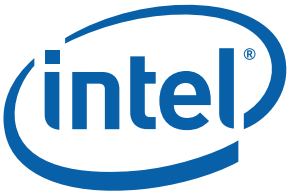


Intel® Virtualization Technology for Directed I/O

Architecture Specification

November 2017

Order Number: D51397-009, Rev. 2.5

Contents

Figures

Tables

Revision History

Date	Revision	Description
March 2006	Draft	• Preliminary Draft Specification
May 2007	1.0	• 1.0 Specification
September 2007	1.1	• Specification update for x2APIC support
September 2008	1.2	• Miscellaneous documentation fixes/clarifications, including BIOS support for NUMA, hot-plug
February 2011	1.3	• Fixed documentation errors; Added BIOS support to report X2APIC_OPT_OUT
January 2012	2.0	• Updated chapter 8 (BIOS requirements) to comprehend platforms with ACPI devices capable of generating DMA requests (such as Low Power Subsystem (LPSS) on client platforms).
August 2013	2.1	• Extended page group request with a stream response requested flag to request stream responses for page requests except the last request in group. • Added an Blocked-On-Fault field to page requests requesting stream response as a hint to indicate the respective fault caused a blocking condition on the endpoint device. • Clarified hardware behavior on page requests received when page request queue is full.
September 2013	2.2	• Added support for Shared Virtual Memory (SVM) capability. • Fixed ANDD structure definition in DMAR ACPI table to support 2-byte length field. • Fixed invalidation granularity encoding for extended IOTLB invalidation descriptor. • Updated bit positions of fields in PASID-State table entry.
October 2014	2.3	• Added support for Interrupt Posting capability support. • Clarified specific registers whose read completions are required to drain various types of interrupt requests generated by the remapping hardware. • Fixed typo in effective memory-type computation for first-level paging entry accesses when nested translations are enabled with Extended Memory Type disabled in second-level translation tables. • Fixed Page Request Status Register and Page Request Event Control Register descriptions to clarify that queueing of any page_req_desc in the page request queue results in hardware setting the Pending Page Request (PPR) field. • Fixed Supervisor Request Enable (SRE) field location from Extended-context-entry to PASID-entry, to distinguish privileged versus non-privileged PASIDs of a device. • Fixed Extended Access Flag Enable (EAFE) field location from PASID-entry to Extended-Context-entry. • Relaxed context-entry programming considerations to clarify software requirement to ensure self-consistency when modifying present root, extended-root, context or extended-context entries. • Reserved Translation Type (TT) field encoding of 110b and 111b in extended-context-entries (previously documented incorrectly as PASID-only translation types).
June 2016	2.4	• Fixed location of PASID Support enumeration field in ECAP_REG from bit28 to bit 40. • Fixed typo in Section 4.2.3 to clarify that for translation-requests-with-PASID with PR=1, remapping hardware supporting supervisor-requests (SRS=1) return PRIV bit as always 1. Previous versions of the spec. incorrectly specified hardware returning PRIV bit as 1 only if the U/S field is 0 in at least one of the first-level paging-structure entries controlling the translation. • Clarified the ordering requirement to be followed by remapping hardware on page request descriptor writes and recoverable fault reporting event interrupt. • Updated Chapter 6 to include Device-TLB invalidation throttling support for SR-IOV devices. New Device-TLB Invalidation Throttling (DIT) capability field added to ECAP_REG. • Updated Chapter 6 to include a new Page-request Drain (PD) flag in inv_wait_dsc for page request draining. • Updated Chapter 7 to include details on page request and page response ordering and draining, including handling of terminal conditions on device with pending page faults. • Added ECAP_REG capability fields to report support for Device-TLB invalidation throttling and page-request draining. • (contd.)

Date	Revision	Description
June 2016	2.4	• (contd.) • Clarified Caching Mode (CM=1) behavior to indicate that the reserved Domain-ID of 0 is used only for context-cache and rest of the caching structures follow same tagging for cached entries for CM=0 and CM =1(including for cached faulting entries when CM=1).
November 2017	2.5	• Updated specification to include support for 5-level paging structures for first-level and second-level translation. Use of 5-level paging for first-level translation is controlled through an explicit enable per PASID-entry. Use of 5-level paging for second-level translation is controlled through the programming of already existing Address Width (AW) field in the context/extended-context entry. New capability bit is added to report support for 5-level paging for first-level translation. Added FL-PML5E and SL-PML5E documentation. • Clarified the contents of address field in Fault Recording register when hardware supports multiple paging modes (E.g., 4-level and 5-level paging). • Fixed typo error in Section 4.1.3 from translation-requests to translated-requests. • Fixed error in Table-14 in Section 7.2.1.4 that had incorrectly listed translation-requests with PASID that fail ERE and SRE checks as returning Unsupported Request in Translation Completion Status and report Fault Reason of 19h and 1Ah. Instead, these conditions result in Recoverable Fault conditions for Translation Requests as the Translation Completion Data Entry returns R=W=U=S=0 (as documented in Table-16 in Section 7.2.2) • Clarified software must wait for the current Protected Memory Enable (PMEN) register control operation to be completed by hardware and reported in the status register before updating it again. • Added Fault code 30h in Appendix A to accomodate implementations logging a non-recoverable fault when a Page (PRS) request is blocked due to Present (P) or Page Request Enable (PRE) fields Clear in corresponding extended-context-entry. • Clarified that if a request-with-PASID with PR=1 (Privileged Request) is received by a remapping hardware implementation that reports SRS (Supervisory Requests Supported) as 0 (not supported), the translation completion entry is forced with PRIV=1 (same value as PR in request) and permissions forced to 0 (R=W=E=0). • Added new DMA_CTRL_PLATFORM_OPT_IN_FLAG in DMAR ACPI table for platform firmware to report compliance about platform initiated DMA restricted to RMRR ranges when transferring control to system software such as on *ExitBootServices()*. System software may program DMA remapping hardware to block DMA outside of RMRR, except for memory explicitly registered by device drivers with system software.

1 Introduction

This document describes the Intel® Virtualization Technology for Directed I/O ("Intel® VT for Directed I/O"); specifically, it describes the components supporting I/O virtualization as it applies to platforms that use Intel® processors and core logic chipsets complying with Intel® platform specifications.

Figure 1-1 illustrates the general platform topology.

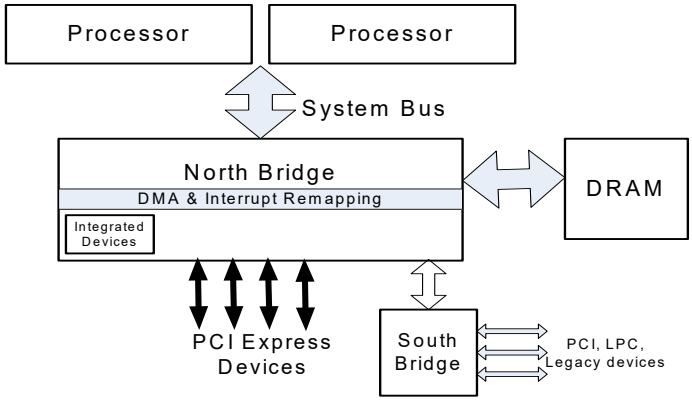


Figure 1-1. General Platform Topology

The document includes the following topics:

- An overview of I/O subsystem hardware functions for virtualization support
- A brief overview of expected usages of the generalized hardware functions
- The theory of operation of hardware, including the programming interface

The following topics are not covered (or are covered in a limited context):

- Intel® Virtualization Technology for Intel® 64 Architecture. For more information, refer to the "*Intel® 64 Architecture Software Developer's Manual, Volume 3B: System Programming Guide*".

1.1 Audience

This document is aimed at hardware designers developing Intel platforms or core-logic providing hardware support for virtualization. The document is also expected to be used by Operating System (OS) and Virtual Machine Monitor (VMM) developers utilizing the I/O virtualization hardware functions.

1.2 Glossary

The document uses the terms listed in the following table.

Table 1. Glossary

Term	Definition
Context	A hardware representation of state that identifies a device and the domain to which the device is assigned.
Context-cache	Remapping hardware cache that stores device to domain mappings
Device-TLB	A translation cache at the endpoint device (as opposed to in the platform).
DMA	Direct Memory Access: Address routed in-bound requests from I/O devices
DMA Remapping	The act of translating the address in a DMA request to a host physical address (HPA).
Domain	A collection of physical, logical, or virtual resources that are allocated to work together. Used as a generic term for virtual machines, partitions, etc.
DMA Address	Address in a DMA request: Depending on the software usage and hardware capabilities, DMA address can be Guest Physical Address (GPA), Guest Virtual Address (GVA), Virtual Address (VA), or I/O Virtual Address (IOVA).
First-Level Paging	Paging structures used for address translation of DMA requests with Process Address Space ID (PASID)
First-Level Caches	Translation caches used by remapping hardware units to cache intermediate (non-leaf) entries of the first-level paging structures. These may include PML5 cache, PML4 cache, PDP cache, and PDE cache.
GAW	Guest Address Width: Physical addressability limit within a partition (virtual machine)
GPA	Guest Physical Address: the view of physical memory from software running in a partition (virtual machine).
Guest	Software running within a virtual machine environment (partition).
GVA	Guest Virtual Address: Processor virtual address used by software running in a partition (virtual machine).
HAW	Host Address Width: the DMA physical addressability limit for a platform.
HPA	Host Physical Address: Physical address used by hardware to access memory and memory-mapped resources.
IEC	Interrupt Entry Cache: A translation cache in remapping hardware unit that caches frequently used interrupt-remapping table entries.
IOTLB	I/O Translation Lookaside Buffer: an address translation cache in remapping hardware unit that caches effective translations from DVA (GPA) to HPA.
I/OxAPIC	I/O Advanced Programmable Interrupt Controller
IOVA	I/O Virtual Address: Virtual address created by software for use in I/O requests.
Interrupt Remapping	The act of translating an interrupt request before it is delivered to the CPU complex.
MGAW	Maximum Guest Address Width: the maximum DMA virtual addressability supported by a remapping hardware implementation.
MSI	Message Signalled Interrupts.

Table 1. Glossary

Term	Definition
Second-Level Caches	Translation caches used by remapping hardware units to cache intermediate (non-leaf) entries of the second-level (SL) paging structures. Depending on the Guest Address Width supported by hardware, these may include SL-PML5 cache, SL-PML4 cache, SL-PDP cache, and SL-PDE cache.
PASID	Process Address Space Identifier: DMA requests with virtual address (or guest virtual address) are tagged with a PASID value that identifies the targeted virtual address space.
PASID-cache	Remapping hardware cache that caches frequently accessed PASID-table entries used to translate DMA requests with PASID.
PASID State Table	Data structure used by hardware to report to software if a given PASID is active at a endpoint device or not. PASID state is used by software to implement optimizations for IOTLB invalidations.
Second-Level Paging	Paging Structures used for address translation of DMA requests without Process Address Space ID (PASID).
Source ID	A 16-bit identification number to identify the source of a DMA or interrupt request. For PCI family devices this is the 'Requester ID' which consists of PCI Bus number, Device number, and Function number.
Root-Complex	Refers to one or more hardware components that connect processor complexes to the I/O and memory subsystems. The chipset may include a variety of integrated devices.
VA	Virtual Address: Virtual address used by software on a host processor.
VMM	Virtual Machine Monitor: a software layer that controls virtualization. Also referred to as hypervisor in this document.
x2APIC	The extension of xAPIC architecture to support 32-bit APIC addressability of processors and associated enhancements.

1.3 References

Table 2. References

Description
Intel® 64 Architecture Software Developer's Manuals http://developer.intel.com/products/processor/manuals/index.htm
PCI-Express® Base Specifications http://www.pcisig.com/specifications/pciexpress
PCI-Express Address Translation Services Specification, Revision 1.1 http://www.pcisig.com/specifications/iov
PCI-Express Process Address Space ID, and PASID Translation ECNs
PCI-Express Alternative Routing-ID Interpretation (ARI) ECN
PCI-Express Single-Root I/O Virtualization and Sharing (SR-IOV) Specification, Revision 1.0 http://www.pcisig.com/specifications/iov
ACPI Specification http://www.acpi.info/
PCI-Express to PCI/PCI-X Bridge Specification, Revision 1.0 http://www.pcisig.com/specifications/pciexpress/bridge

2 Overview

This chapter provides a brief overview of Intel® VT, the virtualization software ecosystem it enables, and hardware support offered for processor and I/O virtualization.

2.1 Intel® Virtualization Technology Overview

Intel® VT consists of technology components that support virtualization of platforms based on Intel processors, thereby enabling the running of multiple operating systems and applications in independent partitions. Each partition behaves like a virtual machine (VM) and provides isolation and protection across partitions. This hardware-based virtualization solution, along with virtualization software, enables multiple usages such as server consolidation, activity partitioning, workload isolation, embedded management, legacy software migration, and disaster recovery.

2.2 VMM and Virtual Machines

Intel® VT supports virtual machine architectures comprised of two principal classes of software:

- **Virtual-Machine Monitor (VMM)**: A VMM acts as a host and has full control of the processor(s) and other platform hardware. VMM presents guest software (see below) with an abstraction of a virtual processor and allows it to execute directly on a logical processor. A VMM is able to retain selective control of processor resources, physical memory, interrupt management, and I/O.
- **Guest Software**: Each virtual machine is a guest software environment that supports a stack consisting of an operating system (OS) and application software. Each operates independently of other virtual machines and uses the same interface to processor(s), memory, storage, graphics, and I/O provided by a physical platform. The software stack acts as if it were running on a platform with no VMM. Software executing in a virtual machine must operate with reduced privilege so that the VMM can retain control of platform resources.

The VMM is a key component of the platform infrastructure in virtualization usages. Intel® VT can improve the reliability and supportability of virtualization infrastructure software with programming interfaces to virtualize processor hardware. It also provides a foundation for additional virtualization support for other hardware components in the platform.

2.3 Hardware Support for Processor Virtualization

Hardware support for processor virtualization enables simple, robust and reliable VMM software. VMM software relies on hardware support on operational details for the handling of events, exceptions, and resources allocated to virtual machines.

Intel® VT provides hardware support for processor virtualization. For Intel® 64 processors, this support consists of a set of virtual-machine extensions (VMX) that support virtualization of processor hardware for multiple software environments by using virtual machines.

2.4 I/O Virtualization

A VMM must support virtualization of I/O requests from guest software. I/O virtualization may be supported by a VMM through any of the following models:

- *Emulation*: A VMM may expose a virtual device to guest software by emulating an existing (legacy) I/O device. VMM emulates the functionality of the I/O device in software over whatever physical devices are available on the physical platform. I/O virtualization through emulation provides good compatibility (by allowing existing device drivers to run within a guest), but pose limitations with performance and functionality.
- *New Software Interfaces*: This model is similar to I/O emulation, but instead of emulating legacy devices, VMM software exposes a synthetic device interface to guest software. The synthetic device interface is defined to be virtualization-friendly to enable efficient virtualization compared to the overhead associated with I/O emulation. This model provides improved performance over emulation, but has reduced compatibility (due to the need for specialized guest software or drivers utilizing the new software interfaces).
- *Assignment*: A VMM may directly assign the physical I/O devices to VMs. In this model, the driver for an assigned I/O device runs in the VM to which it is assigned and is allowed to interact directly with the device hardware with minimal or no VMM involvement. Robust I/O assignment requires additional hardware support to ensure the assigned device accesses are isolated and restricted to resources owned by the assigned partition. The I/O assignment model may also be used to create one or more I/O container partitions that support emulation or software interfaces for virtualizing I/O requests from other guests. The I/O-container-based approach removes the need for running the physical device drivers as part of VMM privileged software.
- *I/O Device Sharing*: In this model, which is an extension to the I/O assignment model, an I/O device supports multiple functional interfaces, each of which may be independently assigned to a VM. The device hardware itself is capable of accepting multiple I/O requests through any of these functional interfaces and processing them utilizing the device's hardware resources.

Depending on the usage requirements, a VMM may support any of the above models for I/O virtualization. For example, I/O emulation may be best suited for virtualizing legacy devices. I/O assignment may provide the best performance when hosting I/O-intensive workloads in a guest. Using new software interfaces makes a trade-off between compatibility and performance, and device I/O sharing provides more virtual devices than the number of physical devices in the platform.

2.5 Intel® Virtualization Technology For Directed I/O Overview

A general requirement for all of above I/O virtualization models is the ability to isolate and restrict device accesses to the resources owned by the partition managing the device. Intel® VT for Directed I/O provides VMM software with the following capabilities:

- *I/O device assignment*: for flexibly assigning I/O devices to VMs and extending the protection and isolation properties of VMs for I/O operations.
- *DMA remapping*: for supporting address translations for Direct Memory Accesses (DMA) from devices.
- *Interrupt remapping*: for supporting isolation and routing of interrupts from devices and external interrupt controllers to appropriate VMs.
- *Interrupt posting*: for supporting direct delivery of virtual interrupts from devices and external interrupt controllers to virtual processors.
- *Reliability*: for recording and reporting of DMA and interrupt errors to system software that may otherwise corrupt memory or impact VM isolation.

2.5.1 Hardware Support for DMA Remapping

To generalize I/O virtualization and make it applicable to different processor architectures and operating systems, this document refers to *domains* as abstract isolated environments in the platform to which a subset of host physical memory is allocated.

DMA remapping provides hardware support for isolation of device accesses to memory, and enables each device in the system to be assigned to a specific domain through a distinct set of paging structures. When the device attempts to access system memory, the DMA-remapping hardware intercepts the access and utilizes the page tables to determine whether the access can be permitted; it also determines the actual location to access. Frequently used paging structures can be cached in hardware. DMA remapping can be configured independently for each device, or collectively across multiple devices.

2.5.1.1 OS Usages of DMA Remapping

There are several ways in which operating systems can use DMA remapping:

- *OS Protection*: An OS may define a domain containing its critical code and data structures, and restrict access to this domain from all I/O devices in the system. This allows the OS to limit erroneous or unintended corruption of its data and code through incorrect programming of devices by device drivers, thereby improving OS robustness and reliability.
- *Feature Support*: An OS may use domains to better manage DMA from legacy devices to high memory (For example, 32-bit PCI devices accessing memory above 4GB). This is achieved by programming the I/O page-tables to remap DMA from these devices to high memory. Without such support, software must resort to data copying through OS “bounce buffers”.
- *DMA Isolation*: An OS may manage I/O by creating multiple domains and assigning one or more I/O devices to each domain. Each device-driver explicitly registers its I/O buffers with the OS, and the OS assigns these I/O buffers to specific domains, using hardware to enforce DMA domain protection. See Figure 2-2.
- *Shared Virtual Memory*: For devices supporting appropriate PCI-Express[1] capabilities, OS may use the DMA remapping hardware capabilities to share virtual address space of application processes with I/O devices. Shared virtual memory along with support for I/O page-faults enable application programs to freely pass arbitrary data-structures to devices such as graphics processors or accelerators, without the overheads of pinning and marshalling of data.

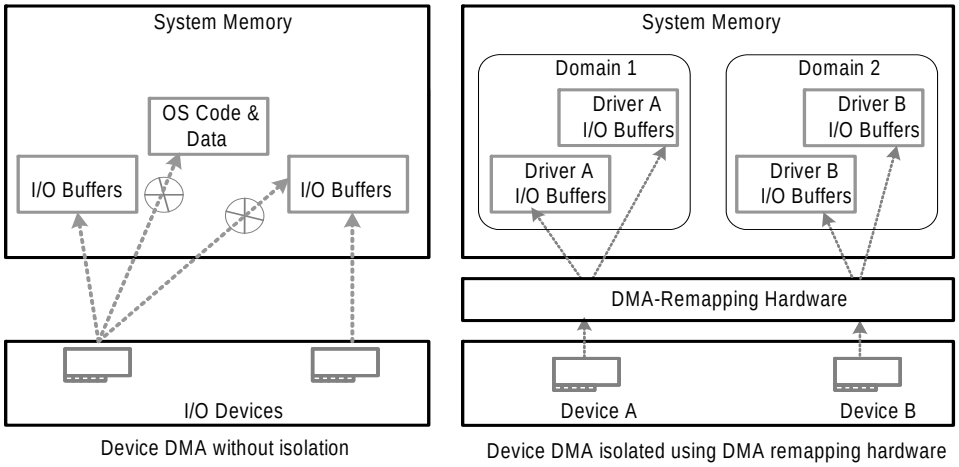


Figure 2-2. Example OS Usage of DMA Remapping

1. Refer to Process Address Space ID (PASID) capability in PCI-Express* base specification.

2.5.1.2 VMM Usages of DMA Remapping

The limitations of software-only methods for I/O virtualization can be improved through direct assignment of I/O devices to partitions. With this approach, the driver for an assigned I/O device runs only in the partition to which it is assigned and is allowed to interact directly with the device hardware with minimal or no VMM involvement. The hardware support for DMA remapping enables this direct device assignment without device-specific knowledge in the VMM. See Figure 2-3.

