AMD I/O Virtualization Technology (IOMMU) Specification

Publication # **48882**	Revision: **2.00**
Issue Date: **3/24/11**	

Advanced Micro Devices

Table of Contents

List of Figures

List of Tables

Revision History

Revision history:

Revision 2.00 - March, 2011:

- Updated Section 1.2 [Definitions].
- Added Section 2.2.7 [Virtualized User Mode Device Accesses].
- Added Section 2.3 [Revision 2 Additions], with subsections.
- Updated Section 2.3.6 [Selecting Translation Tables in a Memory Transaction].
- Added Section 2.3.8 [Interrupt Virtualization (Guest Virtual Interrupt Controller)].
- Updated Section 3.1.1 [Normal Operation].
- Updated Section 3.1.2 [IOMMU Logical Topology].
- Updated Section 3.1.4.4 [Address Translation Requests in the IOMMU Exclusion Range]
- Added Section 3.1.4.7 [Discarding IOMMU TLB Information to Rewalk Page Tables].
- Added Section 3.1.4.10 [Address Translation Response When DTE[Mode]=0].
- Added Section 3.1.4.11 [Page Splintering].
- Updated Section 3.2 [Data Structures], including Figure 4 and Section 3.2.2.1 [Device Table Entry Format].
- Updated Section 3.2.2.1 [Device Table Entry Format], adding Table 3 and updating Table 5.
- Updated Section 3.2.2.2 [Making Device Table Entry Changes] and split out Section 3.2.2.3 [Starting the IOMMU] and Section 3.2.2.4 [Making Guest Interrupt Virtualization Changes].
- Updated Section 3.2.3 [I/O Page Tables for Host Translations].
- Added Section 3.2.6 [I/O Page Tables for Guest Translations].
- Added Section 3.2.7 [Guest and Nested Address Translation].
- Added Section 3.2.7.2 [Calculating Page Table and Page Access Attributes].
- Added Section 3.2.7.5 [Clearing Accessed and Dirty Bits].
- Updated Section 3.2.7.6 [Calculating PCIe Read and Write Attributes for an ATS Response].
- Added Section 3.2.7.7 [PCIe TLP PASID Prefix].
- Added Section 3.2.8 [Guest Virtual APIC Table for Interrupt Virtualization].
- Updated Section 3.3 [Commands].
- Updated Section 3.3.3 [INVALIDATE_IOMMU_PAGES].
- Updated Section 3.3.4 [INVALIDATE_IOTLB_PAGES].
- Updated Section 3.3.6 [PREFETCH_IOMMU_PAGES].
- Updated Section 3.3.7 [COMPLETE_PPR_REQUEST].
- Added Section 3.3.8 [INVALIDATE_IOMMU_ALL].
- Updated Section 3.4 [Event Logging], including Section Table 38: [Event Type Summary]; Table 38 - Table 49.
- Updated Section 3.4.1 [ILLEGAL_DEV_TABLE_ENTRY Event].
- Updated Section 3.4.2 [IO_PAGE_FAULT Event].
- Updated Section 3.4.4 [PAGE_TAB_HARDWARE_ERROR Event].
- Updated Section 3.4.7 [IOTLB_INV_TIMEOUT Event].
- Updated Section 3.4.8 [INVALID_DEVICE_REQUEST Event].
- Added Section 3.4.9 [INVALID_PPR_REQUEST Event]
- Added Section 3.4.10 [EVENT_COUNTER_ZERO Event].
- Added Section 3.4.11 [IOMMU Event Reporting].
- Updated Section 3.5 [Peripheral Page Service Request (PPR) Logging].
- Updated Section 3.5.1 [Peripheral Page Service Request Entry].
- Updated Section 3.7.1 [IOMMU Capability Block Registers].
- Updated Section 3.7.2 [IOMMU MMIO Registers] with hardware error registers.
- Updated Section 3.7.2.1 [MMIO Control and Status Registers].
- Updated Section 4.5 [Software and Platform Firmware Implementation Issues].

- Clarified Section 5 [I/O Virtualization ACPI Tables].
- Clarified Section 5.3 [IOMMU ACPI Table Definitions].
- Updated Section 5.3.3 [I/O Virtualization Hardware Definition (IVHD) Block].
- Updated Section 6 [IOMMU Pseudo Code] with hardware error registers.
- Clarified IOMMU Base Address Low Register [Capability Offset 04h] (alignment requirement).

1 Overview

The I/O Memory Management Unit (IOMMU) is a system function that translates addresses used in DMA transactions, protects memory from disallowed access by I/O devices, and remaps peripheral interrupts.

The IOMMU can be used to:

- Replace the existing GART mechanism.
- Remap addresses above 4GB for devices that do not support 64-bit addressing.
- Allow a guest OS running under a VMM to have direct control of a device.
- Provide page granularity control of device access to system memory.
- Allow a device direct access to user space I/O.
- Filter and remap interrupts.

1.1 Intended Audience

This document provides the IOMMU behavioral definition and associated design notes. It is intended for the use of system designers, chipset designers, and programmers involved in the development of low-level BIOS (basic input/output system) functions, drivers, operating system kernel modules, and hypervisors. The intended user should have prior experience in personal computer design, microprocessor programming, and legacy x86 and AMD64 microprocessor architecture.

1.2 Definitions

- **Accessed bit (A)**. A bit in the page table that indicates the corresponding memory has been read or written. Usually set to 1 by hardware.
- **ACPI.** Advanced Configuration and Power Interface, a specification of industry-standard interfaces enabling OS-directed configuration and other management.
- **APIC**. Advanced programmable interrupt controller (see specifications under the model numbers 82093AA and 82489DX).
- **ARI.** Alternative Routing Information is a PCI-SIG specification that allows a PCI Device to have more than eight PCI Functions but no more than 256.
- **ATS.** Address translation service, a PCI-SIG specification, allows a PCI peripheral to request virtual-to-physical address translation from an IOMMU or TA. The resulting translation may be stored in an IOTLB. ATS is optional on a peripheral. This specification requires the *Address Translation Services 1.1 Specification* or later. See http://www.pcisig.com/specifications/iov/ats/ .
- **BAR**. PCI-defined base address register.
- **BIOS**. Refers to the platform firmware (Basic Input/Output Services).
- **Bounce Buffer**. A buffer located in low system memory for DMA traffic from devices that do not support 64-bit addressing. The OS copies the DMA data to or from the buffer to the real buffer in high memory used by the driver.
- **Cold Reset**. A reset generated by removing and reapplying power to the device.
- **Dirty bit (D)**. A bit in the page table that indicates the corresponding memory has been written. Usually set to 1 by hardware.
- **Device Exclusion Vector (DEV)**. Contiguous arrays of bits in physical memory. Each bit in the DEV table represents a 4KB page of physical memory (including system memory and MMIO). The DEV table is packed as follows: bit[0] of byte 0 controls the first 4K bytes of physical memory; bit[1] of byte 0 controls the second 4K bytes of physical memory; etc.
- **DeviceID**. A 16 bit device identification number consisting of the Bus number, Device number and Function number. Used by an IOMMU to select the nested mapping tables for an address translation or interrupt remapping operation.
- **Device Processing Complex.** A computational unit on the peripheral such as a dedicated function (e.g., NIC,

encryption engine), a graphics processing unit (GPU), or an accelerated computing element (AC)

- **Device Table**. A table in system memory that maps DeviceIDs to DomainIDs and page table root pointers.
- **Device Table Entry (DTE)**. An entry in the device table.
- **Device Virtual Address**. The untranslated address used by a device in a DMA transaction. If the IOMMU is not enabled this address corresponds to the system physical address.
- **Direct Memory Access (DMA)**. A feature that enables a peripheral to access memory without intervention by the central processor.
- **Domain**. See Protection Domain.
- **DomainID.** A 16-bit number chosen by software to identify a domain.
- **GART**. Graphics Address Remapping Table.
- **GPU**. Graphical processing unit, usually used for graphics-specific computation.
- **GPGPU**. A GPU used for general-purpose computation.
- **Guest**. An application or OS run by the host in its own virtual environment.
- **Guest address translation.** Translation for GVA to GPA. May be serviced by an IOMMU or by a private MMU on the peripheral.
- **Guest Physical Address (GPA)**. The x86-canonical virtual address used by a guest operating system in a VM. A GPA is created by using the guest page tables to translate a guest virtual address. The GPA may be further translated to a System Physical Address.
- **Guest Virtual Address (GVA)**. The virtual addresses used by a guest application. A GVA may be translated into a Guest Physical Address. Guest virtual addresses are treated as canonical x86 addresses.
- **Guest Virtual APIC.** The IOMMU (Revision 2) can support the delivery of interrupts to guest VMs without hypervisor intervention. The guest APIC is described in the *AMD Virtual Interrupt Controller Specification, Revision 1.0* or newer.
- **Host Data Path (HDP).** A functional unit that can convert CPU linear addressing to GPU-style tiled or rectangular addressing for improved performance. Often found in advanced graphics processing peripherals.
- **High memory.** In the x86 architecture, this is memory with addresses at or above 4G bytes.
- **Host**. The system software layer responsible for running guests. See also Nested paging and Nested address translation.
- **Hypervisor**. See VMM, Virtual Machine Monitor.
- **IOMMU**. Refers to the I/O Memory Management Unit defined by this specification.
- **IOTLB**. Refers to the I/O Translation Buffer on a peripheral; sometimes called a “remote IOTLB” because it is on the peripheral, remote from the processor.
- **IVHD.** I/O Virtualization Hardware Definition block, an ACPI table defined in Section 5.3.3 [I/O Virtualization Hardware Definition (IVHD) Block].
- **IVMD.** I/O Virtualization Memory Definition block, an ACPI table defined in Section 5.3.9 [I/O Virtualization Memory Definition (IVMD) Block].
- **IVRS**. I/O Virtualization Reporting Structure block, an ACPI table defined in Section 5.3.1 [I/O Virtualization Reporting Structure (IVRS)].
- **LMA.** Local Memory Address; corresponds to the physical address space used on the peripheral to access on-board or private memory. In some peripherals, aperture hardware maps some or all of the local memory address space into the system physical address space. The aperture hardware is usually managed by a device driver in an operating system.
- **Local Memory.** Memory on the peripheral that is typically accessed more quickly than system memory and is usually not coherent with system memory. Part of the local memory may be addressable from the CPU (called "public") and part may be inaccessible from the CPU (called "private"). An aperture mechanism is commonly used to select the portion of local memory that is public.
- **Local Memory Protection Map.** A hardware component that enforces the separation of virtual machine contexts within the local memory of a peripheral.
- **Low memory.** Memory with addresses below 4G bytes.
- **MMIO**. Read or write access to memory mapped resources provided by devices.
- **MMU**. Memory Management Unit.

- **Message Signalled Interrupt (MSI)**. An interrupt that is signalled by generating a posted write to a system-defined physical address.
- **Nested address translation.** Translation for GPA to SPA. May be serviced directly by an IOMMU or by a remote IOTLB. Use of an IOTLB requires ATS and/or PRI.
- **Nested paging.** An optional feature in AMD64 processors, the nested paging feature provides for two levels of address translation, thus eliminating the need for the virtual machine manager to maintain shadow page tables. See *AMD64 Architecture Programmer's Manual, Volume 2: System Programming*, AMD publication number 24593 (APM Volume 2).
- **NW.** A PCI-SIG term (bit) used to signal lack of intent to perform write operations.
- **OpenCL.** A general-purpose parallel programming language for heterogeneous systems such as graphics processing units. See http://www.khronos.org/opencl/ .
- **Page Tables**. A table structure in main memory used to translate an address from one representation to an alternate representation.
- **PASID**. The Process Address Space ID used to identify the application address space within a x86-canonical guest virtual machine. It is used on a peripheral to isolate concurrent contexts residing in shared local memory. Together, PASID and DeviceID uniquely identify an application address space. See *PASID TLP prefix*.
- **PASID TLP prefix**. The IOMMU requires that a virtual address with a PASID carry the PASID value using the PASID TLP prefix. See also PASID and TLP. See the PCI-SIG *PASID TLP Prefix ECN* specification.
- **PCI, PCI-SIG, PCIe, PCI-X**. The PCI-SIG is an industry standards body that defines I/O connection technology, including PCI, PCI-X, and PCIe. See http://www.pcisig.com/home for more information.
- **PDE**. Page directory entry for address translation (see example in Figure 10).
- **Pinned memory.** Memory pages that are to be maintained in real memory all the time. Pinning a memory page prevents the page management software from using it for other purposes. A memory page must typically be pinned before DMA starts and may be unpinned when DMA completes.
- **Platform firmware.** The firmware or software that controls startup and configuration of the platform. Platform firmware is commonly implemented as BIOS or UEFI.
- **PPR**. Peripheral Page Service Request. When the IOMMU receives a valid PRI request, it creates a PPR message to request changes to the virtual address space.
- **PR or P: present.** The present bit in the page table entries as shown in Figure 8, Figure 9, Figure 10, Figure 22, Figure 23, Figure 24, and Figure 25.
- **Pretranslated address.** An address that has been translated to an SPA by a peripheral with an IOTLB.
- **Page Request Interface (PRI).** The Page Request Interface is a PCI-SIG specification that defines how a peripheral requests memory management services from a host OS or hypervisor (e.g., page fault service for the peripheral). PRI is optional on a peripheral, but if PRI is implemented, ATS is required.
- **Private MMU**. A peripheral-specific mechanism to translate addresses generated on the peripheral. In the simplest case, it generates a single bit to indicate the input address is an access to peripheral local memory or to system memory. When present, the private MMU provides guest address translation. On a GPU, a private MMU is often referred to as the VM component of the memory controller.
- **Protection Domain**. A set of address mappings and access rights that can be shared by multiple devices.
- **Page Table Entry, PTE**. A page table translation entry controls virtual-to-physical address translation and memory page access (see example in Figure 9).
- **System Physical Address (SPA)**. The address used by the DRAM controller to specify a specific memory location or the address given to a MMIO device to specify a specific MMIO register.
- **System software.** The software that controls the normal operation of the platform. System software is commonly implemented as a hypervisor or an operating system.
- **TA**. Translation Agent is a PCI-SIG term to refer to the IOMMU table walker.
- **TLB**. Translation Look-aside Buffer is a cache of address translation information usually implemented within an MMU to improve translation speed.
- **TLP**. Transaction Layer Packet is a PCIe term for non-control packets. The TLP packet may have a prefix.
- **UEFI**. Refers to the "Unified Extensible Firmware Interface" specification for platform firmware. See

http://www.uefi.org/home/ .

- **Untranslated address**. A virtual address (GVA or GPA) issued by a peripheral that will be translated to an SPA by the IOMMU. The handling of an untranslated address on a peripheral is outside the scope of this specification.
- **User, U/S, User/Supervisor level**. The IOMMU can provide privilege-level information to a peripheral. The value 0b means supervisor level access is allowed, and 1b means user and supervisor access are allowed. The terms **User** and **U/S** are used, depending on the context.
- **VM**. A virtual machine is created and managed by a hypervisor so that multiple virtual machines can share a single hardware system and run independent operating system instances.
- **VMM, Virtual Machine Monitor**. A VMM is the controlling software for a computer. It manages the physical hardware and VMs to allow multiple operating systems to run concurrently on a computer system. Also known as a hypervisor.

1.3 Bit Attributes

All bit attributes used in this specification are defined in Table 1. These attributes apply to register definitions, device table entries, page table entries, command buffer entries and event log entries.

Table 1: Bit Attribute Definitions

Attribute	Description
HwInit	**Hardware Initialized**: Register fields are initialized by firmware or hardware mechanisms such as pin strapping or serial EEPROM. Fields are read-only after initialization and can only be reset (or write-once by firmware) with a cold reset.
Ignored Ign	**Ignored or Ign:** For an IOMMU register, the state of the field is ignored by the IOMMU, writes may be discarded and reads return undefined results. For a memory location, the contents of the field is ignored by the IOMMU when read, but the value is preserved when the memory location is written by the IOMMU. Note that some ignored fields may be used by other system components (e.g., a memory field in a page table entry that is ignored by the IOMMU may be used by the processor).
RO	**Read-only register:** Register fields are read-only and cannot be altered by software.
RW	**Read-Write register:** Register fields are read-write and may be either set or cleared by software to the desired state.
RW1C	**Read-only status, Write-1-to-clear status register:** Register bits indicate status when read, a set bit indicating a status event may be cleared by writing a 1. Writing a 0 to RW1C bits has no effect.
RW1S	**Write-1-to-set register:** Register bits indicate status of an operation when read, setting the bit initiates the operation. Hardware clears the bit when the operation completes. Writing a 0 to RW1S bits has no effect.
Reserved Resv Res	**Reserved, Resv, or Res:** Reserved for future implementations. Reserved fields in a register must be implemented as read-only zero. Reserved fields in a memory location must be zero.
Unused Un	**Unused or Un:** Field is not used by hardware. Software is allowed to use the field for its own purposes.

2 IOMMU Overview

The I/O Memory Management Unit (IOMMU) extends the AMD64 system architecture with support for address translation and access protection on DMA transfers by peripheral devices. The IOMMU enables several significant enhancements to system-level software:

- Legacy 32-bit I/O device support on 64-bit systems (generally without requiring bounce buffers and expensive memory copies).
- More secure user-level application access to selected I/O devices.
- More secure virtual machine guest operating system access to selected I/O devices.

The IOMMU can be thought of as a combination and generalization of two facilities included in the AMD64 architecture: the Graphics Aperture Remapping Table (GART) and the Device Exclusion Vector (DEV). The GART provides address translation of I/O device accesses to a small range of the system physical address space, and the DEV provides a limited degree of I/O device classification and memory protection. In combination with appropriate software manipulation of processor nested page tables, the IOMMU can provide GART or DEV functionality.

2.1 Architecture Summary

The detailed architecture of the IOMMU is discussed in Section 3 [Architecture]. The remainder of Chapter 2 consists of a brief summary of the architecture of the IOMMU along with a discussion of some anticipated usage models.

The IOMMU extends the concept of protection domains (domains for short) first introduced with the DEV. The IOMMU allows each I/O device in the system to be assigned to a specific domain and a distinct set of I/O page tables. When an I/O device attempts to read or write system memory, the IOMMU intercepts the access, determines the domain to which the device has been assigned, and uses the TLB entries associated with that domain or the I/O page tables associated with that I/O device to determine whether the access is to be permitted as well as the actual location in system memory that is to be accessed.

The IOMMU may include optional support for remote IOTLBs. An I/O device with IOTLB support can cooperate with the IOMMU to maintain its own cache of address translations. This creates a framework for creating scalable systems with an IOMMU in which I/O devices may have different usage models and working set sizes. IOTLB-capable I/O devices contain private TLBs tailored for their own needs, creating a scalable distributed system of TLBs. The performance of IOTLB-capable I/O devices is not limited by the number of TLB entries implemented in the IOMMU. A peripheral with an IOTLB may issue untranslated addresses or pretranslated addresses that are determined from IOTLB entries. Pretranslated addresses are not checked by the IOMMU except to validate that the peripheral has the IOTLB enable bit set (I=1) in the corresponding DTE (see Figure 7 and Table 5).

Revision 2: The IOMMU may include optional support for peripheral page service requests (PPR) for peripherals that use ATS. This creates a mechanism for peripherals and software to reduce the need for pinned pages during I/O. The IOMMU may include optional support for interrupt virtualization. This uses a virtualized guest APIC with memory tables to deliver interrupts to guest VMs.

Major system resources provided by the IOMMU include:
- I/O page tables which the IOMMU uses to provide permission checking and address translation on memory accesses by I/O devices.
- Revision 2: I/O page tables using the `AMD64 long` format.
- A device table that allows I/O devices to be assigned to specific domains and contains pointers to the I/O

devices' page tables.
- An interrupt remapping table which the IOMMU uses to provide permission checking and interrupt remapping for I/O device interrupts.
- Revision 2: A guest virtual APIC mechanism which the IOMMU uses to deliver interrupts to guest VMs.
- Memory-based queues for exchanging command and status information between the IOMMU and the system processor(s). The command queue and event log are implemented by each IOMMU.
- Revision 2: a peripheral page service request queue is optionally implemented by an IOMMU.

In summary, the IOMMU is very similar to the processor's MMU, except that it provides address translation and page protection to memory accesses by peripheral devices rather than memory accesses by the processor and that it provides an interrupt remapping capability. However, compared to the processor's MMU, the IOMMU has a few differences.

The first difference is that the IOMMU provides no direct indication to an I/O device of a failed translation when processing an untranslated posted request.

A second difference is related to the organization of the AMD64 system architecture. AMD64 systems can consist of a number of processor and device nodes connected to each other by HyperTransport™ links. The IOMMU can only process memory transactions that are routed through its node in the system fabric. In a system with multiple links and buses to I/O devices (see Figure 1), multiple IOMMUs are required to ensure that each I/O link or bus has appropriate protection and translation applied. Figure 1 shows the IOMMU implemented in an I/O Hub as one example selected from many possibilities.

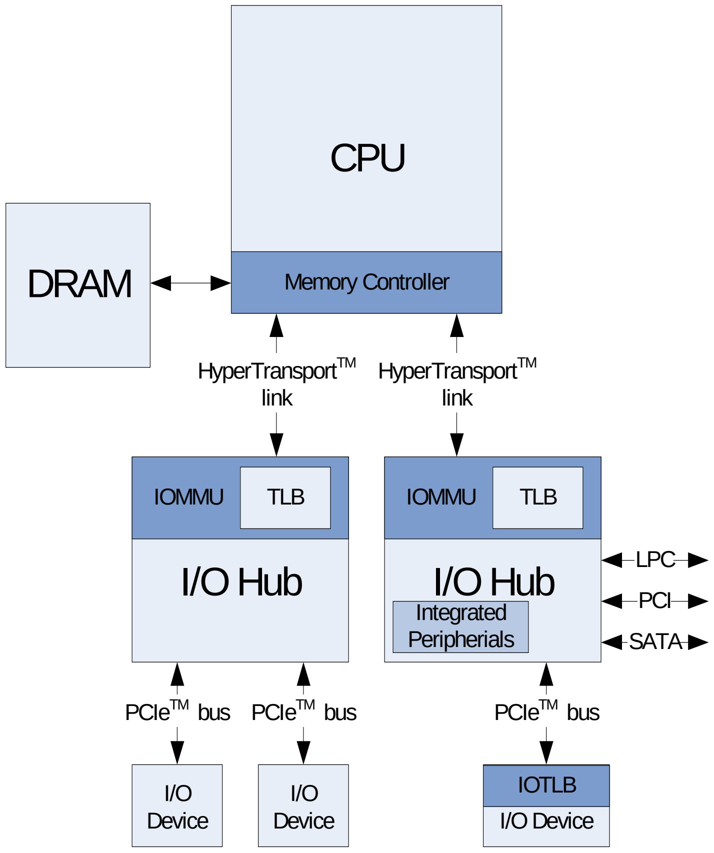


Figure 1: Example Platform Architecture

A third difference is that the IOMMU uses a command queue in memory to accept translation buffer invalidations while the system processors in an SMP use an interrupt synchronization method (the so-called TLB shoot-down interprocessor interrupt).

Revision 2: A fourth difference is that the IOMMU optionally provides a request queue in memory to service peripheral page requests while the system processor uses a fault mechanism.

Revision 2: The IOMMU optionally supports two-level address translation for nested page tables. The nested translations are managed according to the Revision 1 definition and the guest translations are an extension that is available as shown in Table 3. The guest translations are directly compatible with `AMD64 long` page tables supporting 4K byte, 2M byte, and 1G byte pages. The guest address space is flat, has no special ranges or properties, and Table 2 does not apply to guest virtual addresses.

Revision 2: The IOMMU has architectural features designed to support the virtualized guest APIC.

2.2 Usage Models

Six models are discussed to highlight potential uses of the IOMMU in conventional and virtualized systems. These usage models can enhance system security and stability.

2.2.1 Replacing the GART

The GART is a system facility that performs physical-to-physical translation of memory addresses within a graphics aperture. The GART was defined to allow complex graphical objects, such as texture maps, to appear to a graphics co-processor as if they were located in contiguous pages of memory, even though they are actually scattered across randomly allocated pages by most operating systems. The GART translates all accesses to the graphics aperture, including loads and stores executed by the host processor as well as memory reads and writes performed by I/O devices. Only accesses whose system physical addresses are within the GART aperture are translated; however, the results of the translation can be any system physical address.

Unlike the GART, the IOMMU translates only memory accesses by I/O devices. However, with appropriate programming, a host OS can use the IOMMU as a functional equivalent of the GART. First, the host OS must set up its own page tables to perform translations of host processor accesses formerly translated by the GART. Then, to set up the equivalent translations for I/O device-initiated accesses, the host OS must:
- Construct I/O page tables that specify the desired translations.
- Make an entry in the device table pointing to the newly constructed I/O page tables.
- Notify the IOMMU of the newly updated device table entry if NPcache=1.

At this point, all accesses by both the host processor and the graphics device are mapped to the same pages as they would have been by the GART.

If the host OS changes the page protection or translation, it must update both the processor page tables and, if not shared, the I/O page tables, and issue appropriate page-invalidate commands to both the processor and the IOMMU. Unlike the processor, the IOMMU may require page-invalidate commands after any change to the I/O page tables. (AMD64 processors do not require page-invalidate operations after changes to leaf page table entries that add permission and make no change to translation.) Sharing of page tables is discussed in Section 3.2.1 [Updating Shared Tables] and Section 3.2.4 [Sharing AMD64 Processor and IOMMU Page Tables - GPA-to-SPA].

Eventually the host OS may have to tear down the mappings when they are no longer used (e.g., removed from the system). The procedure is similar to setup:

- Mark the device table entry as not valid by setting V=1b, IV=0b, and IntCtl=00b in the device table entry. If transaction pass-through is acceptable, set V=0b and IV=0b.
- Notify the IOMMU of the newly invalidated device table entry.
- Wait for the IOMMU to indicate that the invalidation is complete.
- Finally, de-allocate the I/O page tables.

Since the IOMMU offers no facilities for restarting device accesses to unmapped or protected addresses, all pages that the device might access must be mapped with appropriate permissions. In this respect the IOMMU is similar to the GART.

The IOMMU cannot be used to emulate the GART if processor paging is not enabled; in that case host processor accesses are not translated. This should not be a problem in practice, however, since historically the GART has been used by systems that enable paging on the processor.

In the foregoing procedures for setup and teardown of IOMMU page tables, the order of operations is chosen to prevent the IOMMU from ever looking at device or page table contents before they are initialized. During setup, the I/O page tables are constructed before the pointers are installed, and in teardown the pointers are cleared before the page table is deleted. Similar principles apply to the other applications in this chapter.

2.2.2 Substituting for the DEV

The Device Exclusion Vector is a basic security mechanism that was introduced with Secure Virtual Machine Architecture. Like the IOMMU, the DEV allows I/O devices to be classified into different domains. Associated with each domain is a bit vector, indexed by physical page address, indicating whether I/O devices in that domain are allowed to access the corresponding physical page.

The IOMMU provides protection and translation. If only protection is needed, software can create identity-mapped I/O page tables that specify the desired protection.

2.2.3 32-bit to 64-bit Legacy I/O Device Mapping

With the advent of large physical memories, legacy 32-bit devices that rely on DMA can no longer access arbitrary system memory. This complicates operating systems, which must introduce a distinction between low memory and high memory, and perform appropriate bookkeeping to ensure that legacy I/O devices are only commanded to perform transfers using low memory. The cost is not just complexity: to perform a transfer from a legacy I/O device to high memory, for example, the operating system typically allocates a bounce buffer in low memory, performs the transfer in low memory, and then copies the result to the real destination in high memory. For high-bandwidth I/O devices like disk controllers and network interfaces, the performance cost of bounce buffer allocation and copying can be large.

In some operating systems, the GART has been used to work around this problem. When the OS wishes to perform a transfer between a legacy I/O device and high memory, it allocates a portion of the GART aperture and maps those pages to high memory. It then commands the I/O device to execute the transfer using the address within the GART aperture, which must be located in low memory. Although this approach avoids the cost of bounce buffer copies, it is less than desirable, since the relatively small GART aperture must be shared by all legacy I/O devices and any graphics processors in the system. The device drivers have additional locking and synchronization overhead associated with page allocation and de-allocation in the GART aperture and system performance may be degraded due to serialization waiting for the GART aperture to become available.

The IOMMU provides a better solution. First, IOMMU translation applies to the full range of addresses an I/O device can generate, rather than requiring high-memory transfers to be mapped only within the narrow range of

GART addresses. Moreover, the IOMMU's ability to assign each I/O device to a different domain means that heavily used I/O devices can be given their own sets of I/O page tables, and do not have to contend with other I/O devices for allocation and de-allocation of I/O pages.

2.2.4 User Mode Device Accesses

The IOMMU plays a crucial role in allowing arbitrary I/O devices to be safely controlled by user-level processes, since I/O devices whose memory accesses are translated by the IOMMU can only access pages that are explicitly mapped by the associated I/O page tables. The I/O devices' access can therefore be limited to those pages to which the user processes legitimately have access.

Setting up the IOMMU for user-level I/O to an I/O device may be set up similarly to GART emulation with two differences: first, the mappable address range is the entire range of I/O device-generatable addresses, and secondly the operating system is not necessarily required to make exactly equivalent mappings in the processor page tables (although most likely it will).

Even with the help of the IOMMU, enabling user level I/O device access involves many design considerations. Protecting and remapping DMA is one part of the problem; the other part is interrupt management, for which the IOMMU provides help.

As was the case with GART emulation, system software must assess the need to lock in memory all pages that might ever be accessed by an I/O device controlled by a user-level process. Peripherals that implement an IOTLB or use ATS can use the peripheral page service request mechanism optionally implemented by an IOMMU.

2.2.5 Virtual Machine Guest Access to Devices

The IOMMU can be used to allow unmodified virtual machine guest operating systems to directly access I/O devices. This is really just a special case of allowing user-level access to I/O devices, but there are a few considerations that warrant separate mention.

First of all, a non-VM-aware guest has no current way of informing its Virtual Machine Monitor (VMM) which pages an I/O device might access, so the VMM must lock the entire guest in memory. The VMM’s I/O page tables for the guest should then simply map guest physical addresses to system physical addresses. If the VMM is running the guest under nested paging and is using nested page tables built to be compatible with the IOMMU, then the IOMMU can directly share the host page tables for the guest.

Often a single VM guest has direct access to multiple I/O devices. By design, all I/O devices in the guest that need to see exactly the same I/O page translations can share a DomainID. If all the I/O devices belonging to a given VM guest are assigned to the same domain then the IOMMU can share translation cache entries among any of the guest’s I/O devices.

Finally, guest I/O throughput is often significantly enhanced when guest memory is allocated using large pages on the host system. Then the I/O page tables can similarly use large pages and the IOMMU is more likely to avoid thrashing in its translation cache.

2.2.6 Virtualizing the IOMMU

The IOMMU has been designed so that it can be emulated in software by a VMM that wishes to present its VM guests the illusion that they have an IOMMU.

VMMs that run non-VM-aware guests already intercept and emulate attempts by their guests to access PCI configuration space. Therefore, emulation of the IOMMU configuration registers is straightforward; the emulation can be hooked directly to the existing facilities of the VMM for intercepting PCI configuration space accesses.

The VMM must also arrange to intercept and emulate guest accesses to the IOMMU's MMIO-mapped command registers. Since the overhead of each VMM intercept is high, guest operating systems accessing the IOMMU have better performance when they enqueue batches of commands in the IOMMU's (DRAM-based) command buffer prior to initiating IOMMU command processing via an MMIO register access.

Since an untrusted guest OS cannot be allowed to write in the real device table, the VMM must maintain shadow entries in the real table on behalf of the guest. The IOMMU architecture requires software to issue invalidate-entry commands to the IOMMU after updating device table entries. The VMM can intercept these invalidate commands, look up the corresponding entries in the guest's simulated device table, and make shadow entries in the real device table on behalf of the guest. Note that the DeviceIDs as seen by the guest need not be the same as the real DeviceIDs, and the DomainIDs used by the guest are almost certainly not the same as the DomainIDs used by the VMM in the real device table.

In addition, for each guest VM I/O page table, the VMM must construct a shadow I/O page table. This shadow I/O page table is the page table that is given to the real IOMMU. Unfortunately, since an incomplete I/O device access cannot be restarted, the VMM must construct each guest domain's complete shadow I/O page tables eagerly as soon as the guest enables paging for that domain. The VMM must write-protect guest I/O page tables from the guest in order to intercept all guest updates and propagate the updates to the shadow I/O page tables.

Revision 2: The hypervisor can implement a subset of the IOMMU Revision 2 features by reporting that subset via the IOMMU Extended Feature Register [MMIO Offset 0030h]. The subset of additional features can be implemented using the same techniques described above.

2.2.7 Virtualized User Mode Device Accesses

An IOMMU with two-level translation enforces system protection policies while allowing arbitrary I/O devices to be properly controlled by user-level processes in a virtualized system. As noted in Section 2.2.4 [User Mode Device Accesses], I/O devices whose memory accesses are translated by the IOMMU can only access pages that are explicitly mapped by the associated I/O page tables as granted by the hypervisor and operating system. The I/O devices' access can therefore be limited to only those pages to which the user-level processes legitimately have access when the device supplies PASID information. This means I/O operations can be initiated without hypervisor or operating system intervention.

In addition to address translation, enabling user level I/O device access involves other design considerations such as remapping interrupts.

System software must assess the need to lock in memory all pages that might ever be accessed by an I/O device controlled by a user-level process. Peripherals that use ATS can use the peripheral page service request mechanism when implemented by an IOMMU.

2.3 Revision 2 Additions

Revision 2 of the IOMMU adds:
- guest address translation capabilities while retaining backwards compatibility;
- `AMD64 long` page-table compatibility;

- performance features; and
- PCI-SIG PRI and PASID TLP prefix ECN support.

To identify a Revision 2 IOMMU, software must check that Capability Offset 00h[EFRSup]=1b and IOMMU Extended Feature Register [MMIO Offset 0030h] is non-zero.

2.3.1 Two-level Translation for Guest and Host Address Spaces

Revision 1: one level of address translation is supported. Revision 2: The IOMMU adds an optional layer of guest address translation similar to the processor nested paging capability. The layered address translation may be viewed as a nested address spaces as illustrated in Figure 2. Each address space has a set of address translation tables. The IOMMU can provide guest-physical-to-system-physical address translation as specified in Revision 1 and managed by the hypervisor (sometimes called "L2 translation"). The device table entry is extended to include optional address translation information for guest-virtual-to-guest-physical address translation managed by the guest operating system (sometimes called "L1 translation"). This allows for advanced computation architectures in virtualized systems such as compute-offload, user-level I/O, and accelerated I/O devices. The IOMMU indicates that two-level translation is supported via MMIO Offset 0030h[GTSup]. When supported, two-level translation is activated by programming the appropriate device table entries.

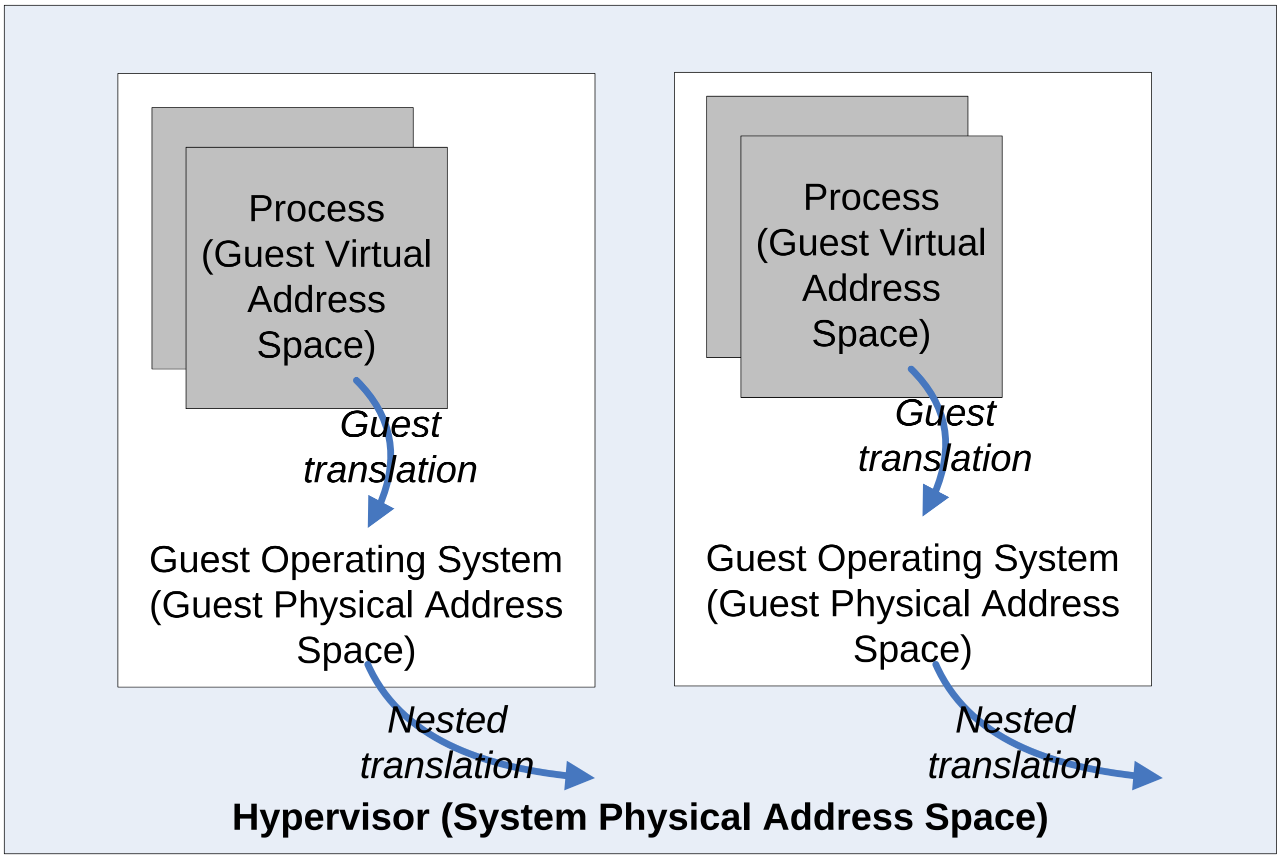

Process (Guest Virtual Address Space)
Guest translation
Guest Operating System (Guest Physical Address Space)
Nested translation
Process (Guest Virtual Address Space)
Guest translation
Guest Operating System (Guest Physical Address Space)
Nested translation
Hypervisor (System Physical Address Space)

Figure 2: Nested Address Spaces

The guest address translation tables can support up to 1048576 (2^{20}) concurrent processes as an architectural limit; an IOMMU may set an implementation limit using MMIO Offset 0030h[PASmax]. The guest address translation tables contain guest physical addresses and the tables are indexed using guest virtual addresses. As a result, the tables are managed by the guest operating system within a virtual machine. The hypervisor manages the nested translation tables and the IOMMU hardware provides mechanisms to keep the tables synchronized and to handle exception conditions. The IOMMU automatically walks address translation tables based on control bits set by the system software.

The IOMMU may be used in three operational modes to do legacy one-level translation, guest and nested translation, and one-level translation with `AMD64 long` page tables. These three modes may be used concurrently for different peripherals.

- For legacy operation, software programs GV=0 in the DTE.
- For guest and nested two-level translation, software checks MMIO Offset 0030h[GTSup]=1. Software is then able to program device table entries for two-level translations.
- For one-level translation with `AMD64 long` page tables, software programs the IOMMU for guest and nested translation but programs DTE[Mode]=000b for the nested translation.

2.3.2 Enhanced `AMD64 long` Page Table Compatibility

Revision 1: The IOMMU can share nested (host) page tables with the processor when the Reserved fields are programmed to zeros. In contrast to the AMD64 CPU, the IOMMU does not rewalk page tables when an access violation is detected using cached information. When the IOMMU detects an access violation in a nested transaction, either from a TLB hit or from a page-table walk (TLB miss), it blocks the access or returns an ATS response with the calculated access privileges. When the IOMMU determines the proper access privileges are present, it allows the requested access or returns an ATS response with the calculated access privileges.

Revision 2: The compatibility of the IOMMU with `AMD64 long` page tables is enhanced. The IOMMU can directly share `AMD64 long` page tables with the processor for guest address translations. The guest page translation tables are strictly compatible with the `AMD64 long` format and semantics, including IOMMU updates to the Accessed and Dirty bits (see Section 3.2.6 [I/O Page Tables for Guest Translations] and Section 3.2.7.4 [Updating Accessed and Dirty Bits in the Guest Address Tables]). When guest translation is used, the IOMMU follows the `AMD64 long` address translation requirements for guest virtual addresses and thus software is not required to issue an invalidation command when it promotes guest access privileges; only when software demotes guest access privileges or removes the guest page (“present to not-present”) must software issue an invalidation. Therefore an ATS request or DMA reference that results in insufficient guest privileges calculated from a TLB entry may be based on stale information. To determine current permissions, the IOMMU must rewalk the guest page tables to recompute access permission using information read from memory. The nested page tables may be read as a consequence of the guest table rewalk. The IOMMU determines the results of the access based on the newly read page table information. The rewalk may require a full walk of both guest and nested translations. Details are in Section 3.2.7 [Guest and Nested Address Translation]). The `AMD64 long` page tables contain information about memory types (PAT); the IOMMU can provide memory type information to a peripheral but does not interpret or validate the information.

2.3.3 Performance Features

Revision 1: Performance counters are implementation-specific.

Revision 2: Three performance-oriented features are available: performance counters, the PREFETCH command, and the FLUSH_ALL command.

2.3.3.1 Performance Counters

Revision 1: Performance counters may be supported by an implementation but are not required in the architecture.

Revision 2: To provide system software with consistent performance monitoring and evaluation mechanisms, an optional set of performance counters are defined (see MMIO Offset 0030h[PCSup]). An implementation may provide additional counters. The counters run independent of processor activity. The counters are organized into counter banks that fit in a 4K byte page so they can be used by a hypervisor or assigned to a guest operating system. The number of counters and counter banks are reported to system software (see Section 3.7.2 [IOMMU MMIO Registers] and Section 5.3.3 [I/O Virtualization Hardware Definition (IVHD) Block]). Each counter bank has controls that filter for devices and event sources of interest. Each event counter is programmed to count events or the duration of the events, and each counter register has an optional signal for thresholding purposes (see Section 3.4.10 [EVENT_COUNTER_ZERO Event]).

2.3.3.2 Loading the IOMMU TLB

Revision 1: The IOMMU loads the TLB in response to peripheral interrupts or accesses to memory.

Revision 2: The PREFETCH_IOMMU_PAGES command gives system software the ability to load the IOMMU TLB with relevant translation information (see Section 3.3.6 [PREFETCH_IOMMU_PAGES]), especially error processing information (Section 3.3.6.1 [Event Processing for PREFETCH_IOMMU_PAGES]).

Support for the prefetch feature is indicated by MMIO Offset 0030h[PreFSup]. If PreFSup=0, a PREFETCH_IOMMU_PAGES command causes the IOMMU to create an error event (Section 3.4.5 [ILLEGAL_COMMAND_ERROR Event]). Because a TLB is a caching structure, the prefetch command must be considered advisory. Even if the IOMMU were to fetch the address translation information for every prefetch command, the TLB entry may be overwritten by other translation information before it is ever used and an attempt to use the translation information would cause a page table walk after all.

The PREFETCH_IOMMU_PAGES command is a hint to the IOMMU that the associated translation records will be needed relatively soon and that the IOMMU should execute a page table walk to load the translation information. Based on internal status and workloads, the IOMMU may fetch the translation information into a TLB. If an entry is already in the TLB, the IOMMU may adjust LRU or other control tags to lengthen cache residency.

2.3.3.3 Flushing the IOMMU TLB

Revision 1: System software flushes the IOMMU caches using the INVALIDATE_DEVTAB_ENTRY command (per DeviceID), the INVALIDATE_IOMMU_PAGES command (per DomainID), and the INVALIDATE_INTERRUPT_TABLE command (per DeviceID).

Revision 2: The INVALIDATE_IOMMU_ALL command may simplify trusted boot, error recovery, and resumption from low-power states (see Section 3.3.8 [INVALIDATE_IOMMU_ALL]). At the completion of an INVALIDATE_IOMMU_ALL command, all IOMMU TLBs are empty, including cached portions of the device table, guest CR3 table, page directory entries, page table entries, and interrupt remapping entries (including the Guest APIC Table Root Pointer). Section 3.3.9 [IOMMU Ordering Rules] describes how outstanding operations must be handled. The operational status of the IOMMU is not affected so translations, command- and event-processing, address translation requests, and peripheral page service request processing continue normally. The contents of the MMIO registers are not affected except to advance the Command Buffer Head Pointer Register [MMIO Offset 2000h] beyond the INVALIDATE_IOMMU_ALL command. The IOMMU may start reloading internal caches with information at any time after the INVALIDATE_IOMMU_ALL command completes. The INVALIDATE_IOMMU_ALL command guarantees ordering as described in Section 3.3.9 [IOMMU Ordering Rules].

2.3.4 Address Translation Services for Guest Virtual Addresses

Revision 1: Address translation services can be used by a peripheral to translate a GPA to an SPA. To translate a GPA to an SPA, a PCIe-connected peripheral issues an ATS request lacking a PASID TLP prefix recognized by the IOMMU (see Section 3.2.7.7 [PCIe TLP PASID Prefix]). The IOMMU evaluates access privileges using cached information and walks the page tables when required. The resulting access privileges are returned in the ATS response.

Revision 2: Address translation services can be used by a peripheral to translate a GVA or GPA to an SPA. To translate a GVA to an SPA, a peripheral connected by PCIe issues an ATS request containing a valid PASID to present flags and a canonical virtual address (see Section 2.3.6 [Selecting Translation Tables in a Memory Transaction] and Section 3.2.7.7 [PCIe TLP PASID Prefix]). An integrated peripheral may use means other than the ATS protocol to present flags and the virtual address, such as wire signals. . The IOMMU evaluates access privileges using cached information for efficiency and walks the page tables when required. To match AMD64 semantics, the IOMMU must rewalk the guest page tables if previously cached information indicate insufficient privileges for the access (see Section 3.2.7.1 [Combining Guest and Host Address Translation] and Table 27). The resulting access privileges are returned in the ATS response. To carry the additional information for a guest address, the IOMMU uses a PCIe TLP prefix containing a valid PASID (see Section 3.2.7.7 [PCIe TLP PASID Prefix]).

The IOMMU must update the Accessed and Dirty bits in the GVA page table while servicing an ATS request as if the peripheral had actually accessed memory (see Section 3.2.7.4 [Updating Accessed and Dirty Bits in the Guest Address Tables]). For the purpose of evaluating GVA Accessed and Dirty bits, the IOMMU must use the access level indicated in the ATS packet. An ATS request for read-only access determines the Accessed bit setting and an ATS request for read-write access determines the Dirty bit setting (see Table 27). When processing a GPA, the IOMMU treats the page tables as read-only.

Software note: Software must issue an invalidation command when it changes A or D bits in a page table entry to 0 from 1. This requirement allows the IOMMU to cache the A & D bits in a TLB for higher performance.

Software issues an INVALIDATE_IOTLB_PAGES command to cause the IOMMU to generate an invalidation request to the peripheral (see Section 3.3.4 [INVALIDATE_IOTLB_PAGES]). An invalidation request sent downstream to the device lacks a valid PASID prefix when the contents are a GPA. An invalidation request sent downstream to the device has a valid PASID prefix when the contents are a GVA and the PASID is in the PASID TLP prefix.

The conditions under which a peripheral with an IOTLB must invalidate a cached translation entry that caused an insufficient-privilege check and obtain a fresh translation using ATS are in Section 3.1.4.8 [Discarding IOTLB Information to Rewalk Page Tables].

2.3.5 Peripheral Page Service Request Support Compatible with PCI-SIG PRI

Revision 1: The IOMMU does not support the PCI-SIG Page Request Interface (PRI) specification.

Revision 2: An IOMMU optionally supports the PCI-SIG PRI specification as a complement to PCI-SIG Address Translation Service (ATS) specification (see Section 3.1.1 [Normal Operation]). The IOMMU support for PRI is called the peripheral page request (PPR) service (see Section 3.5 [Peripheral Page Service Request (PPR) Logging]).

The operating system is usually required to pin memory pages used for I/O; the pinned pages are often allocated from a separate memory pool of limited capacity. ATS and PRI can be used together to enable the peripheral to use unpinned pages for I/O. When processing ATS requests, the IOMMU does not signal events when insufficient access privileges or not-present pages are detected; instead it returns the permissions calculated from the page tables. The peripheral examines the response to determine an appropriate action (e.g., use PRI to request system software to service a page table entry). Use of PPR/PRI allows a peripheral to request the operating system to change the access privileges of the page. Use of ATS with PPR can allow a system to operate efficiently in a reduced memory footprint.

2.3.6 Selecting Translation Tables in a Memory Transaction

Revision 1: A PCIe packet contains a GPA and the originating BDF is used to select GPA-to-SPA translation tables. A PCI-SIG TLP prefix is not interpreted by the IOMMU.

Revision 2: The PCI-SIG defined a method to add information to a transaction called the TLP prefix. The TLP prefix carries added information for a transaction that bears an x86-canonical GVA. When a PCIe transaction has a TLP prefix, the packet contains a GVA and the TLP prefix selects the guest tables for GVA-to-GPA translation; when a PCIe transaction has no TLP prefix, the packet contains a GPA. The originating BDF is used to select GPA-to-SPA translation tables. Details are in Section 3.2.7.7 [PCIe TLP PASID Prefix].

2.3.7 Implementation Considerations to Guarantee Memory Isolation

Revision 1: The IOMMU does not support guest address translation.

Revision 2: With the introduction of guest address translation, there are two implementation considerations discussed in this section: process address space IDs (PASID) and peripheral local memory protection.

The hypervisor typically uses the nested translation layer to separate and isolate guest virtual machines. A peripheral that is directly assigned to the guest VM is contained to the memory space of that virtual machine. The peripheral is unable to change or inspect memory or peripherals belonging to the hypervisor or another virtual machine. Within the guest virtual machine, there are a kernel address space and several process (user) address spaces. Using just nested translation information, a peripheral is usually granted kernel privileges so that it has relatively free access to the entire contents of the guest virtual machine memory. To enable user-level (process) I/O and advanced computation models, the guest translation layer is introduced for separation and isolation of guest processes and I/O. Using guest translation in the IOMMU, a peripheral can be directly assigned to a process in a guest virtual machine or a GPU can run computations in the same address space as a user process. This requires that the process address space be identified to the IOMMU so that the proper translation tables will be used. In other words, each memory transaction must be tagged with the PASID. This architectural specification does not define how the PASID is carried in the system in the general case because each implementation will have different requirements.

In a system that connects a peripheral using an I/O bus, the bus protocol can be extended to carry the originating PASID as well as DeviceID, address, and access type. In the PCI-SIG PCIe specification, the PASID TLP prefix of the bus packet carries the PASID information which is then used by the IOMMU to select the appropriate guest CR3 table (see Section 2.3.4 [Address Translation Services for Guest Virtual Addresses] and Section 3.2.6 [I/O Page Tables for Guest Translations]). This provides memory isolation among processes and virtual machines.

In a system that integrates peripherals and graphics units onto the processor die, it is not necessary to use an I/O bus to connect peripherals to memory. In this case, the PASID can simply be carried on wires or as a tag between the integrated peripheral and the integrated IOMMU. For software compatibility, it is recommended that integrated peripherals emulate ATS behavior and semantics. This provides memory isolation among processes and virtual machines.

Sophisticated, multi-context peripherals that require local memory for performance or security need to offer memory isolation and separation features similar to those provided by the IOMMU. The general architecture of such a device is illustrated in Figure 3. The system (CPU/Chipset, including an IOMMU) is drawn in the upper part of the figure and a multi-context peripheral is drawn in the lower part. Many parts of the peripheral are optional so multiplexors are shown where functions can be by-passed. For example, on the right side of the figure, an access to the system address space may either flow through an IOTLB working with an ATS/PRI unit, or it may flow directly to an IOMMU for service. The device processing complex may represent a GPGPU or other specialized computational engine. It may even represent an advanced peripheral controller such as a NIC with extensive off-load capabilities.

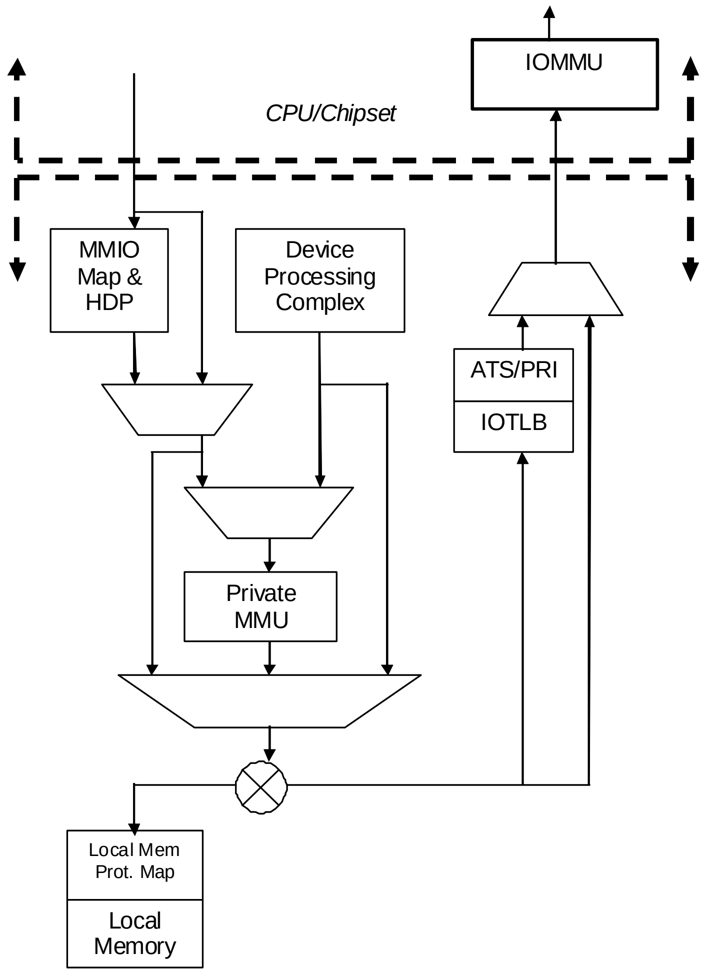


Figure 3: General Architecture of a Sophisticated Peripheral

There are four basic access flows to consider. A data access may originate with the CPU or with the device processing complex and it may terminate in a local memory access or in a system access. Regardless of the source or destination, system guarantees must be honored. The IOMMU and the local memory protection map are used to provide basic enforcement. A peripheral designer may add IOTLB functionality that uses ATS for translation efficiency; PPR/PRI support may be added for advanced function and efficiency. A peripheral may provide a private MMU function for custom address translation and access control.

2.3.8 Interrupt Virtualization (Guest Virtual Interrupt Controller)

Revision 1: Interrupt remapping is supported.

Revision 2: The IOMMU optionally supports interrupt virtualization. Device interrupts can be delivered directly to running guest virtual machines without hypervisor intervention when interrupts are virtualized (see MMIO Offset 0030h[GASup] and MMIO Offset 0018h[GAEn]). This can reduce the delivery latency and overhead of guest VM interrupts. This feature requires compatible APIC virtualization support in the processor. The processor and the IOMMU coordinate to maintain interrupt state in the Guest Virtual APIC Table when delivering interrupts. Interrupt remapping and interrupt virtualization may be enabled independently. See Section 3.2.8 [Guest Virtual APIC Table for Interrupt Virtualization].

3 Architecture

This chapter describes the IOMMU's architecture mainly from a system software point of view. The discussion starts with the normal steady state behavior of the IOMMU once it has been set up, focusing on how the IOMMU handles various device transactions. The following section describes the in-memory data structures used to control the IOMMU, together with the procedures software must follow to correctly update these (shared) data structures. Finally, the chapter concludes with a description of the PCI resources that must be initialized at system startup time to configure the IOMMU.

3.1 Behavior

When the IOMMU is disabled it simply passes all bus traffic through without alteration.

When the IOMMU is enabled, it intercepts requests arriving from downstream devices (which may be HyperTransport™ link or PCI based), performs permission checks and address translation on the requests, and sends translated versions upstream via the HyperTransport™ link to system memory space. Other requests are passed through unaltered (details in Section 3.1.1 [Normal Operation]). PCI devices serviced by a single IOMMU must be on the same PCI Segment Group (see PCI Firmware specification for further details of PCI Segment Groups).

