = 0
    - pqcsr.pqen = 1
    - 轮询 pqcsr.pqon 直到读数为 1
15. 要对 DDT 指针进行编程，首先要确定支持的 device_id 宽度 Dw 和设备上下文数据结构的格式。如果 capabilities.MSI 为 0，则 IOMMU 使用基本格式的设备上下文，否则使用扩展格式的设备上下文。分配一个页面（4 KiB）的内存作为 DDT 的根表。将所分配的内存初始化为全 0，设 B 为所分配内存的 PPN。根据 Dw 和 IOMMU 设备上下文格式确定 DDT 的模式 M，如下所示：
    - 通过写入合法值并读取该值是否被保留，确定 ddtp.iommu_mode 支持的值。如果支持的模式不支持所需的 Dw，则停止并报告失败。
    - 如果使用扩展格式设备上下文，那么
        a. 如果 Dw 小于或等于 6 位，且支持 1LVL，则 M = 1LVL
        b. 如果 Dw 小于或等于 15 位，且支持 2LVL，则 M = 2LVL
        c. 如果 Dw 小于或等于 24 位，且支持 3LVL，则 M = 3LVL
    - 如果使用基本格式的设备上下文，那么
        a. 如果 Dw 小于或等于 7 位，且支持 1LVL，则 M = 1LVL
        b. 如果 Dw 小于或等于 16 位，且支持 2LVL，则 M = 2LVL
        c. 如果 Dw 小于或等于 24 位，且支持 3LVL，则 M = 3LVL

    对 ddtp 寄存器编程如下：
    - temp_ddtp_var.iommu_mode = M
    - temp_ddtp_var.PPN = B

    - ddtp = temp_ddtp_var

IOMMU 已初始化，现在可为 IOMMU 范围内的设备配置设备上下文。

6.3. 失效指南（Guidelines for invalidations）

本节为软件提供了修改 IOMMU 内存数据结构时通过 CQ 向 IOMMU 发送失效命令的指南。软件必须在更新全局可见后执行失效操作。FENCE 指令提供的存储排序和原子指令的获取/释放位也对 IOMMU 观察到的与这些存储相关的数据结构更新进行排序。

软件可使用 IOFENCE.C 命令来确保之前从 CQ 获取的所有命令都已完成并提交。IOFENCE.C 命令中的 PR 和/或 PW 位可设置为 1，以请求将 IOMMU 已处理过的所有先前读取和/或写入请求作为 IOFENCE.C 命令的一部分提交到全局排序点。

6.3.1. 更改设备目录表表项（Changing device directory table entry）

如果软件更改了叶级 DDT 表项（即 device_id = D 的设备上下文 (DC)），则必须执行以下失效处理：

- IODIR.INVAL_DDT，DV=1，DID=D
- 如果 DC.tc.PDTV==1 则 IODIR.INVAL_PDT，DV=1、PV=0 和 DID=D
- 如果 DC.iohgatp.MODE != Bare
    - IOTINVAL.VMA，GV=1，AV=PSCV=0，GSCID=DC.iohgatp.GSCID
    - IOTINVAL.GVMA，GV=1，AV=0，GSCID=DC.iohgatp.GSCID
- 否则
    - 如果 DC.tc.PDTV==1 || DC.tc.PDTV == 0 && DC.fsc.MODE == Bare
        - IOTINVAL.VMA，GV=AV=PSCV=0
    - 否则
        - IOTINVAL.VMA，GV=AV=0，PSCV=1，PSCID=DC.ta.PSCID

如果软件更改了非叶级 DDT 表项，则必须执行以下失效处理：

- IODIR.INVAL_DDT，DV=0

在更改 DDT 表项和 IOMMU 处理使缓存表项失效的失效命令之间，IOMMU 可使用表项的旧值或新值。

6.3.2. 更改进程目录表表项（Changing process directory table entry）

如果软件更改了叶级 PDT 表项（即进程上下文 (PC)，对于 device_id=D 且 process_id=P），则必须执行以下失效处理：

- IODIR.INVAL_PDT 含 DV=1、PV=1、DID=D 和 PID=P
- 如果 DC.iohgatp.MODE != Bare
    - IOTINVAL.VMA，GV=1、AV=0、PV=1、GSCID=DC.iohgatp.GSCID 和 PSCID=PC.PSCID
- 不然
    - IOTINVAL.VMA，GV=0，AV=0，PV=1，PSCID=PC.PSCID