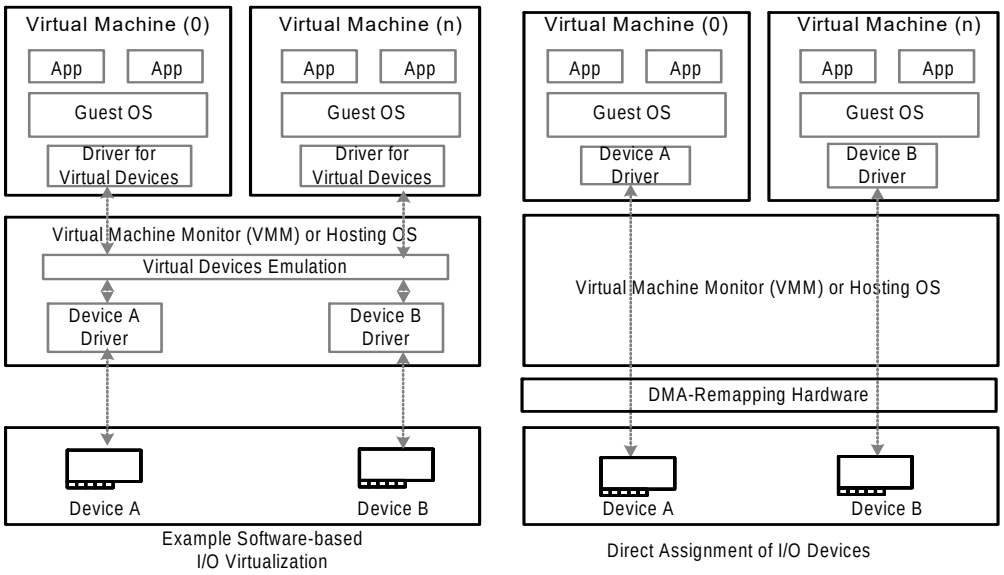


Figure 2-3. Example Virtualization Usage of DMA Remapping

In this model, the VMM restricts itself to enabling direct assignment of devices to their partitions. Rather than invoking the VMM for all I/O requests from a partition, the VMM is invoked only when guest software accesses protected resources (such as configuration accesses, interrupt management, etc.) that impact system functionality and isolation.

To support direct assignment of I/O devices, a VMM must enforce isolation of DMA requests. I/O devices can be assigned to domains, and the remapping hardware can be used to restrict DMA from an I/O device to the physical memory presently owned by its domain. For domains that may be relocated in physical memory, the remapping hardware can be programmed to perform the necessary translation.

I/O device assignment allows other I/O sharing usages — for example, assigning an I/O device to an I/O partition that provides I/O services to other user partitions. Remapping hardware enables virtualization software to choose the right combination of device assignment and software-based methods for I/O virtualization.

2.5.1.3 DMA Remapping Usages by Guests

A guest OS running in a VM may benefit from the availability of remapping hardware to support the usages described in Section 2.5.1.1. To support such usages, the VMM may virtualize the remapping hardware to its guests. For example, the VMM may intercept guest accesses to the virtual remapping hardware registers, and manage a shadow copy of the guest remapping structures that is provided to the physical remapping hardware. On updates to the guest I/O page tables, the guest software performs appropriate virtual invalidation operations. The virtual invalidation requests may be intercepted by the VMM, to update the respective shadow page tables and perform invalidations of

remapping hardware. Due to the non-restartability of faulting DMA transactions (unlike CPU memory management virtualization), a VMM cannot perform lazy updates to its shadow remapping structures. To keep the shadow structures consistent with the guest structures, the VMM may expose virtual remapping hardware with eager pre-fetching behavior (including caching of not-present entries) or use processor memory management mechanisms to write-protect the guest remapping structures.

On hardware implementations supporting two levels of address translations (first-level translation to remap a virtual address to intermediate (guest) physical address, and second-level translations to remap a intermediate physical address to machine (host) physical address), a VMM may virtualize guest OS use of first-level translations (such as for Shared Virtual Memory usages) without shadowing page-tables, but by configuring hardware to perform nested translation of first and second-levels.

2.5.1.4 Interaction with Processor Virtualization

Figure 2-4 depicts how system software interacts with hardware support for both processor-level virtualization and Intel® VT for Directed I/O.

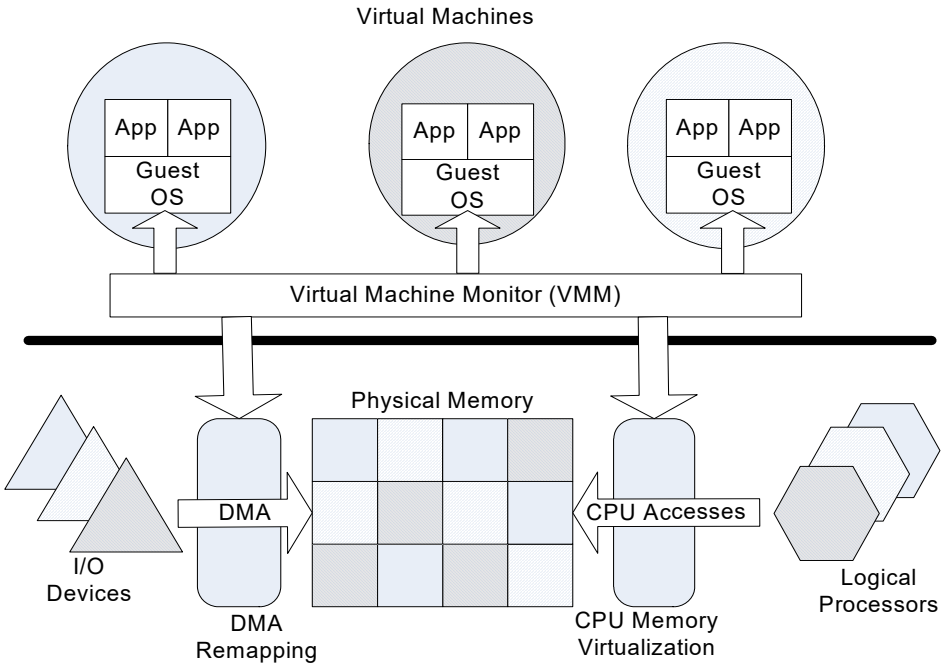


Figure 2-4. Interaction Between I/O and Processor Virtualization

The VMM manages processor requests to access physical memory via the processor's memory management hardware. DMA requests to access physical memory use remapping hardware. Both processor memory management and DMA memory management are under the control of the VMM.

2.5.2 Hardware Support for Interrupt Remapping

Interrupt remapping provides hardware support for remapping and routing of interrupt requests from I/O devices (generated directly or through I/O interrupt controllers). The indirection achieved through remapping enables isolation of interrupts across partitions.

The following usages are envisioned for the interrupt-remapping hardware.

2.5.2.1 Interrupt Isolation

On Intel architecture platforms, interrupt requests are identified by the Root-Complex as write transactions targeting an architectural address range (0xFEEx_xxxxh). The interrupt requests are self-describing (i.e., attributes of the interrupt request are encoded in the request address and data), allowing any DMA initiator to generate interrupt messages with arbitrary attributes.

The interrupt-remapping hardware may be utilized by a Virtual Machine Monitor (VMM) to improve the isolation of external interrupt requests across domains. For example, the VMM may utilize the interrupt-remapping hardware to distinguish interrupt requests from specific devices and route them to the appropriate VMs to which the respective devices are assigned. The VMM may also utilize the interrupt-remapping hardware to control the attributes of these interrupt requests (such as destination CPU, interrupt vector, delivery mode etc.).

Another example usage is for the VMM to use the interrupt-remapping hardware to disambiguate external interrupts from the VMM owned inter-processor interrupts (IPIs). Software may enforce this by ensuring none of the remapped external interrupts have attributes (such as vector number) that matches the attributes of the VMM IPIs.

2.5.2.2 Interrupt Migration

The interrupt-remapping architecture may be used to support dynamic re-direction of interrupts when the target for an interrupt request is migrated from one logical processor to another logical processor. Without interrupt-remapping hardware support, re-balancing of interrupts require software to re-program the interrupt sources. However re-programming of these resources are non-atomic (requires multiple registers to be re-programmed), often complex (may require temporary masking of interrupt source), and dependent on interrupt source characteristics (e.g. no masking capability for some interrupt sources; edge interrupts may be lost when masked on some sources, etc.)

Interrupt-remapping enables software to efficiently re-direct interrupts without re-programming the interrupt configuration at the sources. Interrupt migration may be used by OS software for balancing load across processors (such as when running I/O intensive workloads), or by the VMM when it migrates virtual CPUs of a partition with assigned devices across physical processors to improve CPU utilization.

2.5.2.3 x2APIC Support

Intel® 64 x2APIC architecture extends the APIC addressability to 32-bits (from 8-bits). Refer to *Intel® 64 Architecture Software Developer's Manual, Volume 3B: System Programming Guide* for details.

Interrupt remapping enables x2APICs to support the expanded APIC addressability for external interrupts without requiring hardware changes to interrupt sources (such as I/OxAPICs and MSI/MSI-X devices).

2.5.3 Hardware Support for Interrupt Posting

Interrupt posting includes hardware support for optimized processing of interrupt requests from I/O devices (Physical Functions, or Single Root I/O Virtualization (SR-IOV) Virtual Functions) that are directly assigned to a virtual machine. The following usages are envisioned for the interrupt-posting hardware.

2.5.3.1 Interrupt Vector Scalability

Devices supporting I/O virtualization capabilities such as SR-IOV, virtually increases the I/O fan-out of the platform, by allowing multiple Virtual Functions (VFs) to be enabled for a Physical Function (PF). Any of these PFs or VFs can be assigned to a virtual machine. Interrupt requests from such assigned devices are referred to as virtual interrupts as they target virtual processors of the assigned VM.

Each VF requires its own independent interrupt resources, resulting in more interrupt vectors needed than otherwise required without such I/O virtualization. Without interrupt-posting hardware support, all interrupt sources in the platform are mapped to the same physical interrupt vector space (8-bit vector space per logical CPU on Intel®64 processors). For virtualization usages, partitioning the physical vector space across virtual processors is challenging in a dynamic environment when there is no static affinity between virtual process and logical processors.

Hardware support for interrupt posting addresses this vector scalability problem by allowing interrupt requests from device functions assigned to virtual machines to operate in virtual vector space, thereby scaling naturally with the number of virtual machines or virtual processors.

2.5.3.2 Interrupt Virtualization Efficiency

Without hardware support for interrupt posting, interrupts from devices assigned to virtual machines are processed through the VMM software. Specifically, whenever an external interrupt destined for a virtual machine is received by the CPU, control is transferred to the VMM, requiring the VMM to process and inject corresponding virtual interrupt to the virtual machine. The control transfers associated with such VMM processing of external interrupts incurs both hardware and software overheads.

With hardware support for interrupt posting, interrupts from devices assigned to virtual machines are posted (recorded) in memory descriptors specified by the VMM, and processed based on the running state of the virtual processor targeted by the interrupt.

For example, if the target virtual processor is running on any logical processor, hardware can directly deliver external interrupts to the virtual processor without any VMM intervention. Interrupts received while the target virtual processor is pre-empted (waiting for its turn to run) can be accumulated in memory by hardware for delivery when the virtual processor is later scheduled. This avoids disrupting execution of currently running virtual processors on external interrupts for non-running virtual machines. If the target virtual processor is halted (idle) at the time of interrupt arrival or if the interrupt is qualified as requiring real-time processing, hardware can transfer control to VMM, enabling VMM to schedule the virtual processor and have hardware directly deliver pending interrupts to that virtual processor.

This target virtual processor state based processing of interrupts reduces overall interrupt latency to virtual machines and reduces overheads otherwise incurred by the VMM for virtualizing interrupts.

2.5.3.3 Virtual Interrupt Migration

To optimize overall platform utilization, VMM software may need to dynamically evaluate the optimal logical processor to schedule a virtual processor, and in that process, migrate virtual processors across CPUs.

For virtual machines with assigned devices, migrating a virtual processor across logical processors either incurs the overhead of forwarding interrupts in software (e.g. via VMM generated IPIs), or complexity to independently migrate each interrupt targeting the virtual processor to the new logical

processor. Hardware support for interrupt posting enables VMM software to atomically co-migrate all interrupts targeting a virtual processor when the virtual processor is scheduled to another logical processor.

3 DMA Remapping

This chapter describes the hardware architecture for DMA remapping. The architecture envisions remapping hardware to be implemented in Root-Complex components, such as the memory controller hub (MCH) or I/O hub (IOH).

3.1 Types of DMA requests

Remapping hardware treats inbound memory requests from root-complex integrated devices and PCI-Express attached discrete devices into two categories:

- *Requests without address-space-identifier*: These are the normal memory requests from endpoint devices. These requests typically specify the type of access (read/write/atomics), targeted DMA address/size, and identity of the device originating the request.
- *Requests with address-space-identifier*: These are memory requests with additional information identifying the targeted process address space from endpoint devices supporting virtual memory capabilities. Beyond attributes in normal requests, these requests specify the targeted process address space identifier (PASID), and extended attributes such as Execute-Requested (ER) flag (to indicate reads that are instruction fetches), and Privileged-mode-Requested (PR) flag (to distinguish user versus supervisor access). For details, refer to the Process Address Space ID (PASID) Capability in the PCI-Express specifications.

For simplicity, this document refers to these categories as ***Requests-without-PASID***, and ***Requests-with-PASID***. Previous versions of this specification supported only remapping of requests-without-PASID.

3.2 Domains and Address Translation

A domain is abstractly defined as an isolated environment in the platform, to which a subset of the host physical memory is allocated. I/O devices that are allowed to access physical memory directly are allocated to a domain and are referred to as the domain's assigned devices. For virtualization usages, software may treat each virtual machine as a separate domain.

The isolation property of a domain is achieved by blocking access to its physical memory from resources not assigned to it. Multiple isolated domains are supported in a system by ensuring that all I/O devices are assigned to some domain (possibly a null domain), and that they can only access the physical resources allocated to their domain. The DMA remapping architecture facilitates flexible assignment of I/O devices to an arbitrary number of domains. Each domain has a view of physical address space that may be different than the host physical address space. Remapping hardware treats the address in inbound requests as DMA Address. Depending on the software usage model, the DMA address space may be the Guest-Physical Address (GPA) space of the virtual machine to which the device is assigned, or application Virtual Address (VA) space defined by the PASID assigned to an application, or some abstract I/O virtual address (IOVA) space defined by software. In all cases, DMA remapping transforms the address in a DMA request issued by an I/O device to its corresponding Host-Physical Address (HPA).

For simplicity, this document refers to address in requests-without-PASID as GPA, and address in requests-with-PASID as Virtual Address (VA) (or Guest Virtual Address (GVA), if such request is from a device assigned to a virtual machine). The translated address is referred to as HPA.

Figure 3-5 illustrates DMA address translation. I/O devices 1 and 2 are assigned to domains 1 and 2, respectively. The software responsible for creating and managing the domains allocates system physical memory for both domains and sets up the DMA address translation function. DMA address in requests initiated by devices 1 & 2 are translated to appropriate HPAs by the remapping hardware.

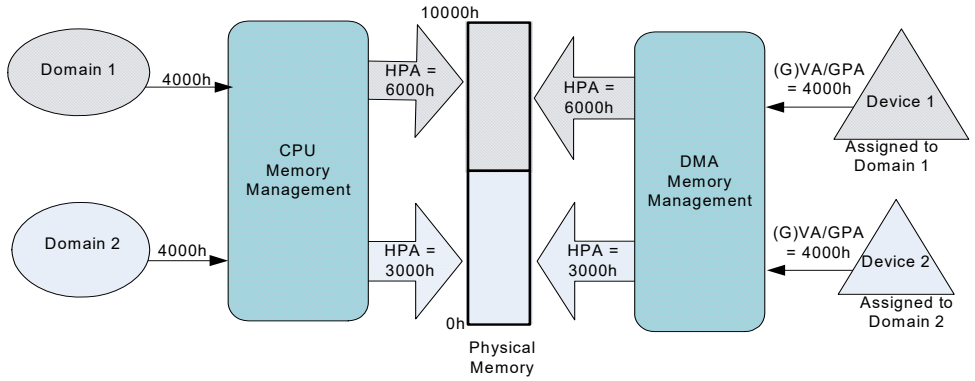


Figure 3-5. DMA Address Translation

The host platform may support one or more remapping hardware units. Each hardware unit supports remapping DMA requests originating within its hardware scope. For example, a desktop platform may expose a single remapping hardware unit that translates all DMA transactions at the memory controller hub (MCH) component. A server platform with one or more core components may support independent translation hardware units in each component, each translating DMA requests originating within its I/O hierarchy (such as a PCI-Express root port). The architecture supports configurations in which these hardware units may either share the same translation data structures (in system memory) or use independent structures, depending on software programming.

The remapping hardware translates the address in a request to host physical address (HPA) before further hardware processing (such as address decoding, snooping of processor caches, and/or forwarding to the memory controllers).

3.3 Remapping Hardware - Software View

The remapping architecture allows hardware implementations supporting a single PCI segment group to expose (to software) the remapping function either as a single hardware unit covering the entire PCI segment group, or as multiple hardware units, each supporting a mutually exclusive subset of devices in the PCI segment group hierarchy. For example, an implementation may expose a remapping hardware unit that supports one or more integrated devices on the root bus, and additional remapping hardware units for devices behind one or a set of PCI-Express root ports. The platform firmware (BIOS) reports each remapping hardware unit in the platform to software. Chapter 8 describes a proposed reporting structure through ACPI constructs.

For hardware implementations supporting multiple PCI segment groups, the remapping architecture requires hardware to expose independent remapping hardware units (at least one per PCI segment group) for processing requests originating within the I/O hierarchy of each segment group.

3.4 Mapping Devices to Domains

The following sub-sections describe the DMA remapping architecture and data structures used to map I/O devices to domains.

3.4.1 Source Identifier

Each inbound request appearing at the address-translation hardware is required to identify the device originating the request. The attribute identifying the originator of an I/O transaction is referred to as the "source-id" in this document. The remapping hardware may determine the source-id of a transaction in implementation-specific ways. For example, some I/O bus protocols may provide the originating device identity as part of each I/O transaction. In other cases (for Root-Complex integrated devices, for example), the source-id may be derived based on the Root-Complex internal implementation.

For PCI-Express devices, the source-id is the requester identifier in the PCI-Express transaction layer header. The requester identifier of a device, which is composed of its PCI Bus/Device/Function number, is assigned by configuration software and uniquely identifies the hardware function that initiated the request. Figure 3-6 illustrates the requester-id[1] as defined by the PCI-Express Specification.

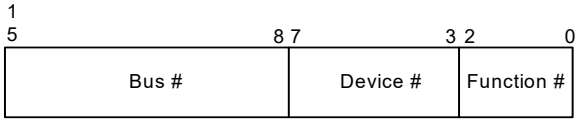


Figure 3-6. Requester Identifier Format

The following sections describe the data structures for mapping I/O devices to domains.