The IOMMU reads three tables in system memory to perform its permission checks, interrupt remapping, and address translations. To avoid deadlock, memory accesses for device tables, page tables, and interrupt remapping tables by the IOMMU use an isochronous virtual channel and may only reference addresses in system memory. Other memory reads originated by the IOMMU to command buffers, event log entries, and optional request queue entries use the normal virtual channel. System performance could be substantially reduced if the IOMMU performed the full table lookup process for every device request it handled. Therefore, implementations of the IOMMU are expected to maintain internal caches for the contents of the IOMMU's in-memory tables, and correct operation of the IOMMU requires system software to send appropriate invalidation commands to the IOMMU when it updates table entries that may have been cached by the IOMMU.

The IOMMU writes to the event log in system memory using the normal virtual channel. The IOMMU can optionally write to the peripheral page service request queue in system memory; these writes use the normal virtual channel.

The IOMMU signals interrupts using standard PCI INTx, MSI, or MSI-X interrupts.

3.1.1 Normal Operation

The typical flow of requests through the IOMMU is as follows:

- Read, write and interrupt transactions generated by the IOMMU are not translated by the IOMMU.
- Transactions arriving from upstream must be passed downstream unaltered.
- Transactions arriving from downstream that are response, fence, or flush commands must be passed upstream unaltered.
- Transactions arriving from downstream that reference addresses within the IOMMU exclusion range must be passed upstream unaltered.
- Memory read and write transactions from downstream result in table lookups in the device table to obtain the DomainID of the requesting I/O device and to locate I/O page tables, and then further table lookups in I/O page tables to perform address translation and permission checking. After performing permission checks and address translation, the IOMMU forwards the resulting transactions upstream if the transaction is allowed from the I/O device.
- Address translation requests from downstream result in table lookups as for memory read and write

transactions. Translated address and access permission information is returned to the requesting peripheral. Software is required to invalidate address translation mappings cached by a peripheral.
- Peripheral page service requests from downstream result in an event log entry if not supported, or result in a peripheral page service request queue entry written to system memory.
- Interrupt addresses are never translated to system memory addresses, but other special address ranges may be optionally treated as memory addresses for translation.
- Interrupts from downstream result in table lookups in the device table and then in the interrupt remapping tables to remap the interrupt. After performing checks and interrupt remapping, the IOMMU forwards the resulting interrupts upstream if the interrupt is allowed from the I/O device.
- Port I/O space transactions from downstream devices result in a device table lookup to determine if the I/O device is allowed to access port I/O space.
- The IOMMU maintains an event log in system memory containing the details of transactions that do not complete normally.
- The IOMMU does not further translate pretranslated memory read and write requests from devices if the I/O device is marked as being able to generate pretranslated addresses.
- The IOMMU processes commands from the command queue.

In addition to passing on transactions from downstream devices, the IOMMU inserts transactions of its own to perform reads to and writes from system memory and to signal interrupts.

The IOMMU is allowed to cache page table and device table contents to speed translations. An invalidation protocol is defined so that software can keep the cache contents consistent with memory when it updates the tables.

3.1.2 IOMMU Logical Topology

Once configured, the IOMMU logically resides on the HyperTransport™ bus between the devices it translates for and the upstream interface. As a result of this logical topology the transactions seen by the IOMMU are defined in terms of HyperTransport™ transactions. Accesses to the HyperTransport™ address range FD_0000_0000h -FF_FFFF_FFFFh, inclusive, have special meanings. The meaning is encoded into various portions of the address as shown in Table 2 and Table 16; complete details are in the *HyperTransport™ I/O Link Specification*. Upstream transactions to these address ranges are controlled by device table control bits, page tables or the interrupt remapping tables. The special address controls do not apply to pretranslated addresses.

Revision 2: The special address controls in Table 2 are interpreted against upstream untranslated addresses (GPA) that lack a PASID TLP prefix. The special address controls do not apply to intermediate translation results during the translation of a guest virtual address to a system physical address.

Table 2: Special Address Controls (GPA)

Base Address	Top Address	Use	Access controlled by
FD_0000_0000h	FD_F7FF_FFFFh	Reserved interrupt address space	See Section 3.4.8 [INVALID_DEVICE_REQUEST Event]
FD_F800_0000h	FD_F8FF_FFFFh	Interrupt/EOI	IntCtl, Interrupt Remapping Tables
FD_F900_0000h	FD_F90F_FFFFh	Legacy PIC IACK	Page Tables
FD_F910_0000h	FD_F91F_FFFFh	System Management	SysMgt, Page Tables
FD_F920_0000h	FD_FAFF_FFFFh	Reserved	Page Tables
FD_FB00_0000h	FD_FBFF_FFFFh	Address Translation	HtAtsResv, Page Tables

Table 2: Special Address Controls (GPA)

Base Address	Top Address	Use	Access controlled by
FD_FC00_0000h	FD_FDFF_FFFFh	I/O Space	IoCtl, Page Tables
FD_FE00_0000h	FD_FFFF_FFFFh	Configuration	Page Tables
FE_0000_0000h	FE_1FFF_FFFFh	Extended Configuration/ Device Messages	Page Tables
FE_2000_0000h	FF_FFFF_FFFFh	Reserved	Page Tables

During configuration, an IOMMU may appear connected in different topologies that are implementation dependent.

3.1.3 IOMMU Event Reporting

The IOMMU must detect and may report several kinds of events that may arise due to unusual hardware or software behavior. When the IOMMU detects an event of any kind and event logging is enabled, it writes an appropriate event entry into the event log located in system memory. In addition, it may optionally signal an interrupt when the event log is written.

Events detected by the IOMMU include I/O page faults as well as hardware memory errors detected when walking the I/O page tables. A detected event may cause a page table or interrupt remapping table walk to terminate before reaching the final memory-translation or interrupt-remap entry. When a walk is terminated early, the event information reported is based on the results calculated in the completed portion of the walk, starting with the DTE.

Software note: the TLB caching behavior of the IOMMU is not defined for an entry causing an events; some implementations may insert an entry in the TLB cache before verifying that it causes no exceptions. System software should invalidate the address that caused the event.

3.1.3.1 IOMMU Event Responses

The IOMMU response to events depends on the type of event detected, the type of transaction that caused the event, and the state of the IOMMU at the time of the event.

If an IOMMU is not enabled or does not support address translation requests, the IOMMU responds to translation requests with a master abort.

If the IOMMU is enabled, it can have one of three event responses:

- For upstream transactions that are master aborted or target aborted, the PCI/Host bridge that is co-located with the IOMMU is the completer of the transaction. Transactions that are target aborted set the legacy Signaled Target Abort bit in a manner consistent with the bus specification over which the transaction was received (secondary port). These aborted transactions should not set any AER bits (if implemented and otherwise applicable).
- Exceptions detected in transactions that target the IOMMU function are not logged in the IOMMU event log. The exceptions are signaled following the rules of the bus specification applicable to the primary bus with which the IOMMU function is associated.
- Exceptions detected in the transactions originating from the IOMMU function signal the event following the rules of the bus specification applicable to the primary bus with which the IOMMU is associated. Additionally, exceptions in command buffer and table walk reads are logged in the IOMMU event log.

A transaction that attempts to use a device table entry beyond the end of the device table is treated as in

Table 40. The size of the device table is defined by the Device Table Base Address Register, MMIO Offset 0000h[Size].

3.1.3.2 I/O Page Faults

The IOMMU may detect page-fault conditions when processing peripheral requests and the response of the IOMMU depends on the type of the request and IOMMU control settings.

A peripheral's memory transaction may result in an I/O page fault. These page faults can arise for a variety of reasons, such as I/O page table entries lacking sufficient permission or memory pages marked not-present. In a traditional processor virtual memory implementation, page faults activate an exception handler that has the option to correct the underlying problem and retry the faulting instruction. The IOMMU has no such option: the underlying HyperTransport™ and PCI bus protocols do not provide a means for the IOMMU to signal a device that it should attempt to retry an access. Consequently, when the IOMMU detects an I/O page fault, it target aborts the faulting request. The IOMMU sets the legacy PCI Signaled Target Abort bit, if appropriate, and records I/O page fault information in its event log when event logging is enabled

For an address translation request, the IOMMU returns the translation result and does not signal a fault (see also Section 3.1.4.5 [Address Translation Requests in the Special Address Range]). The peripheral can examine the translation response to determine if a particular memory transaction would cause an exception. Peripherals may request page fault service as described in Section 3.5 [Peripheral Page Service Request (PPR) Logging].

3.1.3.3 Memory Access Errors

The IOMMU's own memory accesses to its in-memory tables may themselves result in several kinds of errors, including:
- Accesses to nonexistent or non-DRAM addresses because the IOMMU's isochronous virtual channel is restricted to DRAM addresses only.
- Uncorrectable ECC errors.
- Use of reserved values, including invalid or unsupported type codes in device table entries and reserved bits in page table entries.

The IOMMU records all detected memory access errors in its event log when event logging is enabled. Revision 2: Hardware errors may also be stored in the error registers (see Section 3.4.11.2 [I/O Hardware Event Reporting Registers]).

3.1.4 Special Conditions

This section defines the behavior of the IOMMU for particular operating conditions.

3.1.4.1 Zero-byte Read Operations

In some bus architectures, a zero-byte read operation is defined as a special operation with well-defined side effects. Because of these side effects, the IOMMU must permit a zero-byte read operation when a page is marked to allow either read or write access. Further, because the zero-byte read operation returns undefined data in some bus specifications, protecting the contents of a non-readable memory location requires that the IOMMU obscure the returned data for a zero-byte read operation.

Implementation note: methods to obscure the returned data in a zero-byte read operation include returning a constant, a random value, or a predictable value not based on the data contents such as the address.

3.1.4.2 Interrupt Address Range

Accesses to the interrupt address range (Table 2) are defined to go through the interrupt remapping portion of the IOMMU and not through address translation processing. Therefore, when a transaction is being processed as an interrupt remapping operation, the transaction attribute of pretranslated or untranslated is ignored.

Software note: The IOMMU should not be configured such that an address translation results in a special address such as the interrupt address range (see Table 2).

3.1.4.3 Multi-page Address Translation Requests Lacking a PDE

An address translation transaction to the IOMMU can request multiple pages. The page size (stride) is generally determined by the PDE used with level=0 or level=7. The page stride is always a power of two. For situations where there is no relevant PDE (within the IOMMU exclusion range or when the DTE Mode=0), the results returned by the IOMMU are implementation-specific.

3.1.4.4 Address Translation Requests in the IOMMU Exclusion Range

Revision 1: I/O devices may request address translations for addresses in the IOMMU exclusion range, defined by IOMMU Exclusion Base Register [MMIO Offset 0020h] and IOMMU Exclusion Range Limit Register [MMIO Offset 0028h], and may cache the results. When software changes the exclusion range, it must invalidate remote IOTLBs that may contain affected translation entries. Address translation requests to the exclusion range always return permissions that allow reading and writing

Revision 2: An address translation request for a GPA within the exclusion range returns an implementation-defined result.

3.1.4.5 Address Translation Requests in the Special Address Range

I/O device address translation requests for a GPA within special address ranges in Table 2 are controlled by the SysMgt and IoCtl settings in the device table entry (see Section 3.2.2.1 [Device Table Entry Format]) and can either return a translation or cause a target abort.

3.1.4.6 Page Translation Entries Spanning Memory and Special Address Ranges

An IOMMU address translation entry for a GPA may be constructed to cover both conventional memory addresses and special addresses (see Table 2). The DTE[IoCtl] and DTE[SysMgt] fields control IOMMU behavior. To translate a GPA address in a special address range, set the corresponding special address range control in the DTE to direct the IOMMU to translate the desired special address ranges as memory addresses.

3.1.4.7 Discarding IOMMU TLB Information to Rewalk Page Tables

Revision 1: The IOMMU can use address translation information from the TLB or memory to determine access privileges.

Revision 2: When the IOMMU detects an access violation based on cached information, it discards the information in the IOMMU TLB and reloads the translation information from memory. Interrupt remapping information is only loaded from memory on a TLB miss. See Section 2.3.4 [Address Translation Services for Guest Virtual Addresses] for details.

3.1.4.8 Discarding IOTLB Information to Rewalk Page Tables

Revision 1: The peripheral can use address translation information from the IOTLB or obtained via ATS to determine access privileges.

Revision 2: The peripheral can use address translation information from the IOTLB or obtained via ATS to determine access privileges for a nested (host) access. As an AMD extension, a peripheral with an IOTLB must invalidate a cached entry causing an insufficient-privilege failure when R=1 or W=1 in the IOTLB entry for a guest access. The peripheral must then request the guest translation information using ATS and retry the access. If the revised privileges are insufficient for the retry, the peripheral must take appropriate action to abandon the access or issue a PCIe PRI request for escalated privileges.

3.1.4.9 Updating the Accessed and Dirty Bits in Guest Page Tables

Revision 1: The IOMMU does not write page tables so it never updates the Accessed or Dirty bits in the PTE. Software is responsible to manage these bits. It is recommended that a read-only page made available to the peripheral be marked “Accessed”, and that any write-able page made available to the peripheral be marked “Dirty”.

Revision 2: The IOMMU must update the guest page table Accessed and Dirty bits in a manner compatible with the processor, so the IOMMU implements the equivalent of a locked-OR. Specifically, the IOMMU must set the Accessed bit in a locked operation, and it must set the Accessed and Dirty bits in a single locked operation. The IOMMU never clears the Accessed or Dirty bits; software is responsible to clear the bits. The IOMMU is allowed to cache these bits, so software must issue invalidation commands when it clears the bits in PTE. See Section 3.2.7.4 [Updating Accessed and Dirty Bits in the Guest Address Tables] and Section 3.2.7.5 [Clearing Accessed and Dirty Bits] for details.

3.1.4.10 Address Translation Response When DTE[Mode]=0

A peripheral can request address translations when DTE[Mode]=000b; the translated physical address is equal to the supplied virtual address (GPA).

3.1.4.11 Page Splintering

Revision 1: The IOMMU never splinters TLB entries.

Revision 2: When an address is mapped by guest and nested page table entries with different page sizes, the IOMMU TLB entry that is created matches the size of the smaller page (see also *AMD64 Technology, AMD64 Architecture Programmer’s Manual, Volume 2: System Programming,* Page Splintering).

3.1.4.12 Atomic Operations Require Read and Write Permissions

Atomic operations both read and write a page. The IOMMU must permit atomic operations from the peripheral only when the page is marked to allow both read and write operations.

3.1.4.13 INVALIDATE_IOTLB_PAGES and Peripheral Reset

If a peripheral is reset while an INVALIDATE_IOTLB_PAGES command is being executed by the IOMMU (Section 3.3.4 [INVALIDATE_IOTLB_PAGES]), the peripheral may stop processing invalidations and software must process any IOTLB_INV_TIMEOUT events that result (Section 3.4.7 [IOTLB_INV_TIMEOUT Event]).

3.2 Data Structures

Host software must maintain up to seven in-memory data structures for use by the IOMMU. These data structures are:

1. The *device table* is a table indexed by DeviceIDs. Each device table entry contains mode bits, a pointer to the I/O page tables, a pointer to an interrupt remapping control table, a set of control bits, and a 16-bit DomainID. The DomainID acts as an address space identifier, allowing multiple devices sharing the same I/O page tables to share the same translation cache resources on the IOMMU. The page tables must be the same for all devices that share a DomainID.
2. The *I/O page table(s)*: Each device table entry may specify a different I/O page table, or different device table entries may share the same I/O page tables. Each time the IOMMU processes a device access to memory, it looks up the device virtual address in its translation cache and/or the appropriate I/O page tables to determine whether the device has permission, as well as (if permitted) the system physical address to access.
3. The *command buffer:* The IOMMU accepts commands queued by the processor through a circular buffer located in system memory.
4. The *event log*: The IOMMU reports atypical events to the processor by means of another circular buffer, the event log, located in system memory.
5. The *interrupt remapping table(s):* Each device table entry may specify an interrupt remapping table. Each time the IOMMU processes a device interrupt request, it looks up the IRTE to remap the interrupt to the destination with a translated vector.
6. The *peripheral page service request log*: Revision 2. The IOMMU can accept requests from PRI-capable peripherals to service page change requests. These requests are reported in a circular buffer, the PPR log, located in system memory.
7. *The Guest Virtual APIC tables*. Revision 2. The IOMMU can update guest interrupt request status.

Figure 4 illustrates the relationships among the IOMMU data structures. Figure elements with dashed borders are new in IOMMU Revision 2.

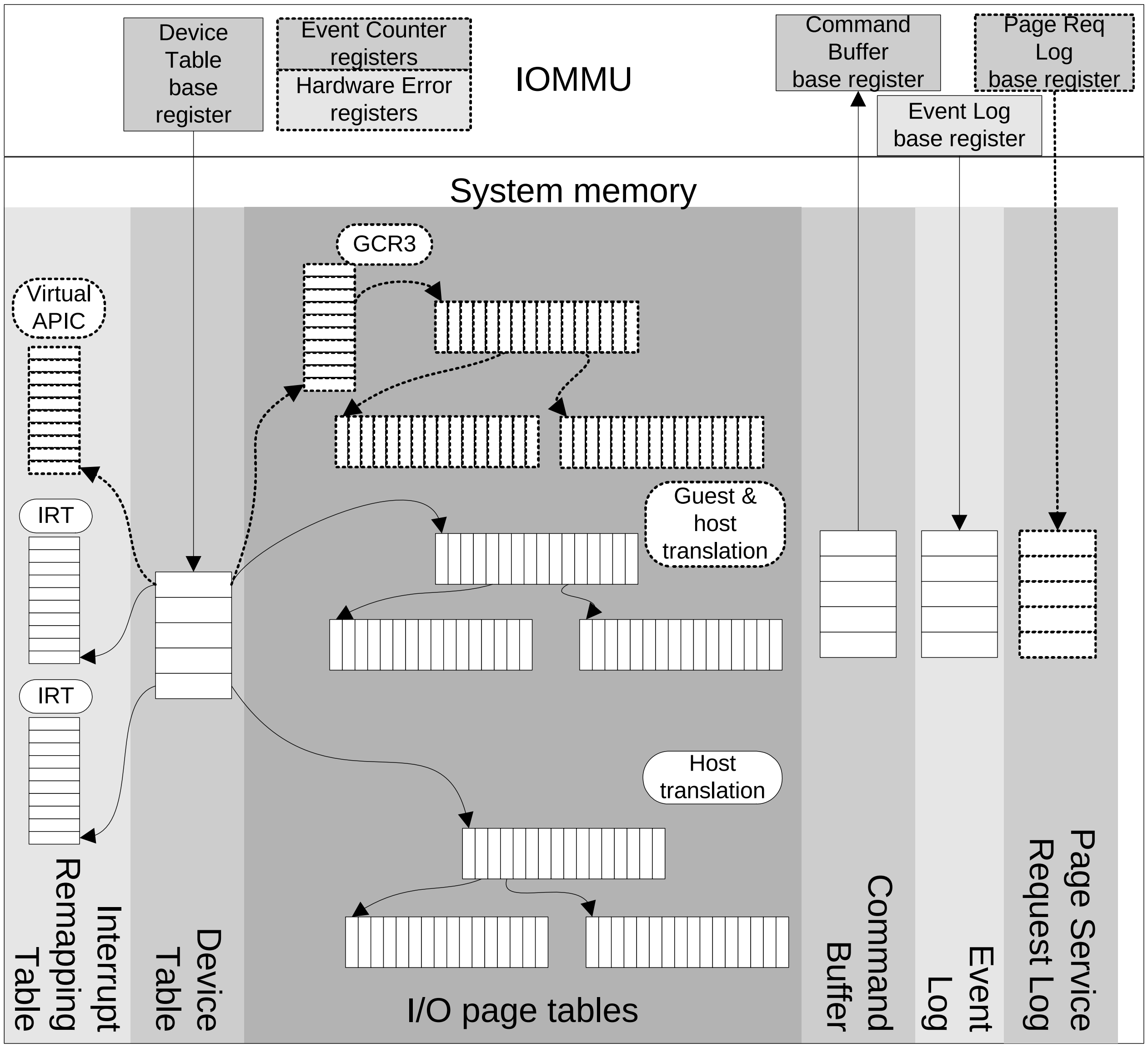


Figure 4: IOMMU Data Structures

Revision 1: The IOMMU supports one-level translation tables for address translation and for interrupt remapping. The event log is the only data structures in system memory that is written by the IOMMU. The maximum size of a virtual address (GPA) is defined in Capability Offset 10h[VAsize] and the maximum size of a physical address (SPA) is defined in Capability Offset 10h[PAsize].

Revision 2: The IOMMU optionally supports both one-level and two-level translation tables (Table 3) as well as guest APIC virtualization, hardware error registers, performance counter registers, and peripheral page-request services. An IOMMU can write to the event log, the peripheral page service request log, the guest virtual APIC tables, and the guest page tables. The maximum size of a guest virtual address (GVA) is defined in Capability Offset 10h[GVAsize].

3.2.1 Updating Shared Tables

The I/O page table structures have been designed so they can be shared among processors and IOMMUs. The table structures (interrupt remapping table, device table, and host I/O page tables) can be shared among

IOMMUs. The guest I/O page table structures are directly compatible with `AMD64 long` page table formats and the IOMMU (Revision 2) accesses and updates the tables so they can be shared with a processor. Shared tables have requirements for correct updates by system software.

When updating a table entry, system software is encouraged to use aligned 64-bit accesses although control bits are defined that allow system software updating a table to use byte accesses.

Each table can also have its contents cached by the IOMMU or peripheral IOTLBs. Therefore, after updating a table entry that can be cached, system software must send the IOMMU an appropriate invalidate command. Information in the peripheral IOTLBs must also be invalidated.

Revision 2: The IOMMU adds support for hardware updates of Accessed and Dirty bits in page tables. The IOMMU is allowed to cache these bits, so software must issue invalidation commands when it clears the bits in memory.

3.2.2 Device Table

I/O devices that originate transactions are identified by a 16-bit DeviceID that is used to index the device table. The content of the DeviceID is fabric-dependent; for example, Figure 5 shows how PCIe® and PCI-X® RequesterIDs are mapped into IOMMU DeviceIDs, and Figure 6 shows how HyperTransport™ UnitIDs are mapped into IOMMU DeviceIDs.

Software note: the mapping of DeviceID from one bus to another is platform specific; consult the platform documentation for details.

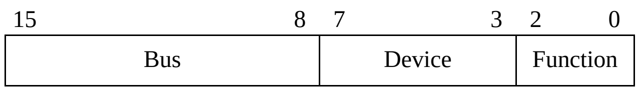
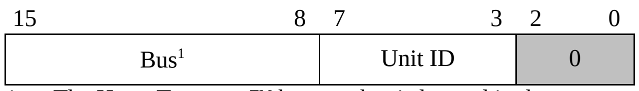


Figure 5: Example DeviceID Derived from Peripheral RequesterID

The number of bits allocated to the Bus, Device, and Function fields varies according to settings in the PCI configuration. The partitioning shown in Figure 5 is a typical example.



1. The HyperTransport™ bus number is located in the Slave/Primary Interface Block associated with the HyperTransport™ link that the traffic was received from.

Figure 6: DeviceID Derived from Peripheral UnitID

The device table is represented as an array of 256-bit entries located in contiguous system memory. Since there are 64K possible DeviceIDs, the device table may be up to 2M bytes in length. The Device Table Base Address Register [MMIO Offset 0000h], controls the system physical address and size of the device table. The device table must be aligned at a 4K byte boundary in system memory, and must be a multiple of 4K bytes in length. The IOMMU must read the entire device table entry in two 128-bit transactions (as defined by the scope of the validity indicators) or a single 256-bit transaction.

When the IOMMU is enabled, any I/O device whose DeviceID is beyond the end of the device table is denied I/O permission (the IOMMU target aborts the access) and all attempted accesses by such I/O devices are logged when event logging is enabled. Revision 2: PRI requests are not validated using the device table and so the IOMMU may create a PPR log entry for an I/O device whose DeviceID is beyond the end of the device table when page requests are enabled (see MMIO Offset 0018h[PPREn]), so software must validate the

DeviceID as part of PPR processing.

If an I/O device uses PCI phantom functions, software must replicate device table entries such that index calculations retrieve the correct entries for any phantom function used by the I/O device.

3.2.2.1 Device Table Entry Format

Revision 1: Device table entries have an address translation portion and an interrupt remapping portion. Control bits independently govern the use of each portion for a given peripheral DeviceID.

Revision 2: Device table entries have an address translation portion, an interrupt remapping portion, and an interrupt virtualization portion; control bits govern the use of each portion for a given DeviceID. The address translation portion has guest and nested translation portions that can be manipulated separately; guest translation cannot operate without nested translation. Consult Table 3 and Table 4 to determine if Revision 1 or Revision 2 functionality is available and enabled. The address translation features in Table 3 may be implemented separately from the interrupt remapping and virtualization features in Table 4; when implemented, address and interrupt features may be enabled and operated independently.

Table 3: Revision Feature Enablement for Address Translation

GTSup (MMIO Offset 0030h)	**GTEn** (MMIO Offset 0018h)	**Device Table Entry Address Translation Settings**			**Revision 1 Address Translation Features Available for Use**	**Revision 2 Address Translation Features Available for Use**
		V	**TV**	**GV**		
0	X	X	X	X	Yes	No
1	0	X	X	X	Yes	No
1	1	0	X	X	Available but not active for the DeviceID	Available but not active for the DeviceID.
1	1	1	0	X	Yes	Available but not active for the DeviceID.
1	1	1	1	0	Yes	Available but guest address translation is not active for the DeviceID.
1	1	1	1	1	Yes	Available and guest address translation is active for the DeviceID.

Table 4: Revision Feature Enablement for Interrupt Remapping and Virtualization

GASup (MMIO Offset 0030h)	**GAEn** (MMIO Offset 0018h)	**Device Table Entry Interrupt Settings**		**Revision 1 Interrupt Features Available for Use**	**Revision 2 Interrupt Features Available for Use**
		IV	**ZV**		
0	X	X	X	Yes	No
1	0	X	X	Yes	No
1	1	0	0	Yes	Available but not active for the DeviceID.

Table 4: Revision Feature Enablement for Interrupt Remapping and Virtualization

GASup (MMIO Offset 0030h)	GAEn (MMIO Offset 0018h)	Device Table Entry Interrupt Settings		Revision 1 Interrupt Features Available for Use	Revision 2 Interrupt Features Available for Use
		IV	ZV		
1	1	1	0	Yes	Interrupt remapping is active for the DeviceID but interrupt virtualization is not.
1	1	0	1	Yes	Interrupt virtualization is active for the DeviceID but interrupt remapping is not.
1	1	1	1	Yes	Interrupt remapping and virtualization are active for the DeviceID.

The device table entry (DTE) format is shown in Figure 7.

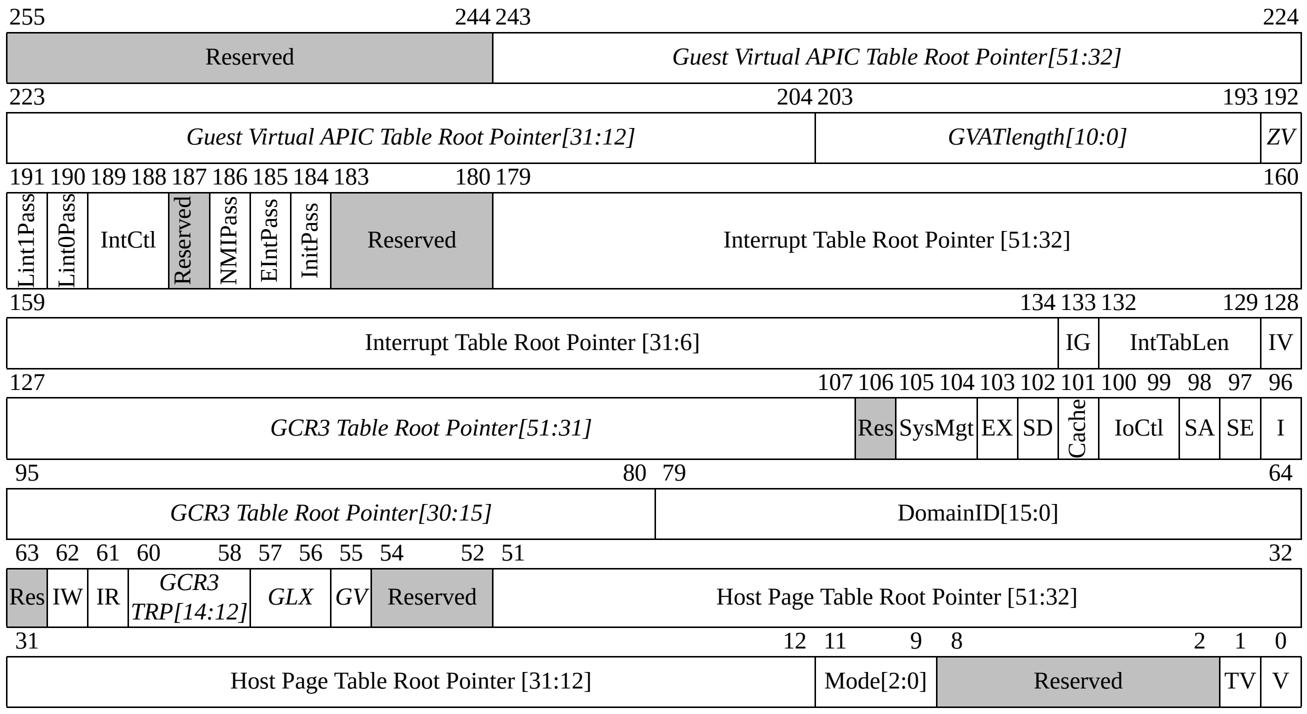


Figure 7: Device Table Entry Fields

Fields in the device table entry are defined in Table 5. Where indicated in Table 5, events are reported as described in Section 3.4.1 [ILLEGAL_DEV_TABLE_ENTRY Event]. Shaded areas mark fields that are reserved; text in *Italics* mark fields that are Reserved in Revision 1 and have a function defined in Revision 2.

Table 5: Device Table Entry Field Definitions

Bits	Description.
255:244	**Reserved**. Revision 1: Reserved. Non-zero bits in this field are reported as an event when IV=1. Revision 2: Reserved. Non-zero bits in this field are reported as an event when IV=1 and ZV=1.

Table 5: Device Table Entry Field Definitions

243:204	**Guest Virtual APIC Table Root Pointer**. Revision 1: Reserved. Non-zero bits in this field are reported as an event when IV=1. Revision 2: The guest virtual APIC table root pointer . The guest virtual APIC table root pointer is ignored by hardware when ZV=0. When MMIO Offset 0030h[GASup]=0, the Guest Virtual APIC Table Root Pointer field must be 0.
203:193	**GVATlength: Guest virtual APIC table length**. Revision 1: Reserved. Non-zero bits in this field are reported as an event when IV=1. Revision 2: GVATlength[10:0]. GVATlength is ignored by hardware when ZV=0. When MMIO Offset 0030h[GASup]=0, the GVATlength field must be 0.
192	**ZV: Guest virtual APIC table valid**. Revision 1: Reserved. A non-zero value in this field is reported as an event when IV=1. Revision 2: This field enables guest virtual interrupt controller processing and indicates that the guest virtual APIC table root pointer and GVATlength fields are meaningful. 0b=IOMMU does not perform guest virtual interrupt controller processing, and the guest virtual APIC table root pointer and GVATlength fields are ignored. 1b=IOMMU performs guest virtual APIC translation using the guest virtual APIC table root pointer and GVATlength fields. ZV and the fields it governs are independent of IV. When MMIO Offset 0030h[GASup]=0, the ZV field must be 0.
191	**Lint1Pass: LINT1 (legacy PIC NMI) pass-through**. This bit enables device initiated LINT1 interrupts to be forwarded by the IOMMU. 1=Device initiated LINT1 interrupts are forwarded unmapped. 0=Device initiated LINT1 interrupts are target aborted by the IOMMU. See Table 8**.**
190	**Lint0Pass: LINT0 (legacy PIC ExtInt) pass-through**. This bit enables device initiated LINT0 interrupts to be forwarded by the IOMMU. 1=Device initiated LINT0 interrupts are forwarded unmapped. 0=Device initiated LINT0 interrupts are target aborted by the IOMMU. See Table 8**.**
189:188	**IntCtl: interrupt control**. This field controls how fixed and arbitrated interrupt messages are handled. Fixed and arbitrated interrupt messages use a HyperTransport™ special addresses as shown in Table 2 and Table 16. 00b=Fixed and arbitrated interrupts target aborted 01b=Fixed and arbitrated interrupts are forwarded unmapped 10b=Fixed and arbitrated interrupts remapped 11b=Reserved See Table 7. If IntCtl=10b, a valid interrupt table root pointer must be present; if !(IntCtl=10b) the interrupt table root pointer is ignored. **Note:** IntCtl=11b is reported as an event when IV=1.
187	**Reserved.** **Note:** Non-zero bits in this field are reported as an event when IV=1**.**
186	**NMIPass: NMI pass-through.** 1=pass through NMI interrupt messages unmapped. 0=NMI interrupt message is target aborted by the IOMMU. See Table 8.
185	**EIntPass: ExtInt pass-through.** 1=pass through ExtInt interrupt messages unmapped. 0=External interrupt message is target aborted by the IOMMU. See Table 8.
184	**InitPass: INIT pass-through.** 1=pass through INIT interrupt messages unmapped. 0=INIT interrupt message handling target aborted by the IOMMU. See Table 8.
183-180	Reserved. **Note:** Non-zero bits in this field are reported as an event when IV=1.
179:134	**Interrupt table root pointer**. The interrupt table root pointer is only used when interrupt translation is enabled (IntCtl=10b). It contains the SPA of the base address of the interrupt remapping table for the I/O device. The interrupt remapping table must be aligned to start on a 128-byte boundary.

Table 5: Device Table Entry Field Definitions

133	**IG: ignore unmapped interrupts.** 1=Supress event logging for interrupt messages causing IO_PAGE_FAULT events. 0=creation of event log entries for IO_PAGE_FAULT events is controlled by SupIOPF in the interrupt remapping table entry (see Section 3.2.5 [Interrupt Remapping Tables]).
132:129	**IntTabLen: interrupt table length**. This field specifies the length of the interrupt remapping table. 0000b = 1 entry 0001b = 2 entries 0010b = 4 entries 0011b = 8 entries ... 1010b = 1024 entries 1011b = 2048 entries 11xxb = reserved **Note:** IntTabLen=11xxb is reported as an event when IV=1.
128	**IV: interrupt map valid.** Revision 1: 1=Interrupt map information in bits [255:129] is valid. 0=Interrupt remapping information in bits [255:129] is not valid and interrupts are passed through the IOMMU unmapped. Revision 2: 1=Interrupt map information in bits [191:129] is valid. 0=Interrupt remapping information in bits [191:129] is not valid and interrupts are passed through the IOMMU unmapped. See Table 7 and Table 8. IV and the fields it governs are independent of ZV.
127:107	**GCR3 Table Root Pointer[51:31]**. Revision 1: Reserved. Non-zero bits in this field are reported as an event when V=1. Revision 2: The guest CR3 table root pointer contains the SPA of the guest CR3 table for the I/O device. The guest CR3 table root pointer may be used by hardware when V=1 and TV=1 and GV=1; it is ignored otherwise. See Section 3.2.6 [I/O Page Tables for Guest Translations].
106	**Reserved.**
105:104	**SysMgt: system management message enable**. Specifies whether device-initiated untranslated memory requests that target the system management address space in Table 2 are blocked, forwarded, or translated by the IOMMU. 00b=Device initiated DMA transactions in the system management address range are return target abort status by the IOMMU. Translation requests return target abort status. 01b=Device initiated system management messages, including INTx messages, are forwarded untranslated by the IOMMU. Upstream reads or non-posted writes return target abort status. Translation requests return target abort status. 10b=Device initiated INTx messages are forwarded by the IOMMU untranslated; device initiated system management messages other than INTx messages return target abort status. Upstream reads and non-posted writes return target abort status. Translation requests return target abort status. 11b=Device initiated DMA transactions in the system management address range are translated by the IOMMU.
103	**EX: allow exclusion**. 1=Accesses from this device that address the IOMMU exclusion range are excluded from translation and access checks. 0=Accesses from this device to the IOMMU exclusion range are translated and checked for access rights. See IOMMU Exclusion Base Register [MMIO Offset 0020h] and IOMMU Exclusion Range Limit Register [MMIO Offset 0028h].
102	**SD: snoop disable**. 1=IOMMU page table walk transactions for this device are not snooped. HyperTransport™ transactions by an IOMMU must not set the coherent bit in page table walk requests for this device. 0=IOMMU page table walk transactions for this device are snooped. HyperTransport™ transactions by an IOMMU must set the coherent bit in page table walk requests for this device. See also the Coherent bit in the IOMMU Control Register [MMIO Offset 0018h].

Table 5: Device Table Entry Field Definitions

101	**Cache: IOTLB cache hint**. Revision 1: 1=the IOMMU avoids caching GPA-to-SPA translation information obtained for ATS requests. 0=the IOMMU caches GPA-to-SPA translation information obtained for ATS requests when the peripheral is directed to issue untranslated addresses (see Table 10). Revision 2: For ATS requests containing a GVA, the IOMMU caches translation information and sets U=0 in an ATS response. **Software note:** It is recommended that software set Cache=0 for peripherals with an IOTLB. 1=Caching of translations for explicit translation requests is not recommended. See Section 3.2.7.3 [Recalculating Present, Read, and Write Access Permissions]**.**
100:99	**IoCtl: port I/O control**. Specifies whether device-initiated port I/O space transactions are blocked, forwarded, or translated. 00b=Device-initiated port I/O is not allowed. The IOMMU target aborts the transaction if a port I/O space transaction is received. Translation requests are target aborted. 01b=Device-initiated port I/O space transactions are allowed. The IOMMU must pass port I/O accesses untranslated. Translation requests are target aborted. 10b=Transactions in the port I/O space address range are translated by the IOMMU page tables as memory transactions. 11b=Reserved. **Note:** IoCtl=00b and IoCtl=01b control the forwarding upstream of port I/O, if it is implemented. **Note:** IoCtl=11b is reported as an event when V=1.
98	**SA: suppress all I/O page fault events**. 1=Suppress event logging for all IO_PAGE_FAULT events caused by memory accesses from this I/O device. **Note:** SA does not affect events logged due to interrupts or IOMMU command processing. **Note:** When V=0 the value of SA is ignored by the IOMMU.
97	**SE: suppress I/O page fault events**. Suppress event logging for IO_PAGE_FAULT events if an IO_PAGE_FAULT event has already been logged in the event log for this I/O device. 1=The IOMMU must only update the event log with an IO_PAGE_FAULT event for the first page fault seen for the device as long as the DeviceID remains in the IOMMU cache. The IOMMU clears all state associated with this bit when an INVALIDATE_DEVTAB_ENTRY command is received for the device or when the DeviceID is replaced in the cache by a different DeviceID. **Software note:** The SE bit controls a mechanism that reduces the number of event log entries on a per-device basis. The degree of filtering depends on the behavior of the device table cache. As such, software should not assume that only a single entry per device is made in the event log. **Note:** SE does not affect events logged due to interrupts or IOMMU command processing. **Note:** When V=0 the value of SE is ignored by the IOMMU
96	**I: IOTLB enable**. Controls IOMMU response to address translation requests from peripherals. 0=IOMMU returns target abort status when it receives an ATS requests from the peripheral. 1=IOMMU responds to ATS requests from the peripheral. This bit does not affect interrupts from the peripheral. If I=1 when Capability Offset 00h[IotlbSup]=0, the results are undefined.
95:80	**GCR3 Table Root Pointer[30:15]**. Revision 1: Reserved. Non-zero bits in this field are reported as an event when V=1. Revision 2: The guest CR3 table root pointer contains the SPA of the top (or only) level of the guest CR3 table for the peripheral. The guest CR3 table root pointer may be used by hardware when V=1 and TV=1 and GV=1. See Section 3.2.6 [I/O Page Tables for Guest Translations]. Must be zero when MMIO Offset 0030h[GTSup]=0.

Table 5: Device Table Entry Field Definitions

79:64	**DomainID**. The DomainID is a 16-bit integer chosen by software that the IOMMU must use to tag its internal translation caches and to mark event log entries. I/O devices with different page tables must be given different DomainIDs. I/O devices that share the same page tables may be given the same DomainID. I/O devices that share the same DTE[DomainID] must have the same settings in the DTE[Mode] and page table root pointer fields, however they may have different values in the DTE[I] and DTE[SysMgt] fields. If devices with the same DTE[DomainID] are given different non-zero values in the DTE[Mode] field or different page table root pointer values, the behavior of the IOMMU is undefined. The value of the DTE[DomainID] recorded in an event log entry is undefined when V=0 and IV=1.
63	Reserved. **Note:** A non-zero value in this field is reported as an event when V=1.
62	**IW: I/O write permission**. Used in the calculation of effective write access with the permission bits in the page tables; if there are no page tables (DTE[Mode]=000b), then this bit defines the I/O write permission. 1=I/O device is allowed to perform DMA write transactions and 0-byte read transactions (see Section 3.1.4 [Special Conditions]); the I/O device is allowed to perform DMA atomic operations when IR is also programmed to allow read access. 0=Device initiated DMA write and atomic transactions are target aborted.
61	**IR: I/O read permission**. Used in the calculation of effective read access with the permission bits in the page tables; if there are no page tables (DTE[Mode]=000b), then this bit defines the I/O read permission. 1=I/O device is allowed to perform DMA read transactions; the I/O device is allowed to perform atomic transactions when IW is also programmed to allow write operations. 0=Device initiated DMA read transactions are target aborted. When both IW and IW are programmed to 0b, device-initiated 0-byte read transactions are target aborted.
60:58	**GCR3 TRP: guest CR3 table root pointer[14:12]**. Revision 1: Reserved. Non-zero bits in this field are reported as an event when V=1. Revision 2: The guest CR3 table root pointer contains the SPA of the top (or only) level of the guest CR3 table for the I/O device. The guest CR3 table root pointer may be used by hardware when V=1 and TV=1 and GV=1. See Section 3.2.6 [I/O Page Tables for Guest Translations]. Must be zero when MMIO Offset 0030h[GTSup]=0.
57:56	**GLX: guest levels translated**. Revision 1: Reserved. A non-zero value in this field is reported as an event when V=1. Revision 2: GLX[1:0] specifies the type of guest CR3 lookup performed by the IOMMU for the I/O device when the device presents an address with a valid PASID. 00b=GCR3 table is single-level. 01b=GCR3 table is two-level. 10b=GCR3 table is three-level. 11b=reserved. The GLX value is ignored when GV=0. See Table 9 and Section 3.2.6.1 [Guest CR3 Table]. Must be zero when MMIO Offset 0030h[GTSup]=0. **Implementation note**: Revision 2: The number of levels in a guest CR3 table supported by hardware is indicated by MMIO Offset 0030h[GLXSup]. **Software note:** Revision 2: For a peripheral using PASID values up to 9 bits, software may program GLX=00b and build one-level GCR3 tables. For a peripheral using PASID values that use more than 9 bits but fewer than 19 bits, software must program GLX=01b and build two-level GCR3 base address tables. For a peripheral using PASID values that use 19 or 20 bits, software must program GLX=10b and build three-level GCR3 base address tables.

Table 5: Device Table Entry Field Definitions

55	**GV: guest translation valid**. Revision 1: Reserved. A non-zero value in this field is reported as an event when V=1. Revision 2: This field controls guest-level translation. 0=IOMMU performs GPA-to-SPA translation only; GLX and the GCR3 table root pointer fields are ignored. 1=IOMMU performs GPA-to-SPA translation or GVA-to-SPA when a valid PASID is provided; GLX and the GCR3 table root pointer values are used for GVA-to-GPA translations. Software programs this bit when guest page translation is available (see Table 3). This bit is meaningful when V=1 and TV=1 and MMIO Offset 0030h[GTSup]=1. Must be zero when MMIO Offset 0030h[GTSup]=0.
54:52	Reserved. **Note:** a non-zero value in this field is reported as an event when V=1.
51:12	**Page Table Root Pointer**. The page table root pointer contains the system physical address of the root page table for the I/O device for GPA-to-SPA translations. The pointer is only used in modes where GPA-to-SPA translation is enabled.
11:9	**Mode: paging mode**. Specify how the IOMMU performs GPA-to-SPA translation on behalf of the device. If GPA-to-SPA translation is enabled, this field specifies the depth of the host page tables associated with the device (see page table root pointer). 000b Translation disabled (Access controlled by IR and IW bits) 001b 1 Level Page Table (provides a 21-bit device virtual address space) 010b 2 Level Page Table (provides a 30-bit device virtual address space) 011b 3 Level Page Table (provides a 39-bit device virtual address space) 100b 4 Level Page Table (provides a 48-bit device virtual address space) 101b 5 Level Page Table (provides a 57-bit device virtual address space) 110b 6 Level Page Table (provides a 64-bit device virtual address space) 111b Reserved **Note:** the page table root pointer for GPA-to-SPA translation is ignored when Mode=000b and when Mode=111b. **Note:** Mode=111b is reported as an event when V=1 and TV=1. See also MMIO Offset 0030h[HATS].
8:2	Reserved. Non-zero bits in this field are reported as an event when V=1.
1	**TV: translation information valid.** 1=Page translation information is valid, specifically IW, IR, the page table root pointer, Mode, and GV. 0=Page translation information is not valid. TV is not meaningful when V=0. See Table 6.
0	**V: valid**. 1=Device table entry bits [127:1] are valid. 0=Device table entry bits [127:1] are invalid and transactions not intercepted by the interrupt remapping portion of the IOMMU are passed through. **Note**: The interrupt remapping portion of the device table entry is controlled by the IV bit. **Software note:** DomainID must be valid when V=1. See Table 6.

The interactions of the V, TV, IV, and IntCtl control bits are stated in Table 6 and Table 7. The interactions of IV and the pass control bits are defined in Table 8. The event log entries for operations causing a target abort are defined in Section 3.4 [Event Logging].

Table 6: V, TV, and GV Fields in Device Table Entry

V	TV	GV	Description
0	X	X	All addresses are forwarded without translation; individual control fields are ignored.
1	0	0	The SysMgt, EX, SD, Cache, IoCtl, SA, SE, and I fields are valid. The value of DomainID is used for event log entries. If the request requires a table walk, the table walk is terminated. The Mode and Host Page Table Root Pointer fields are ignored. Revision 2: The GV, GLX, GCR3 Table Root Pointer fields are ignored.
1	0	1	Revision 1: causes ILLEGAL_DEV_TABLE_ENTRY event (Section 3.4.1 [ILLEGAL_DEV_TABLE_ENTRY Event]). Revision 2: The SysMgt, EX, SD, Cache, IoCtl, SA, SE, and I fields are valid. The value of DomainID is used for event log entries. If the request requires a table walk, the table walk is terminated. The Mode, Host Page Table Root Pointer, GV, GLX, GCR3 Table Root Pointer fields are ignored.
1	1	0	All fields in bits [127:2] are valid and GPA-to-SPA translation is active (see Section 3.2.6 [I/O Page Tables for Guest Translations]).
1	1	1	Revision 1: causes ILLEGAL_DEV_TABLE_ENTRY event. Revision 2: All fields in bits [127:2] are valid and GVA-to-SPA translation is active (see Section 3.2.6 [I/O Page Tables for Guest Translations]).

Table 7: IV and IntCtl Fields in Device Table Entry for Fixed and Arbitrated Interrupts

IV	IntCtl	Description
0	X	All interrupts are forwarded without remapping.
1	00b	All fixed and arbitrated interrupts are target aborted.
1	01b	All fixed and arbitrated interrupts are forwarded without remapping.
1	10b	All fixed and arbitrated interrupts are remapped.
1	11b	Behavior undefined.

Table 8: IV and Pass Fields in Device Table Entry for Selected Interrupts

IV	Pass Field Name	Pass Field=0b	Pass Field=1b
0	X	LINT0, LINT1, SMI, NMI, INIT, and ExtInt interrupts are passed through unmapped.	
1	X	SMI interrupts are passed through unmapped. There is no pass field to control SMI requests.	
1	Lint0Pass	LINT0 interrupts are target aborted.	LINT0 interrupts are passed through unmapped.
1	Lint1Pass	LINT1 interrupts are target aborted.	LINT1 interrupts are passed through unmapped.
1	NMIPass	NMI interrupts are target aborted.	NMI interrupts are passed through unmapped.

Table 8: IV and Pass Fields in Device Table Entry for Selected Interrupts

IV	Pass Field Name	Pass Field=0b	Pass Field=1b
1	INITPass	INIT interrupts are target aborted.	INIT interrupts are passed through unmapped.
1	EIntPass	ExtInt interrupts are target aborted.	ExtInt interrupts are passed through unmapped.

Table 9: GLX and maximum translatable PASID size

MMIO Offset 0030h **GTSup**	**DTE[GV] (see Table 6)**	MMIO Offset 0030h **GLXSup**	**DTE[GLX]**	**Maximum translatable PASID size (bits)**	**Levels in GCR3 table**
0	X	X	XXb	none	-
1	0	X	XXb	none	-
1	1	00b	00b	9	1
1	1	00b	01b, 10b	not defined	-
1	1	01b	00b	9	1
1	1	01b	01b	18	2
1	1	01b	10b	not defined	-
1	1	10b	00b	9	1
1	1	10b	01b	18	2
1	1	10b	10b	20	3
1	1	X	11b	not defined	-

Although Table 9 defines the maximum PASID size that can be translated using a GCR3 table, MMIO Offset 0030h[PASmax] defines the maximum PASID size that can be handled internally by the IOMMU. Figure 16 and Figure 18 illustrate the structure of 1- and 2-level GCR3 tables, respectively. Guest address translation control fields are in Table 18.

Table 10: Cache bit and U bit for ATS requests

U (I/O PTE, Table 14)	Cache (DTE, Table 5)	IOMMU behavior (advised)	Comments
0	0,1	IOMMU not advised to cache results from ATS request	The peripheral issues pretranslated addresses (SPA) for read, write, and atomic operations; the IOMMU is not likely to need translation information.
1	0	IOMMU is advised to cache results from ATS requests	The peripheral issues untranslated addresses (GVA or GPA) for read, write, and atomic operations; the IOMMU needs translation information to process the memory transactions.

U (I/O PTE, Table 14)	Cache (DTE, Table 5)	IOMMU behavior (advised)	Comments
1	1	IOMMU not advised to cache results from ATS requests	The peripheral issues untranslated addresses (GVA or GPA) for read, write, and atomic operations. Note that the IOMMU is likely to walk page tables to obtain the needed translation information.
Implementation note: An ATS response for a GVA always returns U=0 (see Table 14) and software must account for this when deciding if an invalidation operation is required. **Note**: For more information on the U bit, see the PCI *Address Translation Services 1.1 Specification*.			

Table 10 defines the caching behavior of the IOMMU based on the per-device Cache bit in the DTE and the per-page U bit in the PTE. In the PCI *Address Translation Services 1.1 Specification*, the U bit defines whether the peripheral can issue translated or untranslated addresses to access a page for read, write, or atomic operations. When PTE[U]=1, software can use the Cache bit in the DTE to provide a caching hint to the IOMMU.

3.2.2.2 Making Device Table Entry Changes

This section contains information for software that changes the IOMMU tables. Software should issue invalidate commands after certain types of changes to tables and note that I/O device accesses are neither queued nor throttled by the IOMMU. Software may change the interrupt remapping information independently of the address translation information in a device table entry. These operational sequences are general and system conditions may allow optimizations.

Software may change the interrupt remapping information in a device table entry with a single 64-bit write. The change must be followed by an INVALIDATE_DEVTAB_ENTRY command when either the value of IV=1b or the value of V=1b before the change. If a 64-bit operation cannot be used, software may change the interrupt remapping information in the device table entry in the following manner, according to the value of IV before the change in the relevant device table entry.

- If IV=0b before the change, changes can be made in any order as long as the last change is to set to IV=1b; an INVALIDATE_DEVTAB_ENTRY command is required when the V=1b before the change.
- If IV=1b before the change, the following steps may be followed to change interrupt remapping information for fixed and arbitrated interrupts:
  - Set IntCtl=00b in the device table entry to block interrupts; any device-initiated interrupts for the domain are target aborted and, when enabled, logged to the event log.
  - Update the interrupt table root pointer, IG, and IntTabLen.
  - Invalidate the interrupt table if the interrupt table root pointer or IntTabLen was changed (see Section 3.3.5 [INVALIDATE_INTERRUPT_TABLE]).
  - Change IntCtl to cease blocking interrupts from the device (set IntCtl=01b or 10b).
  - Invalidate the device table entry (see Section 3.3.2 [INVALIDATE_DEVTAB_ENTRY]).
- If IV=1b before the change, the following steps change interrupt control information in the device table entry for NMI, Lint0, Lint1, Init, and ExtInt interrupts:
  - Update Lint1Pass, Lint0Pass, IntCtl, NMIPass, EIntPass, and InitPass. The setting of IntCtl can be changed at the same time.
  - Invalidate the device table entry for the device (see Section 3.3.2 [INVALIDATE_DEVTAB_ENTRY]).

Software may change the address translation information in a device table entry with a single 128-bit write operation followed by an INVALIDATE_DEVTAB_ENTRY command when either IV=1b or V=1b before the change. If a 128-bit operation cannot be used, software may change the address translation information in the following ways, according to the values of V and TV before the change.

- If V=0b before the change, address translation changes can be made in any order as long as the last change is to set V=1b. An INVALIDATE_DEVTAB_ENTRY command is required if IV=1b before the change.
- If V=1b before the change, software can use the following steps to set the IOMMU to pass addresses untranslated with access controlled by IR and IW, depending on the value of TV.
    - If TV=0b before the change, set values for IW, IR, Mode=000b, and TV=1b (maintaining V=1b), then issue an INVALIDATE_DEVTAB_ENTRY command. If not done as a 64-bit write, the values of TV and V must be in the final change. Note that the DomainID and other values in bits [127:96] are already valid because V=1b.
    - If TV=1b before the change, software must change IW and IR concurrently with or before changing Mode, and the values of TV and V must be in the final change. Software then issues an INVALIDATE_DEVTAB_ENTRY command.

Revision 2: The IOMMU adds support for hardware updates of Accessed and Dirty bits in page tables. The IOMMU is allowed to cache these bits, so software must issue invalidation commands when it clears the A or D bit in memory.

3.2.2.3 Starting the IOMMU

To start the IOMMU and activate table walking, etc., use the following procedure after a system reset.

- If not previously set, initialize the following registers:
    - the Device Table Base Address Register [MMIO Offset 0000h],
    - the Command Buffer Base Address Register [MMIO Offset 0008h],
    - the Command Buffer Head Pointer Register [MMIO Offset 2000h],
    - the Command Buffer Tail Pointer Register [MMIO Offset 2008h],
    - the IOMMU Exclusion Base Register [MMIO Offset 0020h] and the IOMMU Exclusion Range Limit Register [MMIO Offset 0028h], if used,
    - the Event Log Base Address Register [MMIO Offset 0010h], the Event Log Head Pointer Register [MMIO Offset 2010h] and the Event Log Tail Pointer Register [MMIO Offset 2018h], if used.
- Write the IOMMU Control Register [MMIO Offset 0018h] with EventLogEn=1b (if used), CmdBufEn=1b, and IommuEn=1b. Other IOMMU Control Register [MMIO Offset 0018h] bits should be set as necessary.

The IOMMU is now operational; it processes device transactions and fetches command buffer entries. When enabled, the IOMMU creates event log entries as events occur.

Revision 2: If using peripheral page service requests, software must initialize the PPR queue registers in addition to the other registers before setting IommuEn=1b:

- the PPR Log Base Address Register [MMIO Offset 0038h],
- the IOMMU PPR Log Head Pointer Register [MMIO Offset 2030h], and
- the IOMMU PPR Log Tail Pointer Register [MMIO Offset 2038h].

Software must also write the IOMMU Control Register [MMIO Offset 0018h] with PPRLogEn=1b (if used).

3.2.2.4 Making Guest Interrupt Virtualization Changes

Revision 1: Guest interrupt virtualization is not supported.