从更改 PDT 表项到 IOMMU 处理使缓存表项失效的失效命令之间，IOMMU 可使用表项的旧值或新值。

6.3.3. 更改 MSI 页表表项（Changing MSI page table entry）

如果软件更改了由中断文件号 I 标识的 MSI 页表项，而该页表项与未转换的 MSI 地址 A 相对应，则必须执行以下失效操作：

- IOTINVAL.GVMA 含 GV=AV=1、ADDR[63:12]=A[63:12]和 GSCID=DC.iohgatp.GSCID

要使 MSI 页表中的所有缓存项失效，必须执行以下失效操作：

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

在修改 MSI PTE 和 IOMMU 处理使缓存 PTE 失效的无效命令之间，IOMMU 可使用旧的 PTE 值或新的 PTE 值。PW=1 的 IOFENCE.C 命令可用于确保 IOMMU 先前处理过的所有先前写入（包括 MSI 写入）都提交到全局排序点中，以便系统中的所有 RISC-V 硬件和 IOMMU 都能观察到它们。

6.3.4. 更改第二阶段页表表项（Changing second-stage page table entry）

如果软件更改了虚拟机的叶第二阶段页表表项，而更改又影响了Guest-PPN G 的转换，则必须执行以下失效处理：

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

如果软件更改了虚拟机的非叶第二阶段页表项，则必须执行以下失效操作：

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

DC 具有保存 Guest-PPN 的字段。作为缓存 DC 的一部分，实现可能会将这些字段转换为 Supervisor -PPN。如果第二阶段页表更新影响到 DC 中持有的 Guest-PPN 的转换，那么软件必须使用 IODIR.INVAL_DDT（DV=1）并将 DID 设置为相应的 device_id，使所有此类缓存的 DC 失效。或者，也可以使用 DV=0 的 IODIR.INVAL_DDT 来使所有缓存的 DC 失效。

在改变第二阶段 PTE 和 IOMMU 处理使缓存 PTE 失效的无效命令之间，IOMMU 可以使用旧的 PTE 值或新的 PTE 值。

6.3.5. 更改第一阶段页表表项（Changing first-stage page table entry）

DC 可配置一个第一阶段页表（当 DC.tc.PDTV=0 时），或使用进程目录表中的 process_id 选择的第一阶段页表目录（当 DC.tc.PDTV=1 时）。

当对第一阶段页表进行修改，且第二阶段页表为 Bare 表时，软件必须使用 IOTINVAL.VMA（GV=0，AV 和 PSCV 操作数适合表 9 中规定的修改）执行失效操作。

如果对第一阶段页表进行了修改，而第二阶段页表不是 Bare 表，则软件必须使用 IOTINVAL.VMA（GV=1、GSCID=DC.iohgatp.GSCID 以及表 9 中规定的适合修改的 AV 和 PSCV 操作数）执行失效操作。

从第一阶段 PTE 发生变化到 IOMMU 处理使缓存 PTE 失效的失效命令之间，IOMMU 可以使用旧的 PTE 值或新的 PTE 值。

6.3.6. 访问 (A)/Dirty (D) 位更新和页面推广（Accessed (A)/Dirty (D) bit updates and page promotions）

当 IOMMU 支持硬件管理的 A 和 D 位更新时，如果软件清除了第一阶段和/或第二阶段 PTE 中的 A 和/或 D 位，则软件必须使 IOMMU 可能缓存的相应 PTE 表项失效。如果未执行此类失效操作，则 IOMMU 在处理使用此类条目的后续事务时可能不会设置这些位。

当软件将第一阶段 PT 和/或第二阶段 PT 中的页面升级为超级页面时，如果没有首先清除原始非叶 PTE 的有

效位并使缓存中的转换无效，那么 IOMMU 就有可能缓存多个与单个地址匹配的表项。IOMMU 可以使用旧的非叶 PTE，也可以使用新的非叶 PTE，但在其他方面都有明确规定。

在升级和/或降级页面大小时，软件必须确保原始 PTE 和新 PTE 具有相同的权限和内存类型属性，并且使用原始或新 PTE 转换后确定的物理地址对于任何给定输入都是相同的。IOMMU 在不清除原始 PTE 中的 V 位并执行适当的 IOTINVAL 命令的情况下，唯一支持的 PTE 更新是页面大小升级或降级。如果以这种方式改变其他属性，IOMMU 的行为将由具体实现确定。

6.3.7. 设备地址转换缓存失效（Device Address Translation Cache invalidations）

当第一阶段和/或第二阶段页表被修改时，可能需要对设备中的 DevATC 进行失效处理，这些设备可能已缓存了来自修改后页表的转换。此类页表的失效需要使用 ATS.INVAL 命令生成 ATS 失效。软件必须按照 PCIe ATS 规范 [1] 中定义的规则指定 PAYLOAD。