3.4.2 Root-Entry & Extended-Root-Entry

The root-table functions as the top level structure to map devices to their respective domains. The location of the root-table in system memory is programmed through the Root Table Address Register described in Section 10.4.6. The root-table is 4-KByte in size and contains 256 root-entries to cover the PCI bus number space (0-255). The bus number (upper 8-bits) encoded in a request's source-id field is used to index into the root-entry structure.

Each root-entry contains the following fields:

- **Present flag**: The present field indicates the root-entry is present and the context-table pointer (CTP) field is initialized. Software may Clear the present field for root entries corresponding to bus numbers that are either not present in the platform, or don't have any downstream devices attached. DMA requests processed through root-entries with present field Clear result in translation-fault.
- **Context-table pointer**: The context-table pointer references the context-table for devices on the bus identified by the root-entry. Section 3.4.3 describes context-entries in the context-table.

Section 9.1 provides the exact root-table entry format.

For implementations supporting Extended-Context-Support (ECS=1 in Extended Capability Register), the Root Table Address Register (RTADDR_REG) points to an extended-root-table when Root-Table-Type field in the Register is Set (RTT=1). The extended-root-table is similar to the root-table (4KB in size and containing 256 extended-root-entries to cover the 0-255 PCI bus number space), but has an extended format to reference extended-context-tables.

Each extended-root-entry contains the following fields:

1. For PCI-Express devices supporting Alternative Routing-ID Interpretation (ARI), bits traditionally used for the Device Number field in the Requester-id are used instead to expand the Function Number field.

- **Lower Present flag**: The lower-present field indicates the lower 64-bits of the extended-root-entry is present and the lower-context-table pointer (LCTP) field is initialized. Software may Clear the lower-present field for extended-root-entries corresponding to bus numbers that are either not present in the platform, or don't have downstream devices with device numbers 0-15 attached. DMA requests processed through the lower part of an extended-root-entry with the lower-present field Clear result in translation-fault.
- **Lower Context-table pointer**: The lower-context-table pointer references the lower-context-table for devices with device number 0-15, on the bus identified by the referencing extended-root-entry. Section 3.4.4 describes extended-context-entries in the lower-context-table.
- **Upper Present flag**: The upper-present field indicates the upper 64-bits of the extended-root-entry is present and the upper-context-table pointer (UCTP) field is initialized. Software may Clear the upper-present field for extended-root-entries corresponding to bus numbers that are either not present in the platform, or don't have downstream devices with device numbers 16-31 attached. DMA requests processed through the upper part of an extended-root-entry with the upper-present field Clear result in translation-fault.
- **Upper Context-table pointer**: The upper-context-table pointer references the upper-context-table for devices with device number 16-31, on the bus identified by the referencing extended-root-entry. Section 3.4.4 describes extended-context-entries in the upper-context-table.

Section 9.2 provides the exact extended-root-table entry format.

3.4.3 Context-Entry

A context-entry maps a specific I/O device on a bus to the domain to which it is assigned, and, in turn, to the address translation structures for the domain. The context entries are programmed through memory-resident context-tables. Each root-entry in the root-table contains the pointer to the context-table for the corresponding bus number. Each context-table contains 256 entries, with each entry corresponding to a PCI device function on the bus. For a PCI device, the device and function numbers (lower 8-bits) of source-id are used to index into the context-table. Figure 3-7 illustrates device to domain mapping through root-table.

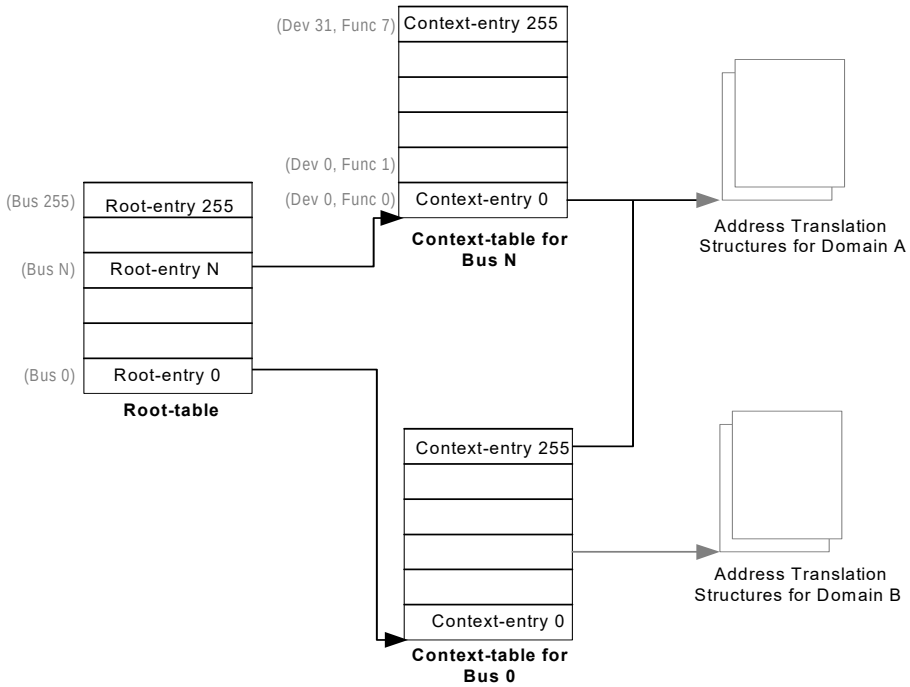


Figure 3-7. Device to Domain Mapping Structures using Root-Table

Context-entries support only requests-without-PASID, and contains the following fields:

- **Present Flag**: The present field is used by software to indicate to hardware whether the context-entry is present and initialized. Software may Clear the present field for context entries corresponding to device functions that are not present in the platform. If the present field of a context-entry used to process a request is Clear, the request is blocked, resulting in a translation-fault.
- **Translation Type**: The translation-type field indicates what types of requests are allowed through the context-entry, and the type of the address translation that must be used for such requests.
- **Address Width**: The address-width field indicates the address-width of the domain to which the device corresponding to the context-entry is assigned.
- **Second-level Page-table Pointer**: The second-level page-table pointer field provides the host physical address of the address translation structure in system memory to be used for remapping requests-without-PASID processed through the context-entry.
- **Domain Identifier**: The domain-identifier is a software-assigned field in a context-entry that identifies the domain to which a device with the given source-id is assigned. Hardware may use this field to tag its caching structures. Context entries programmed with the same domain identifier must reference the same address translation structure. Context entries referencing the same address translation structures are recommended to use the same domain-identifier for best hardware efficiency.
- **Fault Processing Disable Flag**: The fault-processing-disable field enables software to selectively disable recording and reporting of remapping faults detected for requests processed through the context-entry.

Multiple devices may be assigned to the same domain by programming the context-entries for the devices to reference the same translation structures, and programming them with the same domain identifier. Section 9.3 provides the exact context-entry format.

3.4.4 Extended-Context-Entry

For implementations supporting Extended-Context-Support (ECS=1 in Extended Capability Register), when using extended-root-table, each extended-root-entry references a lower-context-table and a upper-context-table. The Lower-context-table is 4-KByte in size and contains 128 extended-context-entries corresponding to PCI functions in device range 0-15 on the bus. The Upper-context-table is also 4-KByte in size and contains 128 extended-context-entries corresponding to PCI functions in device range 16-31 on the bus. Figure 3-8 illustrates device to domain mapping through extended-root-table.

Extended-context-entries are capable of supporting both requests-without-PASID and requests-with-PASID. For requests-without-PASID, it supports the same fields as in the regular context-entry (described above). Section 9.4 provides the exact extended-context-entry format. For requests-with-PASID, extended-context-entries contain the following additional fields:

- **Extended Translation Type**: The translation-type field is extended to provide additional controls to specify how requests with and without PASID should be processed. Extended-context-entries supports two levels of translation, referred to as first-level translation and second-level translation. First-level translation applies to requests-with-PASID. Second-level translation applies to requests-without-PASID. When nested translation is specified in the extended-context-entry, requests-with-PASID are subject to nested first-level and second-level translation.
- **Translation Structure Pointers**: For first-level translation, the extended-context-entry contains a pointer to a PASID-table. Each 8-byte PASID-table-entry corresponds to a PASID value, and contains the root of first-level translation structures used to translate requests-with-PASID tagged with the respective PASID. For second-level translation, extended-context-entry contains a pointer to the second-level page-table, which is the same as the second-level page-table pointer field in the regular context-entry (described in Section 3.4.3).

- **Translation Controls**: These include additional controls such as Page-Global-Enable, Write-Protect-Enable, No-Execute-Enable, Supervisor-Mode-Execute-Protection, etc. that are applied when processing requests-with-PASID.
- **Memory-type Attributes**: Extended-context-entries support fields such as Page-Attribute-Table, Extended-memory-type etc., that are used to compute the effective memory-type for requests-with-PASID from devices operating in the processor coherency domain.
- **Page Request Enable**: The page-request-enable field in the extended-context-entry allows software to selectively enable or disable page-fault requests from the device. When enabled, page-requests from the device are reported to software through a memory-resident page-request-queue. Chapter 7 provides details on page request processing.
- **Deferred Invalidation Controls**: The PASID-state table pointer field enables devices to communicate whether a given address-space (PASID) is active or not at the device. Software can utilize the PASID-state tables for deferred invalidation of cached mappings for inactive PASIDs in translation caches (TLBs). Chapter 6 describes the various translation caching structures and invalidation operations, including deferred invalidation support.

Figure 3-8 illustrates device to domain mapping using extended-root-table.

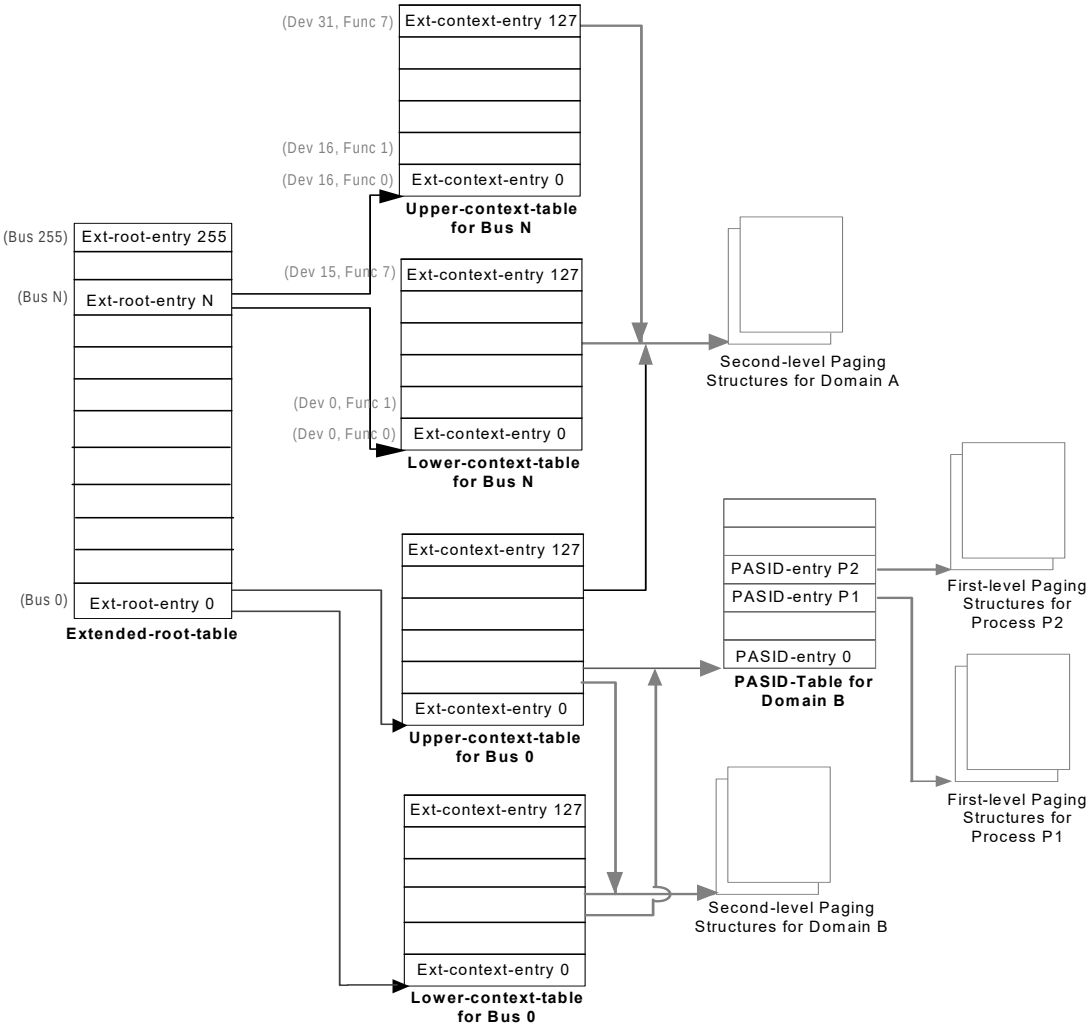


Figure 3-8. Device to Domain Mapping Structures using Extended-Root-Table

3.5 Hierarchical Translation Structures

DMA remapping uses hierarchical translation structures for both first-level translation (for requests-with-PASID) and second-level translation (for requests-without-PASID and for nested translation of requests-with-PASID).

For first-level translation, and second-level translation of requests-without-PASID, the DMA-address in the request is used as the input address. For nested translation of requests-with-PASID, any address generated by first-level translation (both addresses to access first-level translation structures and the output address from first-level translation) is used as the input address for nesting with second-level translation. Section 3.6, Section 3.7 and Section 3.8 provides more details on first-level, second-level, and nested translation respectively.

Every paging structure in the hierarchy is 4-KBytes in size, with 512 8-Byte entries. Remapping hardware uses the upper portion of input address to identify a series of paging-structure entries. The last of these entries identifies the physical address of the region to which the input address translates (called the page frame). The lower portion of the input address (called the page offset) identifies the specific offset within that region to which the input address translates. Each paging-structure entry contains a physical address, which is either the address of another paging structure or the address of a page frame. First-level translation supports 4-level structure and 5-level structure. Second-level translation supports a N-level structure, where the value of N depends on the Guest Address Width (GAW) supported by an implementation as enumerated in the Capability Register.

The paging structures support a base page-size of 4-KBytes. The page-size field in paging entries enable larger page allocations. When a paging entry with the page-size field Set is encountered by hardware on a page-table walk, the translated address is formed immediately by combining the page-base-address in the paging-entry with the unused input address bits. Remapping architecture defines support for 2-MByte and 1-GByte large-page sizes. Implementations report support for large-pages and intermediate-pages through the Capability Register.

Figure 3-9 illustrates the paging structure for translating a 48-bit address to a 4-KByte page.

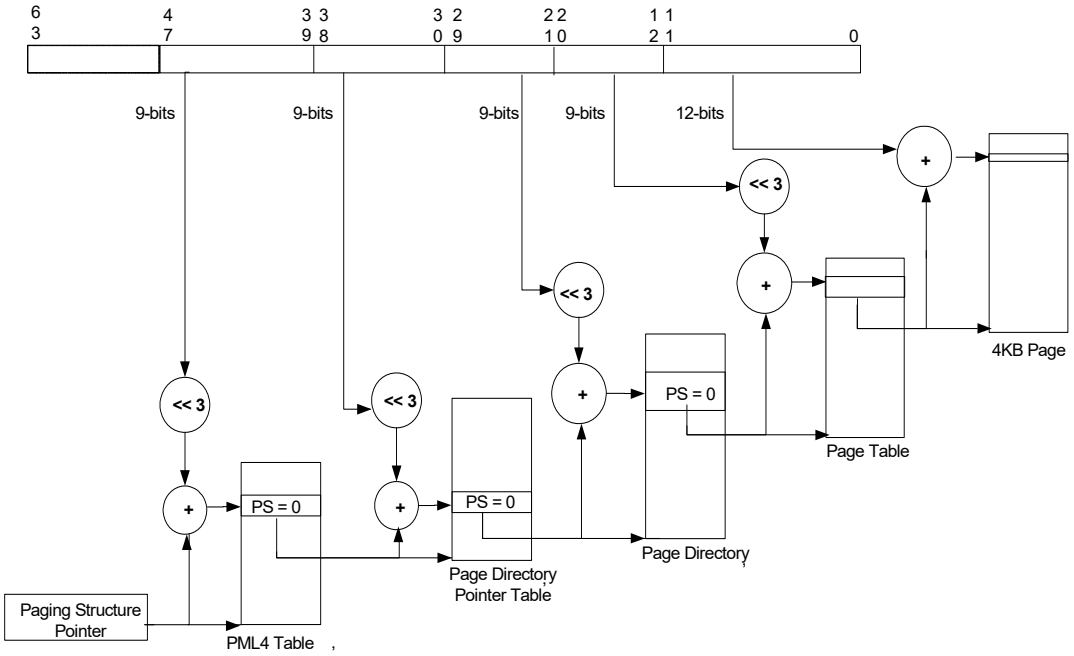


Figure 3-9. Address Translation to a 4-KByte Page

Figure 3-10 illustrates the paging structure for translating a 48-bit address to a 2-MByte large-page.

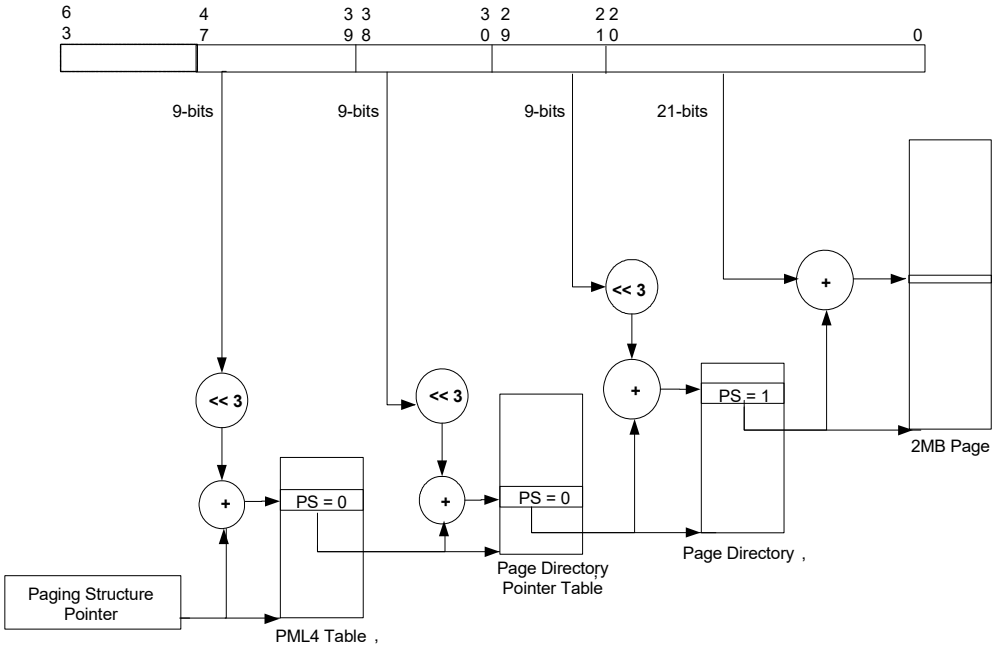


Figure 3-10. Address Translation to a 2-MByte Large Page

Figure 3-11 illustrates the paging structure for translating a 48-bit address to a 1-GByte large-page.