Revision 2: Software may change the guest interrupt virtualization information in a device table entry with a single 64-bit write. The change must be followed by an INVALIDATE_DEVTAB_ENTRY command when the ZV=1 before the change. If a 64-bit operation cannot be used, software may change the guest interrupt virtualization information in the device table entry in the following manner, according to the value of ZV

before the change in the relevant device table entry.
- If ZV=0 before the change, changes can be made in any order as long as the last change is to set ZV=1b.
- If ZV=1 before the change, the following steps may be followed to change interrupt virtualization information:
  - Set ZV=0 in the device table entry to disable interrupt virtualization for the DeviceID; any device-initiated interrupts for the DeviceID are processed using the controls in DTE[159:128].
  - Update the guest APIC table root pointer and invalidate the interrupt table (see Section 3.3.5 [INVALIDATE_INTERRUPT_TABLE]).
  - Set ZV=1 to virtualize interrupts from the device.
  - Invalidate the device table entry (see Section 3.3.2 [INVALIDATE_DEVTAB_ENTRY]).

3.2.3 I/O Page Tables for Host Translations

The IOMMU uses a new page table structure designed to support a full 64-bit device virtual address space while allowing faster translation in many common cases. The IOMMU page tables are a generalization of `AMD64 long` mode page tables. The IOMMU page tables are a multi-level tree of 4K tables indexed by groups of 9 virtual address bits (determined by the level within the tree) to obtain 8-byte entries. Each page table entry is either a page directory entry pointing to a lower-level 4K page table, or a page translation entry specifying a system physical page address. A page translation entry is a page table entry with the Next Level field set to 0h or 7h. A page directory entry is a page table entry with the Next Level field not equal to 0h or 7h. Revision 2: The maximum value of Next Level in a page directory entry is defined in MMIO Offset 0030h[HATS]; exceeding this limit causes an IO_PAGE_FAULT.

The first generalization in the IOMMU page tables compared to AMD64 processor page tables is that directory entries, in addition to specifying the address of the lower page table, also specify the level, or grouping of bits within the virtual address, that is used for the next page table lookup step. This allows the IOMMU to skip page translation steps in cases where the virtual address often contains long strings of 0 bits, such as software architectures that allocate virtual memory sparsely.

The second generalization in the IOMMU page tables is that page translation entries can specify the page size of the translation. The default page size of a translation can be overridden by setting the Next Level bits to 7h. When the Next Level bits are 7h, the size of the page is determined by the first zero bit in the page address, starting from bit 12 (illustrated in Table 11). The page size specified by this method must be larger than the default page size and smaller than the default page size for the next higher level.

The page addresses illustrated in Table 11 are 64-bit values that have been zero-extended from the 52-bit values specified in the DTE and page tables.

Table 11: Example Page Size Encodings

Level	Address Bits																					Page Size	Default Page Size
	63:52**, 51:32*	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12		
1	Page Address																				0	8K	4K
1	Page Address																			0	1	16K	4K
1	Page Address													0	1	1	1	1	1	1	1	1M	4K
2	Page Address											0	1	1	1	1	1	1	1	1	1	4M	2M

* Address bits 51:32 can be used to encode page sizes greater that 4G.
** Address bits 63:52 are zero-extended.

Table 11: Example Page Size Encodings

3	Page Address	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4G	1G
6	7_FFFFh	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Entire cache	NA
6	F_FFFFh	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Undef	Undef
* Address bits 51:32 can be used to encode page sizes greater that 4G. ** Address bits 63:52 are zero-extended.																							

Software Note: The page tables are required to have one PTE for each default page size (see Table 12). When the Next Level bits are equal to 7h, some of the least significant bits of the virtual address indexing the PTE are used for indexing the enlarged physical page, therefore those bits are not unique for indexing the PTE and the PTE must be repeated accordingly. For example, if the physical page is 32K bytes, the 3 least significant bits of the Page Table Level 1 virtual address cannot be used only for indexing within the page table and therefore the PTE must be repeated 8 times for each of the 64 unique PTEs given 4K byte page tables. Another example for 4M byte pages is illustrated in Figure 12. The PTE in the Level-2 page table is replicated twice and bit 21 of the virtual address is used twice for indexing, first to index the Level-2 table of PTEs and again to index into the 4M byte page for the data. The replicated Level-2 PTEs have identical contents and follow the example in Table 11 for a page size of 4M bytes. For larger page sizes, the PTEs must be replicated an appropriate number of times so that more bits of the virtual address can be used for indexing.

Implementation Note: While IOMMU implementations are not strictly required to include translation caches, it is strongly recommended that they include at least a cache for translations of 4K page table entries. IOMMU implementations can cache translations of larger pages by splitting them into multiple 4K cache entries.

The page table pointer for each domain specifies the system physical address and level of the root page table for that domain. Translation of a device virtual address begins by comparing it to the root page table's level. If the address contains any nonzero bits in bit positions higher than the range selected by the root page table's level, translation terminates with an IO_PAGE_FAULT. Otherwise, the appropriate group of virtual address bits is used to fetch a page table entry from the root page table. If this entry is marked not present, translation terminates with an IO_PAGE_FAULT. Otherwise the entry may be a page directory entry pointing to a lower-level page table (in which case the translation process repeats starting at the new page table using the remaining virtual address bits), or it may be a page translation entry containing the final system physical address (in which case the translation process terminates and the remaining device virtual address bits are concatenated with the translation entry's physical address to obtain a translated address). If a translation skips levels and any of the skipped virtual address bits are non-zero, translation terminates with an IO_PAGE_FAULT.

Effective write permission is calculated using the IW bits in the DTE (see Table 5), the I/O PDEs, and the I/O PTE. Device accesses to translated addresses are first checked against these cumulative permissions before being allowed to proceed. IW and IR bits from skipped levels are treated as if they were 1s. For a discussion of guest and host permissions, see Section 3.2.7 [Guest and Nested Address Translation].

Table 12 specifies the virtual address bit groups used for indexing at each level of the page tables, as well as the default page sizes associated with page translation entries fetched from page tables at each level. Figure 8 and Figure 9 illustrate the formats of page table entries. If a page table entry contains nonzero bits in any of the fields marked reserved, if the Next Level field is greater than or equal to the current page table entry table's level, or if a page translation entry's physical address is not aligned to a multiple of the appropriate page size for the current page table entry page table's level, translation terminates with an IO_PAGE_FAULT.

The layout of IOMMU page table entries has been chosen so that the IOMMU can use `AMD64 long` mode processor page tables, provided the Next Level fields (which occupy bit positions ignored by AMD64 processors) are properly initialized according to their level within the processor page tables. (AMD64 processors lack the IOMMU's level skipping facility.) All other page table entry fields used by the IOMMU are either ignored by AMD64 processors, or have the same meaning to both the processor and the IOMMU. For more details on sharing page tables see Section 3.2.4 [Sharing AMD64 Processor and IOMMU Page Tables - GPA-to-SPA].

The U bit in the page tables is an attribute bit passed to peripherals in ATS responses. See Table 10 for the behavior of the IOMMU for settings of the DTE[Cache] and PTE[U] fields.

IOMMU implementations must zero-fill all high-order physical address (SPA).The IOMMU fields are architected to produce a physical address of up to 52 bits, thus physical address bits [63:53] are always zero.

Table 12: Page Table Level Parameters

Page Table Level	Virtual address bits indexing table	Default Page size (bytes) for translation entries
6	63:57	NA
5	56:48	2^{48}
4	47:39	2^{39}
3	38:30	2^{30}
2	29:21	2^{21}
1	20:12	4096

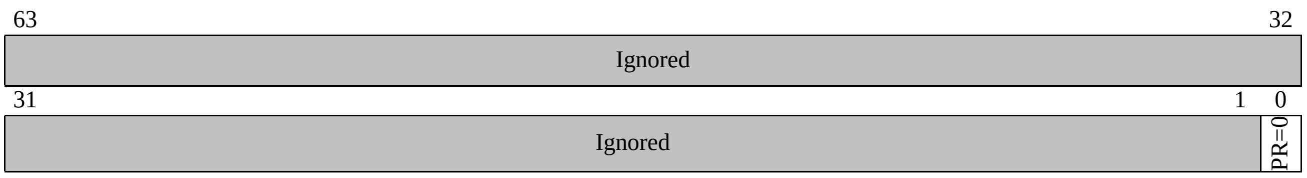


Figure 8: I/O Page Table Entry Not Present (any level)

Table 13: I/O Page Table Entry Not Present Fields, PR=0

Bits	**Description**
63:1	Ignored when PR=0.
0	**PR: Present**. 0=the remainder of the I/O page table entry is ignored and the corresponding memory page is considered not-present (see Section 3.4.2 [IO_PAGE_FAULT Event]). When PR=1, see Table 14 and Table 15.

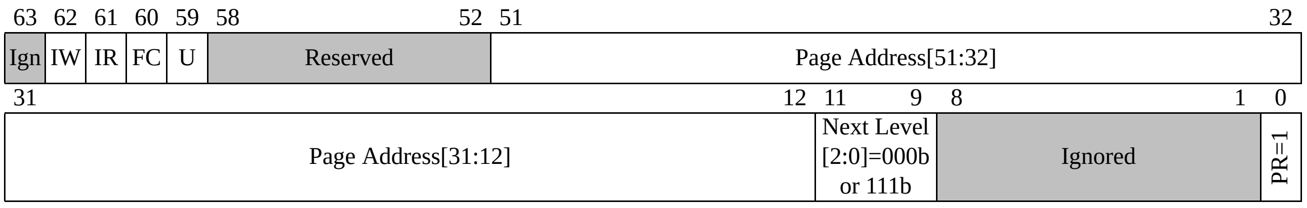
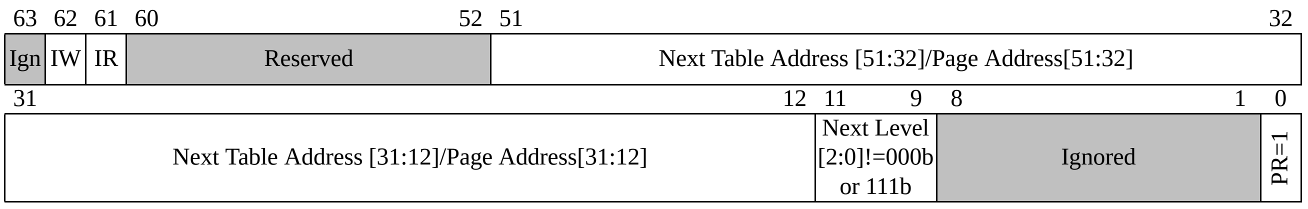


Figure 9: I/O Page Translation Entry (PTE), PR=1

Table 14: I/O Page Translation Entry (PTE) Fields, PR=1

Bits	Description
63	Ignored.
62	**IW: write permission**. 1=write operations are allowed. 0=write operations are not allowed (see Section 3.4.2 [IO_PAGE_FAULT Event]). Effective write permission is calculated using the IW bits in the DTE (see Table 5), the I/O PDEs, and the I/O PTE. At each step of the translation process, I/O write permission (IW) bits from fetched page table entries are logically ANDed into cumulative I/O write permissions for the translation including the IW bit in the DTE. IW bits from skipped levels are treated as if they were 1s. For a discussion of guest and host permissions, see Section 3.2.7 [Guest and Nested Address Translation].
61	**IR: read permission**. 1=read operations are allowed. 0=read operations are not allowed (see Section 3.4.2 [IO_PAGE_FAULT Event]). Effective read permission is calculated using the IR bits in the DTE (see Table 5), the I/O PDEs, and the I/O PTE. At each step of the translation process, I/O read permission (IR) bits from fetched page table entries are logically ANDed into cumulative I/O read permissions for the translation including the IR bit in the DTE. IR bits from skipped levels are treated as if they were 1s. For a discussion of guest and host permissions, see Section 3.2.7 [Guest and Nested Address Translation].
60	**FC: Force Coherent**. The FC bit in the page translation entry is used to specify if DMA transactions that target the page must clear the PCI-defined No Snoop bit. The state of FC is returned to a peripheral in an ATS response. 1=for an untranslated access, the IOMMU sets the coherent bit in the upstream HyperTransport™ request packet. 0=for an untranslated access, the IOMMU passes upstream the coherent attribute from the originating request.
59	**U**. The U bit in the I/O page table entry is an attribute bit passed to a peripheral in an ATS response for a GPA-to-SPA translation. For a GVA-to-SPA translation, hardware must set U=0 in the ATS response. For details, see Table 10 and the *PCI ATS Specification Version 1.1* or newer.
58:52	Reserved.
51:12	**Page Address[51:12]:** Specifies the SPA of the page.
11:9	**NextLevel: next page translation level**. Specifies the level of page translation as described in this section. The value of NextLevel cannot exceed the value of the Mode field in the DTE (Table 5).
8:1	Ignored.
0	**PR: Present**. 1=the remainder of the I/O PTE contains valid information. 0=see Table 13.



63 62 61 60 52 51 32
Ign
IW
IR
Reserved
Next Table Address [51:32]/Page Address[51:32]
31 12 11 9 8 1 0
Next Table Address [31:12]/Page Address[31:12]
Next Level [2:0]!=000b or 111b
Ignored
PR=1

Figure 10: I/O Page Directory Entry (PDE), PR=1

Table 15: I/O Page Directory Entry (PDE) Fields, PR=1

Bits	Description
63	Ignored.

Table 15: I/O Page Directory Entry (PDE) Fields, PR=1

62	**IW: write permission**. 1=write operations are allowed. 0=write operations are not allowed (see Section 3.4.2 [IO_PAGE_FAULT Event]). Effective write permission is calculated using the IW bits in the DTE (see Table 5), the I/O PDEs, and the I/O PTE. Effective write permission is calculated using the IW bits in the DTE (see Table 5), the I/O PDEs, and the I/O PTE. At each step of the translation process, I/O write permission (IW) bits from fetched page table entries are logically ANDed into cumulative I/O write permissions for the translation including the IW bit in the DTE. IW bits from skipped levels are treated as if they were 1s. For a discussion of guest and host permissions, see Section 3.2.7 [Guest and Nested Address Translation].
61	**IR: read permission**. 1=read operations are allowed. 0=read operations are not allowed (see Section 3.4.2 [IO_PAGE_FAULT Event]). Effective read permission is calculated using the IR bits in the DTE (see Table 5), the I/O PDEs, and the I/O PTE. At each step of the translation process, I/O read permission (IR) bits from fetched page table entries are logically ANDed into cumulative I/O read permissions for the translation including the IR bit in the DTE. IR bits from skipped levels are treated as if they were 1s. For a discussion of guest and host permissions, see Section 3.2.7 [Guest and Nested Address Translation].
60:52	Reserved.
51:12	**Next Table Address[51:12]/Page Address[51:12]:** Specifies the SPA of the next page descriptor entry when NextLevel != 000b or 111b; specifies the SPA of the page when NextLevel = 000b or 111b. See discussion in this section.
11:9	**NextLevel: next page translation level**. Specifies the level of page translation as described in this section. The value of NextLevel cannot exceed the value of the Mode field in the DTE (Table 5).
8:1	Ignored.
0	**PR: Present**. 1=the remainder of the I/O PTE contains valid information. 0=see Table 13.

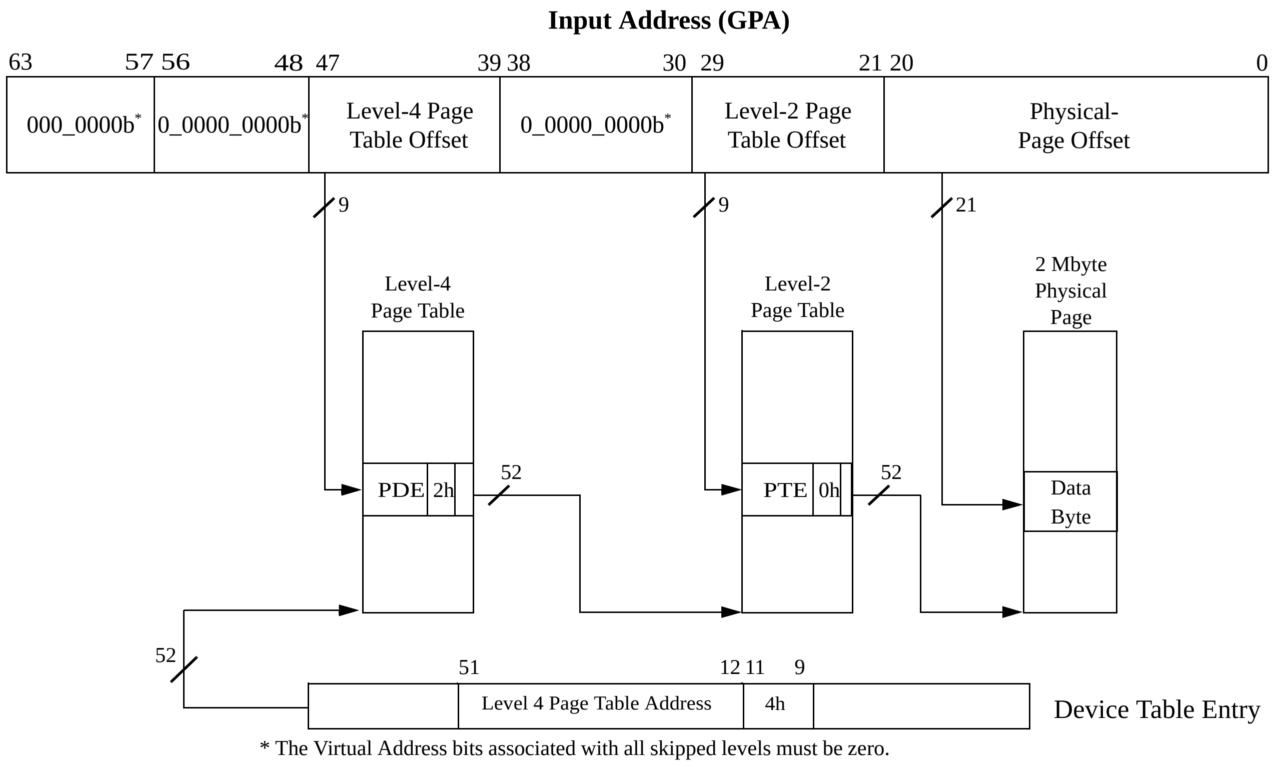


Figure 11: Address Translation Example with Skipped Level and 2M page

Revision 1: The input address in Figure 11 is a GPA from the peripheral. Revision 2: The input address in Figure 11 is a GPA that is supplied by the peripheral or translated from a GVA.

Using the nested page tables, the IOMMU translates the input GPA to an SPA within a 2 Mbyte physical page. The input address is mapped into page table offsets for the levels of address translation. The level-4 page table offset is used to index into the level-4 page table. The level-3 table offset is zero, so the contents of the level-4 page table entry points directly to a level-2 page table. The level-2 page table contains an entry with the next level=0, so that entry points directly to a 2M page and the physical page offset is the 21 low-order bits of the input address.

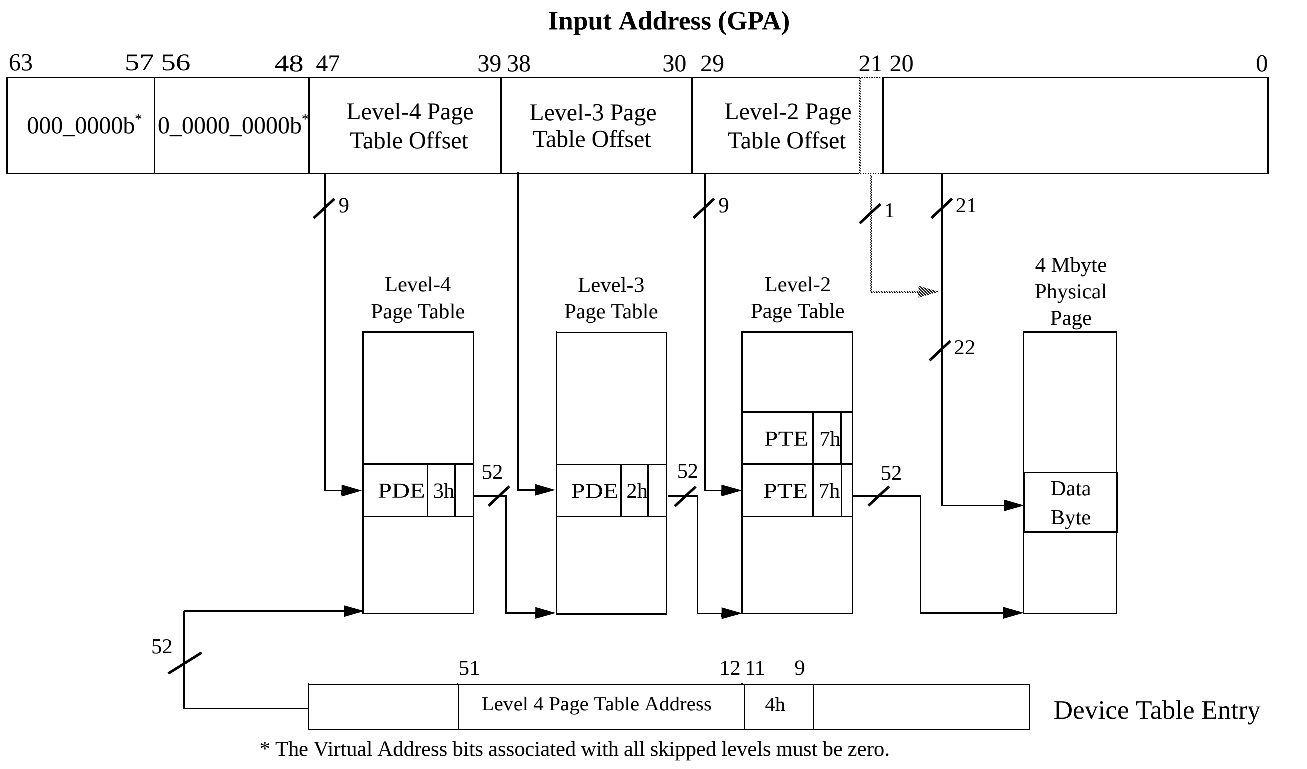


Figure 12: Address Translation Example with Page Size Larger than Default Size

Revision 1: The input address in Figure 12 is a GPA. Revision 2: The top address in Figure 12 is a GPA that is supplied by the peripheral or translated from a GVA.

Using the nested page tables, the IOMMU translates the input GPA to an SPA within a 4 Mbyte physical page. The translations for level-4 and level-3 are conventional and the next level fields are used to indicate contiguous levels of translation with no level skipping. The level-2 table contains paired entries with the next level fields set to 7h; as a result, bit 21 of the input GPA can be treated as an additional offset bit within a larger physical page 4 Mbyte in size. The adjacent PTE values in the level-2 page table must be adjacent 2 Mbyte page base addresses, and the lower base address value must be set so that the page is 4 Mbyte aligned.

3.2.4 Sharing AMD64 Processor and IOMMU Page Tables - GPA-to-SPA

This section outlines the topics to be considered so that the host or GPA-to-SPA page tables may be shared with an IOMMU. A more complete discussion depends on many implementation factors.

AMD64 processors and the IOMMU treat upper virtual address bits [63:48] differently. The processor requires canonical addresses (in which address bits [63:48] are equal to bit 47). By contrast, the IOMMU is designed to support the full PCI 64-bit address space. If 6-level page tables are used, the IOMMU can map any 64-bit address. If fewer than 6 levels are used, the IOMMU requires upper virtual address bits (beyond the range mapped by the page tables) to be 0. This ensures that software can always add levels to page tables without changing the address space as seen by devices.

In `AMD64 long` mode level 4 page tables, the bottom 256 entries of the root page table correspond to positive virtual addresses with bits [63:47] all 0s, and the top 256 entries correspond to negative virtual addresses with bits [63:47] all 1s.

For the IOMMU to directly share processor page tables, at a minimum the Next Level fields in all page table entries must be initialized with correct values for the IOMMU.

Once the Next Level fields are initialized, the IOMMU may directly share exactly the same page tables. In 3-level 32-bit PAE mode this is all that's needed. However, in 4-level long mode software should be aware that processor virtual addresses in the range FFFF_8000_0000_0000h to FFFF_FFFF_FFFF_FFFFh correspond to I/O virtual addresses in the range 0000_8000_0000_0000h to 0000_FFFF_FFFF_FFFFh.

If software requires 64-bit processor virtual addresses to be identical to I/O virtual addresses, including negative addresses, software needs to configure the IOMMU with the 6-level paging structure illustrated in Figure 13, where 4 extra 4K byte page tables (shaded) at levels 6, 5, and 4 are used solely by the IOMMU, and sharing with processor page tables occurs only at levels 3 and below.

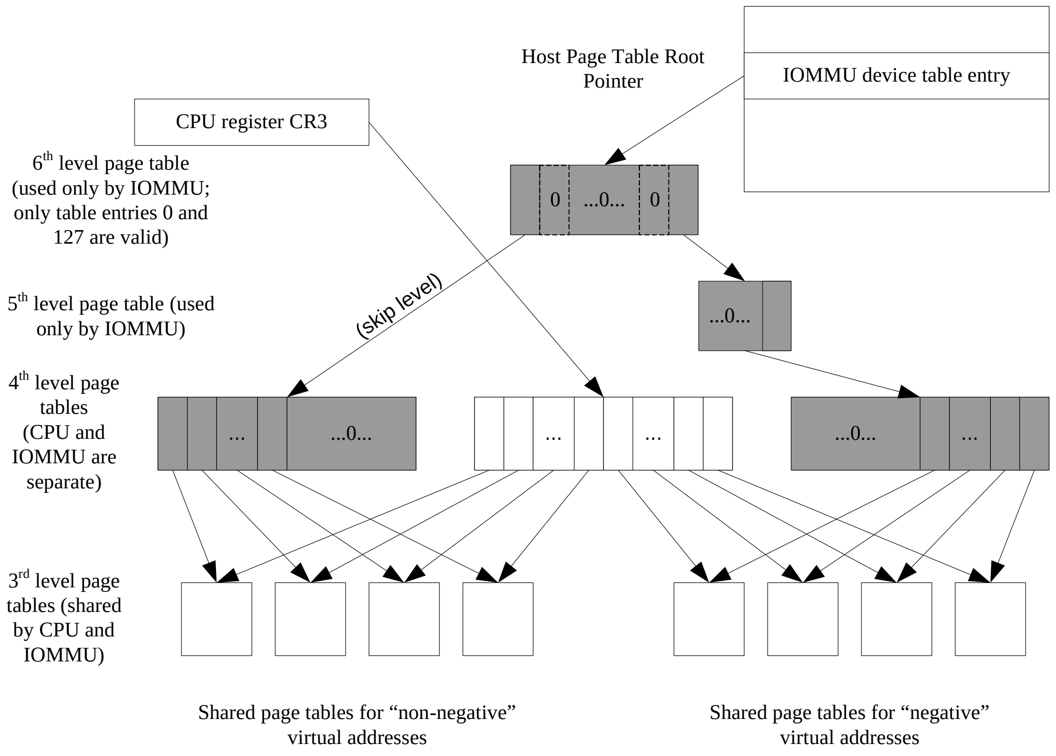

Host Page Table Root Pointer
IOMMU device table entry
CPU register CR3
6th level page table (used only by IOMMU; only table entries 0 and 127 are valid)
0
...0...
0
5th level page table (used only by IOMMU)
(skip level)
...0...
4th level page tables (CPU and IOMMU are separate)
...
...0...
...
...
...0...
...
3rd level page tables (shared by CPU and IOMMU)
Shared page tables for "non-negative" virtual addresses
Shared page tables for "negative" virtual addresses

Figure 13: Sharing AMD64 and IOMMU Host Page Tables with Identical Addressing

3.2.5 Interrupt Remapping Tables

Interrupt messages use a HyperTransport™ interrupt special address range shown in Table 2. All fixed and arbitrated interrupt requests are mapped into the HyperTransport™ address space where they can be remapped by the IOMMU. Other interrupts are handled specially. Startup interrupts cannot originate from I/O devices thus the IOMMU cannot remap them. LINT0, LINT1, NMI, INIT, and External (ExtInt) interrupts are controlled individually using the device table entry control fields (see Table 8 and Table 16). The binary encodings listed in Table 16 are from the HyperTransport™ architecture specification for the MT field.

Revision 2: When interrupt remapping and interrupt virtualization are active (Section 3.2.8 [Guest Virtual APIC Table for Interrupt Virtualization]), interrupts are remapped using the remapping tables and then posted for delivery to a guest VM.

Table 16: IOMMU Interrupt Controls and Actions

Interrupt type (with MT encoding)		Destination Mode (DM)	Controlled by
0000b	Fixed	0b	Device table entry and interrupt remapping table entry
0001b	Arbitrated		
0010b	SMI		Forward unmapped
0011b	NMI		NMIPass
0100b	INIT		InitPass
0110b	ExtInt		EIntPass
1011b	Lint1		Lint1Pass
1110b	Lint0		Lint0Pass
0101b, 0111b, 1000b, 1001b, 1010b, 1100b, 1101b, 1111b	Startup, EOI, EOI		Target abort
0000b-1111b		1b	Device table entry and interrupt remapping table entry

The IOMMU remaps HyperTransport™ addresses for fixed and arbitrated interrupts as shown in the concatenation in Figure 14. The offset created by this concatenation corresponds directly to data bits 10:0 in the originating MSI interrupt message. After reading the interrupt remapping table entry, the IOMMU creates a new interrupt message address by OR'ing IRTE[23:2] with bits [63:2] of HyperTransport™ interrupt address range base (FD_F800_0000h). Interrupt table walks are always coherent.

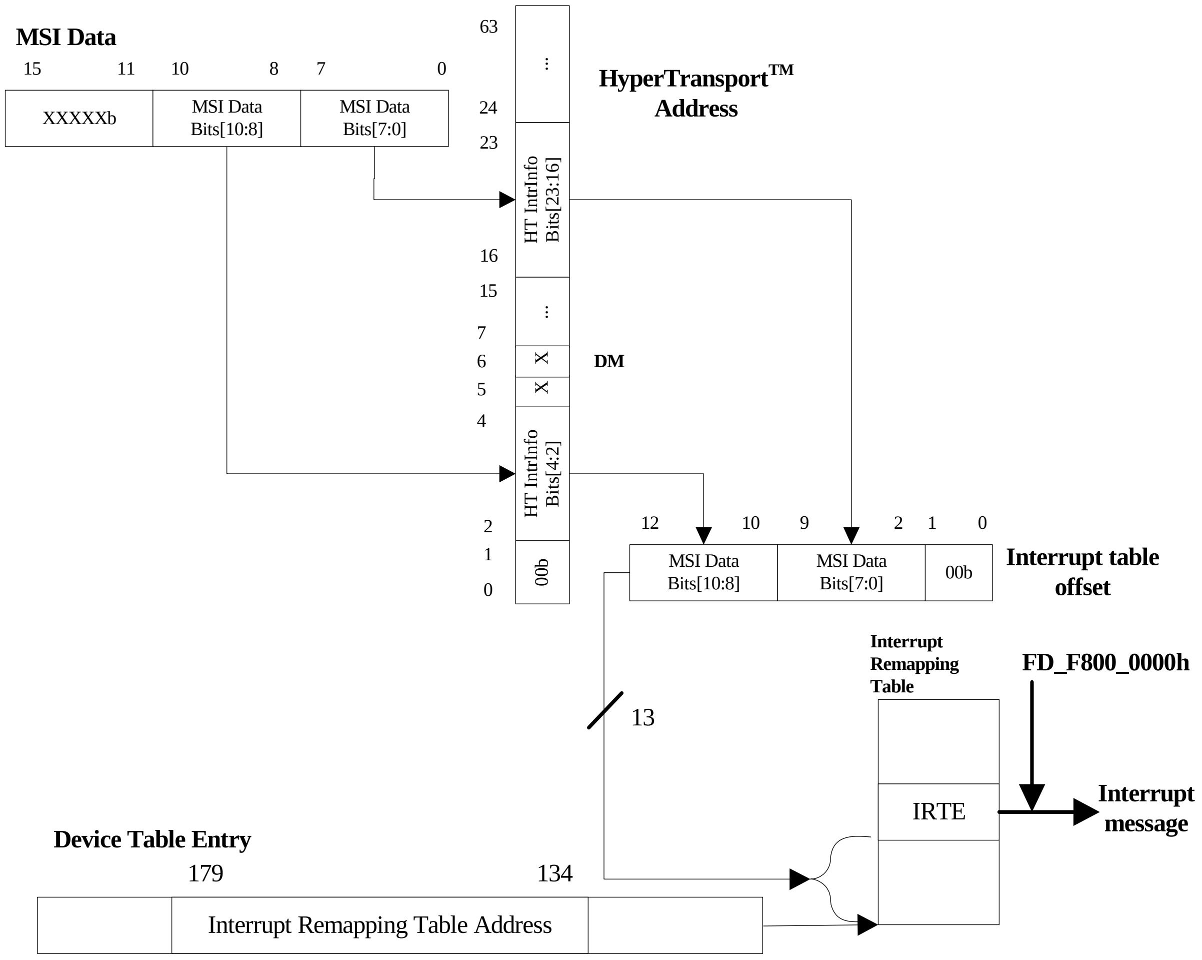


Figure 14: Interrupt Remapping Table Lookup for Fixed and Arbitrated Interrupts

3.2.5.1 Interrupt Remapping Tables

When MMIO Offset 0030h[XTSup]=0, the IOMMU remaps interrupts using the information from the interrupt remapping table entry shown in Figure 15.

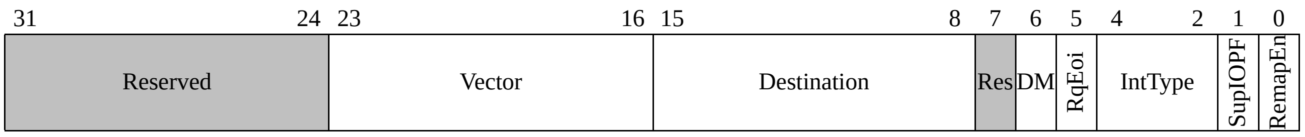


Figure 15: Interrupt Remapping Table Entry

Table 17: Interrupt Remapping Table Fields

Bits	Description
31:24	Reserved.
23:16	**Vector**. Specifies the interrupt vector for the interrupt.
15:8	**Destination**. Specifies the APIC logical or physical address to send the interrupt to.
7	Reserved.
6	**DM: destination mode**. 1=Logical destination mode. 0=Physical destination mode.

Table 17: Interrupt Remapping Table Fields

5	**RqEoi: request EOI**. 1=EOI cycle required. **Software note:** If RqEoi=1, software is responsible for performing the reverse mapping of the vector number.
4:2	**IntType: interrupt type**. This field specifies the type of interrupt message to deliver to the Local APIC. 000b Fixed 001b Arbitrated 010b Reserved 011b Reserved 100b Reserved 101b Reserved 110b Reserved 111b Reserved
1	**SupIOPF:** suppress IO_PAGE_FAULT events. 1=Supress logging when use of this remapping entry causes an IO_PAGE_FAULT. 0=Log event when this entry causes an IO_PAGE_FAULT. See the IG control bit in the device table entry (Section 3.2.2.1 [Device Table Entry Format]).
0	**RemapEn**. 1=Interrupt is remapped. 0=Interrupt is target aborted. **Note:** SupIOPF is meaningful independent of the value of RemapEn.

3.2.6 I/O Page Tables for Guest Translations

Revision 1: guest address translation is not supported.

Revision 2: The use of guest address translation is controlled by values in the DTE (GV and GLX), MMIO Offset 0018h[GTEn], and MMIO Offset 0030h[GTSup, GLXSup]. Software can use guest address translation by programming hardware support as shown as in Table 3. The size of the guest address is defined by MMIO Offset 0030h[GATS]; exceeding this limit generates an IO_PAGE_FAULT.

Table 18: Guest Address Translation Controls

MMIO Offset 0030h		MMIO Offset 0018h	Device Table Entry		**Description**
GTSup	**GLXSup**	**GTEn**	**GV**	**GLX**	**See also Table 9.**
0	XXb	X	MBZ	MBZ	Guest translation is not supported by the IOMMU.
1	XXb	0	X	XX	Guest translation is not active for the IOMMU.
1	XXb	1	0	XX	Guest translation is not active for the DeviceID.
1	00b, 01b	1	1	00b	Guest translation is active. The GCR3 table is a one-level table in system physical memory.
1	01b	1	1	01b	Guest translation is active. The GCR3 table is a two-level table.

When guest address translation is active, the IOMMU will process a guest virtual addresses when it has a valid PASID (see Section 3.2.7.7 [PCIe TLP PASID Prefix]).

3.2.6.1 Guest CR3 Table

When guest translation is active (see Table 18), the DTE contains an SPA pointer to a GCR3 table containing

GPA entries that are structured like processor CR3 values. The GCR3 table root pointer in the DTE is used when a transaction contains a valid PASID. When valid and present, the PASID is used to walk the guest CR3 table. When MMIO Offset 0030h[GLXSup]=00b, hardware supports a one-level lookup table so the table must be a 4 Kbyte page and must be naturally aligned. Figure 16 illustrates a lookup for guest translation tables.

Software note: IOMMU TLBs are not cleared when a value is changed in the guest CR3 table and software must issue invalidation commands (see Section 3.3 [Commands]).

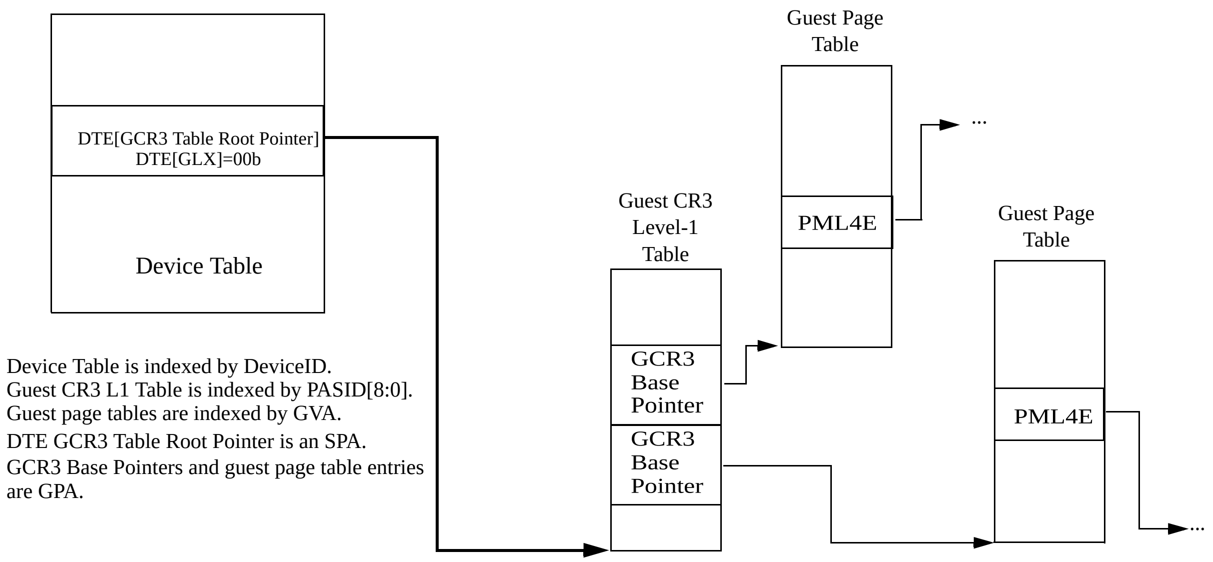

Guest Page Table
DTE[GCR3 Table Root Pointer]
DTE[GLX]=00b
Device Table
...
PML4E
Guest CR3 Level-1 Table
Guest Page Table
GCR3 Base Pointer
GCR3 Base Pointer
PML4E
...
Device Table is indexed by DeviceID.
Guest CR3 L1 Table is indexed by PASID[8:0].
Guest page tables are indexed by GVA.
DTE GCR3 Table Root Pointer is an SPA.
GCR3 Base Pointers and guest page table entries are GPA.

Figure 16: Guest CR3 Table, 1-level

Figure 16 shows how a DTE links to a guest page table using a guest CR3 level-1 table. When guest translation is active (see Table 18) with DTE[GLX]=00b and the peripheral supplies a valid PASID, the PASID[8:0] field is used to index the guest CR3 level-1 table to select a CGR3 base pointer that is the root of a guest page table. The IOMMU ignores PASID[19:9] when DTE[GLX]=01b. The GCR3 table root pointer in the DTE is the SPA of a guest CR3 level-1 table; each valid GCR3 base pointer in the guest CR3 level-1 table is the GPA of a guest page table; the guest page table is a level-4 page table and contains PML4E entries (see AMD64 processor architecture specification). Each valid PML4E in the guest page table is a GPA. The IOMMU translates a GPA in the GCR3 level-1 table and guest page table to a system physical address as needed. Each GCR3 base pointer in Figure 16 is structured as valid bit with a 4 Kbyte-aligned pointer to the guest page table using the formats specified in Table 19 and Figure 17.

Table 19: Guest CR3 Level-1 Table Format

Byte Offset	Guest CR3 Base Table Contents
0	GCR3 Base Pointer entry for PASID$_0$ (GPA)
8	CGR3 Base Pointer entry for PASID$_1$ (GPA)
16	GCR3 Base Pointer entry for PASID$_2$ (GPA)
...	...
4088	GCR3 Base Pointer entry for PASID$_{511}$ (GPA)

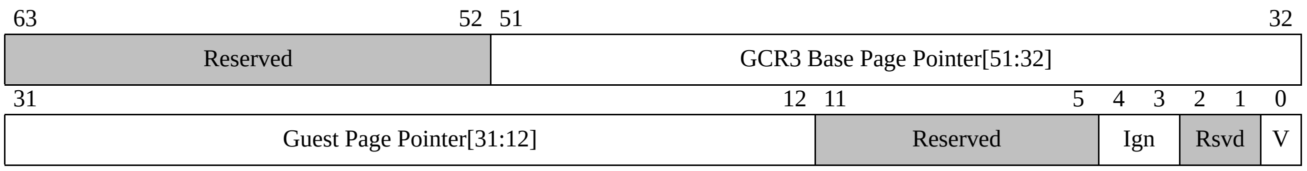


Figure 17: GCR3 Base Pointer Entry Format

Table 20: GCR3 Base Pointer Entry Fields

Bits	Description
63:52	Reserved when V=1. Ignored when V=0.
51:12	**Guest Page Table Pointer**. Specifies a GPA base table address when V=1.
11:5	Reserved when V=1. Ignored when V=0.
4:3	Ignored. The PCD and PWT bits used in the processor CR3 register are ignored by the IOMMU.
2:1	Reserved when V=1. Ignored when V=0.
0	**V: Valid**. Valid bit for the GCR3 base table pointer. 0=the GCR3 base pointer is ignored by hardware. 1=the GCR3 base pointer is the GPA of the root page of a guest page table.

The structure in Figure 16 may be used for PASID values up to 9-bits long and software must program DTE[GLX]=00b. For larger PASID values up to 18-bits, the two-level structure in Figure 18 must be used and software must program DTE[GLX]=01b; for 19- and 20-bits, the three-level structure must be used and software must program DTE[GLX]=10b.

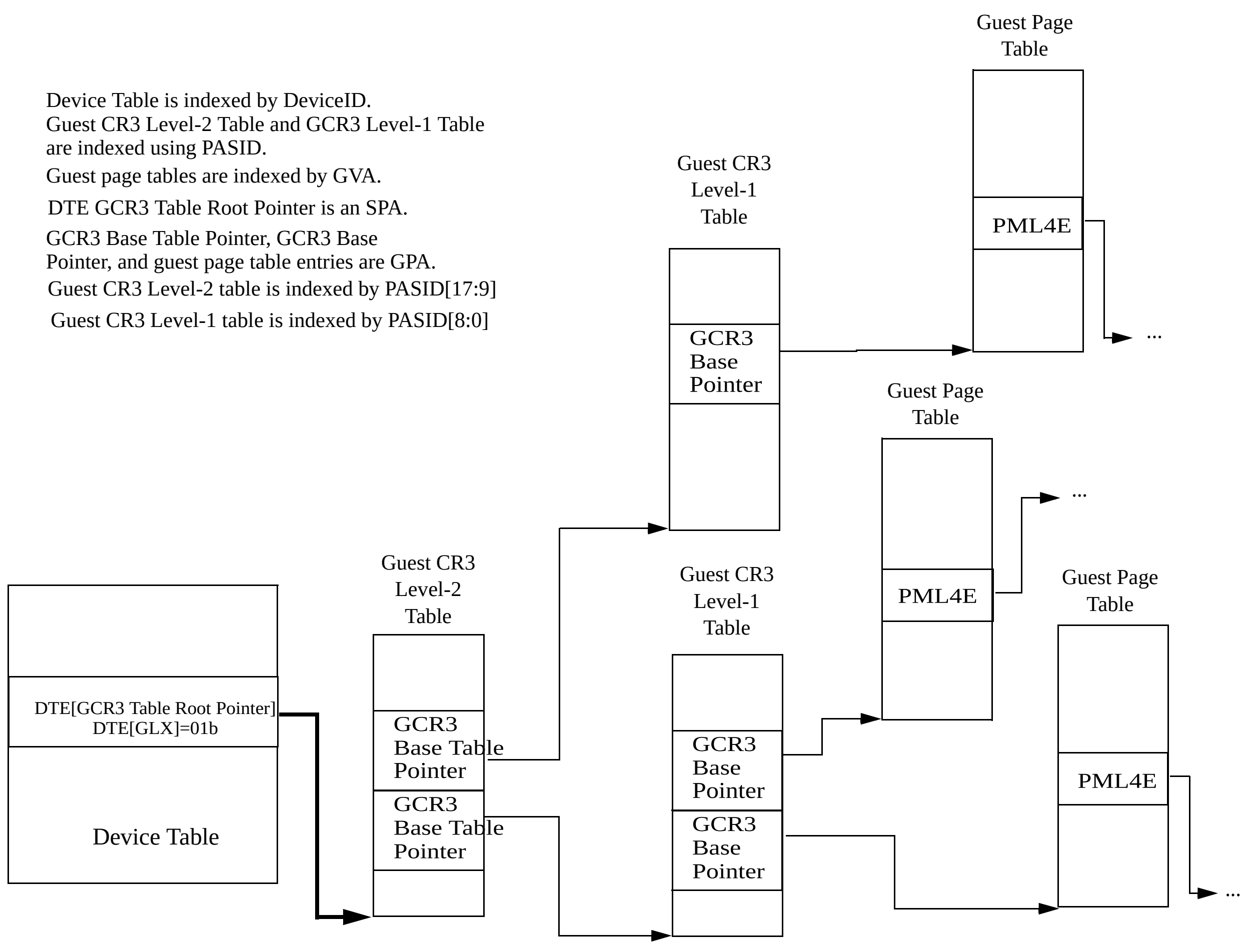

Device Table is indexed by DeviceID.
Guest CR3 Level-2 Table and GCR3 Level-1 Table are indexed using PASID.
Guest page tables are indexed by GVA.
DTE GCR3 Table Root Pointer is an SPA.
GCR3 Base Table Pointer, GCR3 Base Pointer, and guest page table entries are GPA.
Guest CR3 Level-2 table is indexed by PASID[17:9]
Guest CR3 Level-1 table is indexed by PASID[8:0]
DTE[GCR3 Table Root Pointer]
DTE[GLX]=01b
Device Table
Guest CR3 Level-2 Table
GCR3 Base Table Pointer
GCR3 Base Table Pointer
Guest CR3 Level-1 Table
GCR3 Base Pointer
Guest CR3 Level-1 Table
GCR3 Base Pointer
GCR3 Base Pointer
Guest Page Table
PML4E
Guest Page Table
PML4E
Guest Page Table
PML4E
...

Figure 18: Guest CR3 Table, 2-level

Figure 17 shows how a DTE links to a guest page table using a two-level guest CR3 table. When DTE[GLX]=01b and guest translation is active, the GCR3 table root pointer in the DTE is the SPA of a guest CR3 level-2 table; each valid GCR3 base table pointer in the guest CR3 level-2 table is the GPA of a guest CR3 level-1 table; each valid GCR3 base pointer in the guest CR3 level-1 table is the GPA of a guest page table; the guest page table is a level-4 page table and contains PML4E entries (see the AMD64 processor architecture specification). The PML4E in the guest page table is a GPA. When a peripheral supplies an address with a valid PASID and DTE[GLX]=01b, the IOMMU translates the GPA in the GCR3 level-2, GCR3 level-1, and guest page table to an SPA as needed. The guest CR3 level-2 table is indexed using PASID[17:9]; the guest CR3 level-1 table is indexed using PASID[8:0]; and the PASID[19:18] field is ignored. Each GCR3 level-2 base pointer in Figure 18 is structured as valid bit with a 4 Kbyte-aligned pointer to the guest page table using the formats specified in Table 21 and Figure 19; GCR3 level-1 tables use the format specified in Table 19 and Figure 17.

Table 21: Guest CR3 Level-2 Table Format

Byte Offset	Guest CR3 Level-2 Table Contents
0	GCR3 Base Table Pointer to GCR3 Level-1 Table$_0$ (GPA)
8	CGR3 Base Table Pointer to GCR3 Level-1 Table$_1$ (GPA)
16	GCR3 Base Table Pointer to GCR3 Level-1 Table$_2$ (GPA)

Byte Offset	Guest CR3 Level-2 Table Contents
...	...
4088	GCR3 Base Table Pointer to GCR3 Level-1 Table$_{511}$ (GPA)

Each GCR3 base table pointer in Table 21 is structured as a valid bit with a 4K-aligned GPA of a GCR3 table level-1 table.

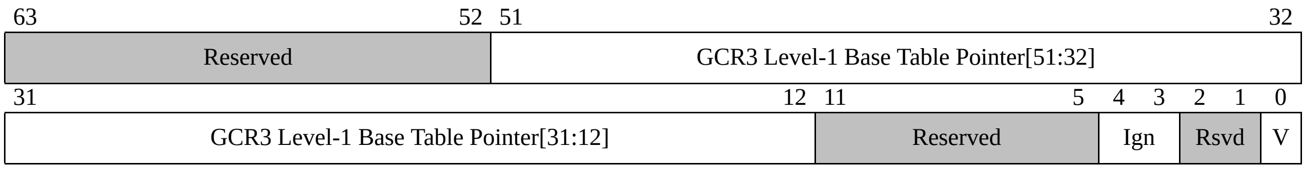

63
52
51
32
Reserved
GCR3 Level-1 Base Table Pointer[51:32]
31
12
11
5
4
3
2
1
0
GCR3 Level-1 Base Table Pointer[31:12]
Reserved
Ign
Rsvd
V

Figure 19: Guest CR3 Level-2 Base Table Pointer Format

Table 22: Guest CR3 Level-2 Base Table Pointer Fields

Bits	Description
63:52	Reserved when V=1. Ignored when V=0.
51:12	**GCR3 Level-1 Base Table Address**. Specifies a GPA base table address when V=1. Ignored when V=0.
11:5	Reserved when V=1. Ignored when V=0.
4:3	Ignored. The PCD and PWT bits used in the processor CR3 are ignored by the IOMMU.
2:1	Reserved when V=1. Ignored when V=0.
0	**V: Valid**. Valid bit for the GCR3 level-1 base table address. 1=the GCR3 base pointer points to a valid table of GCR3 level-1 pointer values. 0=the GCR3 base pointer is ignored by hardware.

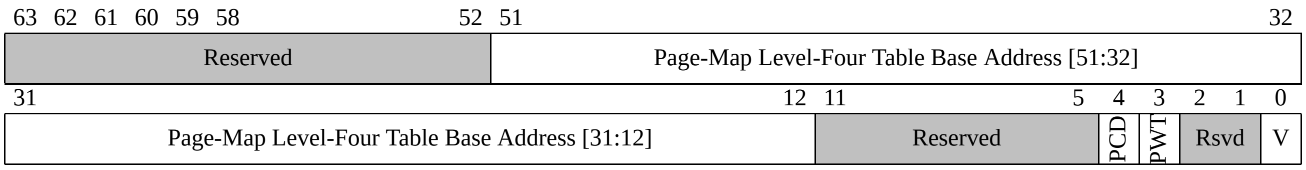

63
62
61
60
59
58
52
51
32
Reserved
Page-Map Level-Four Table Base Address [51:32]
31
12
11
5
4
3
2
1
0
Page-Map Level-Four Table Base Address [31:12]
Reserved
PCD
PWT
Rsvd
V

Figure 20: Guest CR3 Level-1 Entry Format

Table 23: Guest CR3 Level-1 Table Entry Fields

Bits	Description
63:52	Reserved when V=1. Ignored when V=0.
51:12	**PML4 Base Address**. Specifies the GPA base address of the page-map level-4 translation table when V=1. Ignored when V=0.
11:5	Reserved when V=1. Ignored when V=0.
4	**PCD: page-level cache disable bit**. Indicates whether the highest-level page-translation table is cacheable. 1=highest-level table is not cacheable. 0=highest-table is cacheable. An IOMMU implemented outside the processor coherency domain ignores this bit. Used when V=1. Ignored when V=0.

Table 23: Guest CR3 Level-1 Table Entry Fields

3	**PWT: page-level writethrough bit**. Indicates whether the highest-level page-translation table has a writeback or writethrough caching policy. 1=highest-level table has a writethrough caching policy. 0=highest-level table has a writeback caching policy An IOMMU implemented outside the processor coherency domain ignores this bit. Meaningful when V=1. Ignored when V=0.
2:1	Reserved when V=1. Ignored when V=0.
0	**V: Valid.** Valid bit for the PML4 base address. 1=the PML4 base address points to a valid address translation tree. 0=the PML4 base address is ignored.

The IOMMU uses a guest CR3 level-3 table when DTE[GLX]=10b. The guest CR3 level-3 table is pointed to by the DTE and the structure is the same as the guest CR3 level-2 table. The guest CR3 level-3 table is indexed using PASID[19:18], thus it contains four entries and the remainder of the guest CR3 level-3 table is ignored.

The `AMD64 long` page table structure is illustrated in Figure 21. The address translation page tables in Figure 21 contain guest physical addresses that must be translated by the IOMMU to access system memory (PML4E, PDPE, PDE, and PTE). A full nested translation is illustrated in Figure 33. The fields in the AMD64 page table formats are the same for corresponding steps of a translation and are replicated here for clarity. Specifically, the PML4E formats are the same in Figure 22, Figure 27, and Figure 31; the PDPE formats are the same in Figure 23, Figure 28, and Figure 32; and the PDE formats are the same in Figure 24 and Figure 29.

3.2.6.2 AMD64 4K Page Translation

The 4 Kbyte page table formats are defined by the AMD64 processor architecture and interpreted by the IOMMU as in Figure 22, Figure 23, Figure 24, Figure 25, and Table 24. The Page-Map Level-4 Table Address, PML4E, PDPE, PDE, and PTE are guest physical addresses that must be translated by the IOMMU using nested page tables to be system physical addresses (see Section 3.2.3 [I/O Page Tables for Host Translations]).