如果软件生成 ATS Invalidation Request 的速率超过 DevATC 服务的平均速率，那么设备可能会触发流量控制机制来控制速率。这样做的副作用是拥塞蔓延到其他信道和链路，从而导致性能下降。具有 ATS 功能的设备会通过 ATS 能力结构的队列深度字段公布在造成反向压力之前可缓冲的最大无效次数。当设备使用 PCIe SR-IOV 进行虚拟化时，该队列深度由设备的所有 VF 共享。软件必须将队列中未完成的 ATS 失效数量限制在设备公布的限制范围内。

RID 字段用于指定 ATS Invalidation Request 报文目的地的路由 ID。可通过设置 PV=1 并在 PID 中指定 PASID 来执行特定于 PASID 的无效。当 IOMMU 支持多网段时，必须通过设置 DSV=1 并在 DSEG 中提供网段号，用目标网段号限定 RID。

当设备启用 ATS 协议时，除了向 DevATC 提供转换外，IOMMU 仍可在其 IOATC 中缓存转换。即使在设备的设备上下文中启用了 ATS，软件也不得跳过 IOMMU 的转换缓存失效。由于来自 DevATC 的转换请求可能会被 IOMMU 从 IOATC 中满足，为确保正确操作，软件必须在向 DevATC 发送失效之前首先失效 IOATC。

6.3.8. 缓存无效表项（Caching invalid entries）

本规范不允许缓存 V（有效）位为零的第一/第二阶段 PTE、V（有效）位为零的非叶 DDT 表项、V（有效）位为零的设备上下文、V（有效）位为零的非叶 PDT 表项、V（有效）位为零的进程上下文或 V 位为零的 MSI PTE。

将这些表项中的 V 位从 0 改为 1 时，软件无需执行失效操作。

6.4. 重新配置 PMA（Reconfiguring PMAs）

如果平台支持 PMA 的动态重新配置，通常会提供一个机器模式驱动程序来正确配置平台。在某些平台中，这可能涉及平台特定操作，如果 IOMMU 必须参与这些操作，那么机器模式驱动程序就会使用 IOMMU 中的平台特定操作来执行这种重新配置。

6.5. 处理 IOMMU 中断的指南（Guidelines for handling interrupts from IOMMU）

IOMMU 可从 CQ、FQ、PQ 或 HPM 生成中断。每个中断源可配置唯一的向量，也可在一个或多个中断源之间共享一个向量。中断可作为 MSI 或有线信号中断发送。中断处理程序可执行以下操作：

1. 读取 ipsr 寄存器，确定待处理中断的来源
2. 如果设置了 ipsr.cip 位，则 CQ 的中断正在等待处理。
    a. 读取 cqcsr 寄存器。
    b. 通过检查 cmd_to、cmd_ill 和 cqmf 位的状态，确定是否有错误导致中断，如果有，则确定错误原因。

如果这些位中的任何一位被置位，则表示 CQ 遇到错误，命令处理被暂时禁用。

c. 如果发生错误，则纠正错误原因，并通过向 cqcsr 中与纠正错误相对应的位写入 1 来清除这些位。

i. 清除 cqcsr 中的所有错误指示位，重新启用命令处理。

d. 支持有线中断的 IOMMU 可能会被要求在 IOFENCE.C 命令完成时从命令队列中产生一个中断。这一原因由 fence_w_ip 位指示。请注意，当 fence_w_ip 设置为 1 时，命令处理不会停止。软件处理程序可通过向该位写 1 来清除该位，从而在 IOFENCE.C 完成时重新启用来自 CQ 的中断。

e. 向该位写入 1，清除 ipsr.cip。

3. 如果 ipsr.fip 位被置位，则 FQ 等待中断。

a. 读取 fqcsr 寄存器。

b. 通过检查 fqmf 和 fqof 位的状态，确定是否有错误导致中断，如果有，则确定错误原因。如果这两个位中的任何一个被置位，则表示 FQ 发生了错误，故障/事件报告被暂时禁用。

c. 如果发生错误，请更正错误原因，并通过向 fqcsr 中与更正错误相对应的位写入 1 来清除这些位。

i. 清除 cqcsr 中的所有错误指示位，重新启用故障/事件报告。

d. 向该位写入 1，清除 ipsr.fip。

e. 读取 fqt 和 fqh 寄存器。

f. 如果 fqt 的值不等于 fqh 的值，则 FQ 不为空，且包含需要处理的故障/事件报告。

g. 处理需要处理的待处理故障/事件报告，并将其从 FQ 中删除，方法是按已处理记录的数量将 fqh 向前推进。

4. 如果设置了 ipsr.pip 位，则 PQ 的中断正在等待处理。

a. 读取 pqcsr 寄存器。

b. 通过检查 pqmf 和 pqof 位的状态，确定是否有错误导致中断，如果有，则确定错误原因。如果这两个位中的任何一个被置位，则表示 PQ 遇到了错误，“页面请求”报告被暂时禁用。