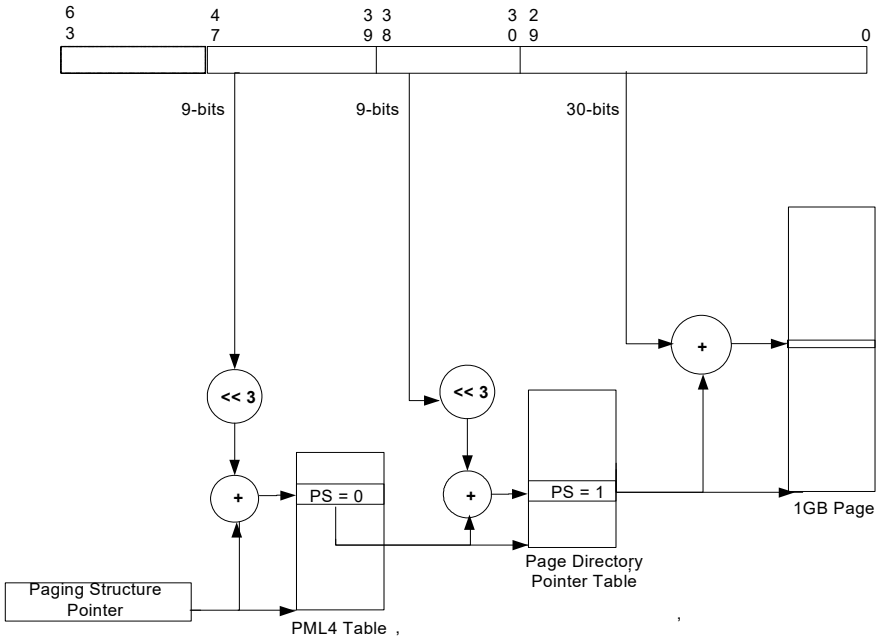


Figure 3-11. Address Translation to a 1-GByte Large Page

3.6 First-Level Translation

Extended-context-entries can be configured to translate requests-with-PASID through first-level translation. Extended-context-entries contain the PASID-table pointer and size fields used to reference the PASID-table. The PASID-number in a request-with-PASID is used to offset into the PASID-table. Each present PASID-entry contains a pointer to the base of the first-level translation structure for the respective process address space. Section 9.5 describes the exact format of the PASID-entry.

First-level translation restricts the input-address to a canonical address (i.e., address bits 63:N have the same value as address bit [N-1], where N is 48-bits with 4-level paging and 57-bits with 5-level paging). A device may perform a local check for canonical address before it issues a request-with-PASID, and handle violations in a device specific manner. Requests-with-PASID arriving at remapping hardware are subject to canonical address checking, and a violation is treated as a translation-fault. Chapter 7 provides details of translation-fault conditions and how they are reported to software.

First-level translation supports the same paging structures as Intel® 64 processors when operating in 64-bit mode. Table 3 gives the different names of the first-level translation structures, that are given based on their use in the translation process. It also provides, for each structure, the source of the physical-address used to locate it, the bits in the input-address used to select an entry from the structure, and details of whether and how such an entry can map a page. Section 9.7 describes the format of each of these paging structures in detail. For implementations supporting 5-level paging for first-level translation, 4-level versus 5-level paging for a request-with-PASID is selected based on the programming of the First level Paging Mode (FLPM) field in the corresponding PASID-entry (see Section 9.5).

Table 3. First-level Paging Structures

Paging Structure	Entry Name	Physical Address of Structure	Bits Selecting Entry	Page Mapping
PML5 table	PML5E	PASID-entry (for 5-level paging)	56:48	N/A
PML4 table	PML4E	PML5E (for 5-level paging); PASID-entry (for 4-level paging)	47:39	N/A
Page-directory-pointer table	PDPE	PML4E	38:30	1-GByte page (if Page-Size (PS) field is Set)
Page directory	PDE	PDPE	29:21	2-MByte page (if Page-Size (PS) field is Set)
Page table	PTE	PDE	20:12	4-KByte page

First-level translation may map input addresses to 4-KByte pages, 2-MByte pages, or 1-GByte pages. Support for 4-KByte pages and 2-MBytes pages are mandatory for first-level translation. Implementations supporting 1-GByte pages report it through the FL1GP field in the Capability Register (see Section 10.4.2). Figure 3-9 illustrates the translation process when it produces a 4-KByte page; Figure 3-10 covers the case of a 2-MByte page; Figure 3-11 covers the case of a 1-GByte page.

The following describe the first-level translation in more detail and how the page size is determined:

- When 5-level page-tables are used for first-level translation, a 4-KByte naturally aligned PML5 table is located at the physical address specified in First-level-page-table-pointer (FLPTPTR) field in the PASID-entry (see Section 9.5). A PML5 table comprises 512 64-bit entries (PML5Es). A PML5E is selected using the physical address defined as follows:
  - — Bits 2:0 are all 0.
  - — Bits 11:3 are bits 56:48 of the input address.
  - — Bits 12 and higher are from FLPTPTR field in the PASID-entry.

Because a PML5E is identified using bits 63:48 of the input address, it controls access to a 256-TByte region of the input-address space.

- When 5-level page-tables are used for first-level translation, a 4-KByte naturally aligned PML4 table is located at the physical address specified in address (ADDR) field in the PML5E (see Table 18). If 4-level page-tables are used for first-level translation, the 4-KByte naturally aligned PML4 table is located at the physical address specified in First-level-page-table-pointer (FLPTPTR) field in the PASID-entry (see Section 9.5). A PML4 table comprises 512 64-bit entries (PML4Es). A PML4E is selected using the physical address defined as follows:
  - Bits 2:0 are all 0.
  - Bits 11:3 are bits 47:39 of the input address.
  - Bits 12 and higher are from the ADDR field in the PML5E (with 5-level page-tables) or from FLPTPTR field in the PASID-entry (with 4-level page-tables).

  Because a PML4E is identified using bits 63:39 of the input address, it controls access to a 512-GByte region of the input-address space.
- A 4-KByte naturally aligned page-directory-pointer table is located at the physical address specified in address (ADDR) field in the PML4E (see Table 19). A page-directory-pointer table comprises 512 64-bit entries (PDPEs). A PDPE is selected using the physical address defined as follows:
  - Bits 2:0 are all 0.
  - Bits 11:3 are bits 38:30 of the input address.
  - Bits 12 and higher are from the ADDR field in the PML4E.

  Because a PDPE is identified using bits 63:30 of the input address, it controls access to a 1-GByte region of the input-address space. Use of the PDPE depends on its page-size (PS) field:
- If the PDPE's PS field is 1, the PDPE maps a 1-GByte page (see Table 20). The final physical address is computed as follows:
  - Bits 29:0 are from the input address.
  - Bits 30 and higher are from the ADDR field in the PDPE.
- If the PDPE's PS field is 0, a 4-KByte naturally aligned page directory is located at the physical address specified in the address (ADDR) field in the PDPE (see Table 21). A page directory comprises 512 64-bit entries (PDEs). A PDE is selected using the physical address defined as follows:
  - Bits 2:0 are all 0.
  - Bits 11:3 are bits 29:21 of the input address.
  - Bits 12 and higher are from the ADDR field in the PDPE.

  Because a PDE is identified using bits 63:21 of the input address, it controls access to a 2-MByte region of the input-address space. Use of the PDPE depends on its page-size (PS) field:
- If the PDE's PS field is 1, the PDE maps a 2-MByte page (see Table 22). The final physical address is computed as follows:
  - Bits 20:0 are from the input address.
  - Bits 21 and higher are from the ADDR field in the PDE.
- If the PDE's PS field is 0, a 4-KByte naturally aligned page table is located at the physical address specified in the address (ADDR) field in the PDE (see Table 23). A page table comprises 512 64-bit entries (PTEs). A PTE is selected using the physical address defined as follows:
  - Bits 2:0 are all 0.
  - Bits 11:3 are bits 20:12 of the input address.
  - Bits 12 and higher are from the ADDR field in the PDE.

  Because a PTE referenced by a PDE is identified using bits 63:12 of the input address, every such PTE maps a 4-KByte page (Table 24). The final page address is translated as follows:

— Bits 11:0 are from the input address.

— Bits 12 and higher are from the ADDR field in the PTE.

If a paging-structure entry's Present (P) field (bit 0) is 0 or if the entry sets any reserved field, the entry is used neither to reference another paging-structure entry nor to map a page. A reference using a input address whose translation would use such a paging-structure entry causes a translation fault (see Chapter 7).

The following bits are reserved with first-level translation:

- If the P field of a paging-structure entry is 1, bits 51:HAW (Host Address Width) are reserved.
- If the P field of a PML5E is 1, the PS field is reserved.
- If the P field of a PML4E is 1, the PS field is reserved.
- If 1-GByte pages are not supported and the P field of a PDPE is 1, the PS field is reserved.
- If the P field and PS field of a PDPE are both 1, bits 29:13 are reserved.
- If the P field and PS field of a PDE are both 1, bits 20:13 are reserved.
- If Extended-Accessed flag is not supported, the EA field in the paging entries are ignored.
- If No-Execute-Enable (NXE) field is 0 in the extended-context-entry and the P field of a paging-structure entry is 1, the Execute-Disable (XD) field is reserved.

3.6.1 Translation Faults

Requests-with-PASID can result in first-level translation faults for either of two reasons: (1) there is no valid translation for the input address; or (2) there is a valid translation for the input address, but its access rights do not permit the access. There is no valid translation if any of the following are true:

- The Root Table Type (RTT) field in Root-table Address register (RTADDR_REG) is 0.
- The input address in the request is not canonical (i.e., address bits 63:N have the same value as address bit [N-1], where N is 48-bits with 4-level paging and 57-bits with 5-level paging).
- Hardware attempt to access a translation entry (extended-root-entry, extended-context-entry, PASID-table entry, or a first-level paging-structure entry) resulted in error.
- The extended-root-entry used to process the request (as noted in Section 3.4.2) has the relevant present field as 0, has invalid programming, or has a reserved bit set.
- The extended-context-entry used to process the request (as noted in Section 3.4.3) has the P field as 0, PASIDE field as 0, ERE field as 0 (for requests with Execute-Requested (ER) field Set), SMEP field as 1 (for requests with Execute-Requests (ER) and Privileged-mode-Requested (PR) fields Set), has invalid programming, T field is programmed to block requests-with-PASID, or has a reserved bit set.
- The PASID-entry used to process the request (as noted in Section 3.6) has the P field as 0, or has the SRE field as 0 (for requests with Privileged-mode-Requested (PR) field Set).
- The translation process for that address (as noted in Section 3.6) used a paging-structure entry in which the P field is 0 or one that sets a reserved bit.

If there is a valid translation for an input address, its access rights are determined as described in Section 3.6.2.

Depending on the capabilities supported by remapping hardware units and the endpoint device, translations faults may be treated as non-recoverable errors or recoverable page faults. Chapter 7 provides detailed hardware behavior on translation faults and reporting to software.

3.6.2 Access Rights

The accesses permitted for a request-with-PASID whose input address that is successfully translated through first-level translation is determined by the attributes of the request and the access rights specified by the paging-structure entries controlling the translation.

Devices report support for requests-with-PASID through the PCI-Express PASID Capability structure. PASID Capability allows software to query and control if the endpoint can issue requests-with-PASID that request execute permission (such as for instruction fetches) and requests with supervisor-privilege. Remapping hardware implementations report support for requests seeking execute permission and requests seeking supervisor privilege through the Extended Capability Register (see ERS and SRS fields in Section 10.4.3).

The following describes how first-level translation determines access rights:

- For requests-with-PASID with supervisor privilege (value of 1 in Privilege-mode-Requested (PR) field) processed through a PASID-entry with SRE (Supervisor Requests Enable) field Set:
  - Data reads (Read requests with value of 0 in Execute-Requested (ER) field)
    - Data reads are allowed from any input address with a valid translation.
  - Instruction Fetches (Read requests with value of 1 in Execute-Requested (ER) field)
    - If No-Execute-Enable (NXE) field in extended-context-entry used to translate request is 0
      - If Supervisor-Mode-Execute-Protection (SMEP) field in extended-context-entry used to translate request is 0, instruction may be fetched from any input address with a valid translation.
      - If Supervisor-Mode-Execute-Protection (SMEP) field in extended-context-entry used to translate request is 1, instruction may be fetched from any input address with a valid translation for which the U/S field (bit 2) is 0 in at least one of the paging-structure entries controlling the translation.
    - If No-Execute-Enable (NXE) field in extended-context-entry used to translate request is 1
      - If Supervisor-Mode-Execute-Protection (SMEP) field in extended-context-entry used to translate request is 0, instruction may be fetched from any input address with a valid translation for which the XD field (bit 63) is 0 in every paging-structure entry controlling the translation.
      - If Supervisor-Mode-Execute-Protection (SMEP) field in extended-context-entry used to translate request is 1, instruction may be fetched from any input address with a valid translation for which, the U/S field is 0 in at least one of the paging-structure entries controlling the translation, and the XD field is 0 in every paging-structure entry controlling the translation.
  - Write requests and Atomics requests
    - If Write-Protect-Enable (WPE) field in extended-context-entry used to translate request is 0, writes are allowed to any input address with a valid translation.
    - If WPE=1, writes are allowed to any input address with a valid translation for which the R/W field (bit 1) is 1 in every paging-structure entry controlling the translation.
- For requests-with-PASID with user privilege (value of 0 in Privilege-mode-Requested (PR) field):
  - Data reads (Read requests with value of 0 in Execute-Requested (ER) field)
    - Data reads are allowed from any input address with a valid translation for which the U/S field is 1 in every paging-structure entry controlling the translation.
  - Instruction fetches (Read requests with value of 1 in Execute-Requested (ER) field)
    - If No-Execute-Enable (NXE) field in extended-context-entry used to translate request is 0, instructions may be fetched from any input address with a valid translation for which the U/S field is 1 in every paging structure entry controlling the translation.

- If No-Execute-Enable (NXE) field in extended-context-entry used to translate request is 1, instructions may be fetched from any input address with a valid translation for which the U/S field is 1 and XD field is 0 in every paging-structure entry controlling the translation.

— Write requests and Atomics requests

- Writes are allowed to any input address with a valid translation for which the R/W field and the U/S field are 1 in every paging-structure entry controlling the translation.

Remapping hardware may cache information from the paging-structure entries in translation caches. These caches may include information about access rights. Remapping hardware may enforce access rights based on these caches instead of on the paging structures in memory. This fact implies that, if software modifies a paging-structure entry to change access rights, the hardware might not use that change for a subsequent access to an affected input address. Refer to Chapter 6 for details on hardware translation caching and how software can enforce consistency with translation caches when modifying paging structures in memory.

3.6.3 Accessed, Extended Accessed, and Dirty Flags

For any paging-structure entry that is used during first-level translation, bit 5 is the Accessed (A) flag. For first-level paging-structure entries referenced through a Extended-context-entry with EAFE=1, bit 10 is the Extended-Accessed flag. For paging-structure entries that map a page (as opposed to referencing another paging structure), bit 6 is the Dirty (D) flag. These flags are provided for use by memory-management software to manage the transfer of pages and paging structures into and out of physical memory.

- Whenever the remapping hardware uses a first-level paging-structure entry as part of input-address translation, it atomically sets the A field in that entry (if it is not already set).
- If the Extended-Accessed-Flag-Enable (EAFE) is 1 in a Extended-context-entry that references a first-level paging-structure entry used by hardware, it atomically sets the EA field in that entry. Whenever EA field is atomically set, the A field is also set in the same atomic operation. For software usages where the first-level paging structures are shared across heterogeneous agents (e.g., CPUs and accelerator devices such as GPUs), EA flag may be used by software to identify pages accessed by non-CPU agent(s) (as opposed to the A flag which indicates access by any agent sharing the paging structures).
- Whenever there is a write to a input address, the remapping hardware atomically sets the D field (if it is not already set) in the paging-structure entry that identifies the final translated address for the input address (either a PTE or a paging-structure entry in which the PS field is 1). The atomic operation that sets the D field also sets the A field (and the EA field, if EAFE=1 as described above).

Memory-management software may clear these flags when a page or a paging structure is initially loaded into physical memory. These flags are “sticky”, meaning that, once set, the remapping hardware does not clear them; only software can clear them.

Remapping hardware may cache information from the first-level paging-structure entries in translation caches (see Chapter 6). These caches may include information about accessed, extended-accessed, and dirty flags. This fact implies that, if software modifies an accessed flag, extended-accessed flag, or a dirty flag from 1 to 0, the hardware might not set the corresponding bit in memory on a subsequent access using an affected input address. Refer to Chapter 6 for details on hardware translation caching and how software can enforce consistency with translation caches when modifying paging structures in memory.

3.6.4 Snoop Behavior

Snoop behavior for a memory access (to a translation structure entry or access to the mapped page) specifies if the access is coherent (snoops the processor caches) or not. The snoop behavior is independent of the memory typing described in Section 3.6.5. When processing requests-with-PASID through first-level translation, the snoop behavior for various accesses is specified as follows:

- Access to extended-root and extended-context-entries are snooped if the Coherency (C) field in Extended Capability Register (see Section 10.4.3) is reported as 1. These accesses are not required to be snooped if the field is reported as 0.
- Access to PASID-table entries are always snooped.
- Accesses to first-level paging-entries (PML5E, PML4E, PDPE, PDE, PTE) are always snooped.
- Access to the page mapped through first-level translation is always snooped (independent of the value of the No-Snoop (NS) attribute in the request).

3.6.5 Memory Typing

The memory type of a memory access (to a translation structure entry or access to the mapped page) refers to the type of caching used for that access. Refer to Intel® 64 processor specifications for definition and properties of each supported memory-type (UC, UC-, WC, WT, WB, WP). Support for memory typing in remapping hardware is reported through the Memory-Type-Support (MTS) field in the Extended Capability Register (see Section 10.4.3). This section describes how first-level translation contributes to determination of memory typing.

- Memory-type has no meaning (and hence ignored) for memory accesses from devices operating outside the processor coherency domain.
- Memory-type applies for memory accesses from devices (such as Intel® Processor Graphics device) operating inside the processor coherency domain.

When processing requests-with-PASID from devices operating in the processor coherency domain, the memory type for any access through first-level translation is computed as follows:

- If cache-disable (CD) field in the extended-context-entry used to process the request is 1, all accesses use memory-type of uncacheable (UC).
- If CD field is 0, the memory-type for accesses is computed as follows:
  - Access to extended-root & extended-context-entries use memory-type of uncacheable (UC).
  - Access to PASID-table entries use memory-type from MTRR (described in Section 3.6.5.2).
  - Memory-type for access to first-level translation-structure entries (PML5E, PML4E, PDPE, PDE, and PTE), and for access to the page itself is computed as follows:
    - First, the memory-type specified by the Page Attribute Table (PAT) is computed. PAT is a 32-bit field in the extended-context-entry, comprising eight 4-bit entries (entry *i* comprises bits $4i+3:4i$). Refer to Section 9.4 for the exact format of the PAT field in the extended-context-entry.
    - Second, the memory-type for the target physical address as specified by the Memory Type Range Registers (MTRRs) is computed. MTRRs provide a mechanism for associating the memory types with physical-address ranges in system memory. MTRR Registers are described in detail in Section 10.4.38, Section 10.4.39, Section 10.4.40, and Section 10.4.41. For details on MTRR usage and configuration requirements refer to Intel® 64 processor specifications.
    - Finally, the effective memory-type is computed by combining the memory-types computed from PAT and MTRR.

The following sub-sections describe details of computing memory-type from PAT, memory type from MTRR, and how to combine them to form the effective memory type.

3.6.5.1 Selecting Memory Type from Page Attribute Table

Memory-type selection from Page Attribute Table requires hardware to form a 3-bit index made up of the PAT, PCD and PWT bits from the respective paging-structure entries. The PAT bit is bit 7 in page-table entries that point to 4-KByte pages and bit 12 in paging-structure entries that point to larger pages. The PCD and PWT bits are bits 4 and 3, respectively, in paging-structure entries that point to pages of any size.

The PAT memory-type comes from entry *i* of the Page Attribute Table in the extended-context-entry controlling the request, where *i* is defined as follows:

- For access to PML5E, *i* = 2*PCD+PWT, where the PCD and PWT values come from the PASID-table entry.
- For access to PML4E with 4-level paging, *i* = 2*PCD+PWT, where the PCD and PWT values come from the PASID-table entry.
- For access to a paging-structure entry X whose address is in another paging structure entry Y (i.e., PDPE, PDE and PTE), *i* = 2*PCD+PWT, where the PCD and PWT values come from Y.
- For access to the physical address that is the translation of an input address, *i* = 4*PAT+2*PCD+PWT, where the PAT, PCD, and PWT values come from the relevant PTE (if the translation uses a 4-KByte page), the relevant PDE (if the translation uses a 2-MByte page), or the relevant PDPE (if the translation uses a 1-GByte page).