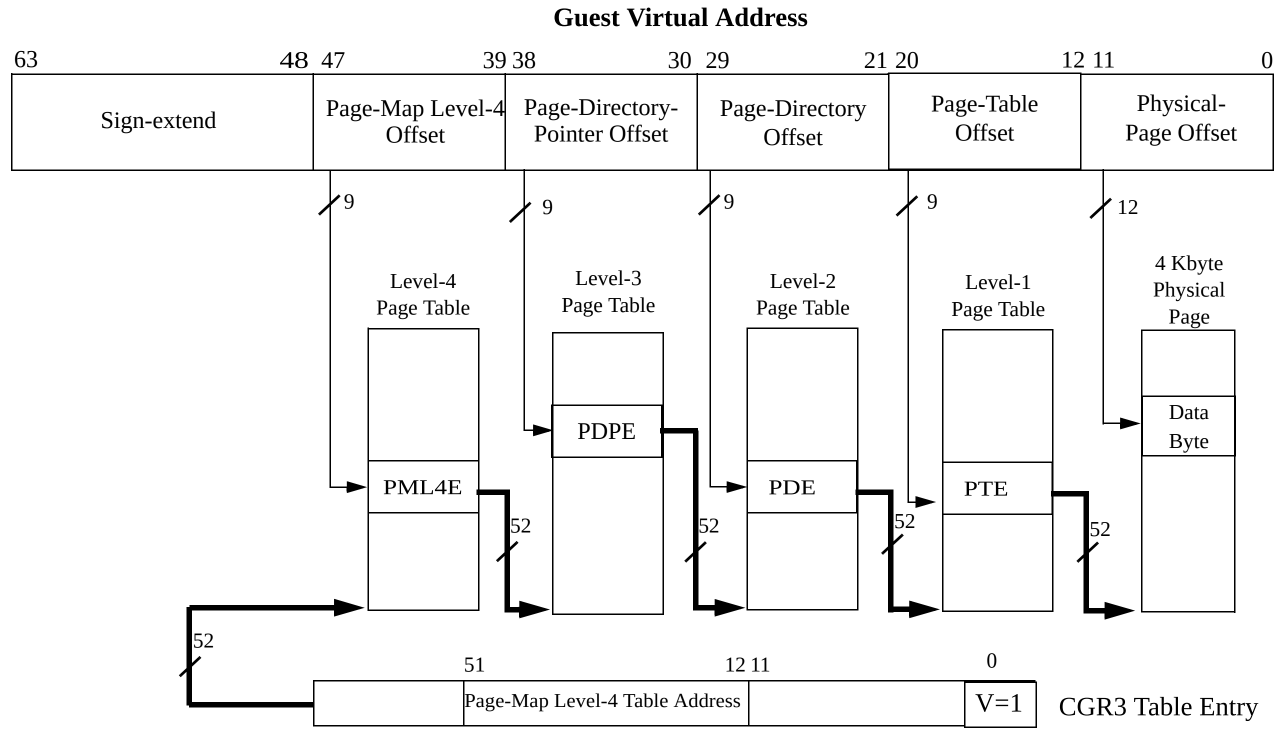


Figure 21: AMD64 Long Mode 4 Kbyte Page Address Translation

63	62–52	51–32
NX	Available	Guest-Physical Page-Directory-Pointer Base Address[51:32]

31–12	11–9	8–7	6	5	4	3	2	1	0
Guest-Physical Page-Directory-Pointer Base Address[31:12]	AVL	MBZ	IGN	A	PCD	PWT	U/S	R/W	P

Figure 22: AMD64 Long Mode 4-Kbyte PML4E Format

63	62–52	51–32
NX	Available	Guest-Physical Page-Directory Base Address[51:32]

31–12	11–9	8–7	6	5	4	3	2	1	0
Guest-Physical Page-Directory Base Address[31:12]	AVL	MBZ	IGN	A	PCD	PWT	U/S	R/W	P

Figure 23: AMD64 Long Mode 4-Kbyte PDPE Format

63	62–52	51–32
NX	Available	Guest-Physical Page-Table Base Address[51:32]

31–12	11–9	8	7	6	5	4	3	2	1	0
Guest-Physical Page-Table Base Address[31:12]	AVL	IGN	0	IGN	A	PCD	PWT	U/S	R/W	P

Figure 24: AMD64 Long Mode 4-Kbyte PDE Format

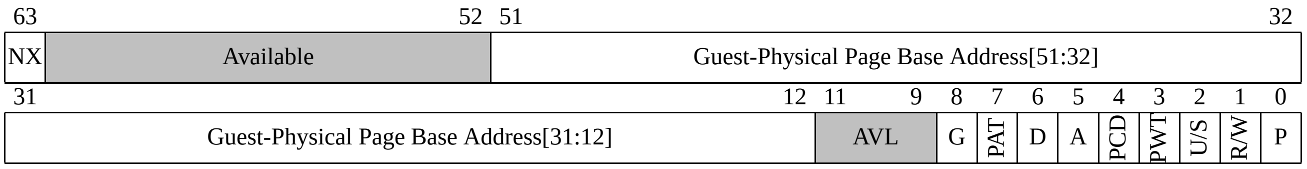

63
52
51
32
NX
Available
Guest-Physical Page Base Address[51:32]
31
12
11
9
8
7
6
5
4
3
2
1
0
Guest-Physical Page Base Address[31:12]
AVL
G
PAT
D
A
PCD
PWT
U/S
R/W
P

Figure 25: AMD64 Long Mode 4-Kbyte PTE Format

Table 24: IOMMU Interpretation of AMD64 Page Table Fields for 4 Kbyte Page Translation

Bits	Description
63	**NX: No execute**. 0=fetch for execution is allowed. 1=fetch for execution is blocked. Ignored by the IOMMU if not implemented (see MMIO Offset 0030h[NXSup]). This bit controls the ability to execute code from all physical pages mapped by the table entry. The no-execute protection check applies to all privilege levels; it does not distinguish between supervisor and user-level accesses.
62:52	**Available**. Ignored by the IOMMU.
51:12	**Guest-Physical Page Base Address**. IOMMU uses same meaning as AMD64 processor; specifies a guest-physical base address when P=1. For 4 Kbyte pages, bits 11:0 are assumed to be zero; for 2 Mbyte pages, bits 20:0 are assumed to be zero; and for 1 Gbyte pages, bits 29:0 are assumed to be zero.
11:9	**AVL: Available**. Ignored by the IOMMU.
8	**G: Global Page**. For 4 Kbyte-page PTE this bit is ignored by the IOMMU.
	IGN: Ignored. For 4 Kbyte-page PML4E, PDPE, and PDE this bit is ignored by the IOMMU.
7	**PAT: Page-Attribute Table**. For 4 Kbyte-page PTE this bit is ignored by the IOMMU.
	MBZ: Must be zero. For 4 Kbyte-page PML4E, PDPE, and PDE this bit must be zero.
6	**D: Dirty**. For 4 Kbyte-page PTE this bit is present in the lowest level of the page-translation hierarchy. This bit indicates whether the page-translation table or the physical page to which this entry points has been written to by a peripheral. The D bit is set to 1 by the IOMMU the first time the a peripheral writes to the physical page. The D bit is never cleared by the IOMMU. See Section 3.2.7 [Guest and Nested Address Translation].
	IGN: Ignored. For 4 Kbyte-page PML4E, PDPE, and PDE this bit is ignored by the IOMMU.
5	**A: Accessed**. This bit indicates whether the page-translation table or the physical page to which this entry points has been accessed by an IOMMU or processor. The A bit is set to 1 by the IOMMU the first time the table or physical page is either read from or written to. The A bit is never cleared by the IOMMU. See Section 3.2.7 [Guest and Nested Address Translation].
4	**PCD: Page-level Cache Disable**. Ignored by the IOMMU.
3	**PWT: Page-level Writethrough**. Ignored by the IOMMU.
2	**U/S: User/Supervisor.** IOMMU uses same meaning as AMD64 processor page tables. 0=access is restricted to supervisor level. 1=both user and supervisor access is allowed. **Software note**: For a peripheral not using U/S, software should program the bit to signal user mode.

Table 24: IOMMU Interpretation of AMD64 Page Table Fields for 4 Kbyte Page Translation

1	**R/W: Read/Write.** This bit controls read/write access to all physical pages mapped by the table entry. 0=access is read-only. 1=access is either read or write. Actual permissions applied to a given page are cumulatively ORed during the page-table walk. The IOMMU converts this status to separate read- and write-enable bits where required.
0	**P: Present**. Present bit indicates whether the page-translation table or guest physical page is loaded in physical memory. 0=page is not present. 1=page is present. When P=0, all the remaining bits in this data structure entry are available to software and are not used by the IOMMU. Entries with P=0 are never cached in an IOMMU TLB nor will the IOMMU set the Accessed or Dirty bit for the table entry.

3.2.6.3 AMD64 2M Page Translation

The 2 Mbyte page table formats are defined by the AMD64 processor architecture and interpreted by the IOMMU as in Figure 27, Figure 28, Figure 29, and Table 25.

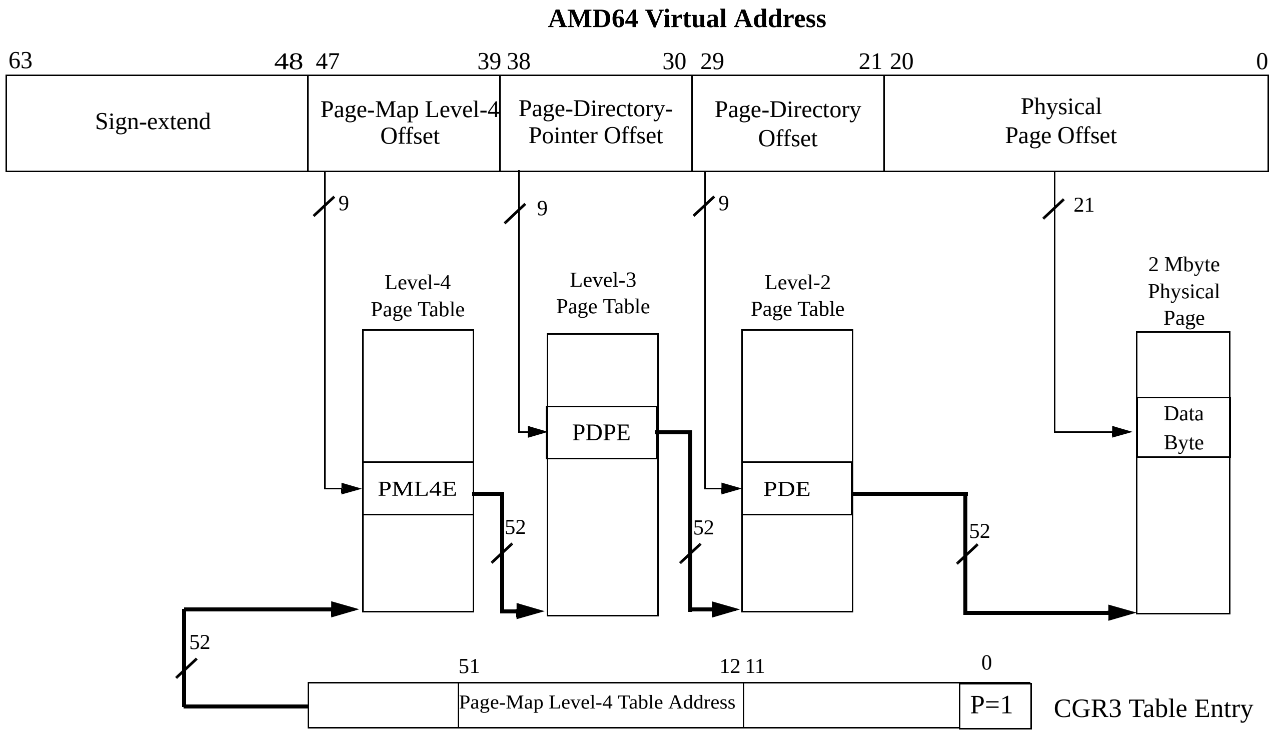


Figure 26: AMD64 Long Mode 2 Mbyte Page Address Translation

63	62–52	51–32
NX	Available	Guest-Physical Page-Directory-Pointer Base Address[51:32]

31–12	11–9	8–7	6	5	4	3	2	1	0
Guest-Physical Page-Directory-Pointer Base Address[31:12]	AVL	MBZ	IGN	A	PCD	PWT	U/S	R/W	P

Figure 27: AMD64 Long Mode 2-Mbyte PML4E Format

63	62–52	51–32
NX	Available	Guest-Physical Page-Directory Base Address[51:32]

31–12	11–9	8–7	6	5	4	3	2	1	0
Guest-Physical Page-Directory Base Address[31:12]	AVL	MBZ	IGN	A	PCD	PWT	U/S	R/W	P

Figure 28: AMD64 Long Mode 2-Mbyte PDPE Format

63	62–52	51–32
NX	Available	Guest-Physical Page-Table Base Address[51:32]

31–21	20–13	12	11–9	8	7	6	5	4	3	2	1	0
Guest-Physical Page-Table Base Address[31:21]	Reserved (MBZ)	PAT	AVL	G	1	D	A	PCD	PWT	U/S	R/W	P

Figure 29: AMD64 Long Mode 2-Mbyte PDE Format

Table 25: IOMMU Interpretation of AMD64 Page Table Fields for 2 Mbyte Page Translation

Bits	Description
63	**NX: No execute**. 0=fetch for execution is allowed. 1=fetch for execution is blocked. Ignored by the IOMMU if not implemented (see MMIO Offset 0030h[NXSup]).
62:52	**Available**. Ignored by the IOMMU.
51:21	**Guest-Physical Page Base Address[51:21]**. Specifies a guest-physical base address when P=1.
20:13	**Guest-Physical Page Base Address[20:13]**. For 2 Mbyte-page PML4E and PDPE, specifies a guest-physical base address when P=1.
	Reserved. For 2 Mbyte-page PDE, must be zero.
12	**Guest-Physical Page Base Address[12]**. For 2 Mbyte-page PML4E and PDPE, specifies a guest-physical base address when P=1.
	PAT: Page Attribute Table. For 2 Mbyte-page PDE, this bit is ignored by the IOMMU.
11:9	**AVL: Available**. Ignored by the IOMMU.
8	**G: Global Page**. For 2 Mbyte-page PTE, this bit is ignored by the IOMMU.
	MBZ. For 2 Mbyte-page PML4E, PDPE, and PDE, this bit must be zero.
7	**1b**. For 2 Mbyte-page PDE, this bit must be 1b.
	MBZ: Must be zero. For 2 Mbyte-page PML4E and PDPE, this bit must be zero.
6	**D: Dirty**. For 2 Mbyte-page PDE, this bit is only present in the lowest level of the page-translation hierarchy. This bit indicates whether the page-translation table or the physical page to which this entry points has been written to by a peripheral. The D bit is set to 1 by the IOMMU the first time the a peripheral writes to the physical page. The D bit is never cleared by the IOMMU. See Section 3.2.7 [Guest and Nested Address Translation].
	IGN: Ignored. For 2 Mbyte-page PML4E and PDPE, this bit is ignored by the IOMMU.
5	**A: Accessed**. This bit indicates whether the page-translation table or the physical page to which this entry points has been accessed by an IOMMU or processor. The A bit is set to 1 by the IOMMU the first time the table or physical page is either read from or written to. The A bit is never cleared by the IOMMU. See Section 3.2.7 [Guest and Nested Address Translation].
4	**PCD: Page-level Cache Disable**. Ignored by the IOMMU.
3	**PWT: Page-level Writethrough**. Ignored by the IOMMU.

Table 25: IOMMU Interpretation of AMD64 Page Table Fields for 2 Mbyte Page Translation

2	**U/S: User/Supervisor.** IOMMU uses same meaning as AMD64 processor. 0=access is restricted to supervisor level. 1=both user and supervisor access is allowed. **Software note**: For a peripheral not using U/S, software should set the bit to signal supervisor mode.
1	**R/W: Read/Write.** This bit controls read/write access to all physical pages mapped by the table entry. 0=access is read-only. 1=access is either read or write. Actual permissions applied to a given page are cumulatively ORed during the page-table walk. The IOMMU converts this status to separate read- and write-enable bits where required.
0	**P: Present**. Present bit indicates whether the page-translation table or guest physical page is loaded in physical memory. 0=page is not present. 1=page is present. When P=0, all the remaining bits in this data structure entry are available to software and are not examined by the IOMMU. Entries with P=0 are never cached in an IOMMU TLB nor will the IOMMU set the Accessed or Dirty bit for the table entry.

3.2.6.4 AMD64 1G Page Translation

The 1 Gbyte page table formats are defined by the AMD64 processor architecture and interpreted by the IOMMU as in Figure 31, Figure 32, and Table 26.

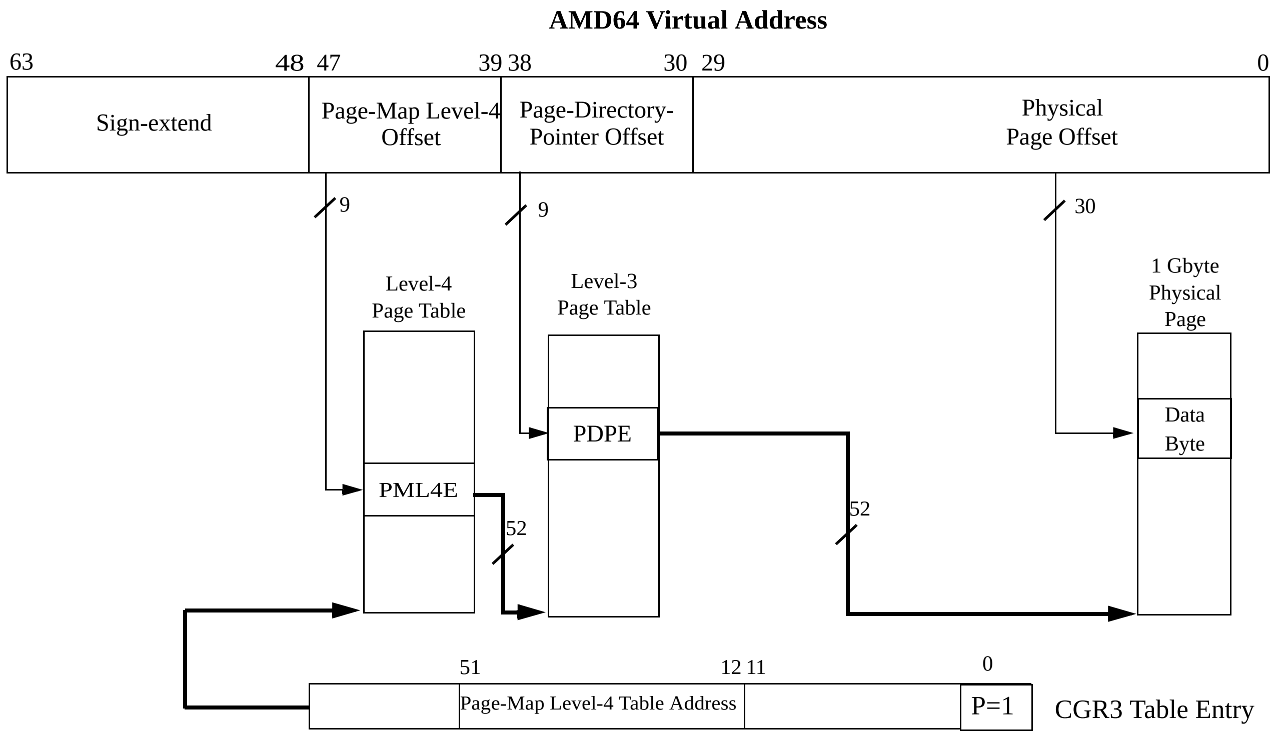


Figure 30: AMD64 Long Mode 1 Gbyte Page Address Translation

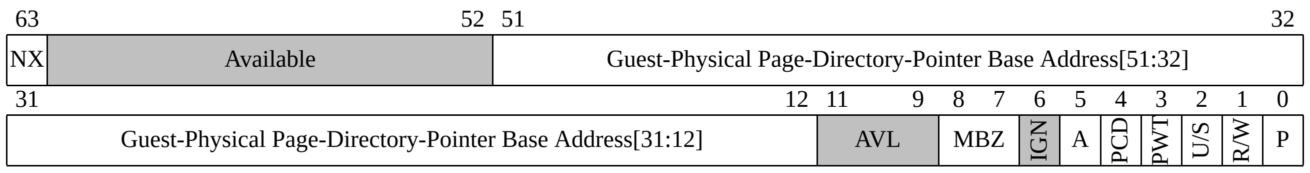

63
52
51
32
NX
Available
Guest-Physical Page-Directory-Pointer Base Address[51:32]
31
12
11
9
8
7
6
5
4
3
2
1
0
Guest-Physical Page-Directory-Pointer Base Address[31:12]
AVL
MBZ
IGN
A
PCD
PWT
U/S
R/W
P

Figure 31: AMD64 Long Mode 1-Gbyte PML4E Format

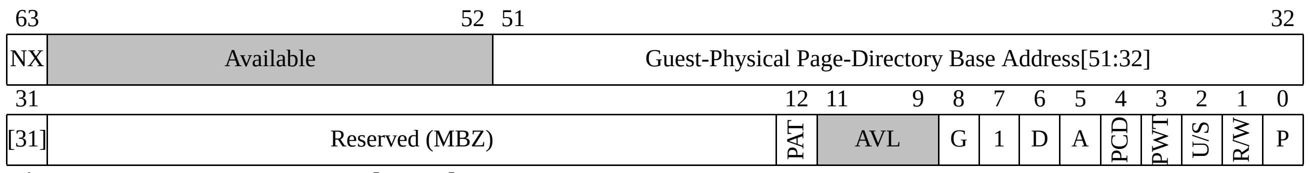


Figure 32: AMD64 Long Mode 1-Gbyte PDPE Format

Table 26: IOMMU Interpretation of AMD64 Long Mode 1 Gbyte Page Table Fields

Bits	Description from AMD64 processor specification
63	**NX: No execute**. 0=fetch for execution is allowed. 1=fetch for execution is blocked. Ignored by the IOMMU if not implemented (see MMIO Offset 0030h[NXSup]).
62:52	**Available**. Ignored by the IOMMU.
51:31	**Guest-Physical Page Base Address[51:31]**. For 1 Gbyte PML4E and PDPE, specifies a guest-physical base address when P=1.
30:13	**Guest-Physical Page Base Address[30:13]**. For 1 Gbyte PML4E, specifies a guest-physical base address when P=1.
	Reserved. For 1 Gbyte-page PDPE, must be zero
12	**Guest-Physical Page Base Address[12]**. For 1 Gbyte PML4E, specifies a guest-physical base address when P=1.
	PAT: Page Attribute Table. For 1 Gbyte-page PDPE this bit is ignored by the IOMMU.
11:9	**AVL: Available**. Ignored by the IOMMU.
8	**G: Global Page**. For 1 Gbyte-page PDPE this bit is ignored by the IOMMU.
	MBZ. For 1 Gbyte-page PML4E this bit must be zero.
7	**1b**. For 1 Gbyte-page PDPE this bit must be 1b.
	MBZ: Must be zero. For 1 Gbyte-page PML4E this bit must be zero.
6	**D: Dirty**. For 1 Gbyte-page PDPE, this bit is only present in the lowest level of the page-translation hierarchy. This bit indicates whether the page-translation table or the physical page to which this entry points has been written to by a peripheral. The D bit is set to 1 by the IOMMU the first time the a peripheral writes to the physical page. The D bit is never cleared by the IOMMU. See Section 3.2.7 [Guest and Nested Address Translation].
	IGN: Ignored. For 1 Gbyte-page PML4E, this bit is ignored by the IOMMU.
5	**A: Accessed**. IOMMU uses same meaning as AMD64 processor. This bit indicates whether the page-translation table or the physical page to which this entry points has been accessed by an IOMMU or processor. The A bit is set to 1 by the IOMMU the first time the table or physical page is either read from or written to. The A bit is never cleared by the IOMMU. See Section 3.2.7 [Guest and Nested Address Translation].
4	**PCD: Page-level Cache Disable**. Ignored by the IOMMU.
3	**PWT: Page-level Writethrough**. Ignored by the IOMMU.
2	**U/S: User/Supervisor.** IOMMU uses same meaning as AMD64 processor. 0=access is restricted to supervisor level. 1=both user and supervisor access is allowed. **Software note**: For a peripheral not using U/S, software should set the bit to select supervisor mode.

Table 26: IOMMU Interpretation of AMD64 Long Mode 1 Gbyte Page Table Fields

1	**R/W: Read/Write.** This bit controls read/write access to all physical pages mapped by the table entry. 0=access is read-only. 1=access is either read or write. Actual permissions applied to a given page are cumulatively ORed during the page-table walk. The IOMMU converts this status to separate read- and write-enable bits where required.
0	**P: Present**. Present bit indicates whether the page-translation table or guest physical page is loaded in physical memory. 0=page is not present. 1=page is present. When P=0, all the remaining bits in this data structure entry are available to software and are not examined by the IOMMU. Entries with P=0 are never cached in an IOMMU TLB nor will the IOMMU set the Accessed or Dirty bit for the table entry.

3.2.6.5 Nested Page Table Walks

A guest translation can require many page table entries to complete. Careful IOMMU cache design can significantly reduce the penalty of page table walks.

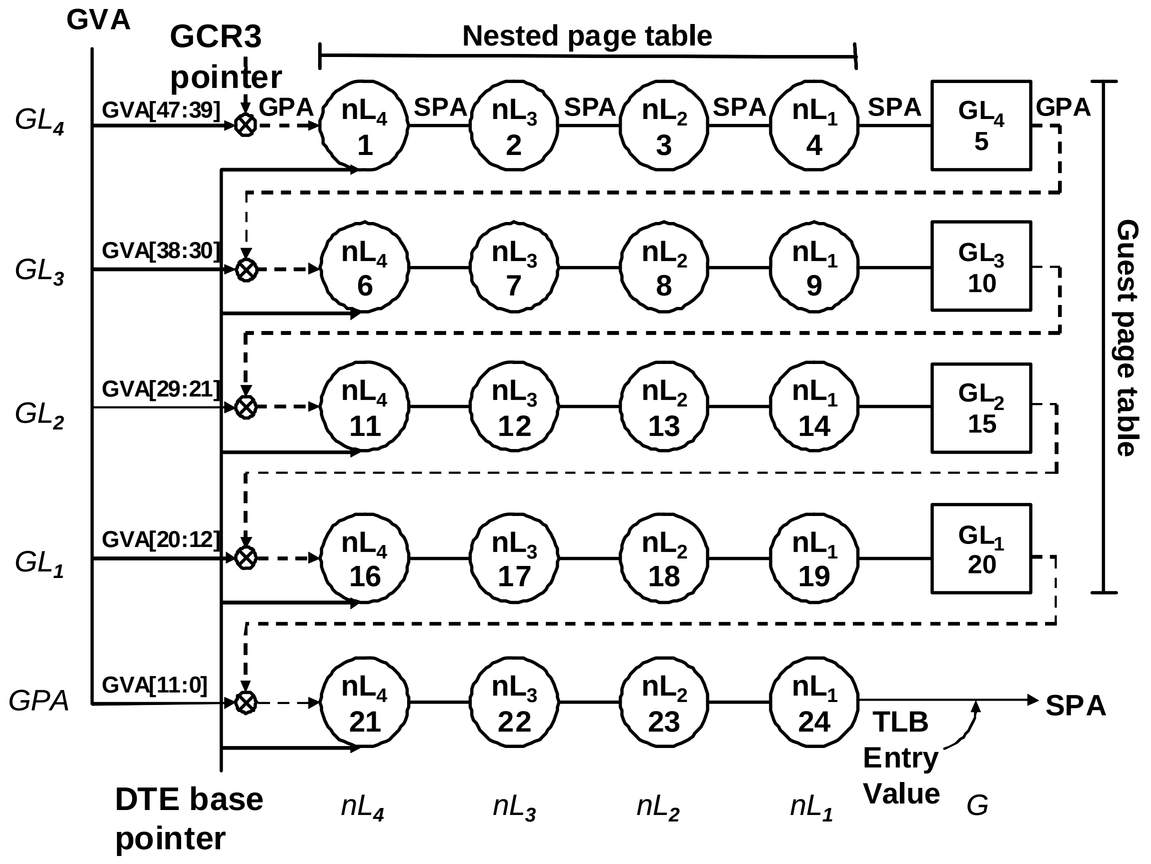


Figure 33: Complete GVA-to-SPA Address Translation

The notation in Figure 33 is adapted from the AMD64 processor architecture specification and uses the notation for processor nested paging. A GVA is shown at the top-left of the figure. The circles indicate GPA translation entries that use the host page table root pointer in the DTE ("DTE base pointer" in the figure). The square entries are GPA entries that are obtained using the guest translation tables pointed to from the GCR3 Table. The square entries on the right edge of the figure correspond to Figure 22, Figure 23, Figure 24, and Figure 25. In the ideal case, an IOMMU TLB cache entry is found containing the required SPA and the translation is complete.

If there is no TLB cache hit, the IOMMU must do a complete page table walk. The GVA must be processed through four layers of guest address translations in the guest physical address space; this is illustrated down the diagram. Each GPA must be translated into the system physical address space to obtain a series of intermediate translation records, illustrated across the diagram.

3.2.7 Guest and Nested Address Translation

Revision 1: Host nested address translation is supported. Guest address translation is not supported.

Revision 2: Guest and nested address translation are supported.

3.2.7.1 Combining Guest and Host Address Translation

The guest and nested (host) translation can be operated in four basic combinations: disabled, together, and each independently. Interrupt remapping is controlled separately by programming DTE[IV].

1. IOMMU address translation is turned off by programming DTE[V]. When DTE[V]=0, no address translation or access checking is performed by the IOMMU for upstream operations from the device. Peripherals have full access to the entire system physical address space. ATS and PRI requests fail.
2. The IOMMU provides GPA-to-SPA address translation by programming DTE[V]=1 and DTE[GV]=0. This operational configuration offers address translation with features such as skip-level tables, large pages (e.g., 8 Kbytes, 16 Kbytes, etc.), and access control. ATS and PRI requests can be enabled.

The next two combinations require that guest address translation is supported by and enabled for the IOMMU (see MMIO Offset 0030h[GTSup] and MMIO Offset 0018h[GTEn]).

3. The IOMMU provides `AMD64 long` compatible GPA-to-SPA address translation when software programs DTE[V]=1 and DTE[GV]=1 and DTE[Mode]=0, with DTE[IR] and DTE[IW] as desired. This configuration enables the nested translation in pass-through mode with guest translation active. ATS and PRI requests can be enabled.
4. The IOMMU provides GVA-to-SPA translation similar to the nested paging provided by the processor. Software programs DTE[V]=1 and DTE[GV]=1 and the associated translation tables. Accesses are translated using the guest tables for GVA-to-GPA and the underlying nested tables for GPA-to-SPA. ATS and PRI can be enabled.

3.2.7.2 Calculating Page Table and Page Access Attributes

Revision 1: The IOMMU calculates read and write attributes as described in Section 3.2.3 [I/O Page Tables for Host Translations].

Revision 2: The IOMMU calculates guest access attributes and nested access attributes for read, write, executable, and user/supervisor permission, and for present, page-accessed and page-dirty attributes. Note that the updating of control bits in the page tables is visible to the CPU when the IOMMU is sharing guest page tables.

- Read (R, IR) permission - The read permission for a page is calculated as a cumulative-AND of the read permission bits in the guest and nested page descriptors and the IR bit in the DTE. For merged R/W bits, the permission is considered “read allowed” when the page is marked “present”.
- Write (W, IW) permission - The write permission for a page is calculated as a cumulative-AND of the write permission bits in the guest and nested page descriptors and the IW bit in the DTE.
- Executable (NX) permission - The NX permission is calculated as a cumulative-OR of the NX permission

bits in the guest page descriptors. The NX permission is ignored when a peripheral supplies a GPA or SPA (i.e., when guest address translation is not used). When enabled, the NX permission bit applies to accesses from either the CPU or the peripherals.
- User/supervisor (U/S) permission - The U/S permission is calculated as a cumulative-AND of the U/S permission bits in the guest page descriptors. The U/S permission is ignored when a peripheral supplies a GPA or SPA (i.e., when guest address translation is not used). When enabled, the U/S permission bit applies to accesses from either the CPU or the peripherals.
- Page accessed (A) attribute - The Accessed attribute is not cumulative. The Accessed attribute bit applies to the page containing the next level of the translation table (PML4E[A] bit refers to the PDPE page, etc.), and the Accessed bit in the PTE refers to the data page.
- Page dirty (D) attribute - The Dirty attribute is not cumulative. The Dirty attribute bit applies to the page containing the next level of the translation table (PML4E[D] bit refers to the PDPE page, etc.), and the Dirty bit in the PTE refers to the data page.
- Present (P) attribute - The Present attribute bit applies to page containing the next level of the translation table (PML4E[P] bit refers to the PDPE page, etc.), and the Present bit in the PTE refers to the data page. The page-table walk terminates when the first non-present page is discovered.

Implementation note: As an optimization, a page table walk may be terminated early as long as the end state is not compromised. For example, a write operation to a memory location may be terminated (IO_PAGE_FAULT) at the first descriptor found for a read-only region.

3.2.7.3 Recalculating Present, Read, and Write Access Permissions

Revision 1: The IOMMU may calculate page-present, read, and write access status from cached or in-memory information.

Revision 2: The IOMMU calculates page-present, read, and write access status from cached or in-memory information; if the result is access-denied using cached information, the IOMMU recalculates page-present, read, and write access status from in-memory information when guest address translation is used by the peripheral.

When guest translation is active (see MMIO Offset 0018h[GTEn]), the IOMMU follows the `AMD64 long` address translation requirements for guest virtual addresses and so software is not required to issue an invalidation command when it promotes guest access privileges or marks a not-present guest page as present. Software is required to issue an invalidation command when it demotes guest access privileges or removes the guest page (“present to not-present”). Therefore, an ATS request or memory reference that results in insufficient guest privileges drawn from a TLB entry may be based on stale information. When the IOMMU detects an access violation using cached guest translation information, it must rewalk the guest page tables to recompute access permission using fresh information read from memory, in the process replacing or discarding cached information. The nested page tables may be read as a consequence of the guest table rewalk. If the retrieved information contains permission control settings that disallow the access then the IOMMU blocks the access; else the IOMMU allows the requested access. An ATS translation request calculates access privileges the same way and returns the computed result. The rewalk may require a full walk of both guest and nested translations (see Section 3.2.7 [Guest and Nested Address Translation]).

Software note: For a peripheral using ATS, software must determine the invalidation requirements and issue appropriate IOTLB invalidation commands.

The `AMD64 long` page tables contain information about memory types (PAT) and the IOMMU ignores these bits when it is outside the coherence domain.

3.2.7.4 Updating Accessed and Dirty Bits in the Guest Address Tables

Revision 1: The IOMMU does not modify Accessed (A) and Dirty (D) bits.

Revision 2: The IOMMU updates A and D bits in guest page descriptors when guest address translation is used by the peripheral transaction. When the IOMMU updates A and D bits in the guest page descriptors, it uses interlocked operations compatible with the processor update operations; however the A and D bits are not defined in the nested page descriptors. Note that the setting of accessed and dirty status bits in the page tables is visible to both the CPU and the peripheral when sharing guest page tables. The IOMMU interlocked operations to update A and D bits must be 64-bit operations and naturally aligned on a 64-bit boundary.

When the IOMMU fetches each needed page table entry, it processes the descriptor differently for memory access requests and for translation requests. For a memory access request, the IOMMU processes the descriptor as follows:

1. Decodes the read and write intent from the memory access.
2. If P=0 in the page descriptor, fail the access.
3. Compare the A & D bits in the descriptor with the read and write intent in the request.
4. If the A or D bits need to be updated in the descriptor:
   - Start atomic operation.
   - Read the descriptor as a 64-bit access.
   - If the descriptor no longer appears to require an update, release the atomic lock with no further action and continue to step 5.
   - Calculate the new A & D bits.
   - Write the descriptor as a 64-bit access.
   - End atomic operation.
5. Continue to the next stage of translation or to the memory access.

For a translation request, the IOMMU processes the descriptor as follows:

1. Decode the read and write intent from the ATS request, including the ATS 1.1 NW bit.
2. If P=0 in the descriptor, return an ATS response with no access (R=W=0).
3. Check the A & D bits in the descriptor against the read and write intent in the translation request.
4. If the descriptor is obtained from the TLB (P=1) and permissions are not adequate to meet the request, discard the TLB entry, rewalk the page table, and re-evaluate the request.
5. If the descriptor has been obtained from a page-table walk, return the indicated permissions.
6. If the A or D bits need to be updated in the descriptor:
   - Start atomic operation.
   - Read the descriptor as a 64-bit access.
   - If the descriptor no longer appears to require an update, release the lock with no further action and continue to step 7.
   - Update the A & D bits.
   - Write the descriptor as a 64-bit access.
   - End atomic operation.
7. Continue to the next stage of translation or return the translation result.

3.2.7.5 Clearing Accessed and Dirty Bits

To clear the Accessed bit in a descriptor, software must modify the PTE in memory and then invalidate the page table entry in the IOMMU for all devices using the translation table. For an example, see the pseudo-code in Section 6.2 [Clear Accessed Bit].

To clear the Dirty bit in a descriptor, software must mark the PTE in memory as not-present (PR=0) and invalidate the page table entry in the IOMMU for all devices using the translation table. When the invalidation is complete, the Dirty bit may be examined or changed.

3.2.7.6 Calculating PCIe Read and Write Attributes for an ATS Response

When translating addresses, the IOMMU must convert between `AMD64 long` page table semantics and PCIe semantics using Table 27.

Table 27: Access Privilege Conversion Table for ATS Request

PTE		ATS Request: NW	IOMMU Action	ATS Response	
P	**RW**			**R**	**W**
0	X	X	Issue ATS response.	0	0
1	0	0	If TLB hit, rewalk and reevaluate using in-memory page table entry.	-	-
			If TLB miss, walk page table and issue ATS response.	1	0
1	0	1	Set A and issue ATS response.	1	0
1	1	0	Set A & D and issue ATS response.	1	1
1	1	1	Set A and issue ATS response.	1	0

The page table contains a present bit (P) and a read/write bit (R/W), the ATS request includes a no-write hint, and the ATS response requires separate read (R) and write (W) permission bits. A key requirement is that the IOMMU provide an ATS response consistent with `AMD64 long` page table semantics for privilege promotions. In general, the IOMMU should return results based on the values found in the TLB. The special case for an ATS request are for pages for which the system software may have elevated the access permissions without issuing an invalidation command to the IOMMU. The system software is required to issue an invalidation command when it reduces access permissions (including marking the page not-present with P=0). Specifically, software must invalidate after removing write or execute permission, after changing P from present to not-present, or after changing U/S from user to supervisor.

3.2.7.7 PCIe TLP PASID Prefix

Revision 1: The PCI-SIG TLP prefix is not interpreted by the IOMMU. A PCIe packet contains a GPA and the originating BDF is used to select GPA-to-SPA translation tables.

Revision 2: The PCI-SIG defined a method to add information to a transaction called the TLP prefix. A PCI-SIG ECN uses the TLP prefix to carry added information for a transaction that bears a GVA; this is called the PASID TLP prefix. The IOMMU inspects the TLP prefixes for a PASID TLP prefix when there are multiple and passes through any remaining TLP prefixes (excluding the PASID TLP prefix if one is found). The IOMMU processes memory transactions with a valid PASID TLP prefix; a PASID TLP prefix used with other types of cycles (e.g., configuration cycles, interrupts) may be ignored by the IOMMU. The IOMMU behavior is undefined when it receives a PASID TLP prefix in a downstream direction.When a PCIe transaction has a PASID TLP prefix containing PNP=0b (see Figure 34 and Table 28) and an untranslated address, the transaction is said to contain a valid PASID.

The IOMMU processes the PASID TLP prefix when enabled and MMIO Offset 0018h[GTEn]=1b and MMIO Offset 0030h[GTSup]=1b. An upstream packet with a valid PASID in the PASID TLP prefix contains an x86-

canonical GVA; an upstream packet without a valid PASID in the PASID TLP prefix or with no PASID TLP prefix and an untranslated address contains a GPA. When the PASID TLP prefix contains a valid PASID, the IOMMU processes the packet using the PMR and Exe bits and the guest translation tables. When the PASID TLP prefix does not contain a valid PASID (PNP=1b), the IOMMU ignores the PASID, PMR, and Exe fields.

The PASID TLP prefix contains a 24-bit payload that is interpreted by the IOMMU in the following way, as specified by the PCI-SIG PCIe ECR for “Process Address Space Identifier Prefix”:

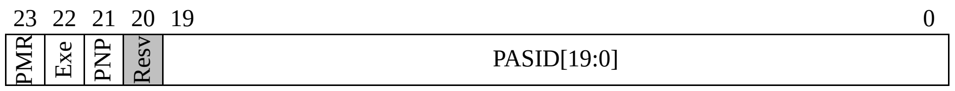


Figure 34: PCIe TLP PASID Prefix Payload Format

Table 28: PCIe TLP Prefix Payload Fields

Bits	Description
23	**PMR: Privileged Mode Requested**. 0=non-privileged (user) request. 1=privileged (supervisor) request.
22	**Exe: execute requested**. 0=no execute permission requested. 1=execute permission requested.
21	**PNP: PASID Not Present**. 0=PASID field contains the PASID to use for the transaction. 1=PASID field is not valid.
20	Reserved. Note: although this bit is reserved, it is passed through the IOMMU as received from the peripheral.
19:0	**PASID[19:0]:** Guest process address space ID.

When a PCIe transaction contains a valid PASID, the packet contains a GVA: the BDF is used to select GPA-to-SPA translation tables, and the PASID TLP prefix contains Exe, PMR, PNP, and PASID information. The PASID is used to select the GVA-to-GPA translation tables. When enabled (see MMIO Offset 0030h[GTSup] and MMIO Offset 0018h[GTEn]), the IOMMU processes the PASID TLP prefix on memory-access PCIe packets, PCIe ATS packets, and PCIe PRI packets. The IOMMU does not process the PASID TLP prefix on an MSI-X or MSI interrupt packet. If an I/O device supplies a PASID TLP prefix that the IOMMU does not process, the IOMMU reports an error in the event log (see Section 3.4.8 [INVALID_DEVICE_REQUEST Event]). When a PCIe memory transaction contains no valid PASID, the packet contains a GPA and the DeviceID is used to select GPA-to-SPA translation tables.

The PCI-SIG defines a TLP prefix as the 32-bit structure shown in Figure 35 and Table 29 and including the payload information from Figure 34.

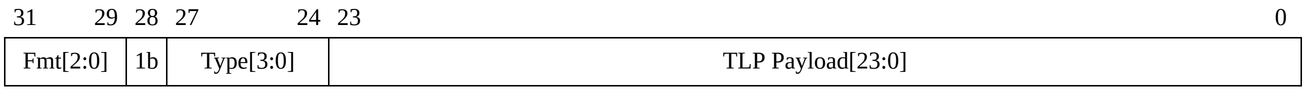


Figure 35: PCI-SIG TLP Prefix Format

Table 29: PCI-SIG TLP Prefix Fields

Bits	Description
31:29	**Fmt[2:0]: TLP Format**. 100b=PASID TLP prefix is interpreted by the IOMMU. 000b-011b and 101b-111b=TLP prefix is not interpreted by the IOMMU.
28	Must be 1b.

Table 29: PCI-SIG TLP Prefix Fields

27:24	**Type[3:0]: TLP Type**. 0001b=PASID type. 0000b and 0010b-1111b=ignored by the IOMMU.
23:0	**TLP Payload[23:0].** For an PASID TLP prefix, the contents are defined in Figure 34 and Table 28.

When the IOMMU receives a TLP prefix with Fmt=001b and Type=100b, it is processed as a PASID TLP prefix and the TLP prefix payload is interpreted to extract the fields in Table 28. If the TLP Prefix Type !=100b, the IOMMU handles the TLP prefix as defined in the PCI-SIG PCIe specifications; see the PCI-SIG TLP prefix definitions for details.

3.2.7.8 Maximum PASID value (PASmax)

The maximum PASID value supported by the IOMMU is calculated as $2^{PASmax+1}$-1 (see MMIO Offset 0030h[PASmax]). Each peripheral may support a smaller value. System software is required to program the guest CR3 tables so that PASID values out-of-range for the peripheral or for the IOMMU are marked not-valid (see Table 17 and Table 19).

3.2.8 Guest Virtual APIC Table for Interrupt Virtualization

Revision 1: The Guest Virtual APIC Table is not supported. Corresponding fields in the DTE must be zero (bits 255:192, inclusive).

Revision 2: The Guest Virtual APIC Table is used to virtualize interrupts. When enabled, the IOMMU can cause interrupts to be delivered directly to running guests without hypervisor intervention for a device with DTE[ZV]=1. When interrupt remapping and interrupt virtualization are both enabled, an incoming device interrupt is first remapped using the IRTE and then delivered using the AMD virtual interrupt controller. Interrupt virtualization requires compatible support in the IOMMU and the processor.

When MMIO Offset 0030h[GASup]=0, the IOMMU does not support interrupt virtualization using the Guest Virtual APIC Table. The Guest Virtual APIC Table and DTE[ZV] are reserved fields for all devices.

When MMIO Offset 0030h[GASup]=1, device interrupt virtualization is enabled for the IOMMU by programming MMIO Offset 0018h[GAEn]=1, DTE[ZV] is programmed to 1 to activate interrupt virtualization for the device, and the Guest Virtual APIC Table Root Pointer in the DTE is programmed to the base address (SPA) of the domain scoreboard.

Further information can be found in the *AMD Virtual Interrupt Controller Specification*, *Revision 1.0* or newer.

3.3 Commands

The host software controls the IOMMU through a shared circular buffer in system memory. The host software writes commands into the buffer and then notifies the IOMMU of their presence by writing a new value to the tail pointer. The IOMMU then reads the commands and executes them at its own pace. The shared command buffer organization was chosen to allow the host software to send commands in batches to the IOMMU, while allowing the IOMMU to set the pace at which commands are actually executed.

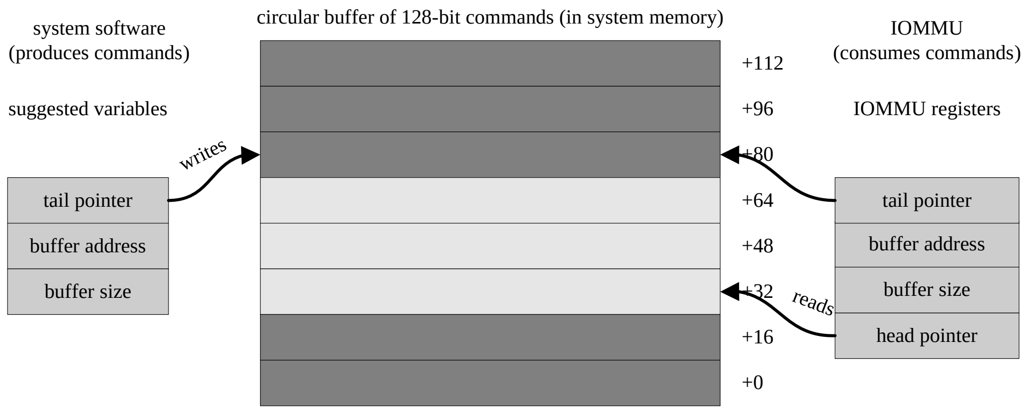


Figure 36: Circular Command Buffer in System Memory

The Command Buffer Base Address Register [MMIO Offset 0008h] is used to program the system physical base address and size of the command buffer. The command buffer occupies contiguous physical memory starting at the programmed base address, up to the programmed size. The size of the command buffer must be a multiple of 4K bytes (to facilitate "mod N" indexing for circularity), and can be as large as 32768 entries (corresponding to a 512 kilobyte buffer). The address of the command buffer must be aligned to a multiple of 4K bytes.

In addition to the Command Buffer Base Address Register [MMIO Offset 0008h], the IOMMU maintains two other registers associated with the command buffer: the Command Buffer Head Pointer Register [MMIO Offset 2000h], an offset from the base address, which points to the next command that the IOMMU can fetch, and the Command Buffer Tail Pointer Register [MMIO Offset 2008h], an offset from the base address, which points to the next command to be written by software. These registers are located in IOMMU MMIO space. When the Command Buffer Base Address Register [MMIO Offset 0008h] register is written, the Command Buffer Head Pointer Register [MMIO Offset 2000h] and the Command Buffer Tail Pointer Register [MMIO Offset 2008h] are reset to the 0. When the Command Buffer Head Pointer Register [MMIO Offset 2000h] and the Command Buffer Tail Pointer Register [MMIO Offset 2008h] are equal the command buffer is empty. The Command Buffer Head Pointer Register [MMIO Offset 2000h] is incremented by the IOMMU after reading a command from the command buffer.

The IOMMU fetches commands in FIFO order from the command buffer. The IOMMU must never refetch a command. The IOMMU must set the Coherent bit in the HyperTransport™ packet when issuing command buffer read requests. Although the IOMMU fetches commands in order, it may execute them concurrently. Software may use the COMPLETION_WAIT command when synchronization is required.

All commands read by the IOMMU take the form of a 4-bit opcode together with two operands, which may be respectively 60 and 64 bits long, for a total of 128 bits (16 bytes) per command:

31 28	27 0	
First opcode dependent operand [31:0]		+00
Opcode[3:0]	First opcode dependent operand [59:32]	+04
Second opcode dependent operand [31:0]		+08
Second opcode dependent operand [63:32]		+12

Figure 37: Generic Command Buffer Entry Format

The COMMAND_HARDWARE_ERROR (Section 3.4.6 [COMMAND_HARDWARE_ERROR Event]) and ILLEGAL_COMMAND_ERROR (Section 3.4.5 [ILLEGAL_COMMAND_ERROR Event]) events cause the IOMMU to halt command processing. If a command buffer entry causes one of these errors, the command head pointer does not advance. Note that the head pointer may have advanced past the command in error. Other activities of the IOMMU, including translations, error logging, and table walks, continue to be processed. Software is required to examine the IOMMU status and event log information to resolve the situation. Command processing is restarted by using the CmdBufEn control bit in the IOMMU Control Register [MMIO Offset 0018h] and status may be determined from CmdBufRun in IOMMU Status Register [MMIO Offset 2020h].

To restart the IOMMU command processing after the IOMMU has halted it, use the following procedure.
- Wait until CmdBufRun=0b in the IOMMU Status Register [MMIO Offset 2020h] so that all commands complete processing as the circumstances allow. CmdBufRun must be 0b to modify the command buffer registers properly.
- Set CmdBufEn=0b in the IOMMU Control Register [MMIO Offset 0018h].
- As necessary, change the following registers (e.g., to relocate the command buffer):
  - the Command Buffer Base Address Register [MMIO Offset 0008h],
  - the Command Buffer Head Pointer Register [MMIO Offset 2000h], and
  - the Command Buffer Tail Pointer Register [MMIO Offset 2008h].
- Any or all command buffer entries may be copied from the old command buffer to the new and software must set the head and tail pointers appropriately.
- Write the IOMMU Control Register [MMIO Offset 0018h] with CmdBufEn=1b and ComWaitIntEn as desired.

The IOMMU now processes command buffer entries.

3.3.1 COMPLETION_WAIT

The COMPLETION_WAIT command allows software to serialize itself with IOMMU command processing. The COMPLETION_WAIT command does not finish until all older commands issued since a prior COMPLETION_WAIT have completely executed.

Implementation note: The COMPLETION_WAIT command may wait to finish after all older commands complete, including prior COMPLETION_WAIT commands. If there are no prior COMPLETION_WAIT commands in the command buffer, the COMPLETION_WAIT command finishes after all older commands. See important considerations in Section 3.3.9 [IOMMU Ordering Rules].

For example, system software that wishes to reclaim pages formerly made available to devices should use the following procedure:
- Mark the page table entry (or entries) not present in the IOMMU's tables.
- Issue appropriate page invalidate commands to the IOMMU.
- Issue a COMPLETION_WAIT command to the IOMMU. When the COMPLETION_WAIT has finished, the IOMMU is designed to ensure that there are no transactions in flight anywhere in the system fabric that read or write the invalidated pages.

Both s=1 and i=1 may be specified in the same COMPLETION_WAIT command.

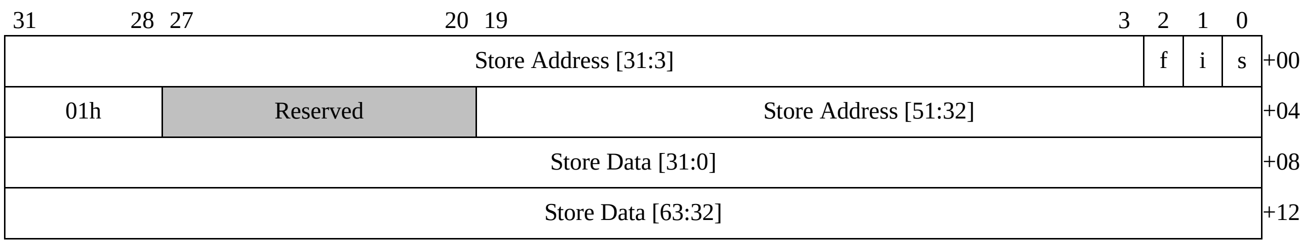


Figure 38: COMPLETION_WAIT Command Format

Table 30: COMPLETION_WAIT Fields

Bits	Description
31:3 +00	**Store Address[31:3]**. The lower portion of the SPA into which the IOMMU may store the Store Data.
2 +00	**f: flush queue**. 0=execution of younger commands may begin at any time. 1=the IOMMU does not begin execution of any younger commands until COMPLETION_WAIT has finished.
1 +00	**i: completion interrupt**. 0=the IOMMU does not set MMIO Offset 2020h[ComWaitInt]. 1=the IOMMU sets MMIO Offset 2020h[ComWaitInt]. See Capability Offset 10h[MsiNum].
0 +00	**s: completion store**. 0=the IOMMU does not write the Store Data value to the Store Address. 1=the IOMMU writes the specified 64-bit Store Data value to the Store Address. Software can use this write to update a semaphore indicating to the waiting process that it can continue execution. The address written by the COMPLETION_WAIT must be located in system memory. **Implementation note:** The write operation must be coherent and not in the isochronous channel. Hardware must not set the PassPw bit when performing this write.
31:28 +04	**01h**. COMPLETION_WAIT command number.
27:20 +04	Reserved.
19:0 +04	**Store Address[51:32]**. The lower portion of the SPA into which the IOMMU may store the Store Data.
31:0 +08	**Store Data[31:0]**. The lower portion of the Store Data.
31:0 +12	**Store Data[63:32]**. The upper portion of the Store Data.

3.3.2 INVALIDATE_DEVTAB_ENTRY

When system software changes a device table entry, it must instruct the IOMMU to invalidate that DeviceID from its internal caches. The IOMMU is then forced to reload the device table entry before DMA from the device is allowed. The IOMMU may reload the device table entry any time after the invalidation has completed.

When software invalidates a DeviceID corresponding to an IOMMU-aware device with its own IOTLB, it should immediately follow INVALIDATE_DEVTAB_ENTRY with an INVALIDATE_IOTLB_PAGES targeted at the same DeviceID and sized to invalidate the full 64-bit address space for the given DeviceID. (Note that on a multi-function device this need only invalidate IOTLB entries for the specified function.)

Note that this command does not invalidate translation cache entries, since they may be in use by other devices sharing the same DomainID. If the DomainID is not shared, software should issue INVALIDATE_IOMMU_PAGES for the DomainID.

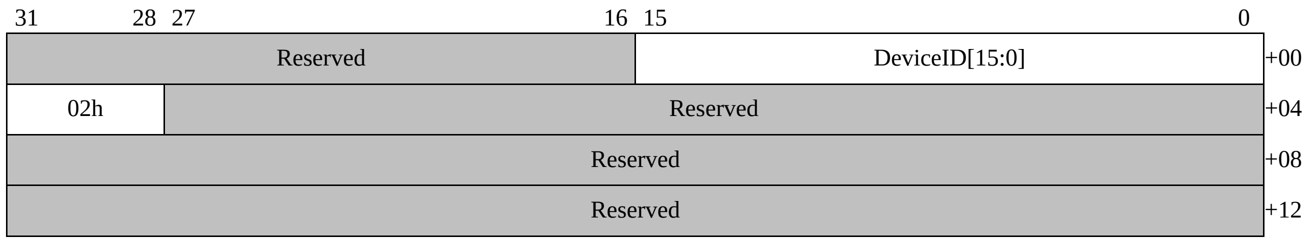


Figure 39: INVALIDATE_DEVTAB_ENTRY Command Format

Table 31: INVALIDATE_DEV_TAB_ENTRY Fields

Bits	Description
31:16 +00	Reserved.
15:0 +00	**DeviceID[15:0]**.
31:28 +04	**02h**. INVALIDATE_DEV_TAB_ENTRY command number.
27:20 +04	Reserved.
31:0 +08	Reserved.
31:0 +12	Reserved.

3.3.3 INVALIDATE_IOMMU_PAGES

The INVALIDATE_IOMMU_PAGES command instructs the IOMMU to invalidate a range of entries in its translation cache for the specified DomainID. The size of the invalidate command is determined by the S bit, and the address. The INVALIDATE_IOMMU_PAGES command must appear as a single atomic operation to the translation engine.

Software note: When issuing INVALIDATE_IOMMU_PAGES commands, the size of each invalidate must be greater than or equal to the size of the largest page being invalidated.

Implementation note: IOMMU implementations are not required to provide optimal support for all of the possible invalidation request sizes. The IOMMU is free to invalidate more than just exactly the requested range of addresses, up to and including its entire translation cache if necessary.

Implementation note: When a guest physical address translation is invalidated, the guest virtual address translations that depend on it must also be invalidated. The IOMMU is permitted to invalidate all guest virtual translations for the DomainID when a guest physical address translation is invalidated.