c. 如果发生错误，请更正错误原因，并通过向 pqcsr 中与更正错误相对应的位写入 1 来清除这些位。

i. 清除 pqcsr 中的所有错误指示位，重新启用“页面请求”报告。

d. 向该位写入 1，清除 ipsr.pip。

e. 读取 pqt 和 pqh 寄存器。

f. 如果 pqt 的值不等于 pqh 的值，则 PQ 不为空，且包含需要处理的“页面请求”报文。

g. 处理需要处理的待处理“页面请求”信息，并将其从 PQ 中删除，方式是将 pqh 按已处理记录数向前推进。

i. 当发生 PQ 溢出时，软件可能会观察到由于 "Page Request（页面请求）"报文被丢弃而导致页面请求组不完整。如果报文中的 "Last Request in PRG（PRG 中的最后一个请求）"标志设为 1，IOMMU 可能会自动响应（见第 2.7 节）此类组中被丢弃的 "页面请求"。软件应忽略此类不完整的组，不为其提供服务。

ii. 在 PQ 溢出时，"PRG 中的最后一个请求 "设置为 1 的 "页面请求 "自动响应预计将导致设备重试 ATS 转换请求。但是，由于 IOMMU 生成的响应并没有实际解决导致设备最初发送 "页面请求 " 的条件，这可能会导致设备再次发送 "页面请求 "报文。如果溢出情况已通过在 PQ 中创建空间得到纠正，这些重试的报文现在可以存储在 PQ 中。

5. 如果设置了 ipsr.pmip 位，则表示 HPM 正在等待中断。

a. 向该位写入 1，清除 ipsr.pmip。

b. 处理性能监控计数器溢出。

6.6. 启用和禁用 ATS 和/或 PRI 的指南（Guidelines for enabling and disabling ATS and/or PRI）

启用 ATS 和/或 PRI：

1. 将设备置于空闲状态，使设备不产生任何事务。
2. 如果设备的设备上下文已经有效，则首先将设备上下文标记为无效，然后向 IOMMU 发送队列命令，使所有缓存的第一/第二阶段页表表项、DDT 表项、MSI PT 表项（如需要）和 PDT 表项（如需要）无效。
3. 对设备上下文进行编程，将 EN_ATS 设置为 1，并根据需要将 T2GPA 字段设置为 1。如果需要，将 EN_PRI 设为 1。如果 EN_PRI 设置为 1，则根据需要将 PRPR 设置为 1。
4. 将设备上下文标记为有效。
5. 启用设备使用 ATS，必要时启用 PRI。

禁用 ATS 和/或 PRI：

1. 将设备置于空闲状态，使设备不产生任何事务。
2. 在设备上禁用 ATS 和/或 PRI
3. 在设备上下文中将 EN_ATS 和/或 EN_PRI 设置为 0。如果 EN_ATS 设置为 0，则设置 EN_PRI 和 T2GPA 为 0。如果 EN_PRI 设置为 0，则 PRPR 设置为 0。
4. 向 IOMMU 发送队列命令，使所有缓存的第一/第二阶段页表项、DDT 项、MSI PT 项（如需要）和 PDT 项（如需要）无效。
5. 向 IOMMU 发送队列命令，通过生成 Invalidation Request 报文使 DevATC 无效。
6. 启用设备中的 DMA 操作。

第 7 章 硬件指南（Hardware guidelines）

本节为平台中 IOMMU 的系统/硬件集成商提供指导。

7.1. 将 IOMMU 集成为 PCIe 设备（Integrating an IOMMU as a PCIe device）

IOMMU 本身可作为 PCIe 设备构建，并可作为专用 PCIe 功能被发现，具有 PCIe 定义的基类 08h、子类 06h 和编程接口 00h [4]。

此类 IOMMU 必须将本规范中定义的 IOMMU 寄存器映射为 PCIe BAR 映射寄存器。

IOMMU 可支持 MSI 或 MSI-X，或两者兼有。支持 MSI-X 时，MSI-X 功能块必须指向 BAR 映射寄存器中的 msi_cfg_tbl，以便系统软件能够为 IOMMU 支持的每个报文配置 MSI 地址和数据对。MSI-X PBA 可位于 IOMMU 的同一 BAR 或其他 BAR 中。建议 IOMMU 支持 MSI-X 功能。