3.6.5.2 Selecting Memory Type from Memory Type Range Registers

Remapping hardware implementations reporting Memory-Type-Support (MTS) field as Set in the Extended Capability Register support the Memory Type Range Registers (MTRRs). These include the MTRR Capability Register (see Section 10.4.38), MTRR Default Type Register (see Section 10.4.39), fixed-range MTRRs (see Section 10.4.40), and variable-range MTRRs (see Section 10.4.41).

Selection of memory-type from the MTRR registers function as follows:

- If the MTRRs are not enabled (Enable (E) field is 0 in the MTRR Default Type Register), then MTRR memory-type is uncacheable (UC).
- If the MTRRs are enabled (E=1 in MTRR Default Type Register), then the MTRR memory-type is determined as follows:
  - If the physical address falls within the first 1-MByte and fixed MTRRs are enabled, the MTRR memory-type is the memory-type stored for the appropriate fixed-range MTRR (see Section 10.4.40).
  - Otherwise, hardware attempts to match the physical address with a memory type set by the variable-range MTRRs ((see Section 10.4.41):
    - If one variable memory range matches, the MTRR memory-type is the memory type stored in the MTRR_PHYSBASEn_REG Register for that range.
    - If two or more variable memory ranges match and the memory-types are identical, then MTRR memory-type is that memory-type.
    - If two or more variable memory ranges match and one of the memory types is UC, then MTRR memory-type is UC.
    - If two or more variable memory ranges match and the memory types are WT and WB, then MTRR memory-type is WT.
    - For overlaps not defined by above rules, hardware behavior is undefined.
  - If no fixed or variable memory range matches, then the MTRR memory-type is the default memory-type from the MTRR Default Type Register (see Section 10.4.39).

3.6.5.3 Selecting Effective Memory Type

When the cache-disable (CD) field in extended-context-entry is 0, the effective memory-type for an access is computed from the PAT memory-type and the MTRR memory-type as illustrated in Table 4 below.

Remapping hardware may cache information from the first-level paging-structure entries in translation caches (see Chapter 6). These caches may include information about memory typing. Hardware may use memory-typing information from these caches instead of from the paging structures in memory. This fact implies that, if software modifies a paging-structure entry to change the memory-typing bits, hardware might not use that change for a subsequent translation using that

Table 4. Effective Memory Types

MTRR Memory Type	PAT Memory Type	Effective Memory Type
UC	UC	UC
	UC-	UC
	WC	WC
	WT	UC
	WB	UC
	WP	UC
WC	UC	UC
	UC-	WC
	WC	WC
	WT	UC
	WB	WC
	WP	UC
WT	UC	UC
	UC-	UC
	WC	WC
	WT	WT
	WB	WT
	WP	WP
WB	UC	UC
	UC-	UC
	WC	WC
	WT	WT
	WB	WB
	WP	WP
WP	UC	UC
	UC-	WC
	WC	WC
	WT	WT
	WB	WP
	WP	WP

entry or for access to an affected input-address. Refer to Chapter 6 for details on hardware translation caching and how software can enforce consistency with translation caches when modifying paging structures in memory.

3.7 Second-Level Translation

Context and extended-context-entries can be configured to support second-level translation. Second-level translation applies to requests-without-PASID, but can also be applied (nested) with first-level translation for requests-with-PASID. This section describes the use of second-level translation for requests-without-PASID. Section 3.8 describes the nested use of second-level translation for requests-with-PASID.

Context and extended-context-entries contain a pointer to the base of the second-level translation structure. Section 9.3 and Section 9.4 describe the exact format of the context and extended-context-entries. Second-level translation restricts input-address to an implementation specific address-width reported through the Maximum Guest Address Width (MGAW) field in the Capability Register. Requests-without-PASID arriving at the remapping hardware are subject to MGAW address checking, and any violations are treated as translation-fault. Chapter 7 provides details of fault conditions and its reporting to software.

Second-level translation uses a hierarchical paging structure as described in Section 3.5. To allow page-table walks with 9-bit stride, the Adjusted Guest Address Width (AGAW) value for a domain is defined as its Guest Address Width (GAW) value adjusted, such that (AGAW-12) is a multiple of 9. The AGAW indicates the number of levels of page-walk. Hardware implementations report the supported AGAWs through the Capability Register. Second-level translation may map input addresses to 4-KByte pages, 2-MByte pages, or 1-GByte pages. Implementations report support in second-level translation for 2-MByte and 1-GByte large-pages through the Capability Register. Figure 3-9 illustrates the translation process for a 4-level paging structure when it produces a 4-KByte page; Figure 3-10 illustrates mapping to a 2-MByte page; Figure 3-11 illustrates mapping to a 1-GByte page.

Table 5. Second-level Paging Structures

Paging Structure	Entry Name	Physical Address of Structure	Bits Selecting Entry	Page Mapping
Second-level PML5 table	SL-PML5E	Context-entry or Extended-Context-entry (with 5-level translation)	56:48	N/A
Second-level PML4 table	SL-PML4E	SL-PML5E (with 5-level translation); Context-entry or Extended-Context-entry (with 4-level translation)	47:39	N/A
Second-level Page-directory-pointer table	SL-PDPE	SL-PML4E (with 4-level translation); Context-entry or Extended-Context-entry (with 3-level translation)	38:30	1-GByte page (if Page Size (PS) field is Set)
Second-level Page directory	SL-PDE	SL-PDPE	29:21	2-MByte page (if Page-Size (PS) field is Set)
Second-level Page table	SL-PTE	SL-PDE	20:12	4-KByte page

Table 5 gives the different names of the second-level translation structures, that are given based on their use in the translation process. It also provides, for each structure, the source of the physical-address used to locate it, the bits in the input-address used to select an entry from the structure, and details of whether and how such an entry can map a page. Section 9.8 describes format of each of these paging structures in detail.

The following describe the second-level translation in more detail and how the page size is determined:

- For implementations supporting a 5-level paging structure for second-level paging, a 4-KByte naturally aligned second-level-PML5 table is located at the physical address specified in the SLPTPTR field in the extended-context-entry (or context-entry). A second-level-PML5 table

comprises 512 64-bit entries (SL-PML5Es). A SL-PML5E is selected using the physical address defined as follows:

— Bits 2:0 are all 0.

— Bits 11:3 are bits MGAW:48 of the input address.

— Bits 12 and higher are from SLPTPTR field in the extended-context-entry (or context-entry).

Because a SL-PML5E is identified using bits MGAW:48 of the input address, it controls access to a 256-TByte region of the input-address space.

- For implementations supporting a 5-level paging structure for second-level translation, a 4-KByte naturally aligned second-level-PML4 table is located at the physical address specified in the address (ADDR) field in the SL-PML5E (see Table 25). For implementations supporting a 4-level paging structure for second-level paging, a 4-KByte naturally aligned second-level-PML4 table is located at the physical address specified in the SLPTPTR field in the extended-context-entry (or context-entry). A second-level-PML4 table comprises 512 64-bit entries (SL-PML4Es). A SL-PML4E is selected using the physical address defined as follows:

  — Bits 2:0 are all 0.

  — Bits 11:3 are bits 47:39 (for 5-level translation) or MGAW:39 (for 4-level translation) of the input address.

  — Bits 12 and higher are from SLPTPTR field in the extended-context-entry (or context-entry).

  Because a SL-PML4E is identified using bits MGAW:39 of the input address, it controls access to a 512-GByte region of the input-address space.

- For implementations supporting 5-level or 4-level paging structures for second-level translation, a 4-KByte naturally aligned page-directory-pointer table is located at the physical address specified in the address (ADDR) field in the SL-PML4E (see Table 26). For implementations supporting 3-level paging structure, the 4-KByte naturally aligned page-directory-pointer table is located at the physical address specified in the SLPTPTR field in the extended-context-entry (or context-entry). A page-directory-pointer table comprises of 512 64-bit entries (SL-PDPEs). A SL-PDPE is selected using the physical address defined as follows:

  — Bits 2:0 are all 0.

  — Bits 11:3 are bits 38:30 (for 4-level or 5-level translation) or MGAW:30 (for 3-level translation) of the input address.

  — Bits 12 and higher are from the ADDR field in the SL-PML4E (or from the SLPTPTR field in the extended-context-entry (or context-entry) for implementations supporting 3-level paging structure).

  Because a SL-PDPE is identified using bits MGAW:30 of the input address, it controls access to a 1-GByte region of the input-address space. Use of the SL-PDPE depends on its page-size (PS) field:

- If the SL-PDPE's PS field is 1, the SL-PDPE maps a 1-GByte page (see Table 27). The final physical address is computed as follows:

  — Bits 29:0 are from the input address.

  — Bits 30 and higher are from the ADDR field in the SL-PDPE.

- If the SL-PDPE's PS field is 0, a 4-KByte naturally aligned second-level page directory is located at the physical address specified in the address (ADDR) field in the SL-PDPE (see Table 28). A second-level page directory comprises 512 64-bit entries (SL-PDEs). A PDE is selected using the physical address defined as follows:

  — Bits 2:0 are all 0.

  — Bits 11:3 are bits 29:21 of the input address.

  — Bits 12 and higher are from the ADDR field in the SL-PDPE.

  Because a SL-PDE is identified using bits MGAW:21 of the input address, it controls access to a 2-MByte region of the input-address space. Use of the SL-PDPE depends on its page-size (PS) field:

- If the SL-PDE's PS field is 1, the SL-PDE maps a 2-MByte page (see Table 29). The final physical address is computed as follows:
  - Bits 20:0 are from the input address.
  - Bits 21 and higher are from the ADDR field in the SL-PDE.
- If the SL-PDE's PS field is 0, a 4-KByte naturally aligned second-level page-table is located at the physical address specified in the address (ADDR) field in the SL-PDE (see Table 30). Such a second-level page-table comprises 512 64-bit entries (SL-PTEs). A SL-PTE is selected using the physical address defined as follows:
  - Bits 2:0 are all 0.
  - Bits 11:3 are bits 20:12 of the input address.
  - Bits 12 and higher are from ADDR field in the SL-PDE.

  Because a SL-PTE referenced by a SL-PDE is identified using bits MGAW:12 of the input address, every such SL-PTE maps a 4-KByte page (Table 31). The final page address is translated as follows:
  - Bits 11:0 are from the input address.
  - Bits 12 and higher are from the ADDR field in the SL-PTE.

If a second-level paging-structure entry's Read (R) and Write (W) fields[1] are both 0 or if the entry sets any reserved field, the entry is used neither to reference another paging-structure entry nor to map a page. A reference using an input address whose translation would use such a paging-structure entry causes a translation error (see Chapter 7).

The following bits are reserved with second-level translation:

- If either the R or W field of a paging-structure entry is 1, bits 51: HAW are reserved.
- If either the R or W field of a SL-PML5E is 1, the PS field is reserved.
- If either the R or W field of a SL-PML4E is 1, the PS field is reserved.
- If 1-GByte pages are not supported and the R or W fields of a SL-PDPE is 1, the PS field is reserved.
- If the R or W fields of a SL-PDPE is 1, and PS field in that SL-PDPE is 1, bits 29:12 are reserved.
- If 2-MByte pages are not supported and the R or W fields of a SL-PDE is 1, the PS field is reserved.
- If either the R or W field of a SL-PDE is 1, and the PS field in that SL-PDE is 1, bits 20:12 are reserved.
- If either the R or W field of a non-leaf paging-structure entry (i.e. SL-PML5E, SL-PML4E, SL-PDPE, or SL-PDE with PS=0) is 1, the SNP (Snoop) field and the TM (Transient Mapping) field are reserved.
- If either the R or W field of a SL-PTE is 1, and Snoop Control (SC) is reported as 0 in Extended Capability Register, the SNP field is reserved.
- If either the R or W field of a SL-PTE is 1, and Device-TLBs (DT) is reported as 0 in Extended Capability Register, the TM field is reserved.

1. Execute (X) field in second-level paging-structure entries is ignored when translating requests-without-PASID. Refer to Section 3.8 for Execute (X) field usage with nested translation of requests-with-PASID.

3.7.1 Translation Faults

Requests-without-PASID can result in second-level translation faults for either of two reasons: (1) there is no valid translation for the input address; or (2) there is a valid translation for the input address, but its access rights do not permit the access. There is no valid translation if any of the following are true:

- A hardware attempt to access a translation entry (root/extended-root entry, context/extended-context entry, or a second-level paging-structure entry) resulted in error.
- The root-entry or extended-root-entry used to process the request (as described in Section 3.4.2) has the relevant present field as 0, has invalid programming, or has a reserved bit set.
- The context-entry or extended-context-entry used to process the request (as described in Section 3.4.3) has the present (P) field as 0, has invalid programming, is programmed to block requests-without-PASID, or has a reserved bit set.
- The input address in the request-without-PASID is above (2^X - 1), where X is the minimum of MGAW and AGAW corresponding to address-width programmed in the context-entry or extended-context-entry.
- The translation process for that address (as described in Section 3.6) used a second-level paging-structure entry in which the R and W fields are both 0 or one that sets a reserved bit.

If there is a valid translation for an input address, its access rights are determined as described in Section 3.7.2. Depending on capabilities supported by remapping hardware and endpoint device, translations faults can be treated as non-recoverable errors or recoverable faults (see Chapter 7 for details).

3.7.2 Access Rights

The accesses permitted for a request-without-PASID whose input address is successfully translated through second-level translation is determined by the attributes of the request and the access rights specified by the second-level paging-structure entries controlling the translation.

Devices can issue requests-without-PASID for reads, writes, or atomics. The following describes how second-level translation determines access rights for such requests:

- Read request without PASID:
  - — Reads are allowed from any input address with a valid translation for which the Read (R) field is 1 in every paging-structure entry controlling the translation.
- Write request without PASID:
  - — Writes are allowed to any input address with a valid translation for which the Write (W) field is 1 in every paging-structure entry controlling the translation.
- Atomics request without PASID:
  - — Atomics requests are allowed from any input address with a valid translation for which the Read (R) and Write (W) fields are both 1 in every paging-structure entry controlling the translation.

Remapping hardware may cache information from the second-level paging-structure entries in translation caches. These caches may include information about access rights. Remapping hardware may enforce access rights based on these caches instead of on the paging structures in memory. This fact implies that, if software modifies a paging-structure entry to change access rights, the hardware might not use that change for a subsequent access to an affected input address. Refer to Chapter 6 for details on hardware translation caching and how software can enforce consistency with translation caches when modifying paging structures in memory.

3.7.3 Snoop Behavior

When processing requests-without-PASID through second-level translation, the snoop behavior for various accesses are specified as follows:

- Access to extended-root and extended-context-entries are snooped if the Coherency (C) field in Extended Capability Register (see Section 10.4.3) is reported as 1. These accesses are not required to be snooped if the field is reported as 0.
- Accesses to second-level paging-entries (SL-PML5E, SL-PML4E, SL-PDPE, SL-PDE, SL-PTE) are snooped if the Coherency (C) field in Extended Capability Register (see Section 10.4.3) is reported as 1. These accesses are not required to be snooped if the field is reported as 0.
- Accesses to a page mapped through second-level translation has snoop behavior as follows:
  - If the Snoop Control (SC) field in extended capability Register is reported as 0, snoop behavior for access to the page mapped through second-level translation is determined by the no-snoop attribute in the request.
  - If the SC field in Extended Capability Register is reported as 1, the snoop behavior for access to the translated address is controlled by the value of the Snoop (SNP) field in the leaf paging-structure entry controlling the second-level translation. If the SNP field in the paging-structure entry is 1, the processor caches are snooped independent of the no-snoop attribute in the request. If the SNP field in the paging-structure entry is 0, the snoop behavior follows the no-snoop attribute in the request.

3.7.4 Memory Typing

This section describes how second-level translation contributes to determination of memory typing for requests-without-PASID.

- Memory-type is ignored for memory accesses from remapping requests from devices operating outside the processor coherency domain.
- Memory-type applies for memory accesses from remapping requests from devices operating inside the processor coherency domain.

When processing requests-without-PASID from devices operating in the processor coherency domain, the memory type for any access through second-level translation is computed as follows:

- If cache-disable (CD) field in the extended-context-entry used to process the request is 1, all accesses use memory-type of uncacheable (UC).
- If cache-disable (CD) is 0 in the extended-context-entry, or if the context-entry is used, the memory-type for accesses is computed as follows:
  - Access to root/extended-root entries and context/extended-context-entries use memory-type of uncacheable (UC).
  - Access to second-level translation entries (SL-PML5E, SL-PML4E, SL-PDPE, SL-PDE, SL-PTE) and the final page use memory-type of write-back (WB).

3.8 Nested Translation

When Nesting Enable (NESTE) field is 1 in extended-context-entries, requests-with-PASID translated through first-level translation are also subjected to nested second-level translation. Such extended-context-entries contain both the pointer to the PASID-table (which contains the pointer to the first-level translation structures), and the pointer to the second-level translation structures. Figure 3-12 illustrates the nested translation for a request-with-PASID mapped to a 4-KByte page through first-level translation, and interleaved through 4-KByte mappings in second-level paging structures.

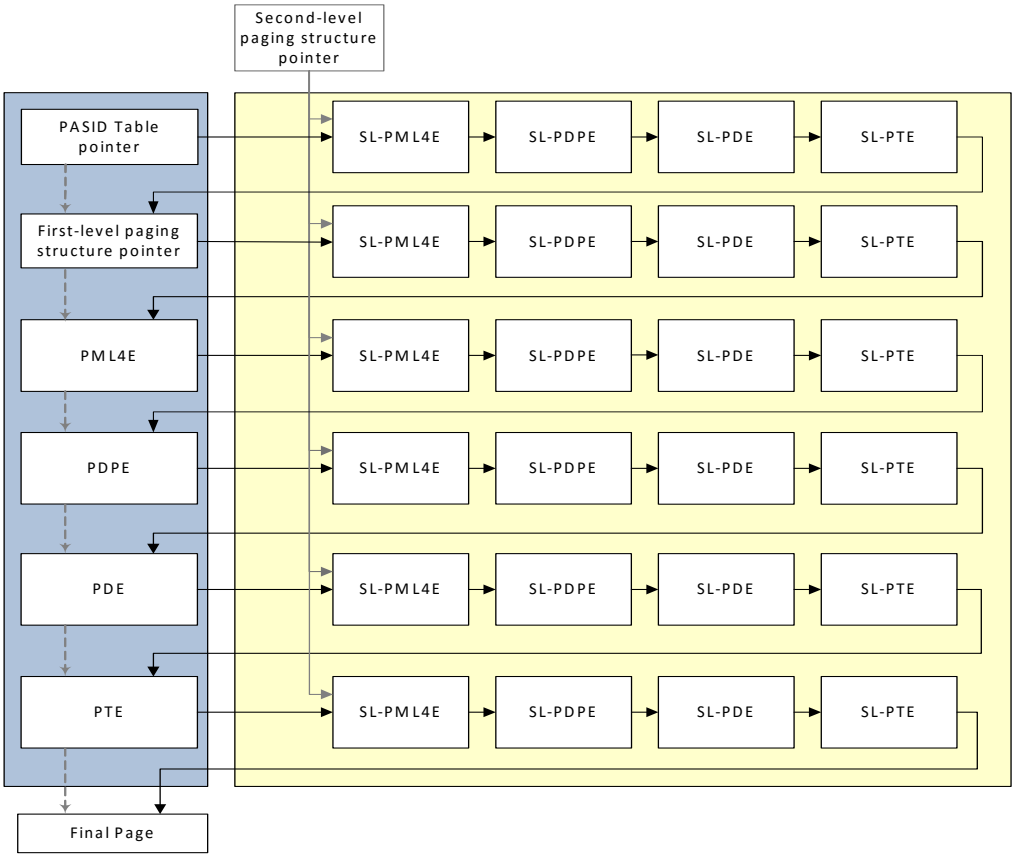


Figure 3-12. Nested Translation with 4-KByte pages

With nesting, all memory accesses generated when processing a request-with-PASID through first-level translation are subjected to second-level translation. This includes access to PASID-table entry, access to first-level paging structure entries (PML5E, PML4E, PDPE, PDE, PTE), and access to the output address from first-level translation. With nested translation, a guest operating system running within a virtual machine may utilize first-level translation as described in Section 2.5.1.3, while the virtual machine monitor may virtualize memory by enabling nested second-level translations.