Software note: To invalidate the guest translation information for a single process address space, issue an INVALIDATE_IOMMU_PAGES command with GN=1, PASID and DomainID as needed, PDE=1, S=1, and

Address[63:12]=7_FFFF_FFFF_FFFFh. The IOMMU invalidates all translation information associated with the DomainID for both nested and guest levels when S=1, PDE=1, GN=0, and Address[63:12]=7_FFFF_FFFF_FFFFh.

Software note: Revision 2: When the IOMMU is configured to update Accessed and Dirty bits, software must issue invalidation commands when it resets A or D from 1 to 0.

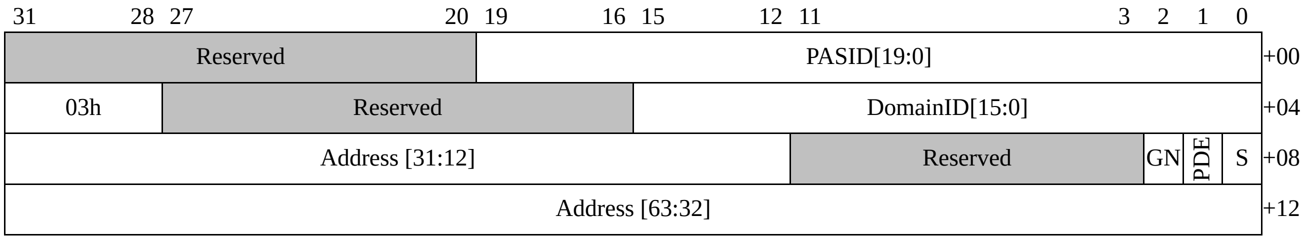


Figure 40: INVALIDATE_IOMMU_PAGES Command Format

Table 32: INVALIDATE_IOMMU_PAGES Fields

Bits	Description
31:20 +00	Reserved.
19:0 +00	**PASID[19:0]**. Revision 1: Must be zero. Revision 2: Must be zero when two-level translation is not enabled (see DTE[GV] in Table 5); ignored when GN=0.
31:28 +04	**03h**. INVALIDATE_IOMMU_PAGES command number.
27:16 +04	Reserved.
15:0 +04	**DomainID[15:0]**.
31:12 +08	**Address[31:12]**. Address to invalidate.
11:3 +08	Reserved.
2 +08	**GN: guest/nested.** Revision 1: Must be zero. Revision 2: 0=Address[52:12] is a GPA so matching nested translations are invalidated and the dependent guest translations must be invalidated if guest address translation is active for any DeviceID within the domain. 1=Address[52:12] is a GVA and matching guest translations are invalidated for the specified PASID. No nested translations are invalidated. **Note**: When two-level translation is not enabled, GN must be zero (see DTE[GV] in Table 5). **Implementation note**: When GN=0, the IOMMU must invalidate the specified guest translations, but it may invalidate more guest translations for the domain, up to and including all guest translations for the domain.
1 +08	**PDE: page directory entries.** 0=only the cached page translation entries are flushed. 1=the cached page directory and page translation entries are flushed. If the range of the INVALIDATE_IOMMU_PAGES command covers all of the pages in a page directory entry and PDE=1, the IOMMU must invalidate the page directory entry in the page directory cache. Revision 2: When GN=1, the dependent guest translation PDEs and PTE must be invalidated and PDE is ignored.

Table 32: INVALIDATE_IOMMU_PAGES Fields

0 +08	**S: size.** 0=the size of the invalidation is 4K bytes. 1=the size of the invalidation is determined by the first zero bit in the address starting from Address[12] (see encoding in Table 11).
31:0 +12	**Address[63:32]**. Address to invalidate.

3.3.4 INVALIDATE_IOTLB_PAGES

The INVALIDATE_IOTLB_PAGES command is only present in IOMMU implementations that support remote IOTLB caching of translations (see Capability Offset 00h[IotlbSup]). This command instructs the specified device to invalidate the given range of addresses in its IOTLB. The size of the invalidate command is determined by the S bit and the address.

Revision 1: The INVALIDATE_IOTLB_PAGES command does not support PASID. Software is required to program GN=0, PASID[15:0]=00h, and Address[63:12] must be a guest physical address.

Revision 2: The INVALIDATE_IOTLB_PAGES command optionally supports PASID and software may program GN=1 (see MMIO Offset 0030h[GTSup]) to specify that Address is a GVA to be translated using PASID.

Software note: The IOMMU does not check the value of DTE[I] before sending the invalidation command to the peripheral.

For more information on the Maxpend and QueueID fields, refer to the PCI *Address Translation Services 1.1 Specification* or newer and the peripheral documentation.

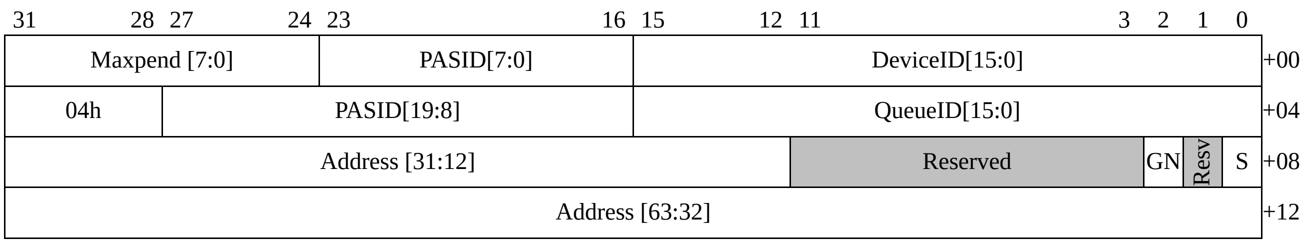


Figure 41: INVALIDATE_IOTLB_PAGES Command Format

Table 33: INVALIDATE_IOTLB_PAGES Fields

Bits	Description
31:24 +00	**Maxpend[7:0]**. The Maxpend field allows software to control the maximum number of simultaneously in-flight INVALIDATE_IOTLB_PAGES transactions that the IOMMU attempts to initiate with any one particular QueueID. The appropriate value for Maxpend is device-dependent and can be obtained from the device's IOTLB capability.
23:16 +00	**PASID[7:0]**. Revision 1: Must be zero. Revision 2: Must be zero when two-level translation is not enabled (see DTE[GV] in Table 5); ignored when GN=0.
15:0 +00	**DeviceID[15:0]**.
31:28 +04	**04h**. INVALIDATE_IOTLB_PAGES command number.

Table 33: INVALIDATE_IOTLB_PAGES Fields

27:16 +04	**PASID[19:8]**. Revision 1: Must be zero. Revision 2: Must be zero when two-level translation is not enabled (see DTE[GV] in Table 5); ignored when GN=0.
15:0 +04	**QueueID[15:0]**. The QueueID is used to limit the outstanding invalidations for all virtual devices sharing the queue for devices that implement multiple virtual functions sharing a single invalidation queue. Some devices implement a physical function and multiple virtual functions in which each physical and virtual function has a unique DeviceID. The QueueID is an abstract number representing the shared queue. When the IOMMU receives an invalidate IOTLB command, the command targets the DeviceID. An implementation may have a single queue (likely associated with the physical function) to receive invalidates for the physical function or any of its virtual functions. To manage the flow control of the unified device invalidate-queue, it is not sufficient to track the outstanding entries based on DeviceID.
31:12 +08	**Address[31:12]**. Address to invalidate.
11:3 +08	Reserved.
2 +08	**GN: guest/nested.** Revision 1: Must be zero. Revision 2: 0=Address[52:12] is a guest physical address and matching nested translations are invalidated, and the corresponding guest translations must be invalidated if guest address translation is active for any DeviceID within the domain. 1=Address[52:12] is a guest virtual address and matching guest translations are invalidated for the specified PASID. No nested translations need to be invalidated. **Note**: When two-level translation is not enabled, GN must be zero (see DTE[GV] in Table 5). **Implementation note**: When GN=0, the IOMMU must invalidate the affected guest translations, but it may invalidate more guest translations for the domain, up to and including all guest translations for the domain.
1 +08	Reserved.
0 +08	**S: size.** 0=the size of the invalidation is 4K bytes. 1=the size of the invalidation is determined by the first zero bit in the address starting from Address[12] (see encoding in Table 11). When S=1, the size of the invalidate is determined by the first zero bit in the address starting from Address[12]. To invalidate the entire contents of an IOTLB, set S=1 and Address[63:32]=7FFF_FFFFh and Address[31:12]=F_FFFFh in the INVALIDATE_IOTLB_PAGES command. When Address[63:32]=FFFF_FFFFh, the IOMMU behavior is undefined.
31:0 +12	**Address[63:32]**. Address to invalidate.

Since both the IOMMU and the remote IOTLB(s) may contain cached translations for a domain, software must take care to perform invalidations in an order that ensures that no stale translations persist anywhere in the system. After updating a domain's page tables, software should first issue an INVALIDATE_IOMMU_PAGES command for the domain; then, if the domain contains any devices with their own IOTLBs, software should follow with INVALIDATE_IOTLB_PAGES commands for each such device.

Revision 2: When GN=0, Address is a guest physical address and PASID[15:0] is ignored by the IOMMU. The GPA is transmitted to the PCIe peripheral without a TLP prefix. When GN=1, Address is a guest virtual address and software programs PASID[15:0] to indicate the process address space to use. The GVA is transmitted to a PCIe peripheral using the TLP prefix.

Implementation note: When issuing the completion notification (Section 3.3.1 [COMPLETION_WAIT]), the IOMMU must ensure that all DMA write transactions that have already been translated have been pushed to the host bridge. A way to meet this is:

- Prior to sending the invalidation completion indication (interrupt or status write) the IOMMU must:
  - Send an upstream Fence command in the base channel if the channel is being used and if the IOMMU supports translating request for more than one upstream stream (more than one unitID is in use).
  - Additionally send an upstream Fence command followed by a Flush command in the isochronous channel if the channel is being used and if the IOMMU supports translating requests in both the isochronous and the base channels. The invalidation completion must wait for the Flush response to be received.

Software Note: In order to flow-control invalidations to functions that share a common invalidation queue, software must set the QueueID to a unique identifier that represents the shared queue. The DeviceID of the physical function associated with the virtual functions may be used as the QueueID to insure the IOMMU issues a limited number of outstanding invalidates to the given queue.

Software Note: To completely tear down address translation for a domain, software should:
- update the IOMMU's in-memory data structures,
- INVALIDATE_DEVTAB_ENTRY for all devices in the domain,
- INVALIDATE_IOMMU_PAGES for the domain, and
- INVALIDATE_IOTLB_PAGES for any IOTLB-capable devices that had been assigned to the domain.

3.3.5 INVALIDATE_INTERRUPT_TABLE

Revision 1: The INVALIDATE_INTERRUPT_TABLE command instructs the IOMMU to invalidate all cached interrupt remapping table entries for the device.

Revision 2: The INVALIDATE_INTERRUPT_TABLE command instructs the IOMMU to invalidate all cached interrupt information for the device, including the guest virtual APIC table base pointer (if cached).

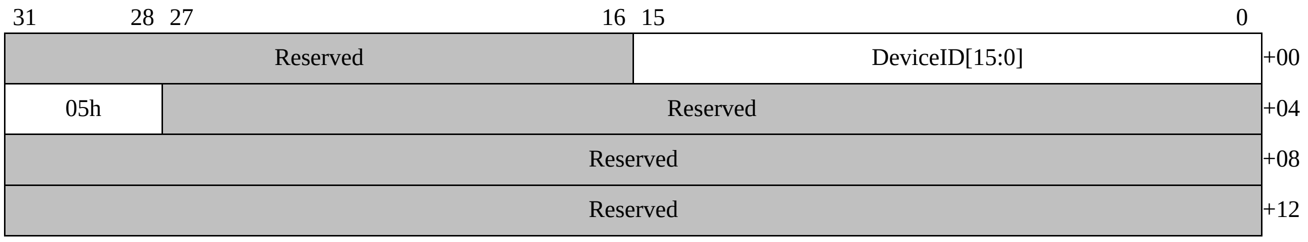


Figure 42: INVALIDATE_INTERRUPT_TABLE Command Format

Table 34: INVALIDATE_INTERRUPT_TABLE command Fields

Bits	Description
31:16 +00	Reserved.
15:0 +00	**DeviceID[15:0]**.
31:28 +04	**05h**. INVALIDATE_INTERRUPT_TABLE command number.

Table 34: INVALIDATE_INTERRUPT_TABLE command Fields

27:20 +04	Reserved.
31:0 +08	Reserved.
31:0 +12	Reserved.

3.3.6 PREFETCH_IOMMU_PAGES

Revision 1: The 06h command is reserved and causes an ILLEGAL_COMMAND_ERROR.

Revision 2: When supported (MMIO Offset 0030h[PreFSup]=1), the PREFETCH_IOMMU_PAGES command instructs the IOMMU to load address translation information into its translation cache for the specified DeviceID. When not supported (MMIO Offset 0030h[PreFSup]=0), the PREFETCH_IOMMU_PAGES (06h) command is reserved and causes an ILLEGAL_COMMAND_ERROR.

The PREFETCH_IOMMU_PAGES command is advisory so the IOMMU may fetch zero or more translation entries in response to the command, not to exceed the value of PFCount[7:0]. An IOMMU treats the PREFETCH_IOMMU_PAGES command as an invalid command when MMIO Offset 0030h[PreFSup]=0. Based on internal status and workloads, the IOMMU may defer fetching the translation information. If an entry is already in the TLB, the IOMMU may adjust LRU or other control tags to lengthen cache residency. The IOMMU calculates permissions for a PREFETCH_IOMMU_PAGES command as it would for a translation that was initiated by device action. Once a translation entry is loaded into the TLB cache by PREFETCH_IOMMU_PAGES, it is subject to ejection and invalidation like any other entry. A PREFETCH_IOMMU_PAGES command must be processed or discarded (ignored) before processing any following invalidation commands that affect the same virtual addresses. The PREFETCH_IOMMU_PAGES command does not affect the contents of remote IOTLB caches.

When GN=0, Address[63:12] is an GPA so the IOMMU walks nested page tables and PASID is ignored. When GN=1, Address[63:12] is a GVA so the IOMMU walks guest and nested page tables and PASID is used to select the guest table.

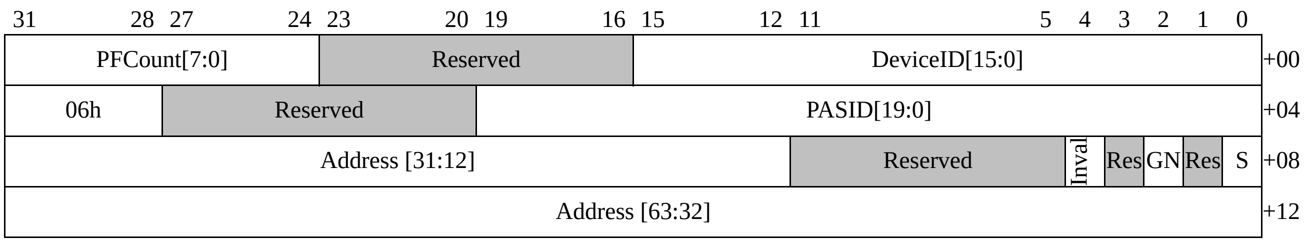


Figure 43: PREFETCH_IOMMU_PAGES Command Format

Table 35: PREFETCH_IOMMU_PAGES Fields

Bits	Description
31:24 +00	**PFCount[7:0]: prefetch count**. Number of translations to prefetch. Zero is treated as a directive to fetch a single translation.

Table 35: PREFETCH_IOMMU_PAGES Fields

23:16 +00	Reserved.
15:0 +00	**DeviceID[15:0]**.
31:28 +04	**06h**. PREFETCH_IOMMU_PAGES command number.
27:20 +04	Reserved.
19:0 +04	**PASID[19:0]**. Must be zero when two-level translation is not enabled (see DTE[GV] in Table 5). Ignored when GN=0.
31:12 +08	**Address[31:12]**. Fetch the address translation information for addresses starting with this value. This is a guest virtual address when GN=1 and is a guest physical address when GN=0.
11:5 +08	Reserved.
4 +08	**Inval: invalidate first**. 0=Prefetch only. 1=Invalidate any matching entry, then prefetch. **Implementation note**: the IOMMU may ignore the prefetch portion of the operation but the invalidation is mandatory if Inval=1.
3 +08	Reserved.
2 +08	**GN: guest/nested.** 0=Address[52:12] is a GPA to be processed through nested translations. 1=reserved (must not be used by software; ignored by hardware).
1 +08	Reserved.
0 +08	**S: size.** 0=the size of the prefetched page is 4K bytes. 1=the size of the prefetched page is determined by the first zero bit in the address starting from Address[12].
31:0 +12	**Address[63:32]**. Address to invalidate.

When Inval=1, the IOMMU must invalidate or replace any existing translation entries in its cache related to Address[63:12]; in effect, this combines an INVALIDATE_IOMMU_PAGES command with a following PREFETCH_IOMMU_PAGES command. When Repl=1, the IOMMU is advised to replace an existing translation entry in the case of a cache conflict.

For a GVA, the PREFETCH_IOMMU_PAGES command is not supported and GN must be programmed to 0. .

For a GPA, software must program GN=0 and the PASID is ignored; the Address field contains a GPA. The size of the prefetch page is determined by the S bit and the Address. If S=0, the size of the prefetch page is 4K bytes. If S=1, the size of the prefetch page is determined by the first zero bit in the address starting from Address[12]. The number of descriptors to fetch is determined by the value of PFCount[7:0]. The PFCount value is unsigned and a PFCount value of 0x00 is treated as 0x01.

3.3.6.1 Event Processing for PREFETCH_IOMMU_PAGES

Revision 1: The 06h command is reserved and causes an ILLEGAL_COMMAND_ERROR.

Revision 2: The IOMMU checks for unusual conditions while processing a PREFETCH_IOMMU_PAGES command. Because such an event would originate as the result of a command in the command queue, neither a Master Abort nor a Target Abort can be caused by a PREFETCH_IOMMU_PAGES command. The PREFETCH_IOMMU_PAGES command never generates an event related to interrupts or interrupt tables.

System hardware events are reported when detected and the IOMMU stops processing the command queue:

- Section 3.4.3 [DEV_TAB_HARDWARE_ERROR Event],
- Section 3.4.4 [PAGE_TAB_HARDWARE_ERROR Event], and
- Section 3.4.6 [COMMAND_HARDWARE_ERROR Event].

These events are reported as soon as detected in the IOMMU hardware event reporting registers (see Section 3.4.11.2 [I/O Hardware Event Reporting Registers]). These events are also reported in the event queue when logging is enabled and an interrupt is signaled when enabled.

Other errors may be reported as soon as possible or postponed until a peripheral access uses the TLB entry:

- Section 3.4.1 [ILLEGAL_DEV_TABLE_ENTRY Event],
- Section 3.4.2 [IO_PAGE_FAULT Event].

An IO_PAGE_FAULT event is not reported when processing an PREFETCH_IOMMU_PAGES command; reporting occurs when the entry is used by a peripheral transaction and is controlled by SA or SE in the corresponding DTE when DTE[V]=1. Table entries marked not-valid or not-present cannot cached by the IOMMU so no error is reported in these cases; the IOMMU will process any error when it rewalks the page tables later to service a peripheral transaction. A PREFETCH_IOMMU_PAGES command does not indicate a read, write, or execute attribute so the IOMMU cannot report an access violation error; the IOMMU will process an access violation when it later services a peripheral transaction.

The following events are never reported as a result of a PREFETCH_IOMMU_PAGES command:

- Section 3.4.7 [IOTLB_INV_TIMEOUT Event],
- Section 3.4.8 [INVALID_DEVICE_REQUEST Event]
- Section 3.4.9 [INVALID_PPR_REQUEST Event], and
- Section 3.4.10 [EVENT_COUNTER_ZERO Event].

If the IOMMU does not prefetch the page table information, a latent problem in the page table structures will not be reported by the IOMMU.

For the purposes of a COMPLETION_WAIT command (see Section 3.3.1 [COMPLETION_WAIT]), the IOMMU may determine that the PREFETCH_IOMMU_PAGES command with Inval=0 completes immediately when the prefetch hint is ignored. A COMPLETION_WAIT command must not signal completion before the completion of a PREFETCH_IOMMU_PAGES command with Inval=1.

Software Note: When issuing PREFETCH_IOMMU_PAGES commands, the size of the prefetch must be greater than or equal to the size of the largest page being prefetched.

3.3.7 COMPLETE_PPR_REQUEST

Revision 1: The 07h command is reserved and causes an ILLEGAL_COMMAND_ERROR.

Revision 2: When supported, the COMPLETE_PPR_REQUEST command is used to instruct the IOMMU to issue a PCIe completion packet for the specified DeviceID with the supplied CompletionTag. When not supported, the COMPLETE_PPR_REQUEST command is reserved and causes an ILLEGAL_COMMAND_ERROR. See MMIO Offset 0030h[PPRSup] to determine if COMPLETE_PPR_REQUEST is supported.

After software has processed a peripheral page service request (see Section 3.5 [Peripheral Page Service Request (PPR) Logging]), it must issue a COMPLETE_PPR_REQUEST command to the IOMMU for the originating DeviceID.

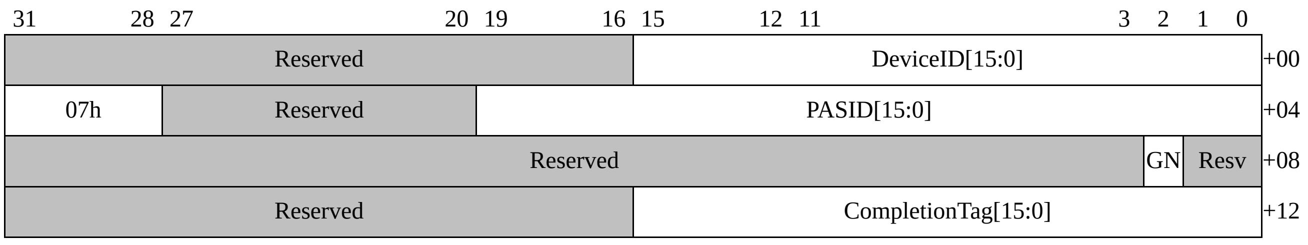


Figure 44: COMPLETE_PPR_REQUEST Command Format

Table 36: COMPLETE_PPR_REQUEST Fields

Bits	Description	
31:16 +00	Reserved.	
15:0 +00	**DeviceID[15:0]**.	
31:28 +04	**07h**. COMPLETE_PPR_REQUEST command number.	
27:20 +04	Reserved.	
19:0 +04	**PASID[19:0]**. Must be zero when two-level translation is not enabled (see DTE[GV] in Table 5); ignored when GN=0.	
31:3 +08	Reserved.	
2 +08	**GN: guest/nested.** 0=PASID is ignored. 1=valid PASID. When two-level translation is not enabled, GN must be zero (see DTE[GV] in Table 5).	
1:0 +08	Reserved.	
31:16 +12	Reserved.	
15:0 +12	**CompletionTag[15:0]**. For PCIe: CompletionTab[15:12] contains the PRI Response Code, CompletionTag[11:9] must be zero, and CompletionTag[8:0] contains the PRI Page Request Group Index taken from the PPRtag in the PAGE_SERVICE_REQUEST request entry (see Section 3.5.1 [Peripheral Page Service Request Entry]). For other bus types: Reserved. **Software note:** The IOMMU does not validate the contents of the CompletionTag field. Software must use the PRI Page Request Group Index from the PPR request.	**CompletionTag[15:0]**. For HyperTransport: Reserved.

For GVA transactions, software should program GN=1, DeviceID, CompletionTag, and PASID. When GN=1, the IOMMU will insert the PASID into the PCIe TLP prefix. For GPA transactions, software should program GN=0, DeviceID, and CompletionTag. When GN=0, the PASID field is ignored by the IOMMU and no PCIe TLP prefix is inserted.

If an error occurs while processing a COMPLETE_PPR_REQUEST command, it is reported in the event log (see Section 3.4.9 [INVALID_PPR_REQUEST Event]).The contents of CompletionTag[15:0] depend on the peripheral bus type; only transactions for the PCIe bus are defined.

3.3.8 INVALIDATE_IOMMU_ALL

Revision 1: The 08h command is reserved and causes an ILLEGAL_COMMAND_ERROR.

Revision 2: When supported, the INVALIDATE_IOMMU_ALL command instructs the IOMMU to clear all address translation and interrupt remapping information from its translation caches for the all DeviceIDs and all Domains. When not supported, the INVALIDATE_IOMMU_ALL command is reserved and causes an ILLEGAL_COMMAND_ERROR. See MMIO Offset 0030h[IASup].

The INVALIDATE_IOMMU_ALL command instructs the IOMMU to invalidate all cached information for interrupt remapping and address translation for guest and nested translations, including cached portions of the device table, the guest CR3 table, page directory entries, page table entries, and interrupt remapping entries. Use of this command generates an ILLEGAL_COMMAND_ERROR event when not supported by the IOMMU (see MMIO Offset 0030h[IASup] and Section 3.4.5 [ILLEGAL_COMMAND_ERROR Event]). The INVALIDATE_IOMMU_ALL command does not affect the contents of any remote IOTLB or IOMMU registers (see Section 3.7 [PCI Resources]) beyond routine command processing updates. Software must issue a INVALIDATE_IOTLB_PAGES command to flush a remote IOTLB. At the completion of an INVALIDATE_IOMMU_ALL command, all IOMMU caches are empty. The results of outstanding page table walks are discarded. Any pending update operations to the page tables for Accessed and Dirty bits must be completed normally. The operational status of the IOMMU is not affected so that translations, command- and event-processing, address translation service, and peripheral page service processing continue normally. The contents of the MMIO registers are not affected except to advance the Command Buffer Head Pointer Register [MMIO Offset 2000h] beyond the INVALIDATE_IOMMU_ALL command. The IOMMU may start reloading internal caches with information at any time after the INVALIDATE_IOMMU_ALL command completes. To invalidate the entire address space of an individual guest, see Section 3.3.3 [INVALIDATE_IOMMU_PAGES].

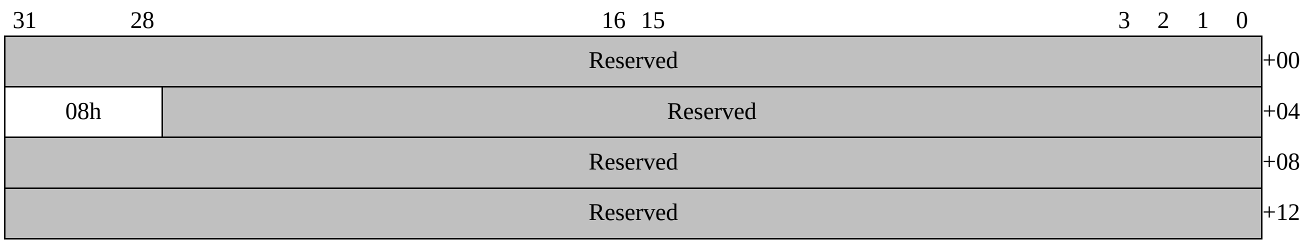


Figure 45: INVALIDATE_IOMMU_ALL Command Format

Table 37: INVALIDATE_IOMMU_ALL Fields

Bits	Description
31:0 +00	Reserved.

Table 37: INVALIDATE_IOMMU_ALL Fields

31:28 +04	**08h**. INVALIDATE_IOMMU_ALL command number.
27:0 +04	Reserved.
31:0 +08	Reserved.
31:0 +12	Reserved.

3.3.9 IOMMU Ordering Rules

The IOMMU must ensure that proper ordering is maintained between invalidation command types and between invalidation commands and the translation process.

3.3.9.1 Invalidation Command Ordering Requirements

The IOMMU must ensure that the following command ordering rules are followed for invalidation commands:
- When an INVALIDATE_IOMMU_PAGES or INVALIDATE_INTERRUPT_TABLE command is received, the IOMMU must ensure that all cache entries associated with any prior INVALIDATE_DEVTAB_ENTRY commands are invalidated from the cache before executing the command.
- When an INVALIDATE_IOTLB_PAGES command is received, the IOMMU must ensure that all cache entries associated with any prior INVALIDATE_DEVTAB_ENTRY or INVALIDATE_IOMMU_PAGES commands are invalidated from the cache before executing the command.

3.3.9.2 Invalidation Commands Interaction Requirements

Invalidation commands are considered completed only when the IOMMU can ensure that there are no DMA transactions in flight anywhere in the system fabric that relied on translation cache contents prior to the invalidation. To ensure that this property is achieved, the IOMMU must follow the following rules:
- The IOMMU must ensure that read responses for all DMA outstanding read transactions that match the invalidation command have been received by the IOMMU.
  - HyperTransport™ tunnels that support address translation can achieve this property by maintaining a counter that is incremented when a non-posted transaction is forwarded to the processor through the tunnel and is decremented when a response is forwarded from the processor through the tunnel. The invalidation command can be considered complete when the counter reaches zero.
    - The tunnel may temporarily block upstream traffic to cause the counter to resolve to zero in a timely manner, ensuring that forward progress of the invalidation command is made.
- The IOMMU must ensure that all DMA write transactions that have already been translated have been pushed to the host bridge by:
  - Prior to sending the invalidation completion indication (interrupt or status write) the IOMMU must:
    - Send an upstream Fence command in the base channel if the IOMMU supports translating requests for more than one upstream stream (more than one unitID is in use).
    - Send an upstream Fence command followed by a Flush command in the isochronous channel if the IOMMU supports translating requests in both the isochronous and the base channels. The invalidation completion must wait for the Flush response to be received.

The IOMMU must ensure that both of these requirements are met prior to executing a subsequent

COMPLETION_WAIT command.

An invalidation command matches an outstanding translation if the command:

- Invalidates the device table entry for the I/O device that caused a translation to be initiated, or
- Invalidates the virtual address range being translated for a device.

3.4 Event Logging

The IOMMU reports events to the host software by means of a shared circular buffer in system memory. The IOMMU writes event records into the buffer. If the IOMMU needs to report an event but finds that the event log is already full, it sets MMIO Offset 2020h[EventOverflow]. The IOMMU can be configured to signal an interrupt whenever the event log is written or overflows using Capability Offset 10h[MsiNum]. The host software increments the IOMMU's head pointer to indicate to the IOMMU that it has consumed event log entries.

Revision 2: The IOMMU generates event log entries for guest translations as well as host translations using Capability Offset 10h[MsiNum]. Because some event reports are caused by hardware events that may make it impossible to update the event log, a set of MMIO registers has been defined to report selected event information. The IOMMU Hardware Event Upper Register [MMIO Offset 0040h] and IOMMU Hardware Event Lower Register [MMIO Offset 0048h] contain the same information found in an event log entry. The IOMMU Hardware Event Status Register [MMIO Offset 0050h] contains status bits that indicate that the information in the three registers is valid (HEV) and if an overflow has occurred (HEO). More information is contained in Section 3.4.11 [IOMMU Event Reporting].

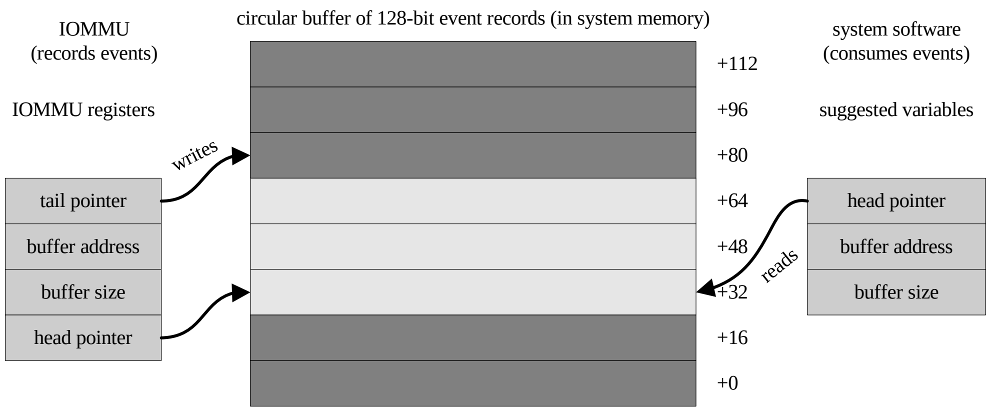


Figure 46: Circular Event Log in System Memory

The Event Log Base Address Register [MMIO Offset 0010h] is used to program the system physical address and size of the event log. The event log occupies contiguous physical memory starting at the programmed base address, up to the programmed size. The size of the event log must be a multiple of 4K bytes (to facilitate "mod N" indexing for circularity), and can be as large as 32768 entries (corresponding to a 512 kilobyte buffer). The address of the event log must be aligned to a multiple of 4K bytes.

In addition to the Event Log Base Address Register [MMIO Offset 0010h], the IOMMU maintains two other registers associated with the event log: the Event Log Head Pointer Register [MMIO Offset 2010h] which points to the next event that software will read, and the Event Log Tail Pointer Register [MMIO Offset 2018h] which points to the next event to be written by the IOMMU. These registers are located in MMIO space. When the Event Log Base Address Register [MMIO Offset 0010h] register is written, the Event Log Head Pointer Register [MMIO Offset 2010h] and the Event Log Tail Pointer Register [MMIO Offset 2018h] are cleared to 0.

When the Event Log Head Pointer Register [MMIO Offset 2010h] and the Event Log Tail Pointer Register [MMIO Offset 2018h] are equal, the event log is empty. The Event Log Tail Pointer Register [MMIO Offset 2018h] is incremented by the IOMMU after writing an event to the event log. The event log is full when all slots but one are used. The event log has overflowed when an event occurs that is to be logged and would otherwise consume the last unused slot. When the event log has overflowed, the EventOverflow bit is set in the IOMMU Status Register [MMIO Offset 2020h] and any data for new events is discarded. An interrupt can be configured to notify software of the new event using Capability Offset 10h[MsiNum]; software should check MMIO Offset 2020h[EventOverflow] to determine if the event log data was discarded. The host software must make space in the event log after an overflow by reading entries (by adjusting the head pointer) or by resizing the log, and event logging may then be restarted.

The IOMMU event logging is disabled after system reset and when the event log overflows. The IOMMU discards event reports until event logging is enabled, setting the EventOverflow bit in the IOMMU Status Register [MMIO Offset 2020h] to indicate the loss of event information. To restart the IOMMU event logging after the event log overflows, use the following procedure.
- Wait until EventLogRun=0b in the IOMMU Status Register [MMIO Offset 2020h] so that all log entries are completed as circumstances allow. EventLogRun must be 0b to modify the event log registers safely.
- Write EventLogEn=0b in the IOMMU Control Register [MMIO Offset 0018h].
- As necessary, change the following registers (e.g., to relocate or resize the event log).
    - the Event Log Base Address Register [MMIO Offset 0010h],
    - the Event Log Head Pointer Register [MMIO Offset 2010h], and
    - the Event Log Tail Pointer Register [MMIO Offset 2018h].
- Write the IOMMU Status Register [MMIO Offset 2020h] with EventOverflow=0b to clear the bit.
- Write the IOMMU Control Register [MMIO Offset 0018h] with EventLogEn=1b and EventIntEn as desired.

The IOMMU now creates event log entries for new events.

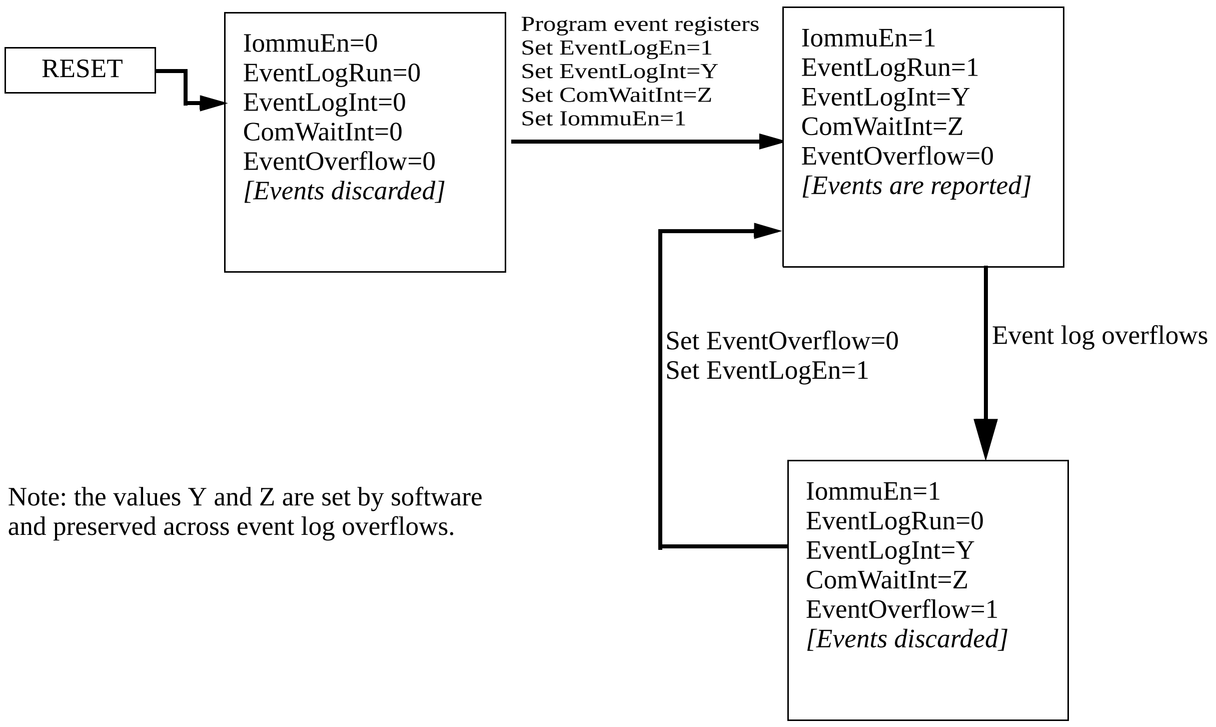


Figure 47: Event log state diagram

All events recorded by the IOMMU consist of a 4-bit EventCode together with two operands, which may be respectively 60 and 64 bits long, for a total of 128 bits (16 bytes) per record. Events that are logged because of errors that occur while performing device table or page table walks always record the DeviceID and address from the transaction that was being translated.

The IOMMU must set the Coherent bit in the HyperTransport™ packet when generating writes to the event log.

31 … 28	27 … 0	
First event code dependent operand [31:0]		+00
EventCode[3:0]	First event code dependent operand [59:32]	+04
Second event code dependent operand [31:0]		+08
Second event code dependent operand [63:32]		+12

Figure 48: Generic Event Log Buffer Entry

Events reported by the IOMMU are listed in Figure 38. The figures that follow give details for each event type with an EventCode as shown embedded in Figure 48.

Table 38: Event Type Summary

EventCode Name	Value	General Error Type	Details
Reserved	0000b	Reserved.	N/A
ILLEGAL_DEV_TABLE_ENTRY	0001b	Non-zero reserved bit or reserved encoding in DTE.	Table 39
IO_PAGE_FAULT (memory transaction or interrupt remapping)	0010b	DeviceID not in the range specified by the device table size.	Table 40
IO_PAGE_FAULT (memory transaction)		PTE programming problems.	
		Virtual address problems.	
		Device attempts to violate page protection settings.	
IO_PAGE_FAULT (interrupt remapping)		IRTE programming problems.	
		Disallowed or malformed interrupt requests.	
DEV_TAB_HARDWARE_ERROR	0011b	Hardware problem as IOMMU reads device table.	Table 41
PAGE_TAB_HARDWARE_ERROR	0100b	Hardware problem as IOMMU accesses page table.	Table 42
ILLEGAL_COMMAND_ERROR	0101b	Invalid command buffer entry.	Table 44
COMMAND_HARDWARE_ERROR	0110b	Hardware problem as IOMMU reads command buffer.	Table 43
IOTLB_INV_TIMEOUT	0111b	Invalidation response not received from IOTLB device.	Table 45
INVALID_DEVICE_REQUEST	1000b	Device attempts access to proscribed address range.	Table 46, Table 47, Table 58
		Device attempts prohibited access.	

Table 38: Event Type Summary

EventCode Name	Value	General Error Type	Details
INVALID_PPR_REQUEST	1001b	Invalid or malformed PRI request from peripheral.	Table 48
		Invalid or malformed COMPLETE_PPR_REQUEST command. **Note**: the COMPLETE_PPR_REQUEST command may be treated as an ILLEGAL_COMMAND_ERROR (see Table 48 for details).	
EVENT_COUNTER_ZERO	1001b	Informational.	Table 61
Reserved	1010b-1111b	Reserved; not used.	N/A

In Table 39 through Table 49, each event type is marked to show that it can be caused only by host software or by hardware.

For details on ILLEGAL_DEV_TABLE_ENTRY events, see Section 3.4.1.

Table 39: ILLEGAL_DEV_TABLE_ENTRY Event Types

Event Type	Cause	IOMMU Response
Non-zero reserved bit in a device table entry.	SW	For a translation request or memory access, Target Abort transaction. For a command, abort the command. Create event log entry if enabled. Signal interrupt if enabled (see Capability Offset 10h[MsiNum]).
Reserved encoding in the IntTabLen field for a device table entry with IntCtl=10b.	SW	
Reserved encoding in the IoCtl field.	SW	
Reserved encoding in the IntCtl field.	SW	

For details on IO_PAGE_FAULT events, see Section 3.4.2.

Table 40: IO_PAGE_FAULT Event Types

Event Type	Cause	IOMMU Response
Memory transaction		

Table 40: IO_PAGE_FAULT Event Types

Event Type	Cause	IOMMU Response
Reserved paging mode in device table entry.	SW	For an untranslated request, Target Abort transaction, and create event log entry if enabled. Signal interrupt if enabled (see Capability Offset 10h[MsiNum]). For a translation request, return response with data and with R and W bits set to 0; no event log entry is created. For a command, abort the command, and create log entry if enabled. Signal interrupt if enabled (see Capability Offset 10h[MsiNum]).
Page size encoding in a PTE that is smaller than the default page size of the PTE.	SW	
Page size encoding in a PTE that is larger than the default page size of the PTE.	SW	
Invalid level encoding in a page table entry, including exceeding limit specified by MMIO Offset 0030h[GATS, HATS].	SW	
A non-zero bit in a bit position higher than root page table's level in DTE. Virtual address bits associated with a skipped page level are not all zero.	SW	
Non-zero reserved bit in a PTE.	SW	
Valid bit not set in page table entry.	HW, SW	
TV bit not set in device table entry for untranslated non-interrupt transaction.	HW, SW	
PASID is outside the range specified by MMIO Offset 0030h[PASmax].	HW, SW	
Device attempts a read transaction to a read protected page.	HW, SW	For untranslated request, Target Abort transaction, and create event log entry if enabled. Signal interrupt if enabled (see Capability Offset 10h[MsiNum]). For translation request, return response with data and with R and W bits from the page translation information; no event log entry is created. Never generated by a command.
Device attempts a write transaction to a write protected page.	HW, SW	
Device attempts an instruction fetch from a no-execute page.	HW, SW	
Interrupt remapping		
Interrupt request that addresses an IRTE that is beyond the end of the table.	HW, SW	For interrupt transaction, Target Abort transaction. Create event log entry if enabled. Signal interrupt if enabled (see Capability Offset 10h[MsiNum]). Never generated by a command.
Non-zero reserved bit in an IRTE.	SW	
Interrupt request that targets an IRTE with RemapEn=0.	HW, SW	
Interrupt request that targets an IRTE with reserved IntType.	SW	
Interrupt request aborted by entry in Table 8 (Pass fields) or Table 16 (entries causing target abort).	HW, SW	
Interrupt transaction with PASID.	HW	

Table 40: IO_PAGE_FAULT Event Types

Event Type	Cause	IOMMU Response
Memory transaction or interrupt remapping		
DeviceID not in the range specified by the device table size.	HW	For a translation request, Master Abort transaction. For a memory transaction, Target Abort transaction. For a command, abort the command. Create event log entry if enabled. Signal interrupt if enabled (see Capability Offset 10h[MsiNum]).

For details on DEV_TAB_HARDWARE_ERROR events, see Section 3.4.3.

Table 41: DEV_TAB_HARDWARE_ERROR Event Types

Event Type	Cause	IOMMU Response
Master abort received on device table read.	HW	For memory access or translation request, Target Abort transaction. For command, abort the command. Create event log entry if enabled. Log event information in the hardware event registers (see Section 3.4.11.2 [I/O Hardware Event Reporting Registers]). Signal interrupt if enabled (see Capability Offset 10h[MsiNum]).
Target abort received on device table read.	HW	
Poisoned data received on device table read.	HW	

For details on PAGE_TAB_HARDWARE_ERROR events, see Section 3.4.4.

Table 42: PAGE_TAB_HARDWARE_ERROR Event Types

Event Type	Cause	IOMMU Response
Master abort received on page table access.	HW	For memory access or translation request, Target Abort transaction. For command, abort the command. Create event log entry if enabled. Log event information in the hardware error registers (see Section 3.4.11.2 [I/O Hardware Event Reporting Registers]). Signal interrupt if enabled (see Capability Offset 10h[MsiNum]).
Target abort received on page table access.	HW	
Poisoned data received on page table access.	HW	

For details on COMMAND_HARDWARE_ERROR events, see Section 3.4.6.

Table 43: COMMAND_HARDWARE_ERROR Event Types

Event Type	Cause	IOMMU Response
Master abort received on command buffer read.	HW	Halt command processing. Create event log entry if enabled. Log event information in the hardware event registers (see Section 3.4.11.2 [I/O Hardware Event Reporting Registers]). Signal interrupt if enabled (see Capability Offset 10h[MsiNum]).
Target abort received on command buffer read.	HW	
Poisoned data received on command buffer read.	HW	

For details on ILLEGAL_COMMAND_ERROR events, see Section 3.4.5.

Table 44: ILLEGAL_COMMAND_ERROR Event Types

Event Type	Cause	IOMMU Response
Non-zero reserved bit in a command buffer entry.	HW, SW	Halt command processing. Create event log entry if enabled. Signal interrupt if enabled (see Capability Offset 10h[MsiNum]).
Unsupported command code in a command buffer entry. **Note**: COMPLETE_PPR_REQUEST is treated as an ILLEGAL_COMMAND_ERROR if not supported (see MMIO Offset 0030h[PPRSup] and Table 48).	SW	
IOMMU receives INVALIDATE_IOTLB_PAGES and does not support IOTLB commands (see Capability Offset 00h[IotlbSup]).	SW	

For details on IOTLB_INV_TIMEOUT events, see Section 3.4.7.

Table 45: IOTLB_INV_TIMEOUT Event Types

Event Type	Cause	IOMMU Response
Invalidation response not received from IOTLB device.	HW	Create event log entry if enabled. Signal interrupt if enabled (see Capability Offset 10h[MsiNum]). Never generated by a command.

For details on INVALID_DEVICE_REQUEST events, see Section 3.4.8 and Table 58.

Table 46: INVALID_DEVICE_REQUEST Event Types (Access)

Event Type	Cause	IOMMU Response
Read request or non-posted write in the interrupt address range.	HW	Target Abort transaction. Create event log entry if enabled. Signal interrupt if enabled (see Capability Offset 10h[MsiNum]). Never generated by a command.
Pretranslated transaction received from an I/O device with I=0 or V=0.	HW	
Port I/O Space request from an I/O device with IoCtl=00b.	HW	
Posted write to the system management address space from an I/O device with SysMgt=00b, or with SysMgt=10b and the message is not an INTx message, or a posted write to the address translation range when Capability Offset 10h[HtAtsResv]=1 (see Table 2).	HW, SW	
Read request or non-posted write in the system management address range (if SysMgt != 11b), or a read or a non-posted-write in the address translation range when Capability Offset 10h[HtAtsResv]=1 (see Table 2). Revision 2: Also, a request with a PASID TLP prefix when Revision 2 features are not available or active for the DeviceID (see Table 3).	HW, SW	
Posted write to the Interrupt/EOI interrupt address range from an I/O device with IntCtl=00.	HW, SW	
Posted write to a reserved interrupt address range (see Table 2).	HW, SW	
Access to the system management address range when SysMgt=11b or to the port I/O space range when IoCtl=10b, while V=1 and TV=0.	HW, SW	

For details on INVALID_DEVICE_REQUEST events, see Section 3.4.8 and Table 58.

Table 47: INVALID_DEVICE_REQUEST Event Types (Translation Request)

Event Type	Cause	IOMMU Response
Translation request in the interrupt space, port I/O space (if IoCtl=0xb), or system management address range (if SysMgt != 11b).	HW	Target Abort transaction. Create event log entry if enabled. Signal interrupt if enabled (see Capability Offset 10h[MsiNum]). Never generated by a command.
Translation request in the system management address range when SysMgt=11b or in the port I/O space range when IoCtl=10b, while V=1 and TV=0.	HW	

Table 47: INVALID_DEVICE_REQUEST Event Types (Translation Request)

Event Type	Cause	IOMMU Response
Translation request with I=0 or V=0.	HW	Master Abort transaction. Create event log entry if enabled. Signal interrupt if enabled (see Capability Offset 10h[MsiNum]). Never generated by a command.
DeviceID outside range.		

For details on INVALID_PPR_REQUEST events, see Section 3.4.9.

Table 48: INVALID_PPR_REQUEST Event Summary

Event Type	Cause	IOMMU Response
PRI request received when Capability Offset 00h[EFRSup]=0 or MMIO Offset 0018h[PPREn]=0 or MMIO Offset 0018h[PPRLogEn]=0.	HW	Target Abort PRI transaction. Create event log entry if enabled. Signal an interrupt if enabled (see Capability Offset 10h[MsiNum]). Never generated by a command.
COMPLETE_PPR_REQUEST command received with GN=1 when guest translation is not enabled.	SW	Create event log entry if enabled. Signal an interrupt if enabled (see Capability Offset 10h[MsiNum]).
Note: COMPLETE_PPR_REQUEST is treated as an ILLEGAL_COMMAND_ERROR if PPR is not enabled (Capability Offset 00h[EFRSup]=0 or MMIO Offset 0018h[PPREn]=0 or MMIO Offset 0018h[PPRLogEn]=0). See also Table 44.		

For details on EVENT_COUNTER_ZERO events, see Section 3.4.10.

Table 49: EVENT_COUNTER_ZERO Event Types

Event Type	Cause	IOMMU Response
Informational; performance counter incremented to equal zero.	SW	Create event log entry if enabled. Signal an interrupt if enabled (see Capability Offset 10h[MsiNum]).

3.4.1 ILLEGAL_DEV_TABLE_ENTRY Event

When the IOMMU performs a lookup in the device table and encounters a device table entry that it does not support or that is formatted incorrectly, the IOMMU writes an ILLEGAL_DEV_TABLE_ENTRY event to the event log as listed in Table 39.

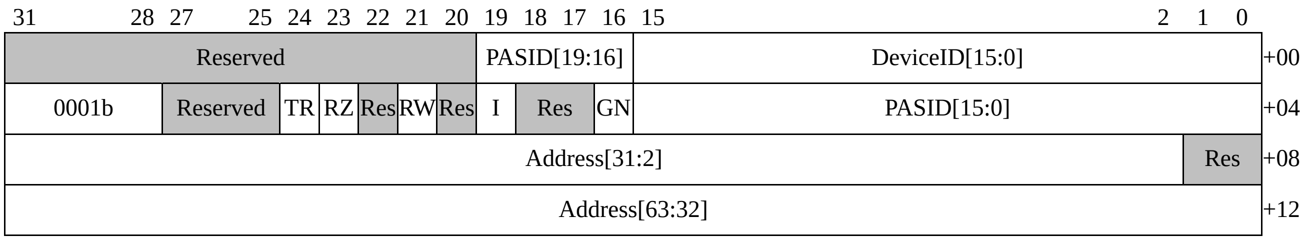


Figure 49: ILLEGAL_DEV_TABLE_ENTRY Event Log Buffer Entry Format

Table 50: ILLEGAL_DEV_TABLE_ENTRY Event Log Buffer Entry Fields

Bits	Description
31:20 +00	Reserved.
31:20 +00	**PASID[19:16]: process space ID**. Revision 1: Reserved. Revision 2: The guest PASID[19:16] from the transaction when GN=1; 0h when GN=0.
15:0 +00	**DeviceID**. Specifies the DeviceID that caused the device table lookup. The address of the malformed device table entry can be determined using the DeviceID field.
31:28 +04	**0001b**. Specifies an ILLEGAL_DEV_TABLE_ENTRY.
27:25 +04	Reserved.
24 +04	**TR: translation**. 1=transaction that caused the device table lookup was a translation request. 0=transaction that caused the device table lookup was a transaction request.
23 +04	**RZ: reserved bit not zero or invalid level encoding.** 1=I/O page fault was caused by a non-zero reserved bit in the device table entry. 0=I/O page fault was caused by an invalid level encoding in the device table entry.
22 +04	Reserved.
21 +04	**RW: read-write.** 1=transaction that caused the device table lookup was a write. 0=transaction that caused the device table lookup was a read. RW is only meaningful when TR=0 and I=0.
20 +04	Reserved.
19 +04	**I: interrupt.** 1=transaction that caused the device table lookup was an interrupt request. 0=transaction that caused the device table lookup was a memory request.
18:17 +04	Reserved.
16 +04	**GN: guest/nested**. Revision 1: Reserved. Revision 2: 0=Transaction contained a GPA. 1=Transaction contained a GVA. See also PASID.
15:0 +04	**PASID[15:0]: process space ID**. Revision 1: Reserved. Revision 2: The guest PASID[15:0] from the transaction when GN=1; 0000h when GN=0.
31:2 +08	**Address[31:2]**. The Address field contains the device virtual address that the device was attempting to access.
1:0 +08	Reserved.
31:0 +12	**Address[63:32]**. The Address field contains the device virtual address that the device was attempting to access.

3.4.2 IO_PAGE_FAULT Event

When the IOMMU performs a lookup in the page tables for a device and encounters an error condition in Table 40, the IOMMU writes the event log with an IO_PAGE_FAULT event as controlled by the SA, SE, IG, and SupIOPF bits (see Figure 7 and Table 5 or Figure 15 and Table 8).

I/O page faults detected for translation requests return a translation-not-present response (R=W=0) to the device and are not logged in the event log.

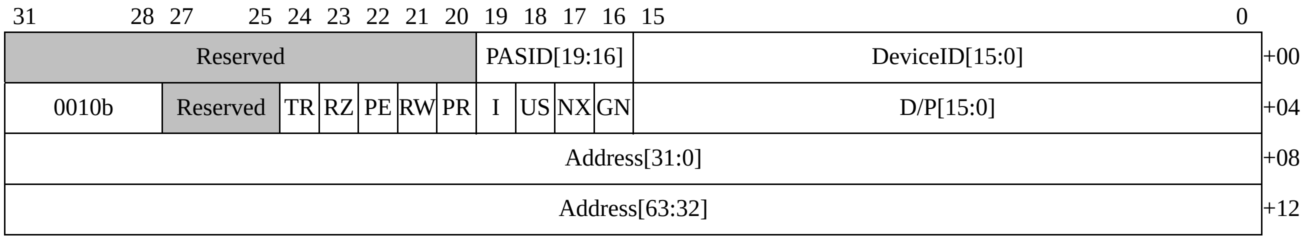


Figure 50: IO_PAGE_FAULT Event Log Buffer Entry Format

Table 51: IO_PAGE_FAULT Event Log Buffer Entry Fields

Bits	Description
31:20 +00	Reserved.
31:20 +00	**PASID[19:16]**. Revision 1: Reserved. Revision 2: The guest PASID[19:16] from the PASID TLP prefix when GN=1; 0h when GN=0. See also D/P below.
15:0 +00	**DeviceID**. Specifies the DeviceID that caused the device table lookup. The address of the device table entry can be determined using the DeviceID field.
31:28 +04	**0010b**. Specifies an IO_PAGE_FAULT entry.
27:25 +04	Reserved.
24 +04	**TR: translation**. 1=transaction that caused the device table lookup was a translation request. 0=transaction that caused the device table lookup was a transaction request.
23 +04	**RZ: reserved bit not zero or invalid level encoding.** 1=I/O page fault was caused by a non-zero reserved bit in the entry. 0=I/O page fault was caused by an invalid level encoding. RZ is only meaningful when PR=1.
22 +04	**PE: permission indicator.** 1=peripheral did not have the permissions required to perform the transaction. 0=peripheral had the necessary permissions. PE is only meaningful when PR=1. Report PE using cumulative read and write permissions as determined during the page walk as accomplished.
21 +04	**RW: read-write.** 1=transaction was a write. 0=transaction was a read. RW is only meaningful when PR=1, TR=0, and I=0.
20 +04	**PR: present.** 1=transaction was to a page marked as present (including V=1b in DTE) or interrupt marked as remapped (RemapEn=1). 0=transaction was to a page marked not present or interrupt marked as blocked (RemapEn=0).
19 +04	**I: interrupt.** 1=transaction was an interrupt request. 0=transaction was a memory request.
18 +04	**US: user-supervisor**. Revision 1: Reserved. Revision 2: 0=Supervisor privileges were asserted. 1=User privileges were asserted.
17 +04	**NX: no execute**. Revision 1: Reserved. Revision 2: NX bit as requested by peripheral when the upstream transaction has a PASID TLP prefix; 0 when the upstream transaction lacks a PASID TLP prefix.
16 +04	**GN: guest/nested**. Revision 1: Reserved. Revision 2: 0=Transaction used a nested address (GPA). 1=Transaction used a guest address (GVA). See also PASID.