7.2. PMA 和 PMP 故障（Faults from PMA and PMP）

IO Bridge 可对来自 IO 设备或由 IOMMU 隐式生成的内存访问调用 PMA 和/或 PMP 检查器，以访问内存数据结构。当内存访问违反 PMA 检查或 PMP 检查时，IO Bridge 可按照第 7.3 节的规定中止内存访问。

7.3. 中止事务（Aborting transactions）

如果中止的事务是 IOMMU 发起的隐式内存访问，那么 IO Bridge 就会向 IOMMU 本身发出此类访问故障信号。此类信号的具体细节由实现定义。

如果中止的事务是写事务，那么 IO Bridge 可能会丢弃写事务；如何丢弃写事务的细节由实现定义。如果 IO 协议要求对写事务做出响应（如 AXI），那么 IO Bridge 可能会生成 IO 协议定义的响应（如 BRESP 上的 SLVERR - 写响应通道）。例如，对于 PCIe，写入事务是 posted 的，当写入事务被丢弃时不会返回任何响应。

如果发生故障的事务是读取，则设备期望完成。IO Bridge 可向设备提供完成。在这种完成过程中，如果返回数据，数据是由实现定义的；通常是一个固定值，如全 0 或全 1。在完成过程中，可能会向设备返回一个状态代码，以指示这种情况。例如，对于 AXI，完成状态由 RRESP（读取数据通道）上的 SLVERR 提供。例如，对于 PCIe，完成状态字段可设置为 "不支持的请求"（UR）或 "完成器中止"（CA）。

7.4. 可靠性、可用性和可维护性（Reliability, Availability, and Serviceability，RAS）

IOMMU 可支持 RAS 架构，该架构规定了启用错误检测、记录检测到的错误（包括其严重性、性质和位置）以及配置向错误处理程序报告错误的方法。

某些错误，如 IOATC 中的错误，可以在检测到错误时通过重新加载缓存的内存数据结构来纠正。预计此类错误不会影响 IOMMU 的运行。

某些错误可能会破坏 IOMMU 的关键内部状态，这些错误可能会导致 IOMMU 进入故障状态。此类状态的例子包括 ddtp、cqb 等寄存器。进入这种故障状态后，IOMMU 可能会要求 IO Bridge 中止所有进入的事务。

有些错误，如在 IOMMU 内部数据路径中发生的损坏，可能无法纠正，但这些错误的影响可能只限于

IOMMU 正在处理的事务。

作为处理事务的一部分，IOMMU 可能需要从 DDT、PDT 或第一/第二阶段页表等内存数据结构中读取数据。数据提供者（内存控制器或高速缓存）可能会检测到所请求的数据有不可纠正的错误，并发出数据已损坏的信号，将错误推迟到 IOMMU 处理。这种将损坏数据的处理推迟到数据消费者的技术通常也被称为数据中毒。此类错误的影响可能只限于导致访问损坏数据的事务。

如果错误影响了正在处理的事务，但 IOMMU 仍可继续提供服务，IOMMU 可中止事务（见第 7.3 节），并通过在 FQ 中队列故障记录来报告故障。例如，对于 PCIe，"Completer Abort (CA)（完成者终止 (CA)）"响应适用于终止事务。以下原因代码用于报告此类故障事务：

- DDT 数据损坏（原因 = 268）
- PDT 数据损坏（原因 = 269）
- MSI PT 数据损坏（原因 = 270）
- MSI MRIF 数据损坏（原因 = 271）
- 内部数据路径错误（原因 = 272）
- 第一/第二阶段 PT 数据损坏（原因 = 274）

如果 IO Bridge 无法将此类延迟错误与其他妨碍 IOMMU 访问内存数据结构的错误区分开来，那么 IOMMU 可能会将此类错误报告为访问故障，而不是使用有区别的数据损坏原因代码。

参考书目

[1] "PCI Express® 基本规范 6.0 版"。[Online].可查阅：pcisig.com/pci-express-6.0- specification。

[2] "RISC-V高级中断架构"。[Online].见：github.com/riscv/riscv-aia。

[3] "RISC-V指令集手册，第二卷：特权体系结构"。[Online].可查阅：github.com/riscv/riscv-ISA-manual。

[4] "PCI代码和ID分配规范1.1修订版"。[Online].可查阅：pcisig.com/sites/ default/files/files/PCI_Code-ID_r_1_11 v24_Jan_2019.pdf。