The first-level translation follows the same process as described in Section 3.6 to map input addresses to 4-KByte, 2-MByte or 1-GByte pages. The second-level translation is interleaved at each step, and follows the process described in Section 3.7 to map input addresses to 4-KByte, 2-MByte or 1-GByte pages.

3.8.1 Translation Faults

Requests-with-PASID subjected to nested translation can result in fault at the first-level translation or any of the second-level translation stages. Translation faults at a level can result from either of two reasons: (1) there is no valid translation for the respective input address; or (2) there is a valid translation for the respective input address, but its access rights do not permit the access.

There is no valid translation if any of the following are true:

- The Root Table Type (RTT) field in Root-table Address register (RTADDR_REG) is 0
- The input address in the request-with-PASID is not canonical (i.e., address bits 63:N have the same value as address bit [N-1], where N is 48-bits with 4-level paging and 57-bits with 5-level paging).
- A hardware attempt to access a translation entry (extended-root-entry, extended-context-entry, PASID-table entry, first-level paging-structure entry, second-level paging-structure entry) resulted in error.
- The extended-root-entry used to process the request (as noted in Section 3.4.2) has the relevant present field as 0, has invalid programming, or has a reserved bit set.
- The extended-context-entry used to process the request (as noted in Section 3.4.3) has the P field as 0, PASIDE field as 0, ERE field as 0 (for requests with Execute-Requested (ER) field Set), SMEP field as 1 (for requests with Execute-Requests (ER) and Privileged-mode-Requested (PR) fields Set), has invalid programming, T field is programmed to block requests-with-PASID, or has a reserved bit set.
- The PASID-entry used to process the request (as noted in Section 3.6) has the P field as 0, or has the SRE field as 0 (for requests with Privileged-mode-Requested (PR) field Set).
- The first-level translation process for the address in the request-with-PASID (as noted in Section 3.6) used a first-level paging-structure entry in which the P field is 0 or one that sets a reserved bit.
- Input address for any of the second-level translation stages is above (2^X - 1), where X is the minimum of MGAW and AGAW corresponding to the address-width programmed in the extended-context-entry used.
- The second-level translation (as noted in Section 3.6) for address of a paging-structure entry (PASID-entry, PML5E, PML4E, PDPE, PDE, PTE) used a second-level paging-structure entry in which both the R and W field is 0, or one that sets a reserved bit.
- The second-level translation (as noted in Section 3.6) for output address from first-level translation (address of final page) used a second-level paging-structure entry in which both the R and W field is 0, or has a reserved field set.

If there is a valid second-level translation for output address from first-level translation (address of final page), its access rights are determined as described in Section 3.8.2.

Depending on the capabilities supported by remapping hardware units and the endpoint device, translations faults may be treated as non-recoverable errors or recoverable page faults. Chapter 7 provides detailed hardware behavior on translation faults and reporting to software.

3.8.2 Access Rights

For requests-with-PASID subjected to nested translation, access rights are checked at both first and second levels of translation.

Access rights checking for first-level translation follows the behavior described in Section 3.6.2.

Access rights for second-level translations function as follows:

- Access to paging structures (First-level paging structure pointer, PML5E, PML4E, PDPE, PDE, PTE) are treated as requests-without-PASID.

  - — Reads of paging structures
    - Read of paging structures are allowed from any input address with a valid translation for which the Read (R) field is 1 in every second-level paging-entry controlling the translation.
  - — Accessed (A), Extended-Accessed (EA), Dirty (D) flag update of first-level paging-structure entries
    - Atomic A/EA/D flag update of first-level paging-entries are allowed from any input address with a valid translation for which the Read (R) and Write (W) fields are 1 in every second-level paging-entry controlling the translation to the respective first-level paging-entry.
- Access to the final page is treated as request-with-PASID. Access rights checking for the final page access functions as follows:
  - — Data reads (Read requests with value of 0 in Execute-Requested (ER) field)
    - Data reads are allowed from any input address with a valid translation for which the Read (R) field is 1 in every second-level paging-structure entry controlling the translation.
  - — Instruction Fetches (Read requests with value of 1 in Execute-Requested (ER) field)
    - If Second-level Execute-Enable (SLEE) field in extended-context-entry used to translate request is 0
      - — Instruction fetches are allowed from any input address with a valid translation for which the Read (R) field is 1 in every second-level paging-entry controlling the translation.
    - If Second-level Execute-Enable (SLEE) field in extended-context-entry used to translate request is 1
      - — Instruction fetches are allowed from any input address with a valid translation for which the Read (R) and Execute (X) fields are both 1 in every second-level paging-entry controlling the translation.
  - — Write requests
    - Writes are allowed from any input address with a valid translation for which the Write (W) field is 1 in every second-level paging-entry controlling the translation.
  - — Atomics requests
    - Atomics requests are allowed from any input address with a valid translation for which both the Read (R) and Write (W) fields are 1 in every second-level paging-entry controlling the translation.

With nested translations, remapping hardware may cache information from both first-level and second-level paging-structure entries in translation caches. These caches may include information about access rights. Remapping hardware may enforce access rights based on these caches instead of on the paging structures in memory. This fact implies that, if software modifies a paging-structure entry to change access rights, the hardware might not use that change for a subsequent access to an affected input address. Refer to Chapter 6 for details on hardware translation caching and how software can enforce consistency with translation caches when modifying paging structures in memory.

3.8.3 Snoop Behavior

When processing requests-with-PASID through nested translation, the snoop behavior for various accesses are specified as follows:

- Access to extended-root and extended-context-entries are snooped if the Coherency (C) field in Extended Capability Register (see Section 10.4.3) is reported as 1. These accesses are not required to be snooped if the field is reported as 0.
- Accesses to second-level paging-entries (SL-PML5E, SL-PML4E, SL-PDPE, SL-PDE, SL-PTE) are snooped if the Coherency (C) field in Extended Capability Register (see Section 10.4.3) is reported as 1. These accesses are not required to be snooped if the field is reported as 0.
- Access to PASID-table entries are always snooped.

- Accesses to first-level paging-entries (PML5E, PML4E, PDPE, PDE, PTE) are always snooped.
- Access to the page mapped through nested (first-level and second-level) translation is always snooped.

3.8.4 Memory Typing

This section describes how nested translation contributes to determination of memory typing for requests-with-PASID.

- Memory-type is ignored for memory accesses from remapping requests from devices operating outside the processor coherency domain.
- Memory-type applies for memory accesses from remapping requests from devices operating inside the processor coherency domain.

When processing requests-with-PASID from devices operating in the processor coherency domain, the memory type for any access through nested translation is computed as follows:

- If cache-disable (CD) field in the extended-context-entry used to process the request is 1, all accesses use memory-type of uncacheable (UC).
- If CD field is 0, the memory-type for accesses is computed as follows:
  - Access to extended-root-entries & extended-context-entries use memory-type of uncacheable (UC).
  - Memory-type for access to second-level translation-structure entries (SL-PML5E, SL-PML4E, SL-PDPE, SL-PDE, and SL-PTE) used to translate access to PASID-table entry, PML5E, PML4E, PDPE, PDE and PTE entries is computed as follows:
    - If extended memory-type enable (EMTE) field in the extended-context-entry used is 0, memory-type of write-back (WB) is used.
    - If EMTE field in extended-context-entry used is 1, memory-type specified in the extended memory-type (EMT) field in the extended-context-entry is used.
  - Memory-type for access to PASID-table entries is computed as follows:
    - If extended memory type enable (EMTE) field in the extended-context-entry used is 0, memory-type from MTRR (as described in Section 3.6.5.2) is used.
    - If EMTE field in extended-context-entry used is 1, memory-type specified in the EMT field in the last (leaf) second-level paging-structure entry used to translate the PASIDPTR field is used.
  - Memory-type for access to first-level translation-structure entries (PML5E, PML4E, PDPE, PDE, and PTE), and memory-type for access to the final page, is computed as follows:
    - First, the first-level memory-type specified by the Page Attribute Table (PAT) is computed. This is identical to the PAT memory-type computation with first-level only translation as described in Section 3.6.5.1.
    - If extended memory-type enable (EMTE) field in the extended-context-entry used is 0, effective memory-type used is computed by combining the first-level PAT memory-type computed above with the MTRR memory-type as described in Section 3.6.5.3.
    - If EMTE field in extended-context-entry used is 1, memory-type is computed as follows:
      - During the second-level translation to access the respective first-level paging entry, the ignore-PAT (IGPT) and extended memory-type (EMT) fields from the last (leaf) second-level translation-structure entry used is fetched.
      - If IGPT field is 1, the PAT memory-type computed from first-level translation is ignored, and memory-type specified by the EMT field is used as the memory-type for the access.
      - If IGPT field is 0, the effective memory-type for the access is computed by combining the first-level PAT memory-type above with the EMT field. The effective memory-type

computation follows the same rules described in Table 4 in Section 3.6.5.3, except memory-type specified by the EMT field is used instead of the MTRR memory-type.

With nesting, remapping hardware may cache information from the first-level and second-level paging-structure entries in translation caches (see Chapter 6). These caches may include information about memory typing. Hardware may use memory-typing information from these caches instead of from the paging structures in memory. flags. This fact implies that, if software modifies a paging-structure entry to change the memory-typing bits, hardware might not use that change for a subsequent translation using that entry or for access to an affected input-address. Refer to Chapter 6 for details on hardware translation caching and how software can enforce consistency with translation caches when modifying paging structures in memory.

3.9 Identifying Origination of DMA Requests

In order to support usages requiring isolation, the platform must be capable of uniquely identifying the requestor (Source-Id) for each DMA request. The DMA sources in a platform and use of source-id in these requests may be categorized as below.

3.9.1 Devices Behind PCI-Express to PCI/PCI-X Bridges

The PCI-Express-to-PCI/PCI-X bridges may generate a different requester-id and tag combination in some instances for transactions forwarded to the bridge's PCI-Express interface. The action of replacing the original transaction's requester-id with one assigned by the bridge is generally referred to as taking 'ownership' of the transaction. If the bridge generates a new requester-id for a transaction forwarded from the secondary interface to the primary interface, the bridge assigns the PCI-Express requester-id using the secondary interface's bus number, and sets both the device number and function number fields to zero. Refer to the PCI-Express-to-PCI/PCI-X bridge specifications for more details.

For remapping requests from devices behind PCI-Express-to-PCI/PCI-X bridges, software must consider the possibility of requests arriving with the source-id in the original PCI-X transaction or the source-id provided by the bridge. Devices behind these bridges can only be collectively assigned to a single domain. When setting up remapping structures for these devices, software must program multiple context entries, each corresponding to the possible set of source-ids. Each of these context-entries must be programmed identically to ensure the DMA requests with any of these source-ids are processed identically.

3.9.2 Devices Behind Conventional PCI Bridges

For devices behind conventional PCI bridges, the source-id in the DMA requests is the requester-id of the bridge device. For remapping requests from devices behind conventional PCI bridges, software must program the context-entry corresponding to the bridge device. Devices behind these bridges can only be collectively assigned to a single domain.

3.9.3 Root-Complex Integrated Devices

Transactions generated by all root-complex integrated devices must be uniquely identifiable through its source-id (PCI requester-id). This enables any root-complex integrated endpoint device (PCI or PCI-Express) to be independently assigned to a domain.

3.9.4 PCI-Express Devices Using Phantom Functions

To increase the maximum possible number of outstanding requests requiring completion, PCI-Express allows a device to use function numbers not assigned to implemented functions to logically extend the Tag identifier. Unclaimed function numbers are referred to as Phantom Function Numbers (PhFN). A device reports its support for phantom functions through the Device Capability configuration register, and requires software to explicitly enable use of phantom functions through the Device Control configuration register.

Since the function number is part of the requester-id used to locate the context-entry for processing a DMA request, when assigning PCI-Express devices with phantom functions enabled, software must program multiple context entries, each corresponding to the PhFN enabled for use by the device function. Each of these context-entries must be programmed identically to ensure the DMA requests with any of these requester-ids are processed identically.

3.10 Handling Requests from Processor Graphics Device

Processor graphics device on Intel platforms is a root-complex integrated PCI-Express device, that is integrated to the processor or chipset package. Processor graphics device may support multiple agents behind it such as the display agent, graphics aperture agent, and the render agents supporting programmable graphics and media pipelines. The render agents support operating within the processor coherency domain, and support two types of GPU contexts: (a) Legacy contexts; and (b) Advanced contexts[1].

- For legacy GPU contexts, all accesses are treated as requests-without-PASID, and input addresses are translated through device specific graphics translation (e.g., through a graphics translation table). If second-level translation is enabled at the remapping hardware, these accesses (access to fetch graphics translation table entries, and access to the pages) are subject to second-level translation.
- For advanced GPU contexts, accesses can be targeted to a physical or virtual address. Accesses to physical address are treated as requests-without-PASID, and can be translated through second-level translation by the remapping hardware. Accesses to virtual address are treated as requests-with-PASID, and are translated through first-level translation (or nested translation) by the remapping hardware. When the GPU is configured to use advanced contexts, the remapping hardware must be configured to allow both requests-with-PASID and requests-without-PASID.

3.11 Handling Requests Crossing Page Boundaries

PCI-Express memory requests are specified to disallow address/length combinations which cause a memory space access to cross a page (4KB) boundary. However, the PCI-Express Specification defines checking for violations of this rule at the receivers as optional. If checked, violations are treated as malformed transaction layer packets and reported as PCI-Express errors. Checking for violations from Root-Complex integrated devices is typically platform-dependent.

Platforms supporting DMA remapping are expected to check for violations of the rule in one of the following ways:

- The platform hardware checks for violations and explicitly blocks them. For PCI-Express memory requests, this may be implemented by hardware that checks for the condition at the PCI-Express receivers and handles violations as PCI-Express errors. DMA requests from other devices (such as Root-Complex integrated devices) that violate the rule (and hence are blocked by hardware) may be handled in platform-specific ways. In this model, the remapping hardware units never receive DMA requests that cross page boundaries.
- If the platform hardware cannot check for violations, the remapping hardware units must perform these checks and re-map the requests as if they were multiple independent DMA requests.

3.12 Handling of Zero-Length Reads

A memory read request of one double-word with no bytes enabled (“zero-length read”) is typically used by devices as a type of flush request. For a requester, the semantics of the flush request allow a device to ensure that previously issued posted writes in the same traffic class have been completed at its destination.

Zero-length read requests are handled as follows by remapping hardware:

1. For details on Processor Graphics device handling of legacy and advanced GPU contexts, refer to the Programmer’s Reference Manual for Intel® Processor HD graphics.

- Implementations reporting ZLR field as Clear in the Capability Register process zero-length read requests like any other read requests. Specifically, zero-length read requests are address-translated based on the programming of the remapping structures. Zero-length reads translated to memory are completed in the coherency domain with all byte enables off. Unsuccessful translations result in translation faults. For example, zero-length read requests to write-only pages in second-level translation are blocked due to read permission violation.
- Implementations reporting ZLR field as Set in the Capability Register handles zero-length read requests same as above, except if it is to a write-only page. Zero-length read requests to write-only pages that do not encounter any faulting conditions other than read permission violation are successfully remapped and completed. Zero-length reads translated to memory complete in the coherency domain with all byte enables off. Data returned in the read completion is obfuscated.

DMA remapping hardware implementations are recommended to report ZLR field as Set and support the associated hardware behavior.

3.13 Handling Requests to Interrupt Address Range

On Intel® architecture platforms, physical address range 0xFEEx_xxxx is designated as the interrupt address range. Write requests without PASID of DWORD length to this range are interpreted by the platform as interrupt requests. For details refer to message signalled interrupts in Intel® 64 Architecture Software Developer's Manual, Volume 3B.

Hardware treats following requests to the interrupt address range as illegal requests and handles them as error:

- Read requests without PASID.
- Atomics requests without PASID.
- Non-DWORD length write requests without PASID.

Write requests without PASID of DWORD length are treated as interrupt requests. Interrupt requests are not subjected to DMA remapping (even if second-level translation structures specify a mapping for this range). Instead, remapping hardware can be enabled to subject such interrupt requests to interrupt remapping. Chapter 5 provides details on the interrupt remapping architecture.

Software must ensure the second-level paging-structure entries are programmed **not** to remap input addresses to the interrupt address range. Hardware behavior is undefined for memory requests remapped to the interrupt address range.

Requests-with-PASID with input address in range 0xFEEx_xxxx are translated normally like any other request-with-PASID through first-level translation (or nested translation).

3.14 Handling Requests to Reserved System Memory

Reserved system memory regions are typically allocated by BIOS at boot time and reported to OS as reserved address ranges in the system memory map. Requests-without-PASID to these reserved regions may either occur as a result of operations performed by the system software driver (for example in the case of DMA from unified memory access (UMA) graphics controllers to graphics reserved memory), or may be initiated by non system software (for example in case of DMA performed by a USB controller under BIOS SMM control for legacy keyboard emulation). For proper functioning of these legacy reserved memory usages, when system software enables DMA remapping, the second-level translation structures for the respective devices are expected to be set up to provide identity mapping for the specified reserved memory regions with read and write permissions.

Platform implementations supporting reserved memory must carefully consider the system software and security implications of its usages. These usages are beyond the scope of this specification. Platform hardware may use implementation-specific methods to distinguish accesses to system reserved memory. These methods must not depend on simple address-based decoding since DMA virtual addresses can indeed overlap with the host physical addresses of reserved system memory.

For platforms that cannot distinguish between device accesses to OS-visible system memory and device accesses to reserved system memory, the architecture defines a standard reporting method to inform system software about the reserved system memory address ranges and the specific devices that require device access to these ranges for proper operation. Refer to Section 8.4 for details on the reporting of reserved memory regions.

For legacy compatibility, system software is expected to setup identity mapping in second-level translation (with read and write privileges) for these reserved address ranges, for the specified devices. For these devices, the system software is also responsible for ensuring that any input addresses used for device accesses to OS-visible memory do not overlap with the reserved system memory address ranges.

3.15 Root-Complex Peer to Peer Considerations

When DMA remapping is enabled, peer-to-peer requests through the Root-Complex must be handled as follows:

- The input address in the request is translated (through first-level, second-level or nested translation) to a host physical address (HPA). The address decoding for peer addresses must be done only on the translated HPA. Hardware implementations are free to further limit peer-to-peer accesses to specific host physical address regions (or to completely disallow peer-forwarding of translated requests).
- Since address translation changes the contents (address field) of the PCI-Express Transaction Layer Packet (TLP), for PCI-Express peer-to-peer requests with ECRC, the Root-Complex hardware must use the new ECRC (re-computed with the translated address) if it decides to forward the TLP as a peer request.
- Root-ports, and multi-function root-complex integrated endpoints, may support additional peer-to-peer control features by supporting PCI-Express Access Control Services (ACS) capability. Refer to ACS capability in PCI-Express specifications for details.

This Page Is Left Intentionally Blank

4 Support For Device-TLBs

The DMA remapping architecture described in Chapter 3 supports address translation of DMA requests received by the Root-Complex. Hardware may accelerate the address-translation process by caching data from the translation structures. Chapter 6 describes details of these translation caches supported by remapping hardware. Translation caches at the remapping hardware is a finite resource that supports requests from multiple endpoint devices. As a result, efficiency of these translation caches in the platform can depend on number of simultaneously active DMA streams in the platform, and address locality of DMA accesses.