Table 51: IO_PAGE_FAULT Event Log Buffer Entry Fields

15:0 +04	**D/P[15:0]: DomainID/PASID[15:0]**. Revision 1: The DomainID from the Device Table Entry. For error conditions that lack a valid DomainID, the reported DomainID is zero. Revision 2: The guest PASID[15:0] from the PASID TLP prefix when GN=1 (see also PASID[19:0] above); the DomainID from the DTE when GN=0. For error conditions that lack a valid PASID or DomainID, the reported value is zero.
31:0 +08	**Address[31:0]**. The Address field contains the device virtual address that the peripheral was attempting to access.
31:0 +12	**Address[63:32]**. The Address field contains the device virtual address that the peripheral was attempting to access.

An interrupt transaction that attempts to use a PASID is not allowed and the event is logged with I=1, GN=1, and the D/P and PASID[19:16] fields contain the PASID when event logging is enabled.

3.4.3 DEV_TAB_HARDWARE_ERROR Event

If the IOMMU triggers a hardware error (master abort, target abort, poisoned data, etc.) while accessing the device table, the IOMMU writes the event log with a DEV_TAB_HARDWARE_ERROR event (see Table 41). In this case the Address field does *not* contain the device virtual address the device was attempting to access, but instead contains the system physical address of the failed device table access. Revision 2: Information on the hardware error registers is contained in Section 3.4.11.2 [I/O Hardware Event Reporting Registers].

Table 52: Event Log Type Field Encodings

Type	Description
00b	Reserved
01b	Master Abort
10b	Target Abort
11b	Data Error

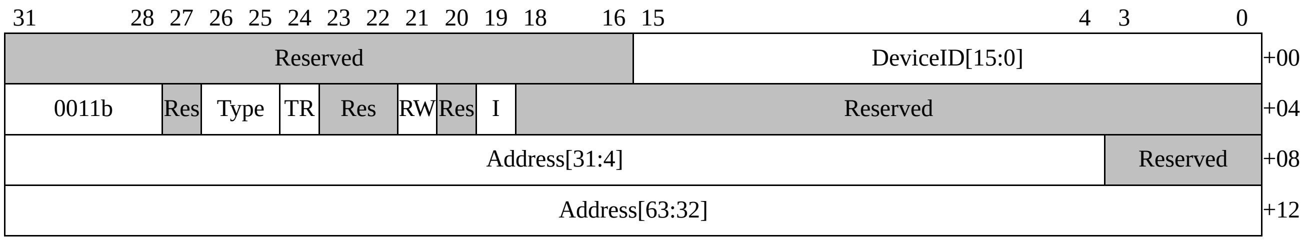


Figure 51: DEV_TAB_HARDWARE_ERROR Event Log Buffer Entry Format

Table 53: DEV_TAB_HARDWARE_ERROR Event Log Buffer Entry Fields

Bits	Description
31:16 +00	Reserved.
15:0 +00	**DeviceID**. Specifies the DeviceID that caused the device table lookup. The address of the device table entry can be determined using the DeviceID field.
31:28 +04	**0011b**. Specifies a DEV_TAB_HARDWARE_ERROR entry.

Table 53: DEV_TAB_HARDWARE_ERROR Event Log Buffer Entry Fields

27 +04	Reserved.
26:25 +04	**Type.** The Type field indicates the type of hardware error that occurred as listed in Table 52.
24 +04	**TR: translation**. 1=transaction that caused the device table lookup was a translation request. 0=transaction that caused the device table lookup was a transaction request.
23:22 +04	Reserved.
21 +04	**RW: read-write.** 1=transaction was a write. 0=transaction was a read. RW is only meaningful when TR=0 and I=0.
20 +04	Reserved.
19 +04	**I: interrupt.** 1=transaction was an interrupt request. 0=transaction was a memory request.
18:0 +04	Reserved.
31:4 +08	**Address[31:4]**. The system physical address of the failed device table access. In this case the Address field does *not* contain the device virtual address the device was attempting to access.
3:0 +08	Reserved.
31:0 +12	**Address[63:32]**. The system physical address of the failed device table access. In this case the Address field does *not* contain the device virtual address the device was attempting to access.

3.4.4 PAGE_TAB_HARDWARE_ERROR Event

If the IOMMU detects a hardware error (master abort, target abort, poisoned data, etc.) while accessing the I/O page tables, the IOMMU writes the event log with a PAGE_TAB_HARDWARE_ERROR event (see Table 42). Revision 2: If the IOMMU detects a hardware error while accessing the guest CR3 table, the guest page tables, or the I/O page tables, the IOMMU writes the event log with a PAGE_TAB_HARDWARE_ERROR event (see Table 42). Data describing the PAGE_TAB_HARDWARE_ERROR event is also written to the hardware error registers (see in Section 3.4.11.2 [I/O Hardware Event Reporting Registers]).

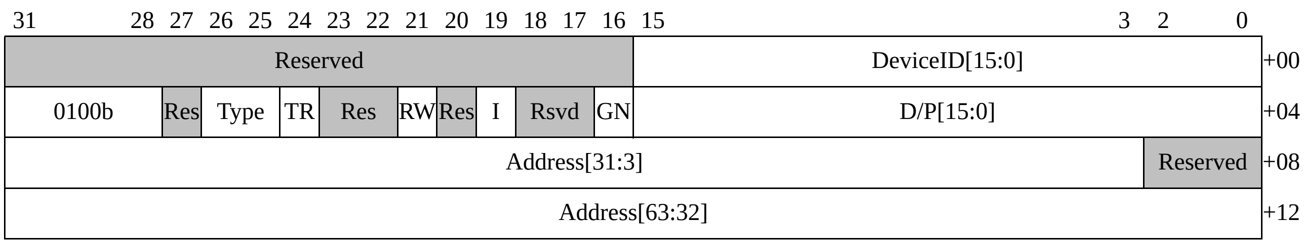


Figure 52: PAGE_TAB_HARDWARE_ERROR Event Log Buffer Entry Format

Table 54: PAGE_TAB_HARDWARE_ERROR Event Log Buffer Entry Fields

Bits	Description
31:16 +00	Reserved.
15:0 +00	**DeviceID**. Specifies the DeviceID that caused the page table lookup. The address of the device table entry can be determined using the DeviceID field.
31:28 +04	**0100b**. Specifies a PAGE_TAB_HARDWARE_ERROR entry.
27 +04	Reserved.
26:25 +04	**Type.** The Type field indicates the type of hardware error that occurred as listed in Table 52.
24 +04	**TR: translation**. 1=transaction that caused the page table lookup was a translation request. 0=transaction that caused the page table lookup was an untranslated request.
23:22 +04	Reserved.
21 +04	**RW: read-write.** 1=transaction was a write. 0=transaction was a read. RW is only meaningful when TR=0 and I=0.
20 +04	Reserved.
19 +04	**I: interrupt.** 1=transaction was an interrupt request. 0=transaction was a memory request.
18:17 +04	Reserved.
16 +04	**GN: guest/nested**. Revision 1: Reserved. Revision 2: 0=The address being translated by the IOMMU is an SPA. 1=The address being translated by the IOMMU is a GPA. Must be zero when MMIO Offset 0030h[GTSup]=0. **Software note:** when GN=1, the error could have been encountered in either the guest CR3 table or in the guest page tables.
15:0 +04	**D/P: DomainID/PASID.** Revision 1: The DomainID of the peripheral that caused the page table lookup. Revision 2: Contains the PASID when GN=1; the DomainID when GN=0.
31:4 +08	**Address[31:4]**. The address of the page table entry.
3:0 +08	Reserved.
31:0 +12	**Address[63:32]**. The SPA of the page table entry. The original address space used by the peripheral is indicated by DeviceID, GN, and PASID. The Address field does *not* contain the address that the device attempted to access.

Software note, Revision 2: When GN=1, the problem may be in the GCR3 table or in the guest page tables.

3.4.5 ILLEGAL_COMMAND_ERROR Event

If the IOMMU reads an invalid command (including an unsupported command code, or a command that incorrectly has reserved bits set), the IOMMU writes the event log with an ILLEGAL_COMMAND_ERROR event (see Table 44). The IOMMU must stop fetching new commands from the command buffer if an ILLEGAL_COMMAND_ERROR event is detected.

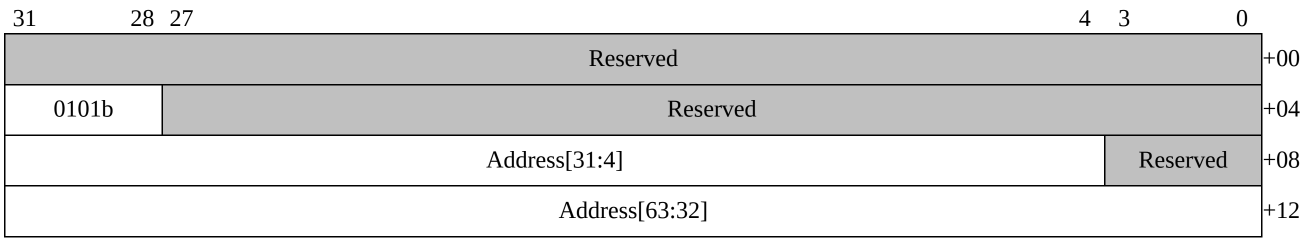


Figure 53: ILLEGAL_COMMAND_ERROR Event Log Buffer Entry Format

Table 55: ILLEGAL_COMMAND_ERROR Event Log Buffer Entry Fields

Bits	Description
31:0 +00	Reserved.
31:28 +04	**0101b**. Specifies an ILLEGAL_COMMAND_ERROR entry.
27:0 +04	Reserved.
31:4 +08	**Address[31:4]**. The system physical address of the invalid command.
3:0 +08	Reserved.
31:0 +12	**Address[63:32]**. The system physical address of the invalid command.

3.4.6 COMMAND_HARDWARE_ERROR Event

If the IOMMU detects a hardware error (master abort, target abort, poisoned data, etc.) while accessing the command buffer, the IOMMU writes the event log with a COMMAND_HARDWARE_ERROR event (see Table 43). The IOMMU must stop fetching new commands from the command buffer if a COMMAND_HARDWARE_ERROR event is detected. Revision 2: Data describing the PAGE_TAB_HARDWARE_ERROR event is also written to the hardware error registers (see Section 3.4.11.2 [I/O Hardware Event Reporting Registers]).

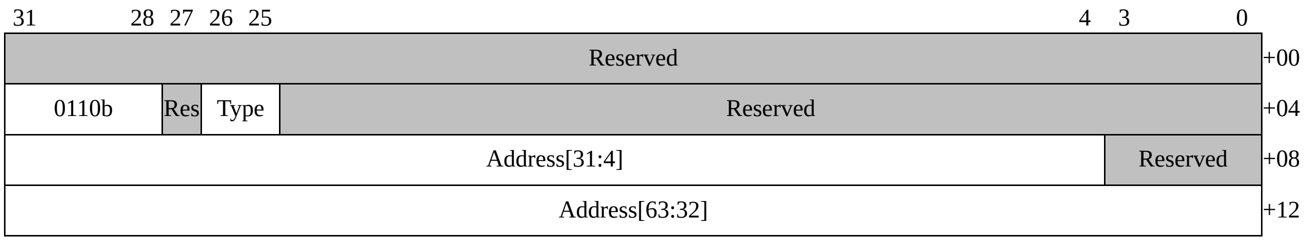


Figure 54: COMMAND_HARDWARE_ERROR Event Log Buffer Entry Format

Table 56: COMMAND_HARDWARE_ERROR Event Log Buffer Entry Fields

Bits	Description
31:0 +00	Reserved.
31:28 +04	**0110b**. Specifies a COMMAND_HARDWARE_ERROR entry.
27 +04	Reserved.
26:25 +04	**Type.** The Type field indicates the type of hardware event that occurred as listed in Table 52.
24:0 +04	Reserved.
31:4 +08	**Address[31:4]**. The system physical address that the IOMMU attempted to access.
3:0 +08	Reserved.
31:0 +12	**Address[63:32]**. The system physical address that the IOMMU attempted to access.

3.4.7 IOTLB_INV_TIMEOUT Event

If the IOMMU sends an invalidation request to a device and does not receive a response before the invalidation timeout timer expires, the IOMMU writes the event log with a IOTLB_INV_TIMEOUT event (see Table 45). See special considerations in Section 3.1.4.13 [INVALIDATE_IOTLB_PAGES and Peripheral Reset].

The Address field contains the system physical address of the invalidation command that timed out.

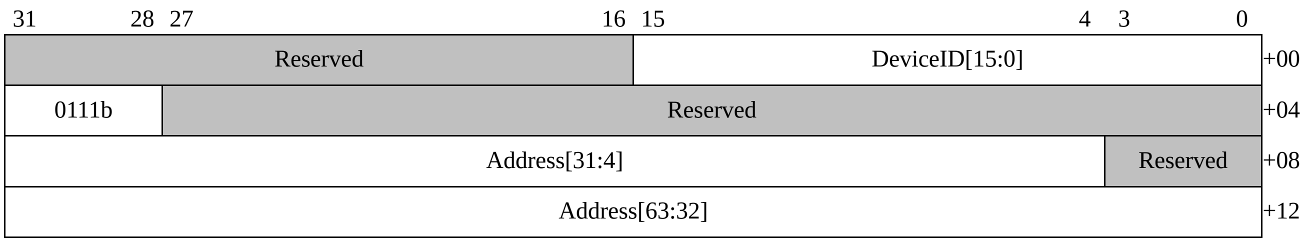


Figure 55: IOTLB_INV_TIMEOUT Event Log Buffer Entry Format

Table 57: IOTLB_INV_TIMEOUT Event Log Buffer Entry Fields

Bits	Description
31:16 +00	Reserved.
15:0 +00	**DeviceID.** Specifies the DeviceID that caused the invalidation timeout. The identity of the device causing the error can be determined using the DeviceID field.
31:28 +04	**0111b**. Specifies a IOTLB_INV_TIMEOUT entry.

Table 57: IOTLB_INV_TIMEOUT Event Log Buffer Entry Fields

27:0 +04	Reserved.
31:4 +08	**Address[31:4]**. The system physical address of the invalidation command that timed out.
3:0 +08	Reserved.
31:0 +12	**Address[63:32]**. The system physical address of the invalidation command that timed out.

3.4.8 INVALID_DEVICE_REQUEST Event

If the IOMMU receives a request from a device that the device is not allowed to perform, the IOMMU writes the event log with a INVALID_DEVICE_REQUEST event (see Table 46). Creation of event log entries for INVALID_DEVICE_REQUEST events is controlled by the IG bit in the device table entry (Figure 7 and Table 5). Depending on the type of the INVALID_DEVICE_REQUEST (Table 58), some of the fields in Figure 56 and Table 59 will not be meaningful.

Table 58: INVALID_DEVICE_REQUEST Type Field Encodings

Type	TR	Description
000b	0b	Read request or non-posted write in the interrupt address range (see Table 2).
001b	0b	Pretranslated transaction received from an I/O device that has I=0 or V=0 in the device's DTE.
010b	0b	Port I/O space transaction received from an I/O device that has IoCtl=00b in the device's DTE.
011b	0b	Posted write to the system management address range received from an I/O device that has SysMgt=00b, or with SysMgt=10b and the message is not a INTx message in the device's DTE, or a posted write to the address translation range when HtAtsResv=1 (see Table 2).
100b	0b	Revision 1: Read request or non-posted write in the system management address range (if SysMgt=10b or 0xb), or a read request or a non-posted write in the address translation range when HtAtsResv=1 (see Table 2); GN=1 for these errors and PASID is ignored. Revision 2: Also, a transaction in any address range with a TLP prefix when Revision 2 features are not available or not active for the I/O device (see Table 3); note that GN=1 for these errors and PASID is valid.
101b	0b	Posted write to the Interrupt/EOI range from an I/O device that has IntCtl=00b in the device's DTE (see Table 2).
110b	0b	Posted write to a reserved interrupt address range (see Table 2).
111b	0b	Transaction to the system management address range when SysMgt=11b or to the port I/O space range when IoCtl=10b, while V=1 and TV=0.

Table 58: INVALID_DEVICE_REQUEST Type Field Encodings

Type	TR	Description
000b	1b	Translation request received from an I/O device that has I=0, or has V=0, or has V=1 and TV=0 in the device's DTE.
001b	1b	Translation request in the interrupt, port I/O space (if IoCtl=0xb), or system management address range (if SysMgt=0xb or 10b); or translation request in the system management address range when SysMgt=11b or in the port I/O space range when IoCtl=10b, while V=1 and TV=0.
010b	1b	Revision 1: Reserved. Revision 2: A translation request for any address with a TLP prefix when Revision 2 features are not available or are not active for the I/O device (see Table 3); note that GN=1 for these errors and PASID is valid.
011b-111b	1b	Reserved.

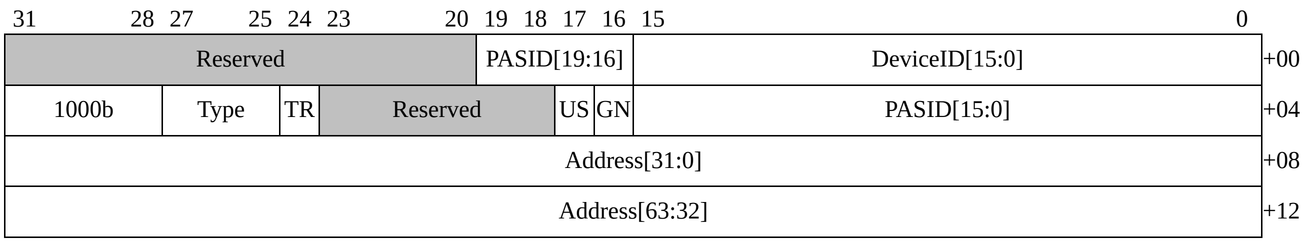


Figure 56: INVALID_DEVICE_REQUEST Event Log Buffer Entry Format

Table 59: INVALID_DEVICE_REQUEST Event Log Buffer Entry Fields

Bits	Description
31:16 +00	Reserved.
	PASID[19:16]: process address space ID. Revision 1: Reserved. Revision 2: The PASID[19:16] when GN=1; 0h when GN=0.
15:0 +00	**DeviceID**. Specifies the DeviceID that caused the page table lookup. The address of the device table entry can be determined using the DeviceID field.
31:28 +04	**1000b**. Specifies an INVALID_DEVICE_REQUEST entry.
27:25 +04	**Type.** The Type field indicates the type of hardware event that occurred as listed in Table 58.
24 +04	**TR: translation**. 1=transaction that caused the page table lookup was a translation request. 0=transaction that caused the page table lookup was a transaction request. See Table 58.
23:18 +04	Reserved.
17	**US: user-supervisor**. Revision 1: Reserved. Revision 2: 0=Supervisor privileges were asserted. 1=User privileges were asserted.
16 +04	**GN: guest/nested**. Revision 1: Reserved. Revision 2: 0=Address is a GPA. 1=Address is a GVA.
15:0 +04	**PASID[15:0]: process address space ID**. Revision 1: Reserved. Revision 2: The PASID when GN=1; 0000h when GN=0.

Table 59: INVALID_DEVICE_REQUEST Event Log Buffer Entry Fields

31:0 +08	**Address[31:0]**. The address that the device attempted to translate or access. See GN.
31:0 +12	**Address[63:32]**. The address that the device attempted to translate or access. See GN.

3.4.9 INVALID_PPR_REQUEST Event

Revision 1: EventCode 1001b is reserved.

Revision 2: An INVALID_PPR_REQUEST event log entry is generated when the peripheral page service request or the completion command has an problem (see Table 48).

When Capability Offset 00h[EFRSup]=0 or MMIO Offset 0030h[PPRSup]=0, the PCIe page request interface (PRI) is not supported by an IOMMU implementation and a COMPLETE_PPR_REQUEST command will cause an ILLEGAL_COMMAND_ERROR event. When PRI is supported (MMIO Offset 0030h[PPRSup]=1), a PRI request from a peripheral is invalid when MMIO Offset 0018h[PPREn]=0 or MMIO Offset 0018h[PPRLogEn]=0.

When the IOMMU receives an invalid PPR request from a peripheral, it writes the event log with an INVALID_PPR_REQUEST event containing RX=0. For certain error conditions noted in Table 48, the IOMMU also target aborts the transaction. The IOMMU continues processing normally. Software is responsible to issue any COMPLETE_PPR_REQUEST command required by the peripheral. The values in the INVALID_PPR_REQUEST event log entry are obtained from the peripheral page service request when RX=0.

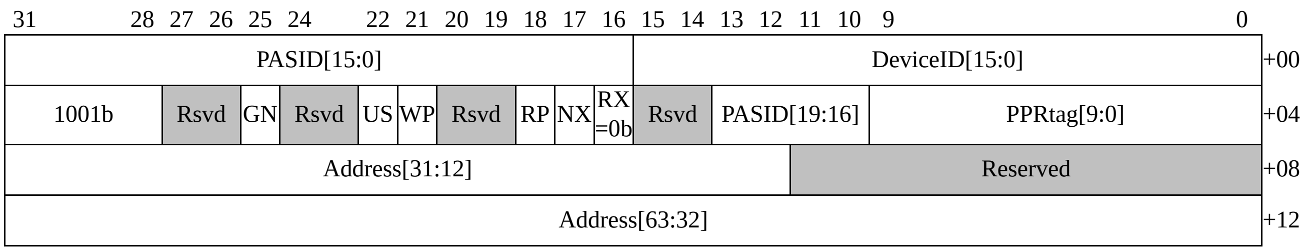


Figure 57: INVALID_PPR_REQUEST Event Log Buffer Entry Format, RX=0

When the IOMMU detects an error while processing a COMPLETE_PPR_REQUEST command, the IOMMU writes the event log with an INVALID_PPR_REQUEST event containing RX=1. The DeviceID and PPRtag fields are extracted from the failed COMPLETE_PPR_REQUEST command and the address of the COMPLETE_PPR_REQUEST command is reported in Address[63:0]. When RX=1, the contents of the WP, RP, and NX fields are undefined and should be ignored by software.

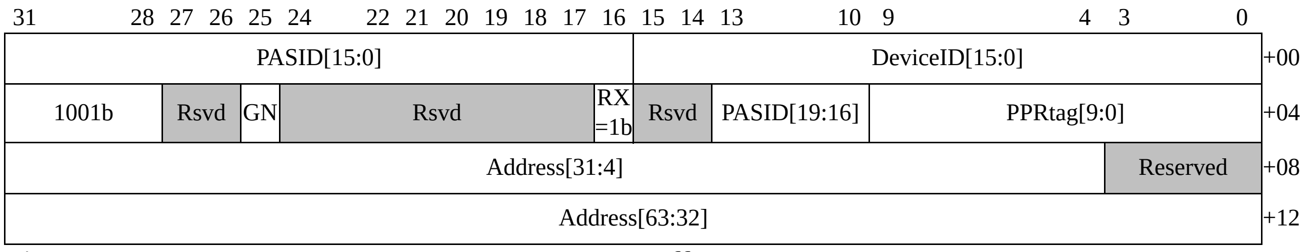


Figure 58: INVALID_PPR_REQUEST Event Log Buffer Entry Format, RX=1

Table 60: INVALID_PPR_REQUEST Event Log Buffer Entry Fields

Bits	Description, RX=0	Description, RX=1
31:16 +00	**PASID[15:0]**. Meaningful if GN=1; must be zero if GN=0.	
15:0 +00	**DeviceID[15:0]**. DeviceID of the peripheral issuing the invalid PPR request.	**DeviceID[15:0]**. DeviceID of the target peripheral (see Section 3.3.7 [COMPLETE_PPR_REQUEST]).
31:28 +04	**1001b**. Specifies an INVALID_PPR_REQUEST entry.	
27:26 +04	Reserved.	
25 +04	**GN: guest/nested**. 0=Address is a GPA and PASID is not meaningful. 1=Address is a GVA and PASID contains the process address space ID.	
24:23 +04	Reserved	
22 +04	**US**. User/supervisor bit as received from the peripheral. See PMR in Table 28.	Reserved.
21 +04	**WP**. Write permission request bit as received from the peripheral.	Reserved.
20:19 +04	Reserved.	
18 +04	**RP**. Read permission request bit as received from the peripheral.	Reserved.
17 +04	**NX**. No-execute permission request bit as received from the peripheral. See Exe in Table 28	Reserved.
16 +04	**RX**. 0=Invalid PAGE_SERVICE_REQUEST received (see Section 3.5.1 [Peripheral Page Service Request Entry]).	**RX**. 1=COMPLETE_PPR_REQUEST failed (see Section 3.3.7 [COMPLETE_PPR_REQUEST]).
15:14 +04	Reserved.	
13:10 +04	**PASID[19:16]**. Meaningful if GN=1; must be zero if GN=0.	
9:0 +04	**PPRtag[9:0]**. The PPRtag field as received from the peripheral. This field contains a protocol-dependent tag. • When the PPR request originated as a PCIe page request message, PPRtag[9] is the L bit and PPRtag[8:0] is the PRG index.	**PPRtag[9:0]**. The PPRtag field as sent to the peripheral. This field contains a protocol-dependent tag. • When the PPR completion targets a PCIe peripheral, PPRtag[9] is the L bit and PPRtag[8:0] is the PRG index. • The PRI Response Code in is not reported (see Table 36).
31:12 +08	**Address[31:12]**. The page address as received from the peripheral.	**Address[31:4]**. The SPA of the invalid PPR completion command.
11:4 +08	Reserved.	

Table 60: INVALID_PPR_REQUEST Event Log Buffer Entry Fields

3:0 +08	Reserved.	
31:0 +12	**Address[63:32]**. The page address as received from the peripheral.	**Address[63:32]**. The SPA of the invalid PPR completion command.

Software note: Software is responsible to take the PCI PRG index from the PPRtag field and use it in the PPR completion command (see Section 3.3.7 [COMPLETE_PPR_REQUEST]).

3.4.10 EVENT_COUNTER_ZERO Event

Revision 1: EventCode 1010b is reserved.

Revision 2: When the IOMMU is programmed to count events and a counter increments to become equal to zero, the IOMMU generates an EVENT_COUNTER_ZERO event. The CounterNote field contains the CounterNote value programmed into the corresponding event register (see IOMMU Counter Report Register [MMIO Offset [40-7F][0-F]28h]). The EVENT_COUNTER_ZERO event log entry is managed by the same controls as other events (see MMIO Offset 0010h[MsiNum] and MMIO Offset 0018h[EventIntEn, EventLogEn]).

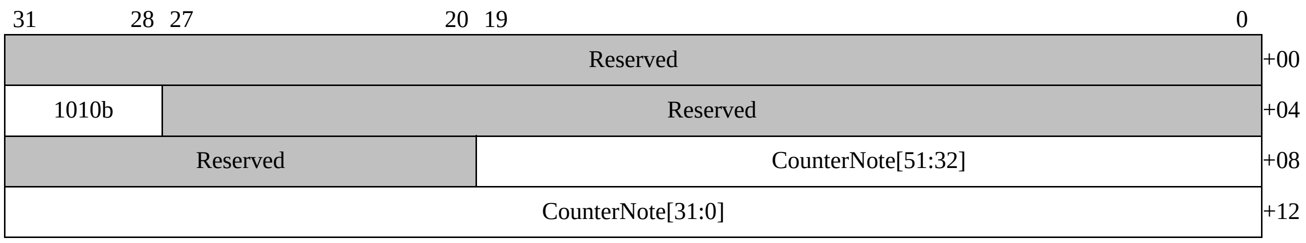


Figure 59: EVENT_COUNTER_ZERO Event Log Buffer Entry Format

Table 61: EVENT_COUNTER_ZERO Event Log Buffer Entry Fields

Bits	Description
31:00 +00	Reserved.
31:28 +04	**1010b**. Specifies a EVENT_COUNTER_ZERO entry.
27:0 +04	Reserved.
31:20 +08	Reserved.
19:0 +08	**CounterNote[51:0]**. The CounterNote value programmed into the corresponding Event Counter Register (see IOMMU Counter Report Register [MMIO Offset [40-7F][0-F]28h]).
31:0 +12	

3.4.11 IOMMU Event Reporting

The IOMMU is designed to identify and report hardware events, software programming problems, completion events, and performance events. Hardware events are the highest priority, programming problems are reported when there are no hardware events detected, and completion and performance events are reported as soon as possible. The IOMMU reports one event for a given activity; because the IOMMU can be highly concurrent, multiple events may be reported in quick succession from different causes. The IOMMU checks for exceptions in a sequence designed to protect system integrity and described in Section 3.4.11.1 [IOMMU Data Validation Sequence].

Revision 2: Hardware events could prevent reporting events via the event log in memory and are reported in IOMMU registers (Section 3.4.11.2 [I/O Hardware Event Reporting Registers]).

3.4.11.1 IOMMU Data Validation Sequence

When the IOMMU processes an interrupt remapping or address translation operation, it follows the data validation sequence in Figure 60. The order of reported events can vary based on the nature of the events and cached information. In the case of multiple events, the IOMMU is only required to report a single event.

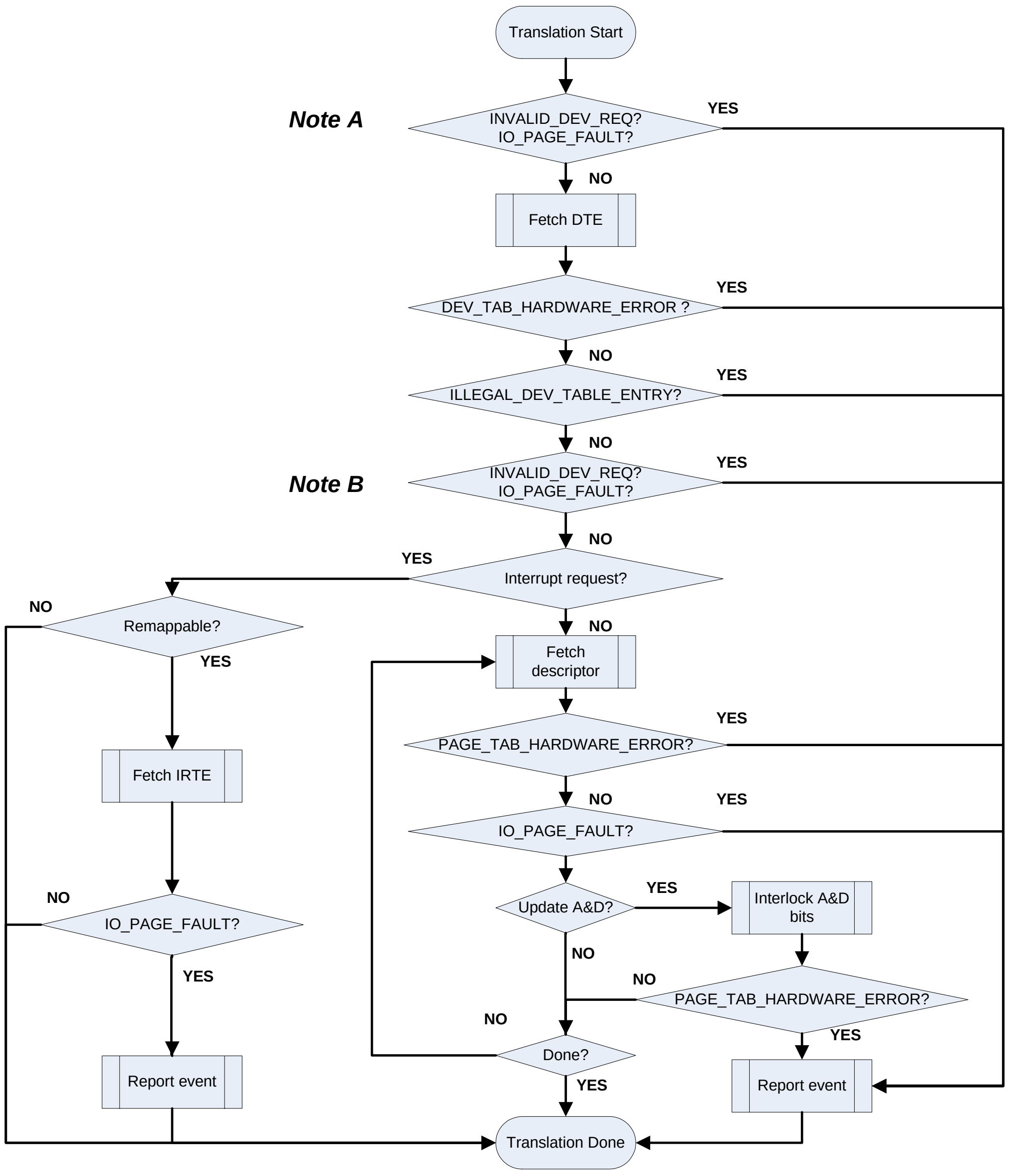


Figure 60: Translation and Remapping Validation Sequence

- **Note A**: these checks may run in parallel and an implementation selects any event to report when it detects multiple events.
- **Note B**: INVALID_DEVICE_REQUEST and IO_PAGE_FAULT checks may run in parallel and an implementation selects any detected event to report when it identifies multiple errors.

The IOMMU initially uses architectural definitions and information programmed in the registers to validate the request (Table 2, Device Table Base Address Register [MMIO Offset 0000h], IOMMU Control Register [MMIO Offset 0018h], IOMMU Exclusion Base Register [MMIO Offset 0020h], IOMMU Exclusion Range Limit Register [MMIO Offset 0028h], and IOMMU Extended Feature Register [MMIO Offset 0030h]). Once the IOMMU has loaded a device table entry, it runs a series of checks. The IOMMU uses architectural definitions to determine if the request requires interrupt remapping or address translation (Table 2). For interrupt remapping, the DTE and IRTE are used with architectural definitions to check for exceptions in sequence (Table 7 and Table 8). For address translation, the IOMMU fetches a series of descriptors and checks for exceptions in sequence. After reporting an event, the IOMMU terminates the translation process.

3.4.11.2 I/O Hardware Event Reporting Registers

Three types of event log entries are caused by memory faults:

- Section 3.4.3 [DEV_TAB_HARDWARE_ERROR Event],
- Section 3.4.4 [PAGE_TAB_HARDWARE_ERROR Event], and
- Section 3.4.6 [COMMAND_HARDWARE_ERROR Event].

Revision 1: All hardware events are written to the event log when logging is enabled. The IOMMU Hardware Event Status Register [MMIO Offset 0050h], IOMMU Hardware Event Upper Register [MMIO Offset 0040h], and IOMMU Hardware Event Lower Register [MMIO Offset 0048h] always read as zero.

Revision 2: Hardware event information is written to the hardware event registers by the IOMMU if MMIO Offset 0030h[HESup]=1. The event log information shown in Figure 48 for the hardware events listed in this section is reported in IOMMU Hardware Event Upper Register [MMIO Offset 0040h] and IOMMU Hardware Event Lower Register [MMIO Offset 0048h] where it can be extracted (e.g., for system diagnostic purposes when memory issues prevent updates to the event log). When logging is enabled, the IOMMU also creates an event log entry. When MMIO Offset 0018h[EventIntEn]=1, the IOMMU signals an interrupt. The hardware events are reported in the hardware event registers even when event logging to memory is not enabled. The information in the hardware event registers is meaningful when MMIO Offset 0050h[HEV]=1. The information in the hardware event registers has overwritten prior information when MMIO Offset 0050h[HEO]=1. Hardware sets MMIO Offset 0050h[HEO]=1 if MMIO Offset 0050h[HEV]=1 when the IOMMU writes new information to IOMMU Hardware Event Upper Register [MMIO Offset 0040h] and IOMMU Hardware Event Lower Register [MMIO Offset 0048h]. HEO is informational and event register overflow does not, itself, cause an error. Software must clear HEV after reading the hardware event registers to prepare the registers to record new information.

3.5 Peripheral Page Service Request (PPR) Logging

Revision 1: Not supported.

Revision 2: Some ATS-capable peripherals can issue requests to the processor to service page service requests using PCIe PRI, the Page Request Interface (see the PCI *Address Translation Services Revision 1.1* specification). An IOMMU that supports peripheral page service requests (MMIO Offset 0030h[PPRSup]=1b) can report these requests to the host software by means of a shared circular buffer in system memory. The IOMMU writes peripheral page service request (PPR) records into the buffer when enabled by MMIO Offset 0018h[PPREn]. The host software increments the IOMMU's PPR request log head pointer to indicate to the IOMMU that it has consumed PPR request log entries. When software has completed processing the PPR request, it uses the IOMMU COMPLETE_PPR_REQUEST command to inform the peripheral of the results (see Section 3.3.7 [COMPLETE_PPR_REQUEST]).

Software note: The IOMMU cannot service PRI requests without software intervention, so it converts them to PPR log entries for software to process. All PRI requests are converted to PPR log entries as long as there is room in the PPR log while MMIO Offset 0018h[PPRRun]=1. To stop the IOMMU from processing all PRI requests, software can program MMIO Offset 0018h[PPRRun]=0; this causes PRI requests to be discarded by the IOMMU. To stop an individual peripheral from issuing PRI requests, software must use the control fields in the peripheral registers.

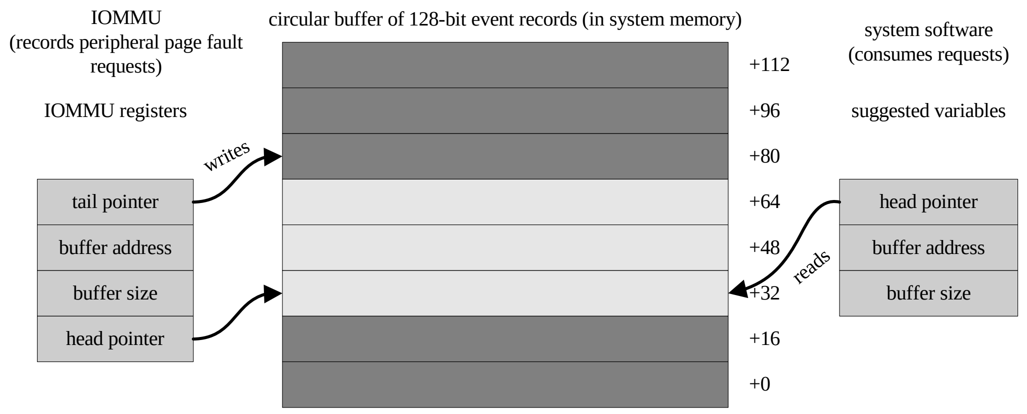


Figure 61: Circular Peripheral Page Service Request Log in System Memory

The PPR Log Base Address Register [MMIO Offset 0038h] is used to program the system physical address and size of the PPR log. The PPR log occupies contiguous physical memory starting at the programmed base address up to the programmed size. The size of the PPR log must be a multiple of 4K bytes (to facilitate "mod N" indexing for circularity), and can be as large as 32768 entries (corresponding to a 512 kilobyte buffer). The base address of the PPR log must be aligned to a multiple of 4K bytes.

In addition to the PPR Log Base Address Register [MMIO Offset 0038h], the IOMMU maintains two other registers associated with the PPR log: the IOMMU PPR Log Head Pointer Register [MMIO Offset 2030h] which points to the next PPR request that software will read, and the IOMMU PPR Log Tail Pointer Register [MMIO Offset 2038h] which points to the next PPR request to be written by the IOMMU. These registers are located in MMIO space. When the PPR Log Base Address Register [MMIO Offset 0038h] register is written, the IOMMU PPR Log Head Pointer Register [MMIO Offset 2030h] and the IOMMU PPR Log Tail Pointer Register [MMIO Offset 2038h] are cleared to 0. When the IOMMU PPR Log Head Pointer Register [MMIO Offset 2030h] and the IOMMU PPR Log Tail Pointer Register [MMIO Offset 2038h] are equal, the PPR request log is empty. The IOMMU PPR Log Tail Pointer Register [MMIO Offset 2038h] is incremented by the IOMMU after writing a PPR request to the log. If the IOMMU needs to report a service request but finds that the PPR log is already full, it sets MMIO Offset 2020h[PPROverflow]. The IOMMU can be configured to signal an interrupt using Capability Offset 10h[MsiNumPPR] whenever the PPR log is written or overflows by setting MMIO Offset 0018h[PPRIntEn]. The PPR request log is full when all slots but one are used. The PPR log has overflowed when a PPR request occurs that is to be logged and would otherwise consume the last unused slot. When the PPR log has overflowed, the MMIO Offset 2020h[PPROverflow] is set, any data for new PPR requests is discarded, and PPR logging is disabled. The host software must make space in the PPR log by reading entries (by adjusting the head pointer) or resizing the log. PPR request logging may then be restarted.

The IOMMU PPR logging is disabled after system reset and when the PPR log overflows. The IOMMU discards PPR requests until PPR logging is enabled, setting IOMMU Status Register [MMIO Offset 2020h][PPROverflow] to indicate the loss of PPR request information. To restart the IOMMU PPR request logging after the PPR log overflows, use the following procedure.

- Wait until IOMMU Status Register [MMIO Offset 2020h][PPRLogRun]=0b so that all request entries are completed as circumstances allow. PPRLogRun must be 0b to modify the PPR log registers safely.
- Write IOMMU Control Register [MMIO Offset 0018h][PPRLogEn]=0b.
- As necessary, change the following registers (e.g., to relocate or resize the PPR log):
    - the PPR Log Base Address Register [MMIO Offset 0038h],
    - the IOMMU PPR Log Head Pointer Register [MMIO Offset 2030h], and

  - the IOMMU PPR Log Tail Pointer Register [MMIO Offset 2038h].
- Write IOMMU Status Register [MMIO Offset 2020h][PPROverflow]=0b to clear the bit.
- Write IOMMU Control Register [MMIO Offset 0018h][PPRLogEn]=1b and IOMMU Control Register [MMIO Offset 0018h][PPRIntEn] as desired.

The IOMMU now creates PPR request log entries for new requests.

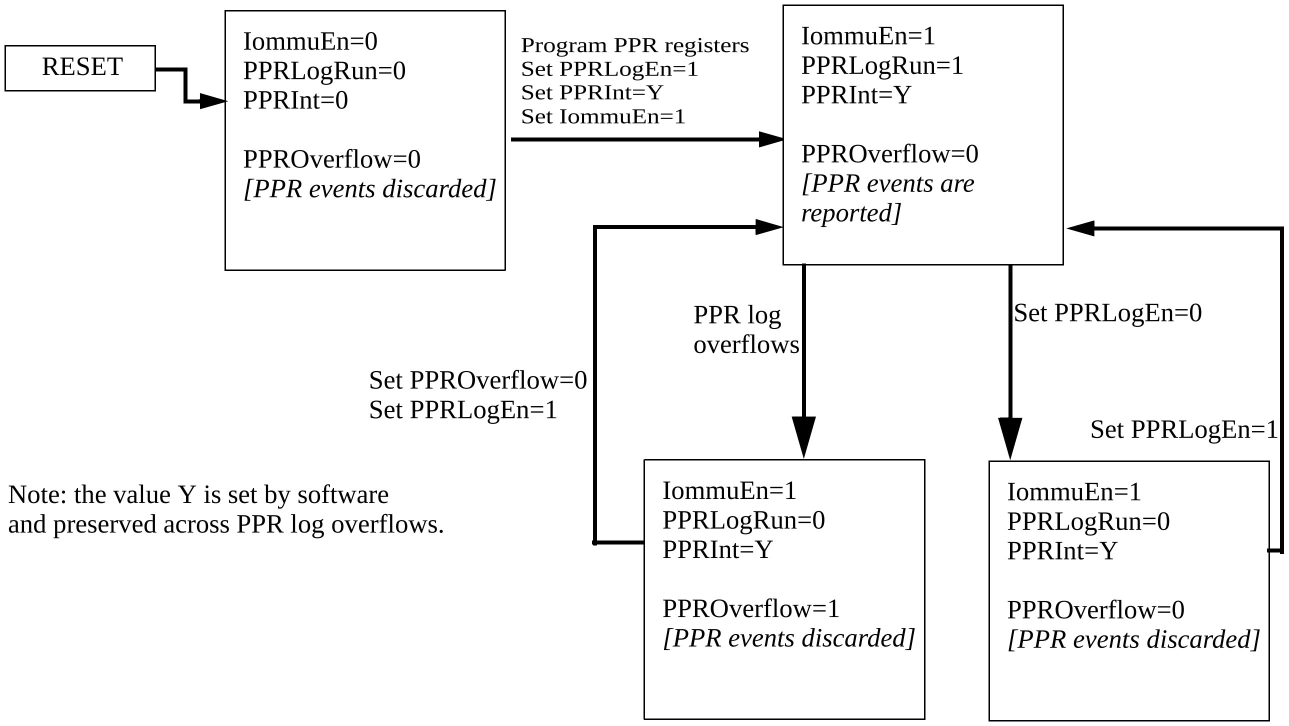


Figure 62: PPR log state diagram

All PPR requests recorded by the IOMMU consist of a 4-bit PPRCode together with two operands, which may be respectively 60 and 64 bits long, for a total of 128 bits (16 bytes) per record.

The IOMMU must set the Coherent bit in the HyperTransport™ packet when generating writes to the PPR log.

31 28	27 0	
First event code dependent operand [31:0]		+00
PPRCode[3:0]	First event code dependent operand [59:32]	+04
Second event code dependent operand [31:0]		+08
Second event code dependent operand [63:32]		+12

Figure 63: Generic Peripheral Page Service Request Log Buffer Entry Format

3.5.1 Peripheral Page Service Request Entry

Revision 1: Not used.

Revision 2: When a peripheral needs memory page services, it issues a special bus request to the IOMMU. If supported (see MMIO Offset 0030h[PPRSup] and MMIO Offset 0018h[PPREn]), the IOMMU converts the special bus request to the PAGE_SERVICE_REQUEST format. When peripheral page service is enabled for the device (see MMIO Offset 0018h[PPRLogEn]), the IOMMU creates a PAGE_SERVICE_REQUEST entry in the PPR log buffer. Certain types of ill-formed PCIe PRI requests are logged in the PPR request log with RZ=1 so that software may attempt recovery (e.g., reserved bit error in Figure 34). When peripheral page service is not enabled, the IOMMU creates an entry in the IOMMU event log to report the error (see Section 3.4.9 [INVALID_PPR_REQUEST Event]). After processing the request, software issues a COMPLETE_PPR_REQUEST command to inform the peripheral that page service processing is complete (see Section 3.3.7 [COMPLETE_PPR_REQUEST]).

If the PCIe PRI request has a PASID TLP prefix with a valid PASID, it is a GVA request and the header contains a PASID. If the PRI request packet lacks a PASID TLP prefix with a valid PASID, it is a nested (GPA) request and the PASID in the log entry must be ignored by software. The presence of a valid PASID is indicated to software by the GN bit in the log entry.

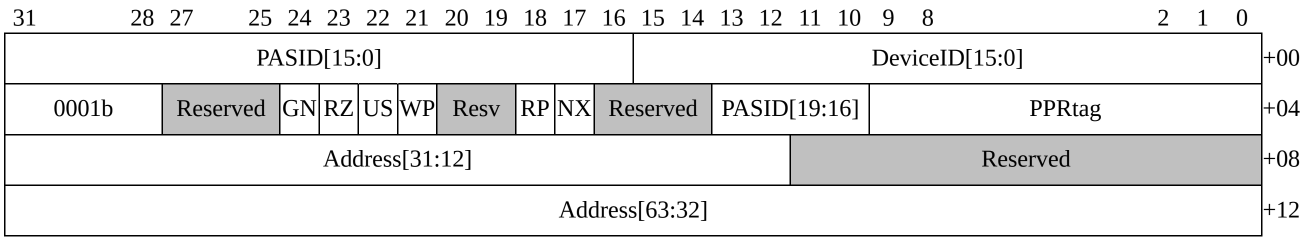


Figure 64: PAGE_SERVICE_REQUEST PPR Log Buffer Entry Format

Table 62: PAGE_SERVICE_REQUEST PPR Log Buffer Entry Fields

Bits	Description
31:16 +00	**PASID[15:0]**. Specifies PASID[15:0] that requested the page service. PASID is valid when GN=1 and is ignored by software when GN=0.
15:0 +00	**DeviceID**. Specifies the DeviceID that requested the page service. The IOMMU domain can be determined using the DeviceID field. The page to be serviced can be determined from the Address field.
31:28 +04	**0001b**. Specifies a PAGE_SERVICE_REQUEST from the peripheral identified in the DeviceID field.
27:25 +04	Reserved.
24 +04	**GN: Guest/nested.** 1=Address[63:12] is a GVA and PASID is valid. 0=Address[63:12] is a GPA and PASID should be ignored by software.
23 +04	**RZ: reserved bit not zero or reserved encoding**. 1=The received peripheral request had a non-zero reserved bit or used a reserved encoding. The rest of the request has been reported as it was received. Software may attempt recovery. 0=The received peripheral request passed all hardware validation checks.
22 +04	**US: User/Supervisor**. The U/S request received from the peripheral.1=user and supervisor access are allowed. 0=supervisor access is allowed.
21 +04	**WP: write permission requested.** 1=peripheral is requesting write access. 0=write access may be denied.
20:19 +04	Reserved.
18 +04	**RP: read permission requested**. 1=peripheral is requesting read access. 0=read access may be denied.

Table 62: PAGE_SERVICE_REQUEST PPR Log Buffer Entry Fields

17 +04	**NX: execute permission requested**. 1=peripheral is requesting NX handling of access requests (instruction fetch requests will be blocked, read requests are to be allowed when RP=1). 0=peripheral instruction fetch access should be handled as a read request.
16:14 +04	Reserved.
13:10 +04	**PASID[19:16]**. Specifies PASID[19:16] that requested the page service. PASID is valid when GN=1 and is ignored by software when GN=0.
9:0 +04	**PPRtag: protocol tag**. This field contains a protocol-dependent tag. When the PPR request originated as a PCIe page request message, PPRtag[9] is the PRI L bit and PPRtag[8:0] is the PRI PRG index; the IOMMU is required to return the PRG index in the response message (see Section 3.3.7 [COMPLETE_PPR_REQUEST]).
31:12 +08	**Address[31:12]**. The address field contains the device virtual address that the device was attempting to access. The minimum invalidation granularity is a 4K byte page so the address is truncated. See also GN and PASID fields.
11:0 +08	Reserved.
31:0 +12	**Address[63:32]**. The Address field contains the device virtual address that the device was attempting to access. See also GN and PASID fields.

The log entry is designed to carry a full set of independent read-, write-, and execute-permission bits; any bits not provided by the underlying peripheral protocol are set to the “permitted” state by the IOMMU (see also MMIO Offset 0030h[NXSup]).

3.6 IOMMU Interrupt Support

The IOMMU uses standard PCI interrupt mechanisms to generate interrupts. The IOMMU must support signaling of either MSI or MSI-X interrupts. The MSI capability must support 64-bit addressing. The IOMMU must not set the PassPW bit when sending interrupts associated with the IOMMU over HyperTransport™ links.

Revision 1: The IOMMU supports generation of an interrupt when the event log is updated or overflows, and when a completion wait command completes (see Capability Offset 10h[MsiNum], MMIO Offset 0018h[ComWaitEn], and MMIO Offset 0018h[EventIntEn]).

Revision 2: The IOMMU supports generation of an interrupt when the event log is updated or overflows, and when a completion wait command completes (see Capability Offset 10h[MsiNum], MMIO Offset 0018h[ComWaitEn], and MMIO Offset 0018h[EventIntEn]), and when the peripheral page service request log is updated or overflows (see Capability Offset 10h[MsiNumPPR] and MMIO Offset 0018h[PPRIntEn]).

3.7 PCI Resources

The IOMMU must be implemented as an independent PCIe Function. Any PCIe Function containing an IOMMU capability block must be a PCIe Endpoint. A peripheral may implement more than one IOMMU within a single PCIe Function. Any PCIe Function containing an IOMMU capability block may not be used for any purpose other than containing an IOMMU. A PCIe Function containing an IOMMU capability block may not have any PCI BAR registers. Configuration and status information for the IOMMU are mapped into PCI configuration space using a PCI capability block. One or more IOMMU capability blocks may be implemented in a PCIe Function. If more than one IOMMU capability block is implemented in a PCIe Function, the

IOMMU must support generating MSI-X interrupts.

If a single IOMMU capability block is implemented in a PCIe Function, the IOMMU may support either MSI or MSI-X or both. The PCIe Function must assign a distinct interrupt vector to each interrupt that can be generated by each IOMMU capability block (i.e., two interrupts per IOMMU capability that has MMIO Offset 0030h[PPRSup]=1 and one interrupt per IOMMU capability block that has MMIO Offset 0030h[PPRSup]=0).

The PCI class is System Base Peripheral (08h) with a subclass of IOMMU (06h) and a programming interface code of 00h, as issued by the PCI-SIG.

The HyperTransport™ UnitID used when an IOMMU generates requests must not be used for any other traffic. A HyperTransport™ UnitID can be shared by multiple IOMMUs within a physical component.

IOMMU registers not specified in Section 3.7 [PCI Resources] return 0s when read and ignore the data when written.

3.7.1 IOMMU Capability Block Registers

The presence of an IOMMU capability block in a PCIe Function indicates the presence of an IOMMU. The IOMMU capability block contains various registers to control the IOMMU and to configure the location of the MMIO registers of the IOMMU.

When Capability Offset 04h[Enable] is written with a 1b, all RW capability registers defined in Section 3.7.1 [IOMMU Capability Block Registers] are locked until the next system reset. This means the registers become read-only and attempts to write them are ignored.

Capability Offset 00h IOMMU Capability Header

This register indicates that this is an IOMMU capability block.

31 28	27	26	25	24	23 19	18 16	15 8	7 0
Reserved	EFRSup	NpCache	HtTunnel	IotlbSup	CapRev	CapType	CapPtr	CapID

Bits	Description
31:28	Reserved.
27	**EFRSup: IOMMU Extended Feature Register support**. Revision 1: RO. Reset 0b. Reserved. The IOMMU Extended Feature Register [MMIO Offset 0030h] is not supported. Revision 2: RO. Reset Xb. 1=Indicates IOMMU Extended Feature Register [MMIO Offset 0030h] is supported. 0=MMIO Offset 0030h is Reserved.
26	**NpCache: not present table entries cached**. RO. Reset Xb. 1=Indicates that the IOMMU caches page table entries that are marked as not present. When this bit is set, software must issue an invalidate after any change to a PDE or PTE. 0=Indicates that the IOMMU caches only page table entries that are marked as present. When NpCache is clear, software must issue an invalidate after any change to a PDE or PTE marked present before the change. **Implementation note:** For hardware implementations of the IOMMU, this bit must be 0b.
25	**HtTunnel: HyperTransport™ tunnel translation support**. RO. Reset Xb. Indicates that the device contains a HyperTransport™ tunnel that supports address translation on the HyperTransport™ interface.

24	**IotlbSup: IOTLB Support**. RO. Reset Xb. Indicates the IOMMU will support ATS translation request messages as defined in PCI ATS 1.0 or later.
23:19	**CapRev: capability revision**. RO. Reset 0_0001b. Specifies the IOMMU interface revision. **Software note**: this value is changed when architectural changes cause an interface incompatibility.
18:16	**CapType: IOMMU capability block type**. RO. Reset 011b. Specifies the layout of the Capability Block as an IOMMU capability block.
15:8	**CapPtr: capability pointer**. RO. Reset XXh. Indicates the location of the next capability block, or 00h if this is the last capability block in the capability list.
7:0	**CapId: capability ID**. RO. Reset 0Fh. Indicates a Secure Device capability block.