One approach to scaling translation caches is to enable endpoint devices to participate in the remapping process with translation caches implemented at the devices. These translation caches on the device is referred to as Device-TLBs (Device Translation lookaside buffers). Device-TLBs alleviate pressure for translation caches in the Root-Complex, and provide opportunities for devices to improve performance by pre-fetching address translations before issuing DMA requests. Device-TLBs can be useful for devices with strict access latency requirements (such as isochronous devices), and for devices that have large DMA working set or multiple active DMA streams. Remapping hardware units report support for Device-TLBs through the Extended Capability Register (see Section 10.4.3). Additionally, Device-TLBs may be utilized by devices to support recoverable I/O page faults. This chapter describes the basic operation of Device-TLBs. Chapter 7 covers use of Device-TLBs to support recoverable I/O page faults.

4.1 Device-TLB Operation

Device-TLB support in endpoint devices requires standardized mechanisms to:

- Request and receive translations from the Root-Complex
- Indicate if a memory request (with or without PASID) has translated or un-translated address
- Invalidate translations cached at Device-TLBs.

Figure 4-13 illustrates the basic interaction between the Device-TLB in an endpoint and remapping hardware in the Root-Complex, as defined by the PCI-Express Address Translation Services (ATS) specification.

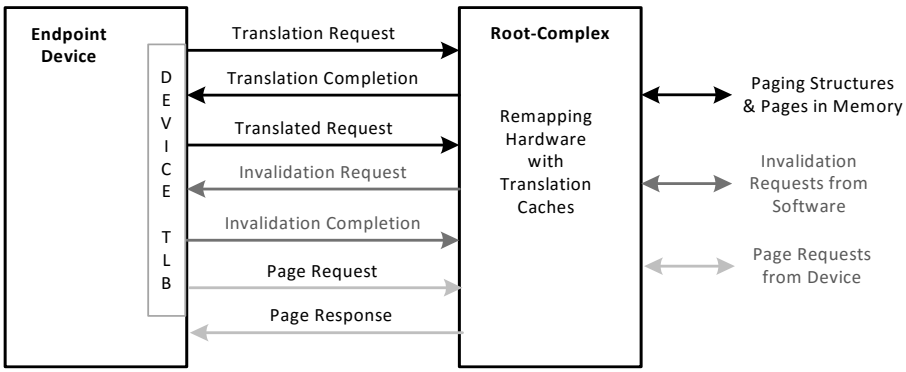


Figure 4-13. Device-TLB Operation

ATS defines the 'Address Type' (AT) field in the PCI-Express transaction header for memory requests. The AT field indicates if transaction is a memory request with 'Untranslated' address (AT=00b), 'Translation Request' (AT=01b), or memory request with 'Translated' address (AT=10b). ATS also define Device-TLB invalidation messages. Following sections describe details of these transactions.

4.1.1 Translation Request

Translation-requests-without-PASID specify the following attributes that are used by remapping hardware to process the request:

- Address Type (AT):
  - AT field has value of 01b to identify it as a translation-request.
- Address:
  - Address field indicates the starting input address for which the translation is requested.
- Length:
  - Length field indicates how many sequential translations may be returned in response to this request. Each translation is 8 bytes in length. If the length field has a value greater than two, then the additional translations (if returned in the translation response) are for sequentially increasing equal-sized pages starting at the requested input address. Refer to PCI-Express ATS specification for more details.
- No Write (NW) flag:
  - The NW flag, when Set, indicates if the endpoint is requesting read-only access for this translation.

Translation-requests-with-PASID specify the same attributes as above, and also specify the additional attributes (PASID value, Execute-Requested (ER) flag, and Privileged-mode-Requested (PR) flag) in the PASID prefix.

4.1.2 Translation Completion

If the remapping hardware was not able to successfully process the translation-request (with or without PASID), a translation-completion without data is returned.

- A status code of UR (Unsupported Request) is returned in the completion if the remapping hardware is configured to not support translation requests from this endpoint.
- A status code of CR (Completer Abort) is returned if the remapping hardware encountered errors when processing the translation-request. If the remapping hardware was able to successfully process a translation-request, a translation-completion with data is returned.

For successful translation-requests-without-PASID, each translation returned in the translation-completion data specifies the following attributes:

- Size (S):
  - Value of 0b in Size field indicates the translation is for a 4-KByte page. If Size field is 1b, the size of the translation is determined by the lowest bit in the Translated Address field (bits 63:12) with a value of 0. For example, if bit 12 is 0, the translation applies to a 8-KByte page. If bit 12 is 1 and bit 13 is 0, the translation applies to a 16-KByte page, and so on. Refer to PCI-Express ATS specification for details on translation size encoding.
- Non-Snooped access flag (N):
  - When Set, the Non-Snooped access field indicates that the translated-requests that use this translation must clear the No Snoop Attribute in the request.
- Untranslated access only flag (U):
  - When Set, the input address range for the translation can only be accessed by the endpoint using untranslated-request.

- Read permission (R):
  - If Set, read permission is granted for the input address range of this translation. If Clear, read permission is not granted for the input address range of this translation.
- Write permission (W):
  - If Set, write permission is granted for the input address range of this translation. If Clear, write permission is not granted for the input address range of this translation.
- Translated Address:
  - If either the R or W field is Set, and the U field is Clear, the translated address field contains the result of the translation for the respective input address. Endpoints can access the page through Translated-requests with this address.

For successful translation-requests-with-PASID, each translation returned in the translation-completion data specifies the same attributes as above, along with following extended attributes:

- Execute permission (EXE):
  - If EXE=R=1, execute permission is granted for the input address range of this translation. Else, execute permission is not granted for the input address range of this translation.
- Privilege Mode Access (PRIV):
  - If Set, R, W and EXE refer to permissions associated with privileged mode access. If Clear, R, W, and EXE refer to permissions associated with non-privileged access.
- Global Mapping (G):
  - If Set, the translation is common across all PASIDs at this endpoint. If Clear, the translation is specific to the PASID value specified in the PASID prefix in the associated Translation-request.

4.1.3 Translated Request

Translated-requests are regular memory read/write/atomics requests with Address Type (AT) field value of 10b. When generating requests to a given input (untranslated) address, the endpoint may lookup the local Device-TLB for cached translation (result of previous translation-requests) for the input address. If a cached translation is found with appropriate permissions and privilege, the endpoint may generate a translated-request (AT=10b) specifying the Translated address obtained from the Device-TLB lookup. Translated-requests are always without PASID, as they reference translated (host physical) address.

4.1.4 Invalidation Request & Completion

Invalidation requests are issued by software through remapping hardware to invalidate translations cached at endpoint Device-TLBs.

Invalidation-requests-without-PASID specify the following attributes:

- Device ID
  - Identity of the device (bus/device/function) whose Device-TLB is the target of invalidation.
- Size (S):
  - Value of 0b in Size field indicates the target of invalidation is a 4-KByte input address range. If Size field is 1b, the input address range to be invalidated is determined by the lowest bit in the Untranslated Address field (bits 63:12) with a value of 0. Refer to the PCI-Express ATS Specification for details on invalidation address size encoding.
- Untranslated Address
  - Specifies the base of the input (untranslated) address range to be invalidated.

The invalidation-requests-with-PASID specify the same attributes as above, along with a global-invalidate flag. If the global-invalidate flag is 1, the invalidation affects across all PASID values. If the global-invalidate flag is 0, the invalidation is required to affect only the PASID value specified in the PASID TLP prefix.

Invalidation requests and completions carry additional tags (ITags) managed by hardware to uniquely identify invalidation requests and completions. Refer to the PCI-Express ATS specification for more details on use of ITags.

4.2 Remapping Hardware Handling of Device-TLBs

Remapping hardware report support for Device-TLBs through the Extended Capability Register (see Section 10.4.3). The translation-type (T) field in the context entries and extended-context-entries can be programmed to enable or disable processing of translation-requests and translated-requests from specific endpoints by remapping hardware. The following sections describe the remapping hardware handling of ATS requests.

4.2.1 Handling of ATS Protocol Errors

The following upstream requests are always handled as Unsupported Request (UR) by hardware:

- Memory read or write request (with or without PASID) with AT field value of 'Reserved' (11b).
- Memory write request (with or without PASID) with AT field value of 'Translation Request' (01b).
- Requests-with-PASID with AT field value of 'Translated' (10b).

The following upstream requests (with or without PASID) are always handled as malformed packets:

- Memory read request with AT field value of 'Translation Request' with any of the following:
  - — Length specifying odd number of DWORDs (i.e. least significant bit of length field is non-zero)
  - — Length greater than N/4 DWORDs where N is the Read Completion Boundary (RCB) value (in bytes) supported by the Root-Complex.
  - — First and last DWORD byte enable (BE) fields not equal to 1111b.
- 'Invalidation Request' message.

When remapping hardware is disabled (TES=0 in Global Status Register), following upstream requests are treated as Unsupported Request (UR).

- Memory requests with non-zero AT field (i.e. AT field is not 'Untranslated').
- ATS 'Invalidation Completion' messages.

4.2.2 Root-Port Control of ATS Address Types

Root-ports supporting Access Control Services (ACS) capability can support 'Translation Blocking' control to block upstream memory requests with non-zero value in the AT field. When enabled, such requests are reported as ACS violation by the receiving root-port. Refer to the ACS Capability in PCI-Express Specifications for more details. Upstream requests that cause ACS violations are blocked at the root-port as error and are not presented to remapping hardware.

4.2.3 Handling of Translation Requests

This section describes the handling of translation-requests when remapping hardware is enabled.

- The requester-id in the translation-request is used to parse the respective root/extended-root and context/extended-context as described in Section 3.4.

- If hardware detects any conditions below that explicitly blocks the translation-requests from this endpoint, a translation-completion is returned with status code of Unsupported Request (UR), and treats it as a translation-fault (see Chapter 7 for translation-fault behavior).
  - — Present (P) field in the root-entry (or extended-root-entry) used to process the translation-request is 0.
  - — Present (P) field in the context-entry (or extended-context-entry) used to process the translation-request is 0.
  - — The translation-type (T) field in the present context-entry (or extended-context-entry) specifies blocking of translation requests (see Section 9.3 and Section 9.4 for description of translation-type field in context-entries and extended-context-entries).
  - — The Root Table Type (RTT) field in the Root-table Address register (RTADDR_REG) used to process the translation-request-with-PASID is 0.
- For translation-requests-without-PASID, if hardware detects any error conditions below, a translation-completion is returned with status code of Completer Abort (CA), and hardware treats it as a translation-fault (see Chapter 7 for translation-fault behavior).
  - — Hardware attempt to access the root-entry (or extended-root-entry) through the Root Table Address (RTA) field in the Root Table Address Register resulted in error.
  - — Hardware attempt to access context-entry (or extended-context-entry) through the Context Table Pointer (CTP) fields in the root-entry (or extended-root-entry) resulted in error.
  - — Hardware detected reserved fields not initialized to zero in a present root-entry (or extended-root-entry).
  - — Hardware detected reserved fields not initialized to zero in a present context-entry (or extended-context-entry).
  - — Hardware detected invalid programming of a present context-entry (or extended-context-entry). For example:
    - The Address Width (AW) field programmed with a value not supported by the hardware implementation.
    - The translation-type (T) field programmed to indicate a translation type not supported by the hardware implementation.
    - Hardware attempt to access the second-level page table base through the second-level page-table pointer (SLPTPTR) field of the context-entry (or extended-context-entry) resulted in error.
  - — Hardware attempt to access a second-level paging entry (SL-PML4E with 5-level translation, SL-PDPE, SL-PDE, or SL-PTE) through the Address (ADDR) field in another paging entry (SL-PML5E with 5-level translation, SL-PML4E, SL-PDPE or SL-PDE) resulted in error.
  - — Hardware detected reserved fields not initialized to zero in a second-level paging entry with at least one of Read (R) and Write (W) field Set.
- For translation-requests-with-PASID, if hardware detects any error conditions below, a translation-completion is returned with status code of Completer Abort (CA), and hardware treats it as a translation-fault (see Chapter 7 for translation-fault behavior).
  - — Hardware attempt to access the extended-root-entry through the Root Table Address (RTA) field in the Root Table Address Register resulted in error.
  - — Hardware attempt to access extended-context-entry through the context-table-pointer fields in the extended-root-entry resulted in error.
  - — Hardware detected reserved fields not initialized to zero in a present extended-root-entry.
  - — Hardware detected reserved fields not initialized to zero in a present extended-context-entry.
  - — Hardware detected invalid programming of a present extended-context-entry. For example:
    - When nested translation is enabled, the Address Width (AW) field programmed with a value not supported by the hardware implementation.

- Invalid or unsupported programming of any of the first-level translation or nested translation control fields in the extended-context-entry.
- The translation-type (T) field is programmed to a type not supported by the hardware implementation.
- Hardware attempt to access the PASID-entry through the PASID-table pointer (PASIDPTR) field of the extended-context-entry resulted in error.
- When nested translation is enabled, hardware attempt to access the second-level page table base through the second-level page-table pointer (SLPTPTR) field of the extended-context-entry resulted in error.

— Hardware attempt to access a FL-PML5E (with 5-level paging) or FL-PML4E (with 4-level paging) through the Address (ADDR) field in a PASID-entry resulted in error.

— Hardware attempt to access a first-level paging entry (FL-PML4E with 5-level paging, FL-PDPE, FL-PDE, or FL-PTE) referenced through Address (ADDR) field in a preceding first-level paging entry (FL-PML5E with 5-level paging, FL-PML4E, FL-PDPE or FL-PDE) resulted in error.

— Hardware detected reserved fields not initialized to zero in a present PASID-entry or present first-level paging entry.

— When nested translation is enabled, hardware attempt to access a second-level paging entry (SL-PML4E with 5-level translation, SL-PDPE, SL-PDE, or SL-PTE) referenced through the Address (ADDR) field in a preceding second-level paging entry (SL-PML5E with 5-level translation, SL-PML4E, SL-PDPE or SL-PDE) resulted in error.

— When nested translation is enabled, hardware detected reserved fields not initialized to zero in a second-level paging entry with at least one of Read (R), Execute (X), and Write (W) field Set. Execute (X) field is applicable only if supported by hardware, and enabled by software.

- If none of the error conditions above are detected, hardware handles the translation-request as below:

— If the input address in the translation-request-without-PASID is within the interrupt address range (0xFEEx_xxxx)[1], a successful translation-completion is issued with R=0, W=1, U=1, and S=0 in the Translation-Completion data. This special handling for translation-requests-without-PASID to interrupt address range is provided to comprehend potential endpoint Device-TLB behavior of issuing translation requests to all of its memory transactions including its message signalled interrupt (MSI) posted writes.

— If remapping hardware encountered any of the conditions below that resulted in either not finding translation for the address specified in the translation-request, or detecting the requested translation lacked both read and write permissions, a translation-completion with status code of Success is returned with R=W=U=S=0 in the translation-completion-data.

- When performing first-level translation for translation-request-with-PASID, hardware detected input address is not canonical (i.e., address bits 63:N have the same value as address bit [N-1], where N is 48-bits with 4-level paging and 57-bits with 5-level paging).
- When performing first-level translation for translation-request-with-PASID, hardware encountered not-present (P=0) PASID-entry.
- When performing first-level translation for translation-request-with-PASID with supervisor privilege (value of 1 in Privilege-mode-requested (PR) field), hardware encountered a present PASID-entry with SRE field value of 0, causing a privilege violation.
- When performing first-level translation for translation-request-with-PASID, hardware encountered a not-present (P=0) first-level-paging-entry along the page-walk, and hence could not complete the page-walk.

1. Translation-requests-with-PASID with input address in the range 0xFEEx_xxxxx is processed normally through the first-level (or nested) translation, like any other translation-request-with-PASID.

- When performing first-level translation for translation-request-with-PASID with user privilege (value of 0 in Privilege-mode-requested (PR) field), hardware encountered a present first-level-paging-entry with U/S field value of 0, causing a privilege violation.
- When performing second-level translation for translation-request-without-PASID, or when performing any second-level step of nested translation for translation-request-with-PASID, hardware detected input address above the minimum of MGAW (reported in Capability Register) and (2^X - 1), where X is the AGAW corresponding to address-width programmed in context-entry or extended-context-entry used to process the request.
- When performing second-level translation for translation-request-without-PASID, or when performing any second-level step of nested translation for translation-request-with-PASID, hardware found a not-present (R=W=0) second-level-paging-entry along the page-walk, and hence could not complete the page-walk.
- Hardware detected that the logical-AND of the Read (R) permission bits and logical-AND of Write (W) permission bits of second-level page-walk to be both Clear.

— If remapping hardware successfully fetched the translation requested, and the translation has at least one of Read and Write permissions, a translation-completion with status code of Success is returned with translation-completion-data as follows:

- **Read (R) bit**: The R bit in the translation-completion data is the effective read permission for this translation.
  - — For translation-requests-without-PASID with NW=0, R bit is 1 if the access rights checking described in Section 3.7.2 allow read access to the page. Else, R bit is 0.
  - — For translation-request-with-PASID, R bit is 1 if the access rights checking (described in Section 3.6.2 for first-level translation, or Section 3.8.2 for nested translation) allow read access to the page. Else, R bit is 0.
- **Write (W) bit**: The W bit in the translation-completion data is the effective write permission for this translation.
  - — For translation-requests with NW=1 (i.e., requests indicating translation is for read-only accesses), remapping hardware reporting no-write-flag support (NWFS=1 in the Extended Capability Register) returns the W bit as always 0. Remapping hardware not supporting no-write-flag (NWFS=0) ignores value of NW field in translation-requests and functions as if NW is 0 (see below).
  - — For translation-requests-without-PASID with NW=0, W bit is 1 if the access rights checking described in Section 3.7.2 allow write access to the page. Else, W bit is 0.
  - — For translation-request-with-PASID with NW=0, W bit is 1 if the access rights checking (described in Section 3.6.2 for first-level translation, or Section 3.8.2 for nested translation) allow write access to the page. Else, W bit is 0.
- **Execute (EXE) bit**: The EXE bit in the translation-completion data is the effective execute permission for this translation.
  - — For translation-requests-without-PASID, this bit is always 0.
  - — For translation-requests-with-PASID with ER=0 (i.e., requests indicating translation is not for instruction fetch), this bit is always 0.
  - — For translation-requests-with-PASID with ER=1 (i.e., requests indicating translation is for instruction fetch), remapping hardware reporting Execute-Requests as not supported (ERS=0 in the Extended Capability Register) returns the EXE bit as always 0. Remapping hardware supporting Execute-Requests (ERS=1) return EXE bit as 1 if the access rights checking (described in Section 3.6.2 for first-level translation, or Section 3.8.2 for nested translation) allow instruction fetch from the page. Else, EXE bit is 0.
- **Privilege Mode (PRIV) bit**: The PRIV bit in the translation-completion data is the effective privilege for this translation.
  - — For translation-requests-without-PASID, this bit is always 0.

  - For translation-requests-with-PASID with PR=0 (i.e., requests indicating translation is not for privileged/supervisor mode access), this bit is always 0.
  - For translation-requests-with-PASID with PR=1 (i.e., requests indicating translation is for privileged/supervisor mode access), remapping hardware reporting supervisor-request-support as not supported (SRS=0 in the Extended Capability Register) returns the PRIV bit as 1 and forces R=W=E=0. Remapping hardware supporting supervisor-requests (SRS=1) return PRIV bit as always 1 (including when SRE=0 in extended-context-entry).