Capability Offset 04h IOMMU Base Address Low Register

This register specifies the lower 32 bits of the base address (SPA) of the IOMMU control registers. This register is locked when IOMMU Base Address Low[Enable] is written with a 1b. Revision 1: the Base Address[31:14] may be set to any value; the Base Address is 16 Kbyte aligned.

Revision 2: When MMIO Offset 0030h[PCSup]=1, the IOMMU event counters are supported therefore the Base Address[31:19] may be set to any value and Base Address[18:14] must be set to 0_0000b such that the Base Address is 512 Kbyte aligned (see Section 3.7.2.4 [MMIO Event Counter Configuration Registers]). When MMIO Offset 0030h[PCSup]=0, the IOMMU event counters are not supported therefore the Base Address[31:14] may be set to any value such that the Base Address is 16 Kbyte aligned.

31 19	18 14	13 1	0
BaseAddress[31:19]	BaseAddress[18:14]	Reserved	Enable

Bits	Description
31:19	**BaseAddress[31:19]**. RW when Capability Offset 04h[Enable]=0. RO when Capability Offset 04h[Enable]=1. Reset 0_0000_0000_0000b. Specifies lower bits of the base address of the IOMMU control registers. Base Address[31:19] may be set to any value.
18:14	**BaseAddress[18:14]**. RW when Capability Offset 04h[Enable]=0. RO when Capability Offset 04h[Enable]=1. Reset 0_0000b. Specifies the lower bits of the base address of the IOMMU control registers. Revision 1: Base Address[18:14] may be set to any value. Revision 2: In order to use the IOMMU event counters Base Address[18:14] must be set to 0_0000b. See MMIO Offset 0030h[PCSup].
13:1	Reserved.
0	**Enable**. RW1S. Reset 0b. 1=IOMMU accepts memory accesses to the address specified in the Base Address Register. When Enable is written with a 1, all RW capability registers defined in Section 3.7.1 [IOMMU Capability Block Registers] are locked until the next system reset. Note: BaseAddress may be changed and locked with the same write operation that sets Enable=1b.

Capability Offset 08h IOMMU Base Address High Register

This register specifies the upper 32 bits of the base address of the IOMMU control registers. This register is locked when IOMMU Base Address Low[Enable] is written with a 1b.

31	0
BaseAddress[63:32]	

Bits	Description
31:0	**BaseAddress[63:32]**. RW when Capability Offset 04h[Enable]=0. RO when Capability Offset 04h[Enable]=1. Reset 0000_0000h. Specifies the upper 32 bits of the base address of the IOMMU control registers.

Capability Offset 0Ch IOMMU Range Register

This register indicates the device and function numbers of the first and last devices associated with the IOMMU. This register is locked when IOMMU Base Address Low[Enable] is written with a 1. Root port devices that have device and function numbers between the first and last device numbers inclusive are supported by the IOMMU and provide full source identification to the IOMMU. Non-root port devices that have device and function numbers between the first and last device numbers inclusive are devices integrated in with the IOMMU and support address translation using the IOMMU. Integrated devices associated with the IOMMU must be located on the same logical bus.

31 24	23 16	15 8	7	6 5	4 0
LastDevice	FirstDevice	BusNumber	RngValid	Reserved	UnitID

Bits	Description
31:24	**LastDevice: last device**. RW when Capability Offset 04h[Enable]=0. RO when Capability Offset 04h[Enable]=1. Reset XXh. Indicates device and function number of the last integrated device associated with the IOMMU. **Note:** an implementation may define this value as RO.
23:16	**FirstDevice: first device**. RW when Capability Offset 04h[Enable]=0. RO when Capability Offset 04h[Enable]=1. Reset XXh. Indicates device and function number of the first integrated device associated with the IOMMU. **Note:** an implementation may define this value as RO.
15:8	**BusNumber: Device range bus number**. RW when Capability Offset 04h[Enable]=0. RO when Capability Offset 04h[Enable]=1. Reset XXh. Indicates the bus number that FirstDevice and LastDevice reside on. **Note:** an implementation may define this value as RO.
7	**RngValid: Range valid.** RW when Capability Offset 04h[Enable]=0. RO when Capability Offset 04h[Enable]=1. Reset Xb. 1b=the BusNumber, FirstDevice, and LastDevice fields are valid. Although the register contents are valid, software is encouraged to use I/O topology information as defined in Section 5 [I/O Virtualization ACPI Tables]. 0b=Software must use I/O topology information as defined Section 5 [I/O Virtualization ACPI Tables]. **Note:** an implementation may define this value as RO.
6:5	Reserved.
4:0	**UnitID: IOMMU HyperTransport™ UnitID**. RW when Capability Offset 04h[Enable]=0. RO when Capability Offset 04h[Enable]=1. Reset X_XXXXb. This field returns the HyperTransport™ UnitID used by the IOMMU. **Note:** an implementation may define this value as RO. **Note:** this field is deprecated and may be set to 0_0000b.

Capability Offset 10h IOMMU Miscellaneous Information Register

This register returns the size of virtual and physical addresses supported by the IOMMU, and the message number for MSI or MSI-X interrupts associated with the IOMMU.

31 27	26 23	22	21 15	14 8	7 5	4 0
MsiNumPPR	Reserved	HtAtsResv	VAsize	PAsize	GVAsize	MsiNum

Bits	Description
31:27	**MsiNumPPR: Peripheral Page Service Request MSI message number**. Revision 1: RO. Reset 0_0000b. Reserved. Revision 2: RO. Reset X_XXXXb. This field must indicate which MSI/MSI-X vector is used for the interrupt message generated by the IOMMU for the peripheral page service request log when MMIO Offset 0030h[PPRSup]=1. MsiNumPPR must be zero when PPRSup=0. For MSI there can only be one IOMMU so this field must be zero (Section 3.7 [PCI Resources]). For MSI-X, the value in this field indicates which MSI-X Table entry is used to generate the interrupt message. Either MSI or MSI-X must be implemented, but not both. This interrupt is not remapped by the IOMMU. If neither MSI nor MSI-X are enabled and a PPR interrupt occurs, the interrupt is silently dropped. **Implementation note:** INTx is not supported for the PPR interrupt.
26:23	Reserved.
22	**HtAtsResv: ATS response address range reserved.** RW when Capability Offset 04h[Enable]=0b. RO when Capability Offset 04h[Enable]=1b. Reset 0b. 1=The HyperTransport™ Address Translation address range for ATS responses is reserved and cannot be translated by the IOMMU. 0=The Address Translation address range can be translated by the IOMMU. See Table 2. **Implementation note**: This bit may be RO if ATS is not supported.
21:15	**VAsize: Virtual Address size.** RO. Reset XXXXXXb. This field must indicate the size of the maximum virtual address processed by the IOMMU. The value is the (unsigned) binary log of the maximum address size. Allowed values are 32, 40, 48, and 64; all other values are reserved. 010_0000b = 32 bits 010_1000b = 40 bits 011_0000b = 48 bits 100_0000b = 64 bits Revision 2: This field defines the size of the GPA.
14:8	**PAsize: Physical Address size.** RO. Reset XXXXXXb. This field must indicate the size of the maximum physical address generated by the IOMMU. The value is the (unsigned) binary log of the maximum address size. Allowed values are 40, 48, and 52; all other values are reserved. 010_1000b = 40 bits 011_0000b = 48 bits 011_0100b = 52 bits Revision 2: This field defines the size of the SPA.

7:5	**GVAsize: Guest Virtual Address size.** Revision 1: RO. Reset 000b. Revision 2: RO. Reset XXXb. This field must indicate the size of the maximum guest virtual address processed by the IOMMU. The allowed size is 48 and all other values are reserved. 000b - 001b = Reserved. 010b = 48 bits 001b - 111b = Reserved.
4:0	**MsiNum: MSI message number**. RO. Reset XXXXXb. This field must indicate which MSI/MSI-X vector is used for the interrupt message generated by the IOMMU for the IOMMU event log. For MSI there can only be one IOMMU so this field must be zero (Section 3.7 [PCI Resources]). For MSI-X, the value in this field indicates which MSI-X Table entry is used to generate the interrupt message. Either MSI or MSI-X can be implemented, but not both. If the MSI-X capability is not enabled, INTx is used to deliver the interrupt. This interrupt is not remapped by the IOMMU. Revision 2: Either MSI or MSI-X must be implemented, but not both. This interrupt is not remapped by the IOMMU. **Implementation note:** INTx is supported for backwards compatibility with IOMMU Revision 1 but its use is discouraged in designs.

3.7.2 IOMMU MMIO Registers

The IOMMU control registers are mapped using the IOMMU Base Address Low Register [Capability Capability Offset 04h] and IOMMU Base Address High Register [Capability Capability Offset 08h] specified in the IOMMU capability block. Software access to IOMMU registers may not be larger than 64 bits. Accesses must be aligned to the size of the access and the size in bytes must be a power of two. Software may use accesses as small as one byte.

3.7.2.1 MMIO Control and Status Registers

MMIO Offset 0000h Device Table Base Address Register

This register specifies the system physical address of the device table.

63 52	51 32
Reserved	DevTabBase

31 12	11 9	8 0
DevTabBase	Reserved	Size[8:0]

Bits	Description
63:52	Reserved.
51:12	**DevTabBase: device table base address**. RW. Reset 00_0000_0000h. Specifies the 4Kbyte-aligned base address of the first level device table.
11:9	Reserved.
8:0	**Size: size of the device table**. RW. Reset 000h. This field contains 1 less than the length of the device table, in multiples of 4K bytes. A minimum size of 0 corresponds to a 4K byte device table and a maximum size of 1FFh corresponds to a 2M byte device table.

MMIO Offset 0008h Command Buffer Base Address Register

This register specifies the system physical address and length of the command buffer.

63 60	59 56	55 52	51 32
Reserved	ComLen	Reserved	ComBase

31 12	11 0
ComBase	Reserved

Bits	Description
63:60	Reserved.
59:56	**ComLen: command buffer length**. RW. Reset 1000b. Specifies the length of the command buffer in power of 2 increments. The minimum size is 256 entries (4K bytes); values less than 1000b are reserved. 0000b - 0111b = Reserved 1000b = 256 entries (4K bytes) 1001b = 512 entries (8K bytes) ... 1111b = 32768 entries (512K bytes)
55:52	Reserved
51:12	**ComBase: command buffer base address**. RW. Reset 00_0000_0000h. Specifies the base address of the command buffer. The base address programmed must be aligned to 4K bytes.
11:0	Reserved

MMIO Offset 0010h Event Log Base Address Register

This register specifies the system physical address and length of the event log.

Software Note: If EventLen or EventBase is changed while the EventLogRun=1, the IOMMU behavior is undefined.

63 60	59 56	55 52	51 32
Reserved	EventLen	Reserved	EventBase

31 12	11 0
EventBase	Reserved

Bits	Description
63:60	Reserved.

59:56	**EventLen: event log length**. RW. Reset 1000b. Specifies the length of the event log in power of 2 increments. The minimum size is 256 entries (4K bytes); values less than 1000b are reserved. 0000b - 0111b = Reserved 1000b = 256 entries (4K bytes) 1001b = 512 entries (8K bytes) ... 1111b = 32768 entries (512K bytes)
55:52	Reserved
51:12	**EventBase: event log base address**. RW. Reset 00_0000_0000h. Specifies the base address of the event log. The base address programmed must be aligned to 4K bytes.
11:0	Reserved

MMIO Offset 0018h IOMMU Control Register

This register controls the behavior of the IOMMU.

63:32
Reserved

31:22	21:18	17	16	15	14	13	12	11	10	9	8	7:5	4	3	2	1	0
Reserved	CRW	GAEn	GTEn	PPREn	PPRIntEn	PPRLogEn	CmdBufEn	Isoc	Coherent	ResPassPW	PassPW	InvTimeOut	ComWaitIntEn	EventIntEn	EventLogEn	HtTunEn	IommuEn

Bits	Description
63:22	Reserved.
21:18	**CRW: ignored.** Revision 1: RO. Reset 0h. Reserved. Revision 2: RW. Reset 0h. Intended for future use. **Software note:** this field is not implemented in Revision 2 but may be defined in future revisions. Software may safely write 0h to this field in Revision 2 and should ignore the value read.
17	**GAEn: Guest APIC enable**. Revision 1: RO. Reset 0b. Reserved. Revision 2: RW. Reset 0b. 0=Device interrupt virtualization is not enabled. 1=device interrupts are updated using the Guest Virtual APIC Table Root Pointer in the DTE and are posted to the processors. Writes to this bit are ignored when MMIO Offset 0030h[GASup]=0.
16	**GTEn: guest translation enable**. Revision 1: RO. Reset 0b. Reserved. Revision 2: RW. Reset 0b. 0=Guest translation disabled. 1=Guest translation may be enabled for a peripheral by programming DTE[GV] (see Table 5). When guest translation is enabled, invalidation semantics are changed (see Section 2.3.2 [Enhanced AMD64 long Page Table Compatibility]). Writes to this bit are ignored when MMIO Offset 0030h[GTSup]=0.
15	**PPREn: peripheral page service request processing enable**. Revision 1: RO. Reset 0b. Reserved. Revision 2: RW. Reset 0b. 1=Peripheral page service requests are processed. 0=PPR requests are treated as invalid device requests (see Section 3.4.8 [INVALID_DEVICE_REQUEST Event]). Writes to this bit are ignored when MMIO Offset 0030h[PPRSup]=0.
14	**PPRIntEn: peripheral page service request interrupt enable**. Revision 1: RO. Reset 0b. Reserved. Revision 2: RW. Reset 0b. 1=An interrupt is signalled when MMIO Offset 2020h[PPRLogInt]=1 using Capability Offset 10h[MsiNumPPR]. 0=An interrupt is not signalled when MMIO Offset 2020h[PPRLogInt]=1. Writes to this bit are ignored when MMIO Offset 0030h[PPRSup]=0.

13	**PPRLogEn: peripheral page service log enable**. Revision 1: RO. Reset 0b. Reserved. Revision 2: RW. Reset 0b. 1=The PPR Log Base Address Register [MMIO Offset 0038h] has been configured and peripheral page service request events are written to the peripheral page service request log when IommuEn has also been set. Writing a 1b to this bit when PPRLogEn=1b has no effect. 0=Peripheral page service request logging is not enabled. Peripheral page service requests are discarded when the peripheral page service request log is not enabled or when MMIO Offset 0030h[PPRSup]=0. When IommuEn=1b and software writes PPRLogEn with 1b, the IOMMU clears the PPRLogOverflow bit and sets the PPRRun bit in the IOMMU Status Register [MMIO Offset 2020h]. The IOMMU can now write new entries to the event log if there are usable entries available. **Note:** writes to this bit are ignored when MMIO Offset 0030h[PPRSup]=0. **Note:** software can read MMIO Offset 2020h[PPRRun] to determine the status of peripheral page service request log writing by the IOMMU. **Note:** the peripheral page service request log and event log are independent. **Software note:** the PPR Log Base Address Register [MMIO Offset 0038h], the IOMMU PPR Log Head Pointer Register [MMIO Offset 2030h], and the IOMMU PPR Log Tail Pointer Register [MMIO Offset 2038h] must be set prior to enabling the event log.
12	**CmdBufEn: command buffer enable.** RW. Reset 0b. 1=Start or restart command buffer processing. When CmdBufEn=1b and IommuEn=1b, the IOMMU starts fetching commands and sets MMIO Offset 2020h[CmdBufRun] to 1b. Writing a 1b to CmdBufEn when CmdBufRun=1b has no effect. 0=Halt command buffer processing. Writing a 0 to CmdBufEn causes the IOMMU to cease fetching new commands although commands previously fetched are completed. The IOMMU stops fetching commands upon reset and after events as specified in Table 3.4. See MMIO Offset 2020h[CmdBufRun]. **Note:** see IOMMU Status Register [MMIO Offset 2020h] to determine the status of command buffer processing. **Note:** writing of event log entries is independently controlled by EventLogEn. **Software note:** the Command Buffer Base Address Register [MMIO Offset 0008h], the Command Buffer Head Pointer Register [MMIO Offset 2000h], and the Command Buffer Tail Pointer Register [MMIO Offset 2008h] must be set prior to enabling the IOMMU command buffer processor.
11	**Isoc: isochronous**. RW. Reset 0b. This bit controls the state of the isochronous bit in the HyperTransport™ read request packet when the IOMMU issues I/O page table reads and device table reads on the HyperTransport™ link. 1=Request packet to use isochronous channel. 0=Request packet to use standard channel. **Note**: Platform firmware should set this bit to 1b for processors that support the isochronous channel.
10	**Coherent: coherent**. RW. Reset 1b. This bit controls the state of the coherent bit in the HyperTransport™ read request packet when the IOMMU issues device table reads on the HyperTransport™ link. 1=Device table requests are snooped by the processor. 0=Device table requests are not snooped by the processor. See SD in Table 5.
9	**ResPassPW: response pass posted write**. RW. Reset 0b. This bit controls the state of the ResPassPW bit in the HyperTransport™ read request packet when the IOMMU issues I/O page table reads and device table reads on the HyperTransport™ link. 1=Response may pass posted requests. 0=Response may not pass posted requests.
8	**PassPW: pass posted write**. RW. Reset 0b. This bit controls the state of the PassPW bit in the HyperTransport™ read request packet when the IOMMU issues I/O page table reads and device table reads on the HyperTransport™ link. 1=Request packet may pass posted requests. 0=Request packet may not pass posted requests.

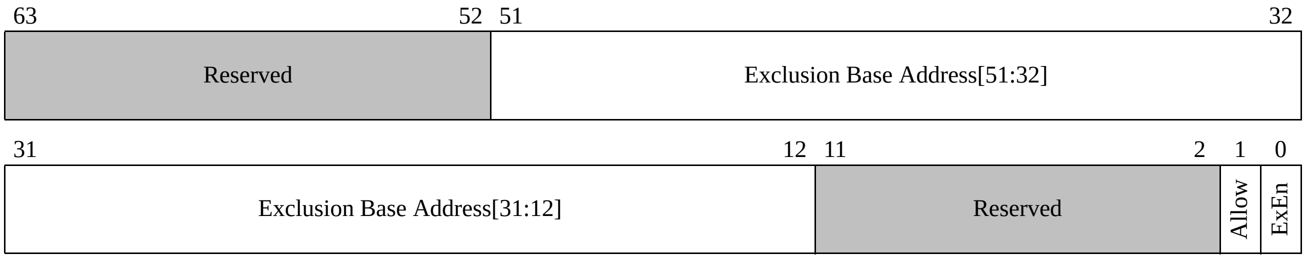
7:5	**InvTimeOut: invalidation time-out**. RW. Reset 000b. This field specifies the invalidation time-out for IOTLB invalidation requests. 000b=No time-out 001b=1 ms 010b=10 ms 011b=100 ms 100b=1 sec. 101b=10 sec. 110b=100 sec. 101b=reserved
4	**ComWaitIntEn: completion wait interrupt enable**. RW. Reset 0b. 1=An interrupt is signalled when MMIO Offset 2020h[ComWaitInt]=1 using Capability Offset 10h[MsiNum].
3	**EventIntEn: event log interrupt enable**. RW. Reset 0b. 1=An interrupt is signalled when MMIO Offset 2020h[EventLogInt]=1 or when MMIO Offset 2020h[EventOverflow]=1 using Capability Offset 10h[MsiNum].
2	**EventLogEn: event log enable**. RW. Reset 0b. 1=The Event Log Base Address Register [MMIO Offset 0010h] has been configured and all events detected are written to the event log when IommuEn has also been set. Writing a 1b to this bit when EventLogEn=1b has no effect. 0=Event logging is not enabled. Events are discarded when the event log is not enabled. When IommuEn=1b and software changes EventLogEn from 0b to 1b, the IOMMU clears the EventOverflow bit and sets the EventLogRun bit in the IOMMU Status Register [MMIO Offset 2020h]. The IOMMU can now write new entries to the event log if there are usable entries available. **Note:** software can read MMIO Offset 2020h[EventLogRun] to determine the status of event log writing by the IOMMU. **Note:** the fetching of commands is independently controlled by CmdBufEn. **Software note:** the Event Log Base Address Register [MMIO Offset 0010h], the Event Log Head Pointer Register [MMIO Offset 2010h], and the Event Log Tail Pointer Register [MMIO Offset 2018h] must be set prior to enabling the event log.
1	**HtTunEn: HyperTransport™ tunnel translation enable**. RW. Reset 0b. 1= Upstream traffic received by the HyperTransport™ tunnel is translated by the IOMMU. 0=Upstream traffic received by the HyperTransport™ tunnel is not translated by the IOMMU. The IOMMU ignores the state of this bit while IommuEn=0. See the HtTunnel bit in the IOMMU Capability Header [Capability Offset 00h].
0	**IommuEn: IOMMU enable**. RW. Reset 0b. 1=IOMMU enabled. All upstream transactions are processed by the IOMMU. The Device Table Base Address Register [MMIO Offset 0000h] must be configured by software before setting this bit. 0=IOMMU is disabled and no upstream transactions are translated or remapped by the IOMMU. When disabled, the IOMMU reads no commands and creates no event log entries. **Software note**: Revision 1: Software must configure EventLogEn and CmdBufEn. Revision 2: Software must configure EventLogEn, CmdBufEn, and PPRLogEn.

MMIO Offset 0020h IOMMU Exclusion Base Register

This register specifies the base device virtual address of the IOMMU exclusion range. Accesses that target addresses in the exclusion range are neither translated nor access checked if the EX bit in the device table is set for the device or if the Allow bit is set in this register.

A translation request for which the IOMMU exclusion range applies and I=1b in the device table entry returns R=1, W=1, and a physical address that equals the requested virtual address. The response to a multi-page translation request in the IOMMU exclusion range is implementation-specific.

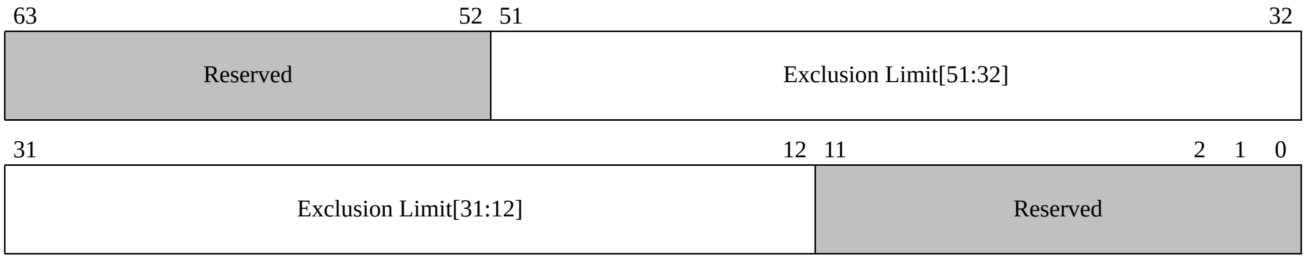
Software note: A peripheral using a remote IOTLB may cache the results of a translation request to the exclusion range, so an INVALIDATE_IOTLB_PAGES command must be issued after changing the IOMMU exclusion range.



Bits	Description
63:52	Reserved.
51:12	**Exclusion range base address**. RW. Reset 00_0000_0000h. Specifies the 4Kbyte-aligned base address of the exclusion range.
11:2	Reserved.
1	**Allow: allow all devices**. RW. Reset 0b. 1=All accesses to the exclusion range are forwarded untranslated. 0=The EX bit in the device table entry specifies if accesses to the exclusion range are translated.
0	**ExEn: exclusion range enable**. RW. Reset 0b. 1=The exclusion range is enabled. 0=the exclusion range is disabled.

MMIO Offset 0028h IOMMU Exclusion Range Limit Register

This register specifies the limit of the IOMMU exclusion range. The lower 12 bits of the limit are treated as FFFh for range comparisons.
Note: when the exclusion base address equals the exclusion limit address, the exclusion range is 4K bytes.



Bits	Description
63:52	Reserved.

51:12	**Exclusion range limit**. RW. Reset 00_0000_0000h. Specifies the 4K byte limit of the exclusion range.
11:0	Reserved.

MMIO Offset 0030h IOMMU Extended Feature Register

Revision 1: Reserved.

Revision 2: This register specifies the extended features supported by the IOMMU.
Note: when Capability Offset 00h[EFRSup]=0b, this register is Reserved and the features described by it are not supported by the IOMMU.

63 … 37	36 … 32
Reserved	PASmax[4:0]

31 … 16	15 14	13 12	11 10	9	8	7	6	5	4	3	2	1	0
Reserved	GLXSup	GATS	HATS	PCSup	HESup	GASup	IASup	EFRW	GTSup	NXSup	XTSup	PPRSup	PreFSup

Bits	Description
63:37	Reserved.
36:32	**PASmax[4:0]: maximum PASID supported**. Revision 1: RO. Reset 0h. Reserved. Revision 2: RO. Reset X_XXXXb. The maximum PASID value supported is calculated as $2^{PASmax+1}$-1. 00h=1-bit PASID 01h=2-bit PASID 02h=3-bit PASID 03h=4-bit PASID 04h=5-bit PASID 05h=6-bit PASID ... ... 0Eh=15-bit PASID 0Fh=16-bit PASID 10h=17-bit PASID 11h=18-bit PASID 12h=19-bit PASID 13h=20-bit PASID 14h-1Fh=Reserved This value is not meaningful when MMIO Offset 0030h[GTSup]=0.
31:16	Reserved.
15:14	**GLXSup**: **Guest CR3 root table level supported**. Revision 1: Reserved. RO. Reset 0b. Revision 2: RO. Reset XXb. Specifies the maximum number of levels supported in a guest CR3 root table. 00b=single-level Guest CR3 base table address translation is supported. 01b=Two-level GCR3 base address table is supported in hardware. 10b=Three-level GCR3 base address table is supported in hardware for 19- and 20-bit PASID values. 11b is reserved. The value of GLXSup is not meaningful when MMIO Offset 0030h[GTSup]=0. See Table 9.
13:12	**GATS[1:0]: Guest Address Translation Size**. Revision 1: RO. Reset 00b. Reserved. Revision 2: RO. Reset XXb. The maximum number of translation levels supported for guest address translation (GVA). This value is not meaningful when MMIO Offset 0030h[GTSup]=0. 00b=4 levels (PML4E) 01b=5 levels 10b=6 levels 11b=Reserved See also Figure 20 and Table 23.

11:10	**HATS[1:0]: Host Address Translation Size**. Revision 1: RO. Reset 00b. Reserved. Revision 2: RO. Reset XXb. The maximum number of host address translation levels supported. 00b=4 levels 01b=5 levels 10b=6 levels 11b=Reserved This field sets an implementation limit on the value of DTE[Mode] in Table 5 and sets an implementation limit on the value of Next Level in Figure 9 and Figure 10. See also Figure 33.
9	**PCSup: Performance counters supported**. Revision 1: RO. Reset 0b. Reserved. Revision 2: RO. Reset Xb. 0=no performance counters are supported. 1=performance counters are supported (see IOMMU Counter Configuration Register [MMIO Offset 4000h] and Section 3.7.2.3 [MMIO Event Counter Control Registers]).
8	**HESup. Hardware Error registers supported.** Revision 1: RO. Reset 0b. Reserved. Revision 2: RO. Reset Xb. 0=Hardware error registers do not report error information. 1=Error information is reported in hardware error registers (see I/O Hardware Event Reporting Registers [3.4.11.2]).
7	**GASup. Guest APIC supported**. Revision 1: RO. Reset 0b. Reserved. Revision 2: RO. Reset Xb. 1=Guest virtual APIC is supported. 0=Guest virtual APIC is not supported.
6	**IASup: INVALIDATE_IOMMU_ALL supported**. Revision 1: RO. Reset 0b. Reserved. Revision 2: RO. Reset Xb. 1=The INVALIDATE_IOMMU_ALL command is supported. 0=The INVALIDATE_IOMMU_ALL command is not supported and will generate an error when used.
5	**EFRW: ignored.** Revision 1: RO. Reset 0b. Reserved. Revision 2: RW. Reset Xb**.** Intended for future use. **Software note:** this field is not implemented in Revision 2 but may be defined in future revisions. Software may safely write 0h to this field in Revision 2 and should ignore the value read.
4	**GTSup: guest translations supported**. Revision 1: RO. Reset 0b. Reserved. Revision 2: RO. Reset Xb. 1=guest address translation is supported. 0=only nested address translation is supported. When GTSup=0, the following values in the DTE must be zero: GV, GLX, and GCR3 Table Root Pointer. See also MMIO Offset 0018h[GTEn].
3	**NXSup: NX supported**. Revision 1: RO. Reset 0b. Reserved. Revision 2: RO. Reset Xb. 1=no-execute protection is supported. 0=no-execute protection is not supported.
2	**XTSup: x2APIC supported**. Revision 1: Reserved. RO. Reset 0b. Revision 2: RO. Reset Xb. The interrupt remapping table is expanded to provide x2APIC interrupt information.
1	**PPRSup: peripheral page service request support**. Revision 1: RO. Reset 0b. Reserved. Revision 2: RO. Reset Xb. 1=Indicates the IOMMU handles page service request events from peripherals, the IOMMU supports the page service request queue, and that the second IOMMU interrupt can be used to signal peripheral page service request events. 0=peripheral page service requests are not supported, the page service request queue is not supported, and the PPR interrupt is not generated by the IOMMU.
0	**PreFSup: prefetch support**. Revision 1: RO. Reset 0b. Reserved. Revision 2: RO. Reset Xb. 1=Indicates IOMMU will accept PREFETCH_IOMMU_PAGES commands (see Section 3.3.6 [PREFETCH_IOMMU_PAGES]). 0=IOMMU treats PREFETCH_IOMMU_PAGES commands as invalid commands.

MMIO Offset 0038h PPR Log Base Address Register

Revision 1: Reserved.

Revision 2: This register specifies the system physical address and length of the peripheral page service request log. Peripheral requests for page service handling are converted to entries in the PPR log. This register is reserved when Capability Offset 00h[EFRSup]=0b or when MMIO Offset 0030h[PPRSup]=0b. Page service requests detected by the IOMMU are reported in the event log (see Section 3.4.2 [IO_PAGE_FAULT Event]).

If PPRLogLen or PPRLogBase is changed while the PPRLogRun=1, the IOMMU response is undefined.

63 60	59 56	55 52	51 32
Reserved	PPRLogLen	Reserved	PPRLogBase

31 12	11 0
PPRLogBase	Reserved

Bits	Description
63:60	Reserved.
59:56	**PPRLogLen: peripheral page service request log length**. RW. Reset 1000b. Specifies the length of the PPR log in power of 2 increments. The minimum size is 256 entries (4K bytes); values less than 1000b are reserved. 0000b - 0111b = Reserved 1000b = 256 entries (4K bytes) 1001b = 512 entries (8K bytes) ... 1111b = 32768 entries (512K bytes)
55:52	Reserved
51:12	**PPRLogBase: peripheral page service request log base address**. RW. Reset 00_0000_0000h. Specifies the base address of the PPR log. The base address programmed must be aligned to 4K bytes.
11:0	Reserved

MMIO Offset 0040h IOMMU Hardware Event Upper Register

Revision 1: Reserved.

Revision 2: This register contains the upper 64-bits or the most recent hardware event detected by the IOMMU.

63 60	59 32
EventCode[3:0]	First event code dependent operand[59:32]

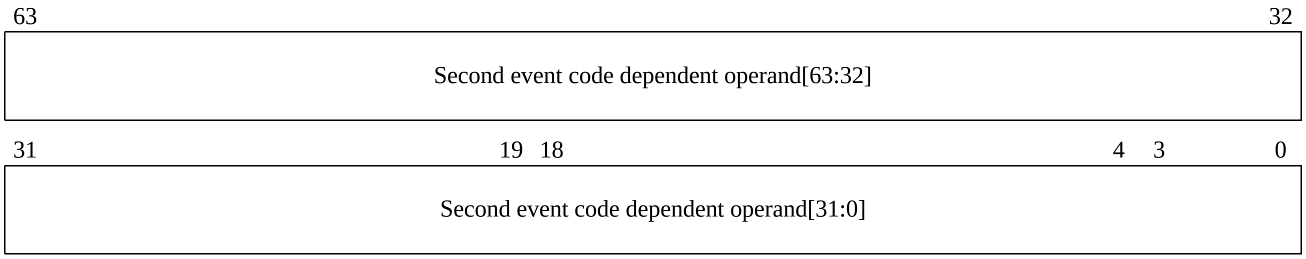
31 0
First event code dependent operand[31:0]

Bits	Description
63:60	**EventCode[3:0]**. RW. Reset 0000b. See Figure 48.
59:0	**First event code dependent operand[59:0]**. RW. Reset 000_0000_0000_0000h. See Figure 48.

MMIO Offset 0048h IOMMU Hardware Event Lower Register

Revision 1: Reserved.

Revision 2: This register contains the lower 64-bits of the most recent hardware event detected by the IOMMU.

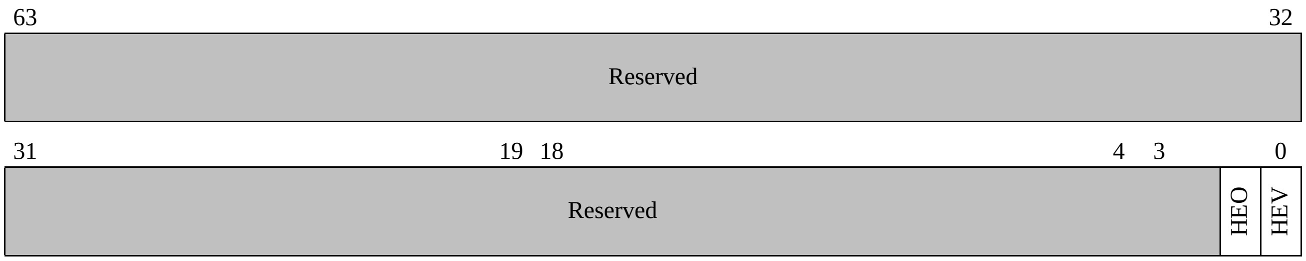

63
32
Second event code dependent operand[63:32]
31
19
18
4
3
0
Second event code dependent operand[31:0]

Bits	Description
63:0	**Second event code dependent operand[59:0]**. RW. Reset 0000_0000_0000_0000h. See Figure 48.

MMIO Offset 0050h IOMMU Hardware Event Status Register

Revision 1: Reserved.

Revision 2: This register contains the lower 64-bits of the most recent hardware event detected by the IOMMU.

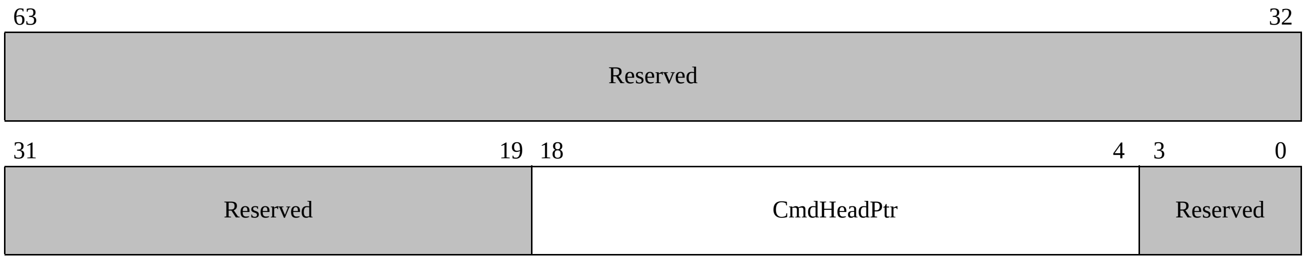


Bits	Description
63:2	Reserved.
1	**HEO: Hardware Event Overflow**. RW. Reset 0. Defines the contents of the IOMMU hardware event registers as having been overwritten. 0=not overwritten. 1=contents overwritten by new information. HEO is not meaningful when HEV=0.
0	**HEV: Hardware Event Valid**. RW. Reset 0. Defines the contents of the IOMMU hardware event registers as valid. 0=register contents not valid. 1=contents valid.

3.7.2.2 MMIO Command and Log Pointer Registers

MMIO Offset 2000h Command Buffer Head Pointer Register

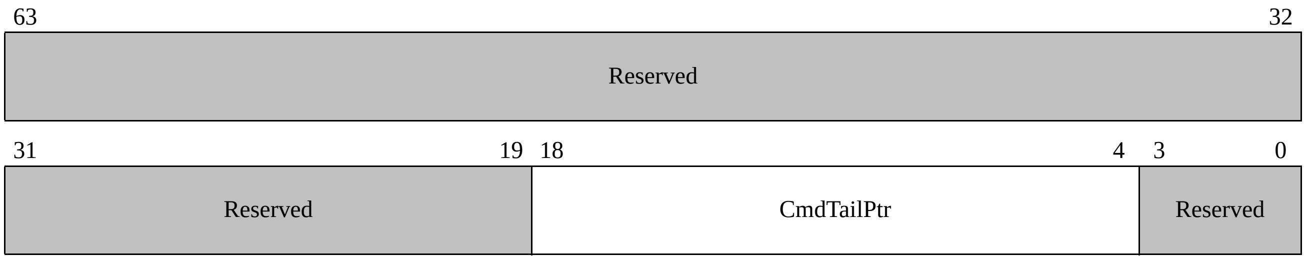
This register points to the offset in the command buffer that will be read next by the IOMMU.



Bits	Description
63:19	Reserved.
18:4	**CmdHeadPtr: command buffer head pointer**. RW. Reset 0000h. Specifies the 128-bit aligned offset from the command buffer base address register of the next command to be fetched by the IOMMU. The IOMMU increments this register, rolling over to zero at the end of the buffer, after fetching and validating the command in the command buffer. After incrementing this register, the IOMMU cannot re-fetch the command from the buffer. If this register is written to by software while CmdBufRun=1b, the IOMMU behavior is undefined. If this register is set by software to a value outside the length specified by MMIO Offset 0008h[ComLen], the IOMMU behavior is undefined.
3:0	Reserved.

MMIO Offset 2008h Command Buffer Tail Pointer Register

This register points to the offset in the command buffer that will be written next by the software.



Bits	Description
63:19	Reserved.
18:4	**CmdTailPtr: command buffer tail pointer**. RW. Reset 0000h. Specifies the 128-bit aligned offset from the command buffer base address register of the next command to be written by the software. Software must increment this field, rolling over to zero at the end of the buffer, after writing a command to the command buffer. If software advances the tail pointer equal to or beyond the head pointer after adding one or more commands to the buffer, the IOMMU behavior is undefined. If software sets the command buffer tail pointer to an offset beyond the length of the command buffer, the IOMMU behavior is undefined.
3:0	Reserved.

MMIO Offset 2010h Event Log Head Pointer Register

This register points to the offset in the event buffer that will be read next by the software.

63 32
Reserved

31 19	18 4	3 0
Reserved	EventHeadPtr	Reserved

Bits	Description
63:19	Reserved.
18:4	**EventHeadPtr: event log head pointer**. RW. Reset 0000h. Specifies the 128 bit aligned offset from the event log base address register that will be read next by software. Software must increment this field, rolling over at the end of the buffer, after reading an event from the event log. If software advances the head pointer beyond the tail pointer, the IOMMU behavior is undefined. If software sets the event log head pointer to an offset beyond the length of the event log, the IOMMU behavior is undefined.
3:0	Reserved.

MMIO Offset 2018h Event Log Tail Pointer Register

This register points to the offset in the event buffer that will be written next by the IOMMU.

63 32
Reserved

31 19	18 4	3 0
Reserved	EventTailPtr	Reserved

Bits	Description
63:19	Reserved.
18:4	**EventTailPtr: event log tail pointer**. RW. Reset 0000h. Specifies the 128-bit aligned offset from the event log base address register that will be written next by the IOMMU when an event is detected. The IOMMU increments this register, rolling over at the end of the buffer, after writing an event to the event log. If this register is written while EventLogRun=1, the IOMMU behavior is undefined. If this register is set by software to a value outside the length specified by MMIO Offset 0010h[EventLen], the IOMMU behavior is undefined
3:0	Reserved.

MMIO Offset 2020h IOMMU Status Register

This register indicates the current status of the IOMMU command and event processing. If interrupts are enabled, the IOMMU signals an interrupt when one of the interrupt status bits is set by hardware and no other interrupts status bits are set. Other bits report the status of command buffer processing and event logging.

63 … 32
Reserved

31 … 8	7	6	5	4	3	2	1	0
Reserved	PPRRun	PPRInt	PPROverflow	CmdBufRun	EventLogRun	ComWaitInt	EventLogInt	EventOverflow

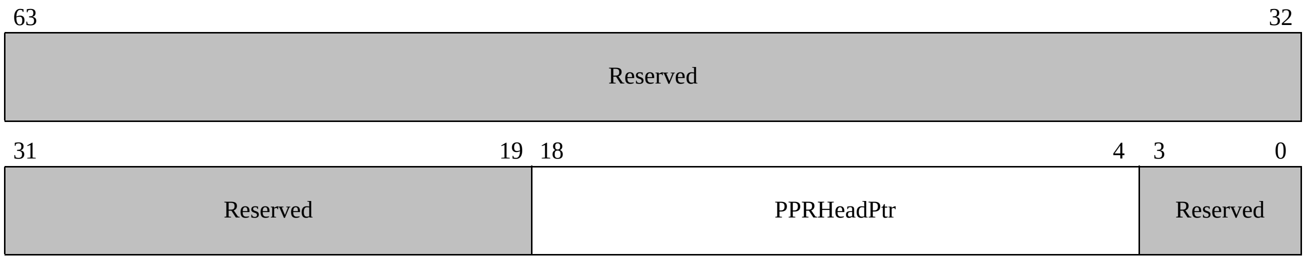
Bits	Description
63:8	Reserved.
7	**PPRRun: peripheral page service request logging is running.** Revision 1: RO. Reset 0b. Reserved. Revision 2: RO. Reset 0b. 1=PPR requests are logged as they occur. 0=PPR requests are discarded without logging. When PPROverflow=1b, the IOMMU does not write new PPR log entries even when PPRRun=1b. When halted, PPR request logging is restarted by using MMIO Offset 0018h[PPRLogEn].
6	**PPRInt: peripheral page service request interrupt**. Revision 1: RO. Reset 0b. Reserved. Revision 2: RW1C. Reset 0b. 1=PPR request entry written to the PPR log by the IOMMU. 0=No PPR entry written to the PPR log by the IOMMU. An interrupt is generated when PPRInt=1b and MMIO Offset 0018h[PPRIntEn]=1b.
5	**PPROverflow: peripheral page service request log overflow**. Revision 1: RO. Reset 0b. Reserved. Revision 2: RW1C. Reset 0b. 1=IOMMU PPR log overflow has occurred. This bit is set when a new peripheral page service request is to be written to the PPR log and there is no usable entry in the PPR log, causing the new event information to be discarded. An interrupt is generated when PPROverflow=1b and MMIO Offset 0018h[PPRIntEn]=1b (see Capability Offset 10h[MsiNumPPR]). No new PPR log entries are written while this bit is set.
4	**CmdBufRun: command buffer is running.** RO. Reset 0b. 1=the IOMMU may fetch commands from the command buffer. 0=IOMMU does not fetch commands from the command buffer. The IOMMU stops command processing after COMMAND_HARDWARE_ERROR (Section 3.4.6 [COMMAND_HARDWARE_ERROR Event]) and ILLEGAL_COMMAND_ERROR (Section 3.4.5 [ILLEGAL_COMMAND_ERROR Event]) events. When CmdBufRun=0, the IOMMU will not fetch commands until software programs MMIO Offset 0018h[CmdBufEn]. Implementation note: CmdBufRun is level-sensitive; once set to 1, it does not change to 0 until command processing stops for cause; and once set to 0, it does not change to 1 until MMIO Offset 0018h[CmdBufEn] is written with 1 by software.
3	**EventLogRun: event logging is running.** RO. Reset 0b. 1=events are logged as they occur. 0=event reports are discarded without logging. When EventOverflow=1b, the IOMMU does not write new event log entries even when EventLogRun=1b. When halted, event logging is restarted by using MMIO Offset 0018h[EventLogEn].
2	**ComWaitInt: completion wait interrupt**. RW1C. Reset 0b. 1=COMPLETION_WAIT command completed. This bit is only set if the i bit is set in the COMPLETION_WAIT command. An interrupt is generated when ComWaitInt=1b and MMIO Offset 0018h[ComWaitIntEn]=1b (see Capability Offset 10h[MsiNum]).

1	**EventLogInt: event log interrupt**. RW1C. Reset 0b. 1=Event entry written to the event log by the IOMMU. 0=No event entry written to the event log by the IOMMU. An interrupt is generated when EventLogInt=1b and MMIO Offset 0018h[EventIntEn]=1b.
0	**EventOverflow: event log overflow**. RW1C. Reset 0b. 1=IOMMU event log overflow has occurred. This bit is set when a new event is to be written to the event log and there is no usable entry in the event log, causing the new event information to be discarded. An interrupt is generated when EventOverflow=1b and MMIO Offset 0018h[EventIntEn]=1b. No new event log entries are written while this bit is set.

MMIO Offset 2030h IOMMU PPR Log Head Pointer Register

Revision 1: Reserved.

Revision 2: This register points to the offset in the peripheral page service request log entry that will be read next by the software.

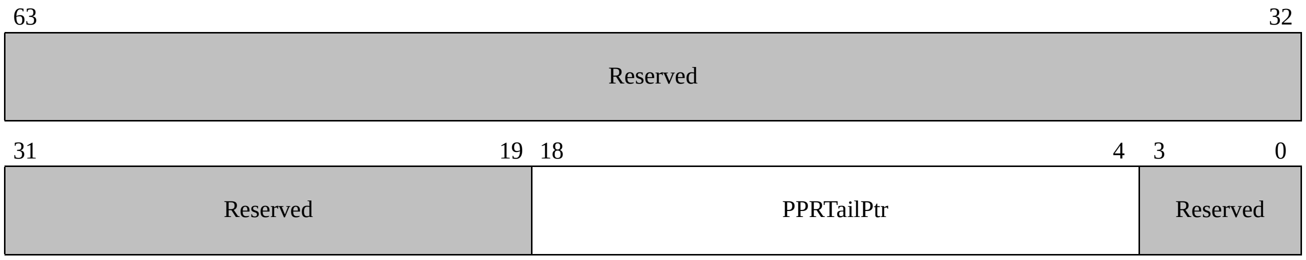


Bits	Description
63:19	Reserved.
18:4	**PPRHeadPtr: peripheral page service log head pointer**. RW. Reset 0000h. Specifies the 128-bit aligned offset from the PPR log base address register that will be read next by software. Software must increment this field, rolling over at the end of the buffer, after reading a PPR request entry from the PPR event log. If software advances the head pointer beyond the tail pointer, the IOMMU behavior is undefined. If software sets the PPR log head pointer to an offset beyond the length of the PPR log, the IOMMU behavior is undefined.
3:0	Reserved.

MMIO Offset 2038h IOMMU PPR Log Tail Pointer Register

Revision 1: Reserved.

Revision 2: This register points to the offset in the peripheral page service request log that will be written next by the IOMMU.



Bits	Description
63:19	Reserved.
18:4	**PPRTailPtr: peripheral page service log tail pointer**. RW. Reset 0000h. Specifies the 128-bit aligned offset from the PPR log base address register that will be written next by the IOMMU when a PPR request is detected. The IOMMU increments this register, rolling over at the end of the buffer, after writing a PPR request to the PPR log. If this register is written while PPRLogRun=1, the IOMMU behavior is undefined. If this register is set by software to a value outside the length specified by MMIO Offset 0038h[PPRLen], the IOMMU behavior is undefined
3:0	Reserved.

3.7.2.3 MMIO Event Counter Control Registers

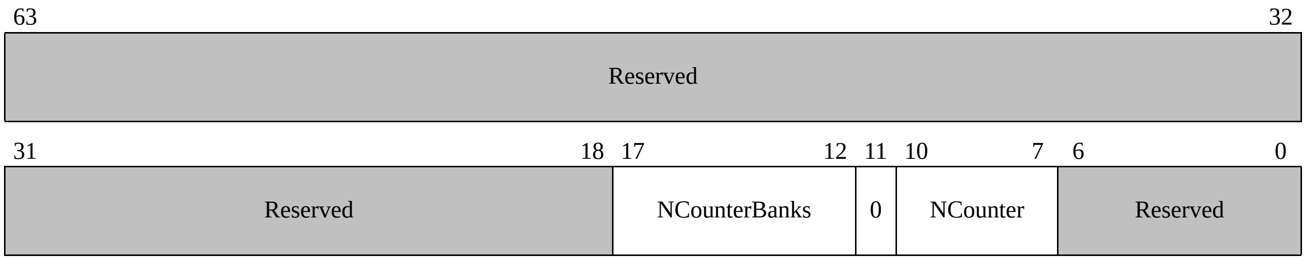
Revision 1: The next set of registers, MMIO Offset 4000h through MMIO Offset 4018h, are reserved.

Revision 2: The next set of registers, MMIO Offset 4000h through MMIO Offset 4018h, control the IOMMU event counters.

MMIO Offset 4000h IOMMU Counter Configuration Register

Revision 1: Reserved.

Revision 2: This register reports the type and number of counters available to software.

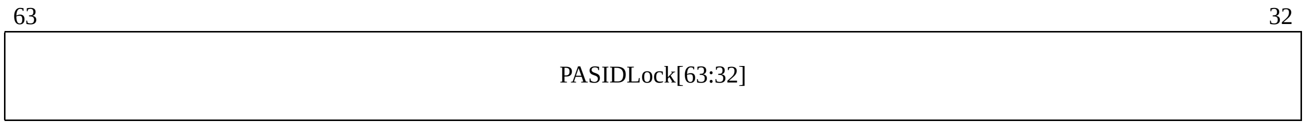


Bits	Description
63:18	Reserved.
17:12	**NCounterBanks[5:0]: number of IOMMU counter banks**. RO. Reset XXh. The number of counter banks supported by the IOMMU. Each bank contains two or more counter register and control registers as specified by NCounter. For each counter bank, a corresponding control bit is in IOMMU Counter PASID Bank-Lock Register [MMIO Offset 4008h],IOMMU Counter Domain Bank-Lock Register [MMIO Offset 4010h], and IOMMU Counter DeviceID Bank-Lock Register [MMIO Offset 4018h]. Each supported event counter bank is in a distinct, consecutive 4K byte page. The limit of 63 counter banks is architectural and an implementation may set a lower value. 0=No event counter banks supported. 1-63=The number of event counter banks supported. **Note**: IOMMU event counter banks are numbered starting with 0.
11	Reserved.
10:7	**NCounter[3:0]: number of counters per counter bank**. RO. Reset Xh. Reports the number of individual counters in each IOMMU counter bank. Each counter bank contains the same number of counters. 0=No counters supported. 1=Reserved. 2-15=number of counters in each counter bank.
6:0	Reserved.

MMIO Offset 4008h IOMMU Counter PASID Bank-Lock Register

Revision 1: Reserved.

Revision 2: This register locks the corresponding PASID-match register, IOMMU PASID Match Register [MMIO Offset [40-7F][0-F]10h]. When a PASID-match register is locked, the register can be read but writes are ignored.



31	0
PASIDLock[31:0]	

Bits	Description
63:0	**PASIDLock: PASID lock enable**. RW. Reset 0. 0=Corresponding counter bank of PASID-match registers is unlocked. 1=locked (writes are ignored). For each bit in PASIDLock, the corresponding PASID-match registers in an IOMMU counter bank may be changed. See IOMMU PASID Match Register [MMIO Offset [40-7F][0-F]10h]. Bit positions above the value reported in MMIO Offset 4000h[NCounterBanks] are ignored when written and return zero when read. The counter banks are numbered starting with zero; PASIDLock[0] controls bank 0, etc. **Software note**: this register should be managed by trusted software.

MMIO Offset 4010h IOMMU Counter Domain Bank-Lock Register

Revision 1: Reserved.

Revision 2: This register locks the corresponding Domain-match counter bank registers, IOMMU Domain Match Register [MMIO Offset [40-7F][0-F]18h]. When a Domain-match register is locked, the register can be read but writes are ignored.

63	32
DomainLock[63:32]	

31	0
DomainLock[31:0]	

Bits	Description
63:0	**DomainLock: Domain lock enable**. RW. Reset 0. 0=Corresponding counter bank of Domain-match registers is unlocked. 1=locked (writes are ignored). For each bit in DomainLock, the corresponding Domain-match registers in an IOMMU counter bank may be changed. See IOMMU Domain Match Register [MMIO Offset [40-7F][0-F]18h]. Bit positions above the value reported in MMIO Offset 4000h[NCounterBanks] are ignored when written and return zero when read. The counter banks are numbered starting with zero; DomainLock[0] controls bank 0, DomainLock[1] controls bank 1, etc. **Software note**: this register should be managed by trusted software.

MMIO Offset 4018h IOMMU Counter DeviceID Bank-Lock Register

Revision 1: Reserved.

Revision 2: This register locks the corresponding DeviceID-match counter bank registers, IOMMU DeviceID Match Register [MMIO Offset [40-7F][0-F]20h]. When a DeviceID-match register is locked, the register can be read but writes are ignored.

63	32
DevIDLock[63:32]	

31 … 0
DevIDLock[31:0]

Bits	Description
63:0	**DevIDLock: DeviceID lock enable**. RW. Reset 0. 0=Corresponding counter bank of DeviceID-match registers is unlocked. 1=locked (writes are ignored). For each bit in DevIDLock, the corresponding DeviceID-match registers in an IOMMU counter bank may be changed. See IOMMU DeviceID Match Register [MMIO Offset [40-7F][0-F]20h]. Bit positions above the value reported in MMIO Offset 4000h[NCounterBanks] are ignored when written and return zero when read. The counter banks are numbered starting with zero; DevIDLock[0] controls bank 0, etc. **Software note**: this register should be managed by trusted software.

3.7.2.4 MMIO Event Counter Configuration Registers

Revision 1: The next set of registers, MMIO Offset [40-7F][0-F]00h through MMIO Offset [40-7F][0-F]28h, are reserved.

Revision 2: The next set of registers, MMIO Offset [40-7F][0-F]00h through MMIO Offset [40-7F][0-F]28h, are spaced at 4K byte page boundaries. There are a variable number of counter registers and counter register banks implemented as specified in MMIO Offset 4000h[NCounterBanks, NCounter]. The MMIO addresses are decoded as shown in Figure 65. The base address is the IOMMU MMIO base address defined by IOMMU Base Address Low Register [Capability Offset 04h] and IOMMU Base Address High Register [Capability Offset 08h]. Note that the use of IOMMU event counters affects the value programmed into IOMMU Base Address Low Register [Capability Offset 04h].

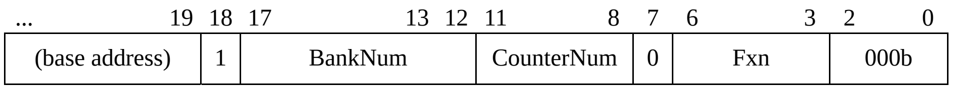


Figure 65: IOMMU Counter Register Address Decode

Table 63: Counter Bank Addressing (MMIO)

Bits	Description
63:19	MMIO base address of IOMMU counter registers.
18	1.
17:12	**BankNum: Bank number**. Selects counter bank. The maximum value is defined by MMIO Offset 4000h[NCounterBanks].
11:8	**CounterNum: Counter number**. Selects counter within bank. The maximum value is defined by MMIO Offset 4000h[NCounter].
7	Must be zero.
6:3	**Fxn: Function**. This field selects the functional register within the counter set. +00h: IOMMU Counter Register [MMIO Offset [40-7F][0-F]00h]. +08h: IOMMU Counter Source Register [MMIO Offset [40-7F][0-F]08h]. +10h: IOMMU PASID Match Register [MMIO Offset [40-7F][0-F]10h]. +18h: IOMMU Domain Match Register [MMIO Offset [40-7F][0-F]18h]. +20h: IOMMU DeviceID Match Register [MMIO Offset [40-7F][0-F]20h]. +28h: IOMMU Counter Report Register [MMIO Offset [40-7F][0-F]28h]. +30h through +78h: Reserved.
2:0	Byte alignment.

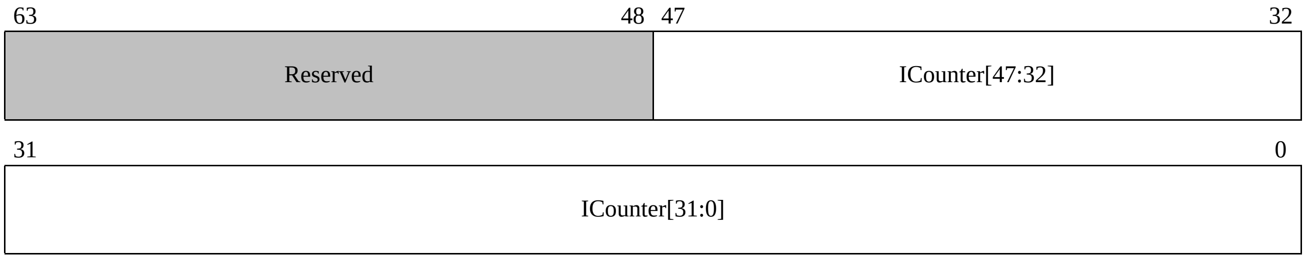
Software note: because each counter bank is aligned to a 4 Kbyte page, the counter banks can be assigned to different guests for direct access after programming IOMMU Counter PASID Bank-Lock Register [MMIO Offset 4008h], IOMMU Counter Domain Bank-Lock Register [MMIO Offset 4010h], and IOMMU Counter DeviceID Bank-Lock Register [MMIO Offset 4018h].

.

MMIO Offset [40-7F][0-F]00h IOMMU Counter Register

Revision 1: Reserved.

Revision 2: This register counts events as programmed by IOMMU Counter Source Register [MMIO Offset [40-7F][0-F]08h] and IOMMU PASID Match Register [MMIO Offset [40-7F][0-F]10h]. When the ICounter value increments to zero, an event is optionally written to the event log (see IOMMU Counter Report Register [MMIO Offset [40-7F][0-F]28h] and Section 3.4.10 [EVENT_COUNTER_ZERO Event]) and the counter continues incrementing. To cause an interrupt at a threshold value, software must set ICounter to the 2’s complement of the desired threshold value.

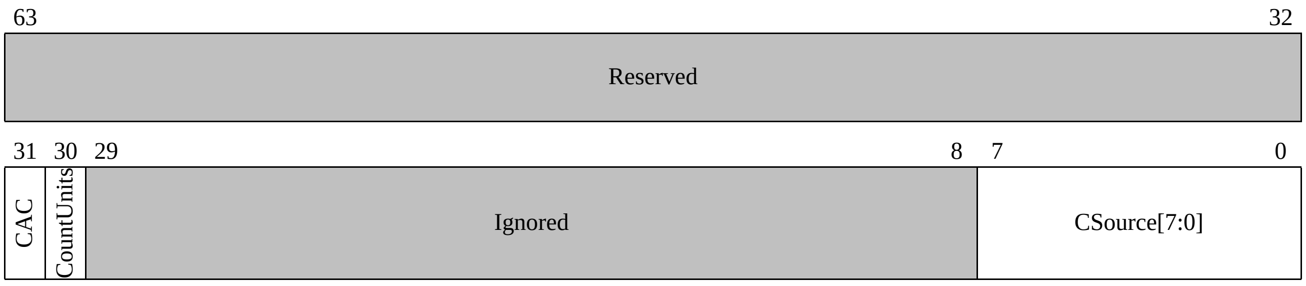


Bits	Description
63:48	Reserved.
47:28	**ICounter**. RW. Reset 0. Reports the counter value. The counter counts up continuously, wrapping at the maximum value and continuing to count. There is no overflow indicator.

MMIO Offset [40-7F][0-F]08h IOMMU Counter Source Register

Revision 1: Reserved.

Revision 2: This register selects an event source for the corresponding counter.



Bits	Description
63:32	Reserved.
31	**CAC: counter source architectural or custom**. RW. Reset 0. Selects architectural counter input group (Table 64) or custom input group. 0=architectural counters as defined in Table 64. 1=implementation-defined counters. **Software note:** Unless otherwise specified, selecting a counter marked Reserved returns undefined results.

30	**CountUnits**. RW. Reset 0. 0=Counter counts events (level). 1=Counter counts clocks (edges). Meaningful when CAC=0; implementation-specific when CAC=1.
29:8	Ignored. When CAC=0, writes to this field are ignored and reads return 0. When CAC=1, this field is implementation-specific.
7:0	**CSource: counter source**. RW. Reset 0. Selects event counter input from the choices in Table 64 when CAC=0; selects an implementation-specific counter input when CAC=1.

Table 64: Architectural Counter Input Group, CAC=0b

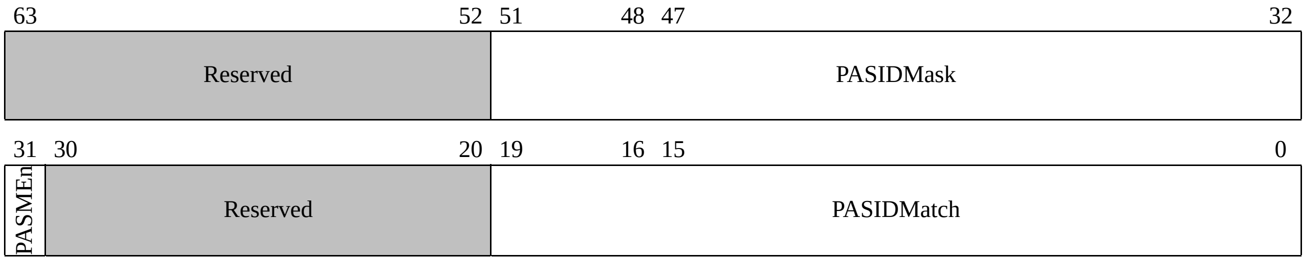
Value (CSource)	Architectural Counter Input Group Selection
0	No events. **Note**: CountUnits=0 stops the counter and CountUnits=1 is a free-run counter.
1	Peripheral memory operations passed-through, untranslated.
2	Peripheral memory operations passed-through, pretranslated.
3	Peripheral memory operations passed-through, via Exclusion Range.
4	Peripheral memory operations target aborted.
5	Peripheral memory operations translated, total.
6	Peripheral memory operations translated, IOMMU TLB hit PTE.
7	Peripheral memory operations translated, IOMMU TLB missed PTE.
8	Peripheral memory operations translated, IOMMU TLB hit PDE.
9	Peripheral memory operations translated, IOMMU TLB missed PDE.
10	Peripheral memory operations, DTE cache hit.
11	Peripheral memory operations, DTE cache miss.
12	IOMMU page table read operations due to memory translation, total.
13	IOMMU page table read operations due to memory translations, nested.
14	IOMMU page table read operations due to memory translations, guest.
15	Peripheral interrupt operations remapped, DTE cache hit.
16	Peripheral interrupt operations remapped, DTE cache miss.
17	IOMMU commands processed (total).
18	IOMMU commands processed, invalidations (total).
19	IOMMU TLB invalidations (total).
20-255	Reserved (treated as CSource=0 and CountUnits=0).

The IOMMU Counter Register [MMIO Offset [40-7F][0-F]00h] is incremented when IOMMU PASID Match Register [MMIO Offset [40-7F][0-F]10h], IOMMU Domain Match Register [MMIO Offset [40-7F][0-F]18h], and IOMMU DeviceID Match Register [MMIO Offset [40-7F][0-F]20h] match or are ignored.

MMIO Offset [40-7F][0-F]10h IOMMU PASID Match Register

Revision 1: Reserved.

Revision 2: This register contains the PASID filter mask and the PASID for which to count events in the corresponding counter register. The incoming PASID is ANDed with the PASIDMask field and the result is compared to the PASIDMatch field. If the comparison result is the same value, the event is enabled to be counted in the corresponding counter.

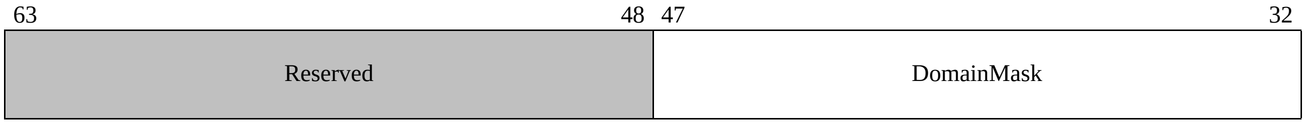


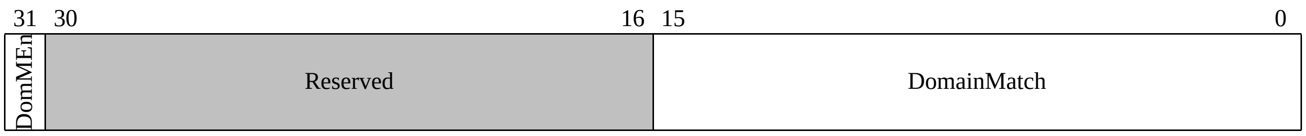
Bits	Description
63:52	Reserved
51:32	**PASIDMask**. RW. Reset 0. This bit-mask is ANDed with the PASID of the transaction to decide to count the corresponding event. 0=count events for all values of incoming PASID. 0_0001h-F_FFFFh=bit-wise mask ANDed with incoming PASID.
31	**PASMEn: PASID match enable**. RW. Reset 0. 0=PASID is ignored. 1=Filtered PASID must match to count an event. An event with no PASID tag is only counted when PASMEn=0.
30:20	Reserved.
19:0	**PASIDMatch**. RW. Reset 0. This value is compared to the masked (filtered) value of the incoming PASID of the transaction to decide to count the corresponding event. The event is counted if PASIDMatch is equal to the masked incoming PASID; the event is not counted if they are not equal.

MMIO Offset [40-7F][0-F]18h IOMMU Domain Match Register

Revision 1: Reserved.

Revision 2: This register contains the Domain filter mask and the Domain for which to count events in the corresponding counter register. The incoming Domain is ANDed with the DomainMask field and the result is compared to the DomainMatch field. If the comparison result is the same value, the event is enabled to be counted in the corresponding counter.



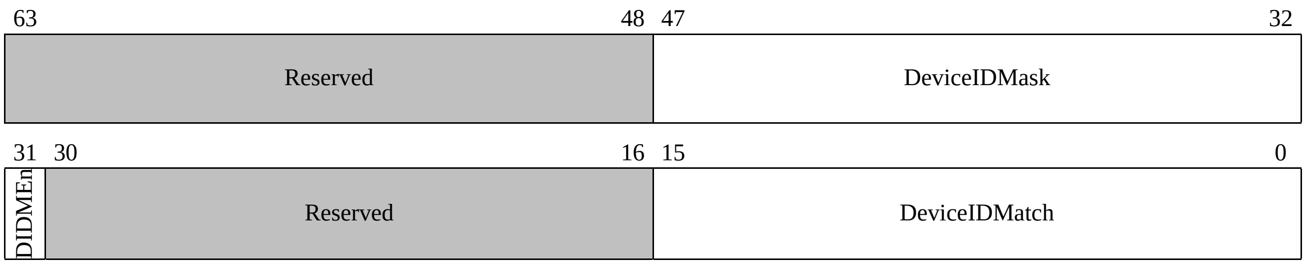


Bits	Description
63:48	Reserved.
47:32	**DomainMask**. RW. Reset 0. This bit-mask is ANDed with the Domain of the transaction to decide to count the corresponding event. 0=count events for all values of incoming Domain. 0001h-FFFFh=bit-wise mask ANDed with incoming Domain.
31	**DomMEn: Domain match enable**. RW. Reset 0. 0=Domain is ignored. 1=Filtered Domain must match to count an event.
30:16	Reserved.
15:0	**DomainMatch**. RW. Reset 0. This value is compared to the masked (filtered) value of the incoming Domain of the transaction to decide to count the corresponding event. The event is counted if DomainMatch is equal to the masked incoming PASID; the event is not counted if they are not equal.

MMIO Offset [40-7F][0-F]20h IOMMU DeviceID Match Register

Revision 1: Reserved.

Revision 2: This register contains the DeviceID filter mask and the DeviceID for which to count events in the corresponding counter register. The incoming DeviceID is ANDed with the DeviceIDMatch field and the result is compared to the DeviceIDMatch field. If the comparison result is the same value, the event is enabled to be counted in the corresponding counter.

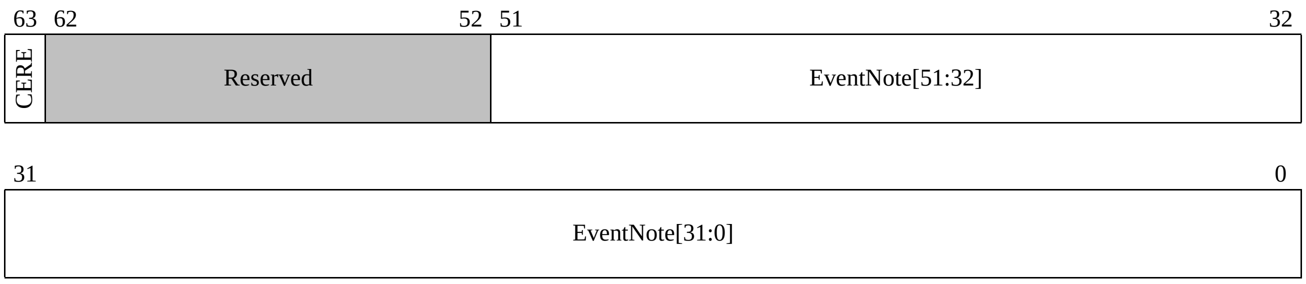


Bits	Description
63:48	Reserved.
47:32	**DeviceIDMask**. RW. Reset 0. This bit-mask is ANDed with the DeviceID of the transaction to decide to count the corresponding event. 0=count events for all values of incoming DeviceID. 0001h-FFFFh=bit-wise mask ANDed with incoming DeviceID.
31	**DIDMEn: DeviceID match enable**. RW. Reset 0. 0=DeviceID is ignored. 1=Filtered DeviceID must match to count an event.
30:16	Reserved.
15:0	**DeviceIDMatch**. RW. Reset 0. This value is compared to the masked (filtered) value of the incoming DeviceID of the transaction to decide to count the corresponding event. The event is counted if DeviceIDMatch is equal to the masked incoming DeviceID; the event is not counted if they are not equal.

MMIO Offset [40-7F][0-F]28h IOMMU Counter Report Register

Revision 1: Reserved.

Revision 2: This register hold information for the optional event log entry generated when the event counter wraps to zero. The counters continue to count after they wrap to zero.



Bits	Description
63	**CERE: Counter Event Report Enable**. RW. Reset 0. 0=no event report when counter wraps to zero. 1=IOMMU writes an EVENT_COUNTER_ZERO event log entry when the counter wraps to zero. The counter-wrap event is treated like any other event (see Section 3.4 [Event Logging]). **Software note**: the counter-wrap event is delivered promptly but without a latency guarantee.
62:52	Reserved.
51:0	**EventNote**. RW. Reset 0_0000_0000_0000h. When CERE=1 and the corresponding counter is incremented and wraps to zero, EventNote[51:0] is reported in the EVENT_COUNTER_ZERO event log entry (see Section 3.4.10 [EVENT_COUNTER_ZERO Event]).

4 Implementation Considerations

This chapter discusses issues that are primarily of concern to IOMMU implementers.

The IOMMU specification is intended to allow a wide range of implementations with different cost and performance trade-offs. Potential implementation technology may range from ASIC to full custom. Capacity and organization of the IOMMU's translation caches can vary substantially depending on technology, die budgets, and product requirements. The IOMMU can be integrated with a chipset (typically as part of some existing HyperTransport™ bridge) or built as a standalone component (which can act as a HyperTransport™ bridge or tunnel).

4.1 Caching and Invalidation Strategies

All IOMMU implementations should have some form of translation cache that allows the IOMMU to determine the disposition of device accesses quickly without having to re-walk the IOMMU tables for each separate device access. The translation cache is likely to be the largest portion of the IOMMU's die area budget in all but the smallest implementations. Consequently the IOMMU specification has been written to allow flexibility in the design of the translation cache.

Plausible implementations range from direct mapped RAM structures to fully associative CAM structures, with the expectation that most implementations are set associative. Furthermore, implementers may choose to flatten the multi-stage IOMMU table walk into a single cache array lookup, or, alternatively, may choose to use a similar multi-stage organization for internal translation cache lookups.

The IOMMU's translation cache must support the following operations:

- Lookup — when the IOMMU processes an access by a particular device to a specified device virtual address, it applies protection checks and translation transformations using information obtained using DeviceID and device virtual address.
- Invalidate device — discard any translation cache contents that depend on a specific device table entry.
- Invalidate virtual address (within domain) — discard any cached translations for a virtual address within the specified domain.

Typical IOMMU implementations are likely to be built with ASIC design flows, where CAM cells are expensive compared to RAM cells. The main implication of this is that direct support for different page sizes is likely to require a combination of separate arrays and/or multiple entries within arrays, causing both fills and invalidations to require time-consuming search-and-destroy algorithms.

The IOMMU is designed to support three main usage models:

- Direct user process access to a single device like a graphics controller;
- Direct virtual machine guest access to a collection of devices that have been dedicated to that guest; and
- A single non-virtualized OS using the IOMMU to enforce device to system memory access controls.

When a user process directly controls a single device, the total memory footprint for the device's accesses is likely to be a modest fraction of the process's own memory footprint. Moreover, the user process has direct knowledge of the specific device, so there is a good chance that the device's access pattern is controllable for good locality. In this case the main consideration for achieving performance is to ensure that the IOMMU translation cache is large enough.

By contrast, the potential memory footprint of a virtual machine guest's devices is the entire memory of the guest. Often the access pattern may be poorly controlled, as determined by the guest operating system's workload (of which the VMM likely has no specific knowledge), and, moreover, consists of interactions with a variety of devices under the control of different guest device drivers and subsystems, with diverse memory allocation strategies. In the case of a non-paravirtualized guest, a VMM's strategy for improving performance is probably to set up I/O page tables using the largest available page size and assume that the IOMMU can share the same translation cache entries among multiple devices. It is for this reason that the IOMMU table structure includes a DomainID that can be shared for multiple DeviceIDs: since the IOMMU uses translation cache entries tagged by *{DomainID, I/O virtual address}* it automatically shares translations among multiple devices assigned to the same domain.

Based on these considerations, designers should consider a two-stage organization for the IOMMU translation cache:

- The first stage should map *DeviceID* to {*DomainID, I/O page table base address}*. Most systems have only a few distinct DeviceIDs, so the capacity of the first stage can be small. The one complication is that DeviceIDs are not very random and tend to be clustered, so, to avoid conflicts, this stage should either be highly associative or use a good DeviceID hash function.
- The second stage should map *{DomainID, device virtual address*} to *{system physical address, protection}*. This stage should have (at least) hundreds of entries. This stage should explicitly include the DomainID in set index hashing (rather than just using the DomainID as a tag), so that different domains with similar memory layouts do not compete for the same translation cache entries. (Server consolidation environments are likely to create many domains with very similar memory layouts.)

In addition, since the latency of IOMMU access to system memory can be high, implementers should consider a *page directory cache (PDC)* to accelerate processing of translation cache misses. This cache should map *{DomainID, device virtual address}* to *page directory entry (PDE)*, so that the IOMMU can quickly calculate the address of the final PTE needed to resolve a translation cache miss. This way, most translation cache misses can be resolved in a single memory access by the IOMMU, rather than requiring a full multi-stage table walk. The page directory cache could also double as a large-page translation cache, since for large pages the PDE is also the PTE.

4.2 IOMMU Topologies

The IOMMU's architecture is designed to accommodate a variety of system fabrics and topologies. There can be multiple IOMMUs, located at a variety of places in the system fabric. Some requestor ID information can be lost at bridges between busses or bus types, so it is advantageous to locate IOMMUs in bridges. The mapping of bus requesterIDs to IOMMU DeviceIDs depends on both the bus type as well as the IOMMU's location in the system fabric. In most other respects, the IOMMU's behavior is bus-independent.

The most basic implementation of the IOMMU takes the form of a HyperTransport™ tunnel.

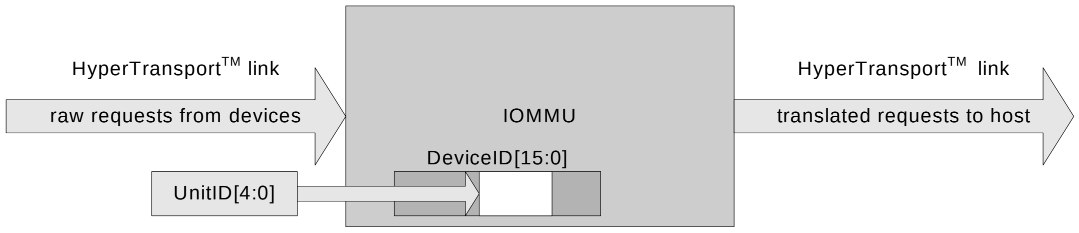


Figure 66: IOMMU in a Tunnel

The advantage of this approach is that it can be easily retrofitted to an existing system design. The main limitation of this approach is that the HyperTransport™ specification defines only 5 bits of UnitID information to identify the originators of requests, so the IOMMU can provide distinct translations for at most 31 downstream devices. If downstream nodes include any bridges, the IOMMU is unable to distinguish between different devices beyond the bridges, since bridged requests use the UnitID of the bridge. A possible solution is to include a separate IOMMU on each downstream bus; each IOMMU can then be programmed not to rewrite transactions whose UnitID proves they have already passed through another IOMMU. Software must understand the system topology to correctly coordinate multiple IOMMUs. If a downstream HyperTransport™ device is a PCIe® root complex or a PCI-X® host bridge, the device can implement the RequesterID mapping capability to assign specific UnitIDs to PCIe® or PCI-X® devices.

An IOMMU implemented in a PCIe®- or PCI-X®-to-HyperTransport™ bridge can exploit the larger PCIe® or PCI-X® RequesterID namespace to provide better discrimination between downstream devices when translating requests:

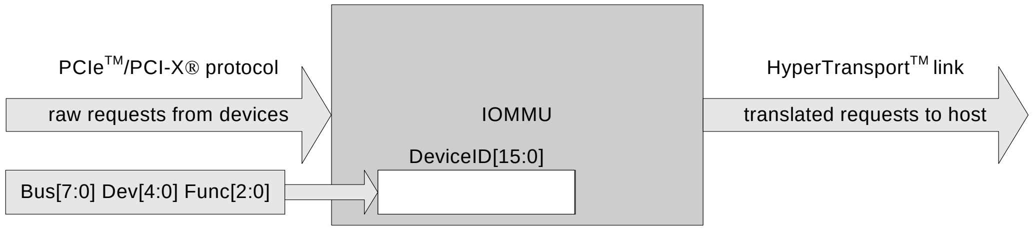

PCIe™/PCI-X® protocol
raw requests from devices
IOMMU
HyperTransport™ link
translated requests to host
DeviceID[15:0]
Bus[7:0] Dev[4:0] Func[2:0]

Figure 67: IOMMU in a Peripheral Bus Bridge

Since most future commodity devices are expected to be on a PCIe® bus, this is likely to be the most common implementation of the IOMMU for low-cost systems.

Large systems may want a scalable IOMMU building block. Such systems may choose to implement a hybrid HyperTransport™ tunnel / PCIe® root complex component or a HyperTransport™ tunnel / PCI-X® host bridge component combining the above ideas:

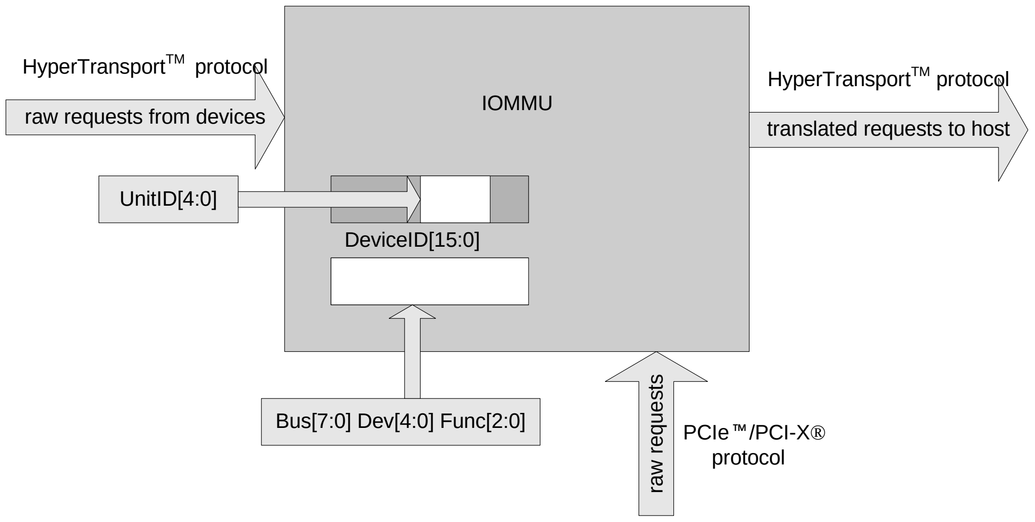


Figure 68: Hybrid IOMMU

Hybrid IOMMUs can be chained together to build large systems:

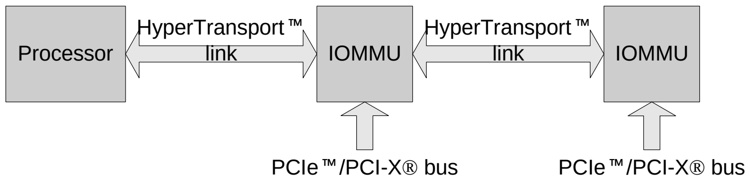


Figure 69: Chained Hybrid IOMMU in a Large System

4.3 Issues Specific to the HyperTransport™ Architecture

This section discusses implementation considerations that are specific to IOMMUs attached to a HyperTransport™ link.

The HyperTransport™ specification requires devices (especially tunnels and bridges) to interoperate with other devices in ways that ensure correctness and maintain performance. Among other requirements, HyperTransport™ devices must make certain transaction ordering guarantees and must ensure they operate without deadlocks.

A key requirement in the HyperTransport™ specification is that posted requests must be able to pass non-posted requests. The introduction of the IOMMU, however, means that posted requests (e.g. writes to memory) may spawn non-posted requests (I/O page table walks) that must complete before the posted request can be allowed to progress further.

To avoid deadlocks, the IOMMU requires a dedicated virtual channel for its I/O page table walk requests. This ensures that, the IOMMU’s page table walks on behalf of posted requests can complete, regardless of the completion status of other non-posted traffic in the fabric. The IOMMU also requires that the host bridge process its requests without spawning any requests to other devices. In other words, the IOMMU’s table structures must be located solely in system memory.

The IOMMU can share its virtual channel with other traffic as long the other traffic is also guaranteed to make forward progress. In practice, this means that any other devices sharing the IOMMU’s page walk channel must also restrict their non-posted traffic solely to accessing system memory.

To allow the IOMMU to support different AMD processors with different isochronous capabilities the IOMMU control registers contain bits that control the state of the PassPW bits, the coherent bit and the isochronous bit in the HyperTransport™ link read request issued by the IOMMU.

4.4 Chipset Specific Implementation Issues

Chipsets that implement both an IOMMU and a legacy PCI or AGP bridge must provide source identification to identify uniquely DMA traffic as originating from the PCI or AGP bus.To provide this identification, the IOMMU must use the requesterID of the PCI or AGP bridge to perform translations for DMA transactions from the legacy bus.

4.5 Software and Platform Firmware Implementation Issues

Because of the flexible architecture of the IOMMU, it is unlikely that any single system software implementation uses all the features, topologies, or options. The following constraints are strongly recommended:

- An IOMMU should be a root-complex device (i.e., appear directly on the bus at the top of the PCI tree

hierarchy).
- Some system software may prohibit an IOMMU from appearing under a PCI-to-PCI bridge.
- To ensure the IOMMU is recognized and configured properly, the platform firmware should perform the initial configuration of the IOMMU so that it is accessible to system software when control is handed off by the platform firmware.
- The platform firmware should describe the IOMMU in an ACPI table as defined in Section 5 [I/O Virtualization ACPI Tables]. The table must include all information necessary to identify, configure, and access the IOMMU.
- System firmware must ensure the IOMMU configuration is preserved or restored across power-management state changes.

5 I/O Virtualization ACPI Tables

This section defines the ACPI tables to describe the platform configuration information for IOMMU control fields. Key features of the I/O virtualization ACPI tables are listed here:
1. The I/O virtualization ACPI tables describe the IOMMUs present in a system with their attributes.
2. The I/O virtualization ACPI tables describe the system I/O topology relevant to the IOMMU.
3. The I/O virtualization ACPI tables identify any peripheral devices that cannot be otherwise enumerated.
4. The I/O virtualization ACPI tables override the information available in IOMMU hardware registers. System software is required to honor the ACPI settings.
5. The I/O virtualization ACPI tables prescribe access to memory regions used by SMI/SMM, platform firmware, and platform hardware. These are generally exclusion ranges to be configured by system software.

The I/O virtualization ACPI tables are created in memory by the platform firmware.

5.1 IOMMU Control Flow

The IOMMU start-up procedure flows through several stages. In general, the IOMMU PCI configuration space is initially configured by the platform firmware and later the primary operational manipulations are done by system software. Some settings are programmed into the IOMMU hardware at design time (e.g., virtual address size, physical address size, MSI interrupts). Certain hardware features can be overridden by the platform designer and so are defined in ACPI settings (e.g., the exclusion range, remote IOTLB support), such as when certain features are not included in platform qualification testing or are reserved for use by the platform firmware.

At system reset, the IOMMU returns to a default state. Following system reset, platform firmware is able to program essential platform-specific information into the IOMMU, mostly through the PCI configuration space registers (e.g., MMIO base address). Some additional settings are made by the platform firmware in the MMIO space (e.g., tunnel enable) while other settings are made by system software in the MMIO space (e.g., the coherent bit). Finally, system software must initialize and manage IOMMU control and operational tables allocated in memory (e.g., the device tables). Some of these control and operational settings must be configured according to policies determined by the platform firmware, so they are communicated to system software via the ACPI tables (e.g., selected interrupt controls, system management controls).

Once configuration is complete, the IOMMU is enabled by system software and begins processing transactions from peripherals. From this point, the IOMMU is under the control of the system software.

5.2 Future Work

Although this specification allows the placement of IOMMU translation units outside the root complex, current platform implementations are cautioned against such designs. This document does not define the ACPI methods or data structures necessary to hot-plug a peripheral controller containing an IOMMU.

5.3 IOMMU ACPI Table Definitions

There are three types of data blocks that may defined in IOMMU ACPI tables:
1. I/O Virtualization Reporting Structure (IVRS),
2. I/O Virtualization Hardware Definition (IVHD), and
3. I/O Virtualization Memory Definition (IVMD).

Each is described in detail in the following sections.

The IOMMU ACPI tables inform system software which IOMMU will service DMA operations and interrupts from the DeviceIDs possible in the system, whether the DeviceID is actually populated or not. The IVHD data block reports ranges when hot-plug and SR-IOV devices are possible. Each DeviceID is described by one IVHD entry which may be a select or select-alias record or is part of a range or an alias range. If a given DeviceID exists but can generate neither DMA nor interrupts (ever), it need not be listed in the IOMMU ACPI tables. A DeviceID used as an alias must be included in a IVHD record (either “select” or “range”). The simplest IOMMU ACPI table contains one DeviceID range for each IOMMU; a system with one IOMMU may report as little as a single range covering all DeviceIDs (0x0000-0xFFFF).

5.3.1 I/O Virtualization Reporting Structure (IVRS)

There is one I/O Virtualization Reporting Structure (IVRS) in a system that contains an IOMMU.

Table 65: I/O Virtualization Reporting Structure (IVRS)

Byte Offset	**I/O Virtualization Reporting Structure (IVRS)**							
00	Signature (“IVRS”)				Length (bytes)			
08	Revision	Check	OEM ID					
16	OEM Table ID							
24	OEM Revision				Creator ID			
32	Creator Revision				IVinfo			
40	Reserved (0000_0000_0000_0000h)							
--	**I/O Virtualization Definition Blocks (IVHD, IVMD)**							
48	(IVHD, IVMD)							
Offset:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**

Table 66: IVRS fields

IVRS field name	**Offset**	**Size (bytes)**	**Value**	**Definition**
Signature	00	4	“IVRS”	I/O Virtualization Reporting Structure (ASCII)
Length	04	4	Length in bytes	Length in bytes of device reporting structure in bytes, including IVHD and IVMD structures
Revision	08	1	1h	IVRS revision number.
Check	09	1		Checksum of entire structure must equal zero
OEM ID	10	6		Identifies platform OEM
OEM Table ID	16	8		Specified by OEM
OEM Revision	24	4		Specified by OEM
Creator ID	28	4		Vendor ID of the utility that created the table
Creator Revision	32	4		Revision of the utility that created the table
IVinfo	36	4		I/O virtualization information common to all IOMMU units in a system
Reserved	40	8	0	Reserved for future use; must be zero
(varies)	48+			I/O Virtualization Definition blocks: IVHD or IVMD

Software note: IOMMU Revision 2 hardware implementations are backwards-compatible with IOMMU Revision 1, so the IVRS Revision field should not be used to distinguish between Revision 1 and Revision 2 implementations.

Table 67: IVRS IVinfo fields

IVinfo field name	Bits	Definition
Reserved	31:23	Must be zero.
HtAtsResv	22	ATS address translation range reserved.
VAsize	21:15	Virtual address size (common). Revision 2 note: This is the guest physical address (GPA) size.
PAsize	14:8	Physical address size (common). Revision 2 note: This is the system physical address (SPA) size.
GVAsize	5:7	Revision 1: Reserved (must be zero). Revision 2: Guest virtual address size (common).
Reserved	4:0	Must be zero.

5.3.2 I/O Virtualization Definition Blocks

There are two types of data blocks that can be passed from platform firmware to system software in the I/O Virtualization ACPI table: I/O Virtualization Hardware Definition (IVHD) blocks, and I/O Virtualization Memory Definition (IVMD) blocks. A set of definition blocks begins with an IVHD, optionally followed by IVMD blocks. Each IVMD blocks pertains to the IVHD block that precedes it.

5.3.3 I/O Virtualization Hardware Definition (IVHD) Block

The I/O Virtualization ACPI entry must contain at least one I/O Virtualization Hardware Definition (IVHD) block to specify key parameters of the IOMMU being described. The Bus and Device/Function (UnitID) specify the IOMMU defined by the IVHD block; the Capability Offset is required in case the function implements multiple IOMMU capabilities. An IVHD block is required for each IOMMU in the platform. The flags are initially programmed into the IOMMU register fields by the platform firmware. They are communicated in the IVHD to the system software for reference when changes are made to the IOMMU registers by system software.

All peripherals that can generate transactions processed by an IOMMU must be defined in the ACPI tables. The IVHD block entries describe the I/O topology (start and end of a range, or single entries) of I/O devices and slots served by an IOMMU.

Implementation note: all DeviceID values served by an IOMMU must be reported in an IVHD record including DeviceIDs not yet populated (including, but not limited to, virtual functions and empty hot-plug slots).

The IVHD device entry also specifies the setting of IOMMU device table entry fields for the DeviceID identified in the IVHD. The IVHD entries must always be aligned to 8-bytes, so a "pad" entry of 0000h should be used to fill out a short entry. There must be at least one IVHD device entry to describe at least one I/O device or slot governed by the IOMMU; an IOMMU may govern multiple ranges and singletons of I/O devices. The IVHD device entry defining the start of a range must come before the IVHD entry defining the end of the range, and the two entries must be adjacent in the ACPI IVHD table. For peripherals that use source identification other than their own DeviceID, alias entries must be used. An IOMMU and the peripherals it serves must be on the same PCI Segment Group defined in the IVHD block. At this time, only PCI Segment Group 0 is supported. The IVHD length field specifies the number of bytes in the IVHD block, starting from the Type field. In Table 68, there must be at least one IVHD entry and there may be many.

Software note: an indication of support for the performance counter feature can be derived from the IOMMU EFR values (see Figure 70 and Table 72). When IOMMU_EFR[PNBanks]=0 and IOMMU_EFR[PNCounters]=0, then the performance counter feature is not supported (see MMIO Offset 0030h[PCSup]).

Table 68: I/O Virtualization Hardware Definition (IVHD) fields

Byte offset	I/O Virtualization Hardware Definition (IVHD) block								Relative offset
48	Type (10h)	IVHD flags	Length		DeviceID		Capability offset		+0
56	IOMMU base address								+8
64	PCI Segment Group		IOMMU info		Reserved / IOMMU EFR				+16
72	IVHD device entry (4-byte)				...				+24
Offset:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**-**

Table 69: IVHD type definitions

IVHD field name	Offset	Size (bytes)	Value	Definition
Type	48	1	10h	I/O virtualization hardware definition block
Flags	49	1		Settings for selected IOMMU control fields (see Table 70)
Length	50	2	(bytes)	Size of IVHD in bytes, including IVHD device entries and starting from Type field
DeviceID	52	2		DeviceID of IOMMU
Capability offset	54	2		Offset in Capability space for control fields of IOMMU
IOMMU base address	56	8		Base address of IOMMU control registers in MMIO space
PCI Segment Group	64	2	0000h	PCI Segment Group number
IOMMU info	66	2		Interrupt numbers and UnitID (see Table 71)
IOMMU EFR	68	4		Revision 1: Reserved for future use; must be zero. Revision 2: Extended Feature Report (see Table 72).
IVHD device entries	72+	4, 8, 16, or 32		IVHD device entries (see Section 5.3.5 [IVHD 4-byte Device Entries], Section 5.3.6 [IVHD 8-byte Device Entries]).

Table 70: IVHD flag fields

IVHD flags	Bits	Definition
PPRSup	7	Defines peripheral page service support to system software (see MMIO Offset 0030h[PPRSup]).
PreFSup	6	Defines PREFETCH_IOMMU_PAGES support to system software (see MMIO Offset 0030h[PreFSup]).
Coherent	5	Recommended setting for Coherent control bit to system software (see MMIO Offset 0018h[Coherent]). The recommended value is 1b.
IotlbSup	4	Defines remote IOTLB support to system software (see Capability Offset 00h[IotlbSup]).
Isoc	3	Recommended setting for Isoc control bit to system software (see MMIO Offset 0018h[Isoc]).
ResPassPW	2	Recommended setting for ResPassPW to system software (see MMIO Offset 0018h[ResPassPW]).
PassPW	1	Recommended setting for PassPW to system software (see MMIO Offset 0018h[PassPW]).
HtTunEn	0	Recommended setting for HtTunEn to system software (see MMIO Offset 0018h[HtTunEn]).

Table 71: IVHD IOMMU info fields

IVHD IOMMU info fields	Bits	Definition
Reserved	15:13	Reserved.
UnitID	12:8	Unit ID number (see Capability Offset 0Ch[UnitID])
Reserved	7:5	Reserved.
MSInum	4:0	MSI message number for event log (see Capability Offset 10h[MSInum])

31 30	29 28	27 23	22 17	16 13	12 8	7	6	5	4 3	2		0
HATS	GATS	MsiNumPPR	PNBanks	PNCounters	PASmax	HESup	GASup	IASup	GLXSup	GTSup	NXSup	XTSup

Figure 70: IVHD IOMMU Extended Feature Report fields format

Table 72: IVHD IOMMU Extended Feature Report fields

IVHD IOMMU EFR fields	Bits	Definition
HATS	31:30	Revision 1: Reserved. Revision 2: Host address translation size (see MMIO Offset 0030h[HATS]).

IVHD IOMMU EFR fields	Bits	Definition
GATS	29:28	Revision 1: Reserved. Revision 2: Guest address translation size (see MMIO Offset 0030h[GATS]). This value must be zero when MMIO Offset 0030h[GTSup]=0.
MsiNumPPR	27:23	Revision 1: Reserved. Revision 2: MsiNumPPR for peripheral page service requests (see Capability Offset 00h[MsiNumPPR]); must be 0_0000b when PPRSup=0.
PNBanks	22:17	Revision 1: Reserved. Revision 2: Number of performance counter banks (see MMIO Offset 4000h[NCounterBanks]).
PNCounters	16:13	Revision 1: Reserved. Revision 2: Number of performance counters per counter bank (see MMIO Offset 4000h[NCounter]).
PASmax	12:8	Revision 1: Reserved. Revision 2: Maximum PASID value supported (see MMIO Offset 0030h[PASmax]). Must be ignored if PPRSup=0.
HESup	7	Revision 1: Reserved. Revision 2: Hardware Error Registers supported (Section 3.4.11.2 [I/O Hardware Event Reporting Registers]).
GASup	6	Revision 1: Reserved. Revision 2: AMD Virtual Interrupt Controller supported (see MMIO Offset 0030h[GASup]).
IASup	5	Revision 1: Reserved. Revision 2: INVALIDATE_IOMMU_ALL supported (see MMIO Offset 0030h[IASup]).
GLXSup	4:3	Revision 1: Reserved. Revision 2: Number of guest CR3 tables supported (see MMIO Offset 0030h[GLXSup]).
GTSup	2	Revision 1: Reserved. Must be zero. Revision 2: Guest translation supported (see MMIO Offset 0030h[GTSup]).
NXSup	1	Revision 1: Reserved. Must be zero. Revision 2: NX supported for I/O (see MMIO Offset 0030h[NXSup]).
XTSup	0	Revision 1: Reserved. Must be zero. Revision 2: x2APIC supported for peripherals (see MMIO Offset 0030h[XTSup]).

5.3.4 IVHD device entry sizes

Device entries in the IVHD table can be 4-, 8-, 16-, or 32-bytes long. The two uppermost bits of the device type define the entry length.

Table 73: IVHD device entry length codes

IVHD device entry type range (decimal)	IVHD device entry type range (hexadecimal)	Uppermost 2-bits	IVHD device entry length (bytes)
0-63	00h-3Fh	00b	4 bytes
64-127	40h-7Fh	01b	8 bytes
128-191	80h-BFh	10b	16 bytes
192-255	C0h-FFh	11b	32 bytes

IVHD entries can be used to describe one or more PCI bus-device-function (BDF) or HyperTransport™ bus-unit addresses. When IVHD entries describe a range of addresses, the DeviceID address is treated as if it were 16-bit integer so that, for example, DeviceID 0100h (Bus 1, Device 0, Function 0) follows DeviceID 00FFh (Bus 0, Device 31, Function 7), and DeviceID 0518h (Bus 5, Device 3, Function 0) follows DeviceID 0517h (Bus 5, Device 2, Function 7).

5.3.5 IVHD 4-byte Device Entries

The 4-byte IVHD device entry is structured to contain a single DeviceID with related IVHD flags. A 4-byte IVHD device entry must be aligned to a 4-byte boundary.

Table 74: IVHD device entry fields (4-byte)

Byte offset:	**+0**	**+1**	**+2**	**+3**
Byte contents:	Device entry type	DeviceID		Data setting

Table 75: IVHD device entry type codes (4-byte)

Byte 0: IVHD device entry type (4-byte)	**Bytes 1 & 2: DeviceID**	**Byte 3: Device entry data setting**	**Entry Definition** (see Table 76)
0	0000h	00h	4-byte pad (use for alignment)
1 (All)	(ignored)	Data setting	Data setting applies to all DeviceIDs controlled by the IOMMU.
2 (Select)	DeviceID of entry	Data setting	Data setting applies to selected DeviceID.
3 (Start of range)	DeviceID of entry	Data setting	Data setting applies to all devices from start of range (inclusive). Range is terminated with entry type of 4 (end of range).
4 (End of range)	DeviceID of entry	Reserved	Previous data setting (from start of range or alias start of range) applies to all peripherals to the end of range (inclusive). The data setting is reserved and must be zero.
5-63 (05h-3Fh)	Reserved	Reserved	Reserved.

Table 76: IVHD device entry data setting fields

IVHD device entry data setting fields	**Bits**	**Definition**
Lint1Pass	7	Identifies a device able to assert LINT1 interrupts
Lint0Pass	6	Identifies a device able to assert LINT0 interrupts
SysMgt[1:0]	5:4	Identifies a device able to assert System Management messages (e.g. VID/FID)
Reserved	3	Reserved; must be zero.
NMIPass	2	Identifies a device able to assert NMI interrupts
EIntPass	1	Identifies a device able to assert ExtInt interrupts
INITPass	0	Identifies a device able to assert INIT interrupts

The fields contained in Table 76 are defined in Table 5. Extended data settings are defined in Table 78.

5.3.6 IVHD 8-byte Device Entries

The 8-byte IVHD device entry is used to convey more information than a single DeviceID address. Fields in an 8-byte IVHD device table entry marked as Reserved or Res must be zero. An 8-byte IVHD device entry must be aligned to a 8-byte boundary.

Table 77: IVHD device entry type codes (8-byte)

Byte 0: IVHD device entry type (8-byte)	**Bytes 1 & 2**	**Byte 3** (see Table 76)	**Byte 4**	**Bytes 5 & 6**	**Byte 7**	**Entry Definition**
64 (40h)	0000h	00h	00h	0000h	00h	8-byte pad (use for alignment)
65	0000h	00h	00h	0000h	00h	Reserved
66 (Alias select)	*DeviceID (a):* Actual peripheral DeviceID	*Data setting:* IVHD device entry data setting	00h	*DeviceID (b):* DeviceID used as source by peripheral	00h	Data setting applies to DeviceID (a); peripheral uses DeviceID (b) as source identification information.
67 (Alias start of range)	*DeviceID (a):* Actual peripheral DeviceID	*Data setting:* IVHD device entry data setting	00h	*DeviceID (b):* DeviceID used as source by peripheral	00h	Data setting applies to all peripherals from start of range (inclusive); all peripherals in range use DeviceID (b) as source identification information. Range is terminated with an IVHD entry type 4 (end of range). Note that all peripherals in the range use the same DeviceID (b).
68-69	0000h	00h	00h	0000h	00h	Reserved
70 (Extended select)	*DeviceID* of entry	*Data setting*	Extended data setting (see Table 78)			Data setting and extended data setting apply to selected DeviceID (see Table 78).
71 (Extended start of range)	*DeviceID* of entry	*Data setting*	Extended data setting (see Table 78)			Data setting and extended data setting apply to all devices from start of range (inclusive). Range is terminated with an IVHD entry type 4 (end of range); see Table 78.
72 (Special Device)	0000h	*Data setting*	*Handle:* see Table 79	*DeviceID:* DeviceID used as source by peripheral	*Variety*: see Table 79	Device not normally identified through enumeration. Variety encodes IOAPIC or HPET, and *Handle* contains the I/O APIC ID or the HPET Number, respectively.
73-127	0000h	00h	0000_0000h			Reserved

An Alias device type entry is used for each peripheral that does not use its own DeviceID information in bus transactions. For example, peripherals downstream of a bridge device that use the DeviceID of the bridge must have a corresponding Alias Select or Alias Start of Range entry to inform system software which IOMMU device table entry will be used for translation information.

When an Alias device type entry is used, the IVHD block cannot contain a device type entry of type 1 (ALL), 2 (Select), or 3 (Start of Range) that includes the same peripherals. When the (type 67, type 4) IVHD type pair is used to define a range, all the included peripherals use the same DeviceID (b) as a DeviceID and thus the same IOMMU device table entry. An extended entry of type 70 or type 71 is used when the extended attributes in Table 78 must be expressed.

Table 78: IVHD device entry extended data setting fields

IVHD device entry extended data setting fields	Bits	Definition
AtsDisabled	31	Identifies an I/O device that must be prevented from issuing address translation requests. 1b=block ATS requests; 0b=allow ATS.
Reserved	30:0	Reserved; must be zero.

A peripheral not identified in the normal enumeration process requires a Special Device table entry of type 72. The variety of the peripheral and the associated tag are provided as show in Table 79.

Table 79: IVHD special device entry - variety and handle fields

IVHD special device entry variety	Variety value	Handle definition
IOAPIC	01h	The I/O APIC ID from the APCI MADT.
HPET	02h	The HPET Number from the HPET table.
Reserved	00h, 03h-FFh	Reserved.

5.3.7 16-byte Device Entries

All 16-byte device entry type values (type 128-191) are reserved and may not be used. A 16-byte IVHD device entry must be aligned to a 16-byte boundary.

5.3.8 32-byte Device Entries

All 32-byte device entry type values (type 192-255) are reserved and may not be used. A 32-byte IVHD device entry must be aligned to a 32-byte boundary.

5.3.9 I/O Virtualization Memory Definition (IVMD) Block

Platform firmware may have memory usage requirements to communicate to system software based on its needs or on hardware characteristics. Platform firmware can inform system software of memory usage restrictions or requirements by using I/O Virtualization Memory Definition (IVMD) blocks. Each IVMD entry may be per-device, specifying the DeviceID to which the entry applies, or the IVMD entry may apply to all devices and the DeviceID is ignored. System software is expected to use the information in the IVMD blocks when it programs the IOMMU.

Table 80: I/O Virtualization Memory Definition (IVMD) format

Byte offset	**I/O Virtualization Memory Definition (IVMD) block**								**Relative offset**
0	Type	Flags	Length		DeviceID		Auxiliary data		+0
8	Reserved (0000_0000_0000_0000h)								+8
16	IVMD start address								+16
24	IVMD memory block length								+24
Offset:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**-**

The IVMD block fields are defined in Table 81. Note that a memory definition block may apply to a particular peripheral device, multiple peripheral devices, or all peripheral devices. When a memory block is defined for multiple peripheral devices, but not all, the IVMD definition is repeated for each discontiguous peripheral device or range to which the memory definition applies. If IR=0b and IW=0b in the IVMD flags field, then the memory range is not to be mapped into the peripheral device address space and the unity flag field must be 0b. To prevent a memory range from ever being mapped into any peripheral device address space, use IR=0b and IW=0b in the IVMD flags field and IVMD type=20h (all devices).

Table 81: IVMD fields

IVMD field name	**Offset**	**Size (bytes)**	**Value**	**Definition**
IVMD type	0	1	20h=all peripherals; 21h=specified peripheral; 22h=peripheral range	I/O virtualization memory definition block
IMVD Flags	1	1	(see Table 82)	Flags for memory block
IVMD length	2	2	32	Length of IVMD entry in bytes

DeviceID	4	2		Type 20h: field reserved;
				type 21h: DeviceID;
				type 22h: starting DeviceID of range

Auxiliary data	6	2		Types 20h, 21h: field reserved; type 22h: ending DeviceID of range (inclusive)
Reserved	8	8	0	Reserved
Start address (physical)	16	8		System-physical address of start of memory block

IVMD field name	Offset	Size (bytes)	Value	Definition
Memory block length	24	8	Length in bytes	Length of memory block; system software may round up to 4K byte boundary

Table 82: IVMD flags definitions

IVMD Flags fields	Bits	Definition
Reserved	7:4	Reserved; must be zero.
ExclusionRange	3	Exclusion range. 1b=included in exclusion range, 0b=not in exclusion range. **Note:** IR, IW, and Unity are ignored when ExclusionRange=1b.
IW	2	Write permission. 1b=writeable, 0b=not writeable.
IR	1	Read permission. 1b=readable, 0b=not readable.
Unity	0	Unit address mapping. 1b=virtual addresses must be the same value as physical addresses. 0b=any virtual address translation may be used.

The IVMD flag field applies to individual devices when IVMD type=21h, to all devices in a system when IVMD type=20h, and to all devices in the DeviceID range when IVMD type=22h.

6 IOMMU Pseudo Code

6.1 IOMMU Page Walker Pseudo Code

```
//
// Page table walker for IOMMU.
// -- for single-level page tables.
//
// Inputs:
// {dte} is partial device table entry (lower 64-bits),
// {dva} is device virtual address,
// extern int errno assumed.
//
// Return value is a (possibly synthetic) 64-bit "pte" suitable for storing
// in a TLB, with the following fields valid:
//    [62]       (cumulative) I/O write permission
//    [61]       (cumulative) I/O read permission
//    [60]       FC bit
//    [59]       U bit
//    [51:12]    system physical page address
//    [5:0]      how many VA bits to append
//
// The caller of this routine is responsible for read and write permission
// checks, and for checks that the dte is valid for translation.
// The caller is responsible for special access permission check in the
// case of a 0-length read.
// This routine performs all other checks, and exits by raising an
// exception (instead of returning a value) if any problem is found.
//

#define LARGEST_VA(LEVEL)    ((0x1000ull << ((LEVEL) * 9)) - 1)
#define VABITS(LEVEL)            (((LEVEL) * 9) + 3)

#define IOPERM      0x6000000000000000       // 6000_0000_0000_0000h - IR and IW bits in PTE
#define RESV_BITS 0x1FF0000000000000      // 1FF0_0000_0000_0000h - Reserved bits in PTE
#define U_FC_BITS 0x1800000000000000       // 1800_0000_0000_0000h - U, FC bits in PTE
#define BITS_51_12 0xFFFFFFFFFF000         // 000F_FFFF_FFFF_F000h - bit mask [51:12]

uint64
iopagewalk(uint64 dte, uint64 dva)
{
   uint64 pdte = dte;
   uint64 ioperm = pdte & IOPERM;
   uint64 pa = pdte & BITS_51_12;           // dte bits [51:12]
   uint oldlevel = 7, level = (pdte >> 9) & 7, vabits = 63;

   if (level == 7)
      raise DEVTAB_RESERVED_LEVEL;

   if (level == 0)
      return ioperm | pa | vabits;
```

```
    while (level != 0) {
        uint64 skipbits = LARGEST_VA(oldlevel - 1) - LARGEST_VA(level);
        if ((dva & skipbits) != 0)
            raise PAGE_NOT_PRESENT;
        uint offset = (dva >> (level * 9)) & 0xFF8;
        pdte = read_memory_qword(pa + offset);
        if ((errno == E_MEM_ERROR) & (MMIO_Offset_0030h[HESup])) {
            update_IOMMU_error_registers();
            errno=0;
            append_event_log(event_info);
            if ((errno == E_MEM_ERROR) & (MMIO_Offset_0030h[HESup])) {
                update_IOMMU_error_registers();
                event_logging(off);
            }
            raise PAGE_TAB_HARDWARE_ERROR;
        }
        if ((pdte & 1) == 0)
            raise PAGE_NOT_PRESENT;

        oldlevel = level;
        level = (pdte >> 9) & 7;
        uint64 reserved_bits = RESV_BITS;
        if (level == 0 || level ==7) {
            reserved_bits &= ~U_FC_BITS;  // U and FC bits are not reserved for PTEs
            ioperm |= pdte & U_FC_BITS;    // merge U and FC bits into result
        }
        if ((pdte & reserved_bits) != 0)
            raise PDTE_RESERVED_BITS;
        ioperm &= pdte;
        pa = pdte & BITS_51_12;           // pte bits [51:12]
        oldlevel = level;
        level = (pdte >> 9) & 7;
        if (level == 0x7) {
            uint64 tmp = pa ^ BITS_51_12;
            vabits = bsf(tmp) + 1;                    // find first 0 in pa[51:12] - see BSF instruction
            if ((vabits >= VABITS(oldlevel + 1)) || (vabits <= VABITS(oldlevel)))
                raise PDTE_RESERVED_BITS;
            pa &= ~((1ull << vabits) - 1);
            return ioperm | pa | vabits
        }
        if (level >= oldlevel)
            raise PDTE_RESERVED_BITS;
    }

    if ((pa & LARGEST_VA(oldlevel - 1) != 0)
        raise PDTE_RESERVED_BITS;

    return ioperm | pa | VABITS(oldlevel);
}
```

6.2 Clear Accessed Bit

This is discussed in Section 3.2.7.5 [Clearing Accessed and Dirty Bits].

```
void
reset_a(uint64 virtual_address, list_t DeviceID_list)   // reset A bit in PTE using a simple algorithm
{
#define AMASK    0x20                                  // Accessed bit in PTE
   pte_t * pte_p = va_to_ptep(virtual_address);       // get pointer to PTE for virtual address
   pte_t    pte_temp;
   did_t    DeviceID;
   int      completion_pending;

   pte_temp = *pte_p;                                 // take sample of PTE, including A bit
   if (pte_temp & AMASK){                             // A==1 means page recently accessed
      lock_and64((uint64)(~AMASK), pte_p);            // interlocked AND to update PTE in memory
      foreach(DeviceID in DeviceID_list)
         IOMMU_enqueue_command(INVALIDATE_IOMMU_PAGES, DeviceID, virtual_address);
      completion_pending = 1;
      IOMMU_enqueue_command(COMPLETION_WAIT, &completion_pending, 0);
      while(completion_pending)   ;                   // spinlock
      // invalidation completed
      page_was_accessed(virtual_address);
   }                                                  // end if
}                                                     // end reset_a()
```

7 Register List

The following is a list of all storage elements, context, and registers provided in this document. Page numbers, register mnemonics, and register names are provided.