- **Global Mapping (G) bit**: The G bit in the translation-completion data is the effective privilege for this translation.
  - For translation-requests-without-PASID, this bit is always 0.
  - For translation-requests-with-PASID, this bit is the G bit from the leaf first-level paging-structure entry (PTE for 4-KByte page, PDE for 2-MByte page, PDPE for 1-GByte page) used to translate the request.
- **Non-snooped access (N) bit**: The N bit in the translation-completion data indicates the use of No-Snoop (NS) flag in accesses that use this translation.
  - For translation-requests-without-PASID, remapping hardware reporting snoop-control as not supported (SC=0 in the Extended Capability Register) always return the N bit as 0. Remapping hardware supporting snoop-control (SC=1) return the SNP bit in the second-level paging-structure entry controlling the translation as the N bit (SL-PTE or second-level paging-structure entry with PS=1).
  - For translation-requests-with-PASID, this bit is always 1.
- **Untranslated access only (U) bit**: The U bit in the translation-completion data indicates the address range for the translation can only be accessed using untranslated-requests.
  - For translation-requests-without-PASID, this bit is the TM (transient-mapping) bit from the second-level paging-structure entry controlling the translation (SL-PTE or second-level paging-structure entry with PS=1).
  - For translation-requests-with-PASID that are subject to only first-level translation, this bit is always 0.
  - For translation-requests-with-PASID that are subject to nested translation, this bit is the TM (transient-mapping) bit from the second-level paging-structure entry used to translate the page (SL-PTE or second-level paging-structure entry with PS=1).
- **Size (S) bit**: The S bit in the translation-completion data indicates the page size for the translation.
  - This bit is 0 if translation returned is for 4-KByte page.
  - This bit is 1 if translation returned if for page larger than 4-KByte. In this case, the size of the translation is determined by the lowest bit in the Translated Address field (bits 63:12) with a value of 0. For example, if bit 12 is 0, the translation applies to a 8-KByte page. If bit 12 is 1 and bit 13 is 0, the translation applies to a 16-KByte page, and so on. Refer to PCI-Express ATS specification for details on translation size encoding.
- **Translated Address (ADDR)**: If either R or W bit is 1, and U bit is 0, the ADDR field in the translation-completion data contains the result of the translation.
  - For translation-requests-without-PASID, this is the translated address from the second-level translation.
  - For translation-requests-with-PASID that are subject to first-level translation only, this is the output address from the first-level translation.
  - For translation-requests-with-PASID that are subject to nested translation, this is the output address from the nested translation.

4.2.3.1 Accessed, Extended Accessed, and Dirty Flags

While processing a translation-request-with-PASID, remapping hardware manages the accessed, extended-accessed, and dirty flags in the first-level paging-structure entries as follows:

- For translation-requests-with-PASID, remapping hardware atomically sets the A field (if it is not already set) in each first-level paging-structure entry used to successfully translate the request.
- For translation-requests-with-PASID, remapping hardware atomically sets the EA field (if it is not already set) in each first-level paging-structure entry used to successfully translate the request, if the Extended-context-entry used to process the request has a value of 1 for the Extended-Accessed-Flag-Enable (EAFE) field.
- For translation-requests-with-PASID with NW=0, remapping hardware reporting No-Write-Flag Support (NWFS=1 in Extended Capability Register) atomically sets the D field (if it is not already set) in the first-level paging-structure entry that identifies the final translated address for the input address (i.e., either a PTE or a paging-structure entry in which the PS field is 1). Remapping hardware not supporting No-Write-Flag (NWFS=0) atomically sets the D field, ignoring the value of the NW field in the translation-request-with-PASID.

As described above, the accessed, extended accessed, and dirty flags are set early at the time of processing a translation-request, and before the endpoint access the page using the translation returned in the translation-completion.

Setting of Accessed, Extended Accessed, and Dirty flags in the first level paging-structure entries are subject to the access rights checking described in Section 3.6.2 (or Section 3.8.2, when nested translations are enabled).

4.2.3.2 Translation Requests for Multiple Translations

Translation-requests for multiple mappings indicate a length field greater than 2 DWORDs. Hardware implementations may handle these requests in any one of the following ways:

- Always return single translation
  - Hardware fetches translation only for the starting address specified in the translation-request, and a translation-completion is returned depending on the result of this processing. In this case, the translation-completion has a Length of 2 DWORDs, Byte Count of 8, and the Lower Address indicates a value of Read Completion Boundary (RCB) minus 8.
- Return multiple translations
  - Hardware fetches translations starting with the address specified in the translation-request, until a Completer Abort (CA) or Unsupported Request (UR) condition as described in Section 4.2.3 is detected, or until a translation with different page-size than the previous translation in this request is detected. Remapping hardware is also allowed to limit fetching of translations to those that are resident within a cache line. When returning multiple translations (which may be less than the number of translations requested), hardware must ensure that successive translations must apply to the untranslated address range that abuts the previous translation in the same completion. Refer to the PCI-Express ATS specification for requirements on translation-completions returning multiple mappings in one or two packets.

4.2.4 Handling of Translated Requests

This section describes the handling of Translated-requests when remapping hardware is enabled.

- If translated-request has a PASID prefix, it is treated as malformed request.
- If translated-request is a write or atomics request to the interrupt address range (0xFEEx_xxxx), it is treated as Unsupported Requests (UR). Also, all translated (and untranslated) requests that are reads to interrupt address range always return UR.
- If hardware detects any conditions below, the translated-request is blocked (handled as UR) and is treated as translation fault (see Chapter 7 for translation-fault behavior).

— Hardware attempt to access the root-entry (or extended-root-entry) through the Root Table Address (RTA) field in the Root Table Address Register resulted in error.

— Present (P) field in the root-entry (or extended-root-entry) used to process the translated-request is 0.

— Hardware detected reserved fields not initialized to zero in a present root-entry (or extended-root-entry).

— Hardware attempt to access context-entry (or extended-context-entry) through the Context Table Pointer fields in the root-entry (or extended-root-entry) resulted in error.

— Present (P) field in the context-entry (or extended-context-entry) used to process the translated-request is 0.

— Hardware detected reserved fields not initialized to zero in a present context-entry (or extended-context-entry).

— Hardware detected invalid programming of a present context-entry (or extended-context-entry). For example:

  • The Address Width (AW) field programmed with a value not supported by the hardware implementation.

  • The translation-type (T) field programmed to indicate a translation type not supported by the hardware implementation.

— The translation-type (T) field in the present context-entry (or extended-context-entry) specifies blocking of translated-requests (see Section 9.3 and Section 9.4 for description of translation-type field in context-entries and extended-context-entries).

- If none of the error conditions above are detected, the translated-request is processed as pass-through (i.e., bypasses address translation).

4.3 Handling of Device-TLB Invalidations

The Address Translation Services (ATS) extensions to PCI-Express defines the wire protocol for the Root-Complex to issue a Device-TLB invalidation request to an endpoint and to receive the Device-TLB invalidation completion responses from the endpoint.

For remapping hardware supporting Device-TLBs, software submits the Device-TLB invalidation requests through the invalidation queue interface of remapping hardware. Section 6.5.2 describes the queued invalidation interface details.

Hardware processes a Device-TLB invalidation request as follows:

- Hardware allocates a free invalidation tag (ITag). ITags are used to uniquely identify an invalidation request issued to an endpoint. If there are no free ITags in hardware, the Device-TLB invalidation request is deferred until a free ITag is available. For each allocated ITag, hardware stores a counter (*InvCmpCnt*) to track the number of invalidation completions received with this ITag.
- Hardware starts an invalidation completion timer for this ITag, and issues the invalidation request message to the specified endpoint. If the invalidation command from software is for a first-level mapping, the invalidation request message is generated with the appropriate PASID prefix to identify the target PASID. The invalidation completion time-out value is recommended to be sufficiently larger than the PCI-Express read completion time-outs.

Hardware processes a Device-TLB invalidation response received as follows:

- ITag-vector in the invalidation completion response indicates the ITags corresponding to completed Device-TLB invalidation requests. The completion count in the invalidation response indicates the number of invalidation completion messages expected with the same ITag-vector and completion count.
- For each ITag Set in the ITag-vector, hardware checks if it is a valid (currently allocated) ITag for the source-id in the invalidation completion response. If hardware detects an invalid ITag, the

invalidation completion message is dropped by hardware. The error condition is reported by setting the Invalidation Completion Error (ICE) field in the Fault Status Register (see Section 10.4.9), and depending on the programming of the Fault Control Register a fault event may be generated.

- If above checks are completed successfully, for each ITag in the ITag-vector, the corresponding *InvCmpCnt* counter is incremented and compared with the 'completion count' value in the invalidation response ('completion count' value of 0 indicates 8 invalidation completions). If the comparison matches, the Device-TLB invalidation request corresponding to the ITag is considered completed, and the ITag is freed.
- If the invalidation completion time-out expires for an ITag before the invalidation response is received, hardware frees the ITag and reports it through the ITE field in the Fault Status Register. Depending on the programming of the Fault Control Register a fault event may be generated. Section 6.5.2.10 describes hardware behavior on invalidation completion time-outs.

This Page Is Left Intentionally Blank

5 Interrupt Remapping and Interrupt Posting

This chapter discuss architecture and hardware details for interrupt-remapping and interrupt-posting.

5.1 Interrupt Remapping

The interrupt-remapping architecture enables system software to control and censor external interrupt requests generated by all sources including those from interrupt controllers (I/OxAPICs), MSI/MSI-X capable devices including endpoints, root-ports and Root-Complex integrated end-points.

Interrupts generated by the remapping hardware itself (Fault Event and Invalidation Completion Events) are not subject to interrupt remapping.

Interrupt requests appear to the Root-Complex as upstream memory write requests to the interrupt-address-range 0xFEEX_XXXXh. Since interrupt requests arrive at the Root-Complex as write requests, interrupt-remapping is co-located with the remapping hardware units. The interrupt-remapping capability is reported through the Extended Capability Register.

5.1.1 Identifying Origination of Interrupt Requests

To support domain-isolation usages, the platform hardware must be capable of uniquely identifying the requestor (Source-Id) for each interrupt message. The interrupt sources in a platform and use of source-id in these requests may be categorized as follows:

- Message Signaled Interrupts from PCI-Express Devices
  - For message-signaled interrupt requests from PCI-Express devices, the source-id is the requester identifier in the PCI-Express transaction header. The requester-id of a device is composed of its PCI Bus/Device/Function number assigned by configuration software and uniquely identifies the hardware function that initiated the I/O request. Section 3.4.1 illustrates the requester-id as defined by the PCI-Express specification. Section 3.9.4 describes use of source-id field by PCI-Express devices using phantom functions.
- Message Signaled Interrupts from Root-Complex Integrated Devices
  - For message-signaled interrupt requests from root-complex integrated PCI or PCI-Express devices, the source-id is its PCI requester-id.
- Message Signaled Interrupts from Devices behind PCI-Express to PCI/PCI-X Bridges
  - For message-signaled interrupt requests from devices behind PCI-Express-to-PCI/PCI-X bridges, the requester identifier in those interrupt requests may be that of the interrupting device or the requester-id with the bus number field equal to the bridge's secondary interface's bus number and device and function number fields value of zero. Section 3.9.1 describes legacy behavior of these bridges. Due to this aliasing, interrupt-remapping hardware does not isolate interrupts from individual devices behind such bridges.
- Message Signaled Interrupts from Devices behind Conventional PCI bridges
  - For message-signaled interrupt requests from devices behind conventional PCI bridges, the source-id in those interrupt requests is the requestor-id of the legacy bridge device. Section 3.9.2 describes legacy behavior of these bridges. Due to this, interrupt-remapping hardware does not isolate message-signaled interrupt requests from individual devices behind such bridges.

- Legacy pin interrupts
  - For devices that use legacy methods for interrupt routing (such as either through direct wiring to the I/OxAPIC input pins, or through INTx messages), the I/OxAPIC hardware generates the interrupt-request transaction. To identify the source of interrupt requests generated by I/OxAPICs, the interrupt-remapping hardware requires each I/OxAPIC in the platform (enumerated through the ACPI Multiple APIC Descriptor Tables (MADT)) to include a unique 16-bit source-id in its requests. BIOS reports the source-id for these I/OxAPICs via ACPI structures to system software. Refer to Section 8.3.1.1 for more details on I/OxAPIC identity reporting.
- Other Message Signaled Interrupts
  - For any other platform devices that are not PCI discoverable and yet capable of generating message-signaled interrupt requests (such as the integrated High Precision Event Timer - HPET devices), the platform must assign unique source-ids that do not conflict with any other source-ids on the platform. BIOS must report the 16-bit source-id for these via ACPI structures described in Section 8.3.1.2.

5.1.2 Interrupt Request Formats On Intel® 64 Platforms

Interrupt-remapping on Intel® 64 platforms support two interrupt request formats. These are described in the following sub-sections.

5.1.2.1 Interrupt Requests in Compatibility Format

Figure 5-14 illustrates the interrupt request in Compatibility format. The Interrupt Format field (Address bit 4) is Clear in Compatibility format requests. Refer to the Intel® 64 Architecture software developer's manuals for details on other fields in the Compatibility format interrupt requests. Platforms without interrupt-remapping capability support only Compatibility format interrupts.

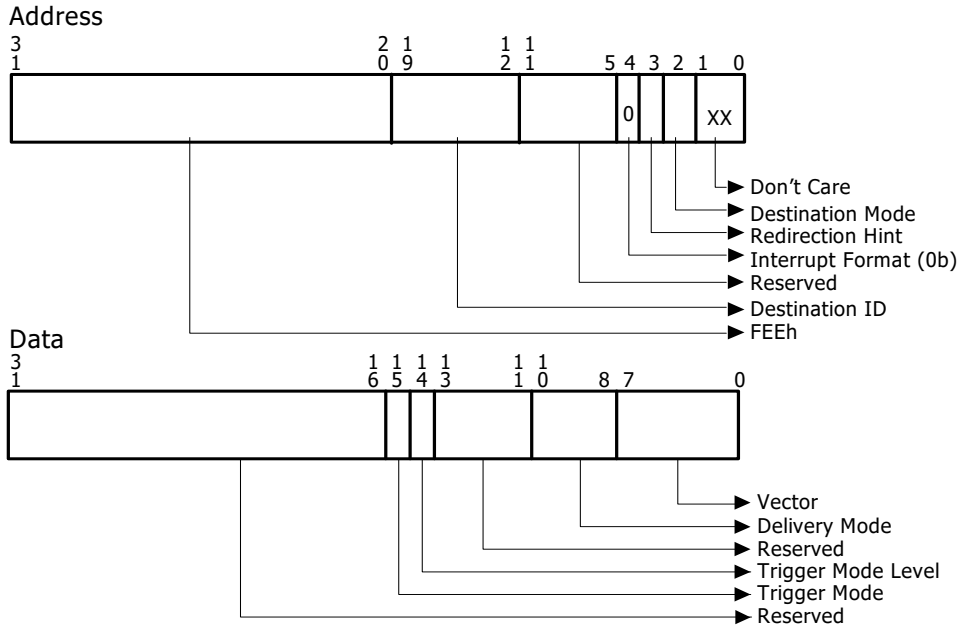


Figure 5-14. Compatibility Format Interrupt Request

5.1.2.2 Interrupt Requests in Remappable Format

Figure 5-15 illustrates the Remappable interrupt request format. The Interrupt Format field (Address bit 4) is Set for Remappable format interrupt requests. Remappable interrupt requests are applicable only on platforms with interrupt-remapping support.

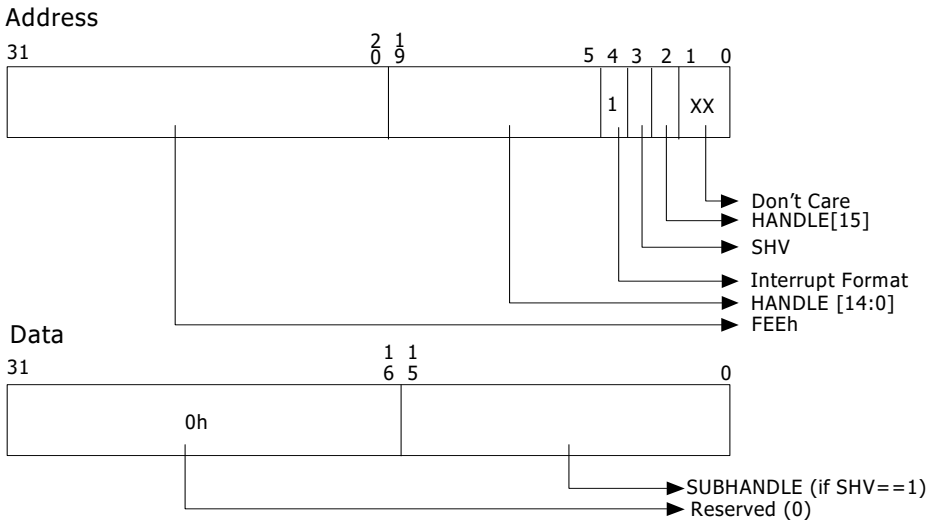


Figure 5-15. Remappable Format Interrupt Request

Table 6 describes the various address fields in the Remappable interrupt request format.

Table 6. Address Fields in Remappable Interrupt Request Format

Address Bits	Field	Description
31: 20	Interrupt Identifier	DWORD DMA write request with value of FEEh in these bits are decoded as interrupt requests by the Root-Complex.
19: 5	Handle[14:0]	This field along with bit 2 provides a 16-bit Handle. The Handle is used by interrupt-remapping hardware to identify the interrupt request. 16-bit Handle provides 64K unique interrupt requests per interrupt-remapping hardware unit.
4	Interrupt Format	This field must have a value of 1b for Remappable format interrupts.
3	SubHandle Valid (SHV)	This field specifies if the interrupt request payload (data) contains a valid Subhandle. Use of Subhandle enables MSI constructs that supports only a single address and multiple data values.
2	Handle[15]	This field carries the most significant bit of the 16-bit Handle.
1:0	Don't Care	These bits are ignored by interrupt-remapping hardware.

Table 7 describes the various data fields in the Remappable interrupt request format.

Table 7. Data Fields in Remappable Interrupt Request Format

Data Bits	Field	Description
31:16	Reserved	When SHV field in the interrupt request address is Set, this field treated as reserved (0) by hardware. When SHV field in the interrupt request address is Clear, this field is ignored by hardware.
15:0	Subhandle	When SHV field in the interrupt request address is Set, this field contains the 16-bit Subhandle. When SHV field in the interrupt request address is Clear, this field is ignored by hardware.

5.1.3 Interrupt Remapping Table

Interrupt-remapping hardware utilizes a memory-resident single-level table, called the Interrupt Remapping Table. The interrupt remapping table is expected to be setup by system software, and its base address and size is specified through the Interrupt Remap Table Address Register. Each entry in the table is 128-bits in size and is referred to as Interrupt Remapping Table Entry (IRTE).Section 9.10 illustrates the IRTE format.

For interrupt requests in Remappable format, the interrupt-remapping hardware computes the 'interrupt_index' as below. The Handle, SHV and Subhandle are respective fields from the interrupt address and data per the Remappable interrupt format.

```
if (address.SHV == 0) {
    interrupt_index = address.handle;
} else {
    interrupt_index = (address.handle + data.subhandle);
}
```

The Interrupt Remap Table Address Register is programmed by software to specify the number of IRTEs in the Interrupt Remapping Table (maximum number of IRTEs in an Interrupt Remapping Table is 64K). Remapping hardware units in the platform may be configured to share interrupt-remapping table or use independent tables. The interrupt_index is used to index the appropriate IRTE in the interrupt-remapping table. If the interrupt_index value computed is equal to or larger than the number of IRTEs in the remapping table, hardware treats the interrupt request as error.

Unlike the Compatibility interrupt format where all the interrupt attributes are encoded in the interrupt request address/data, the Remappable interrupt format specifies only the fields needed to compute the interrupt_index. The attributes of the remapped interrupt request is specified through the IRTE referenced by the interrupt_index.The interrupt-remapping architecture defines support for hardware to cache frequently used IRTEs for improved performance. For usages where software may need to dynamically update the IRTE, architecture defines commands to invalidate the IEC. Chapter 6 describes the caching constructs and associated invalidation commands.

5.1.4 Interrupt-Remapping Hardware Operation

The following provides a functional overview of the interrupt-remapping hardware operation:

- An interrupt request is identified by hardware as a DWORD sized write request to interrupt address ranges 0xFEEx_xxxx.
- When interrupt-remapping is not enabled (IRES field Clear in Global Status Register), all interrupt requests are processed per the Compatibility interrupt request format described in Section 5.1.2.1.
- When interrupt-remapping is enabled (IRES field Set in Global Status Register), interrupt requests are processed as follows:

— Interrupt requests in the Compatibility format (i.e requests with Interrupt Format field Clear) are processed as follows:

- If Extended Interrupt Mode is enabled (EIME field in Interrupt Remapping Table Address Register is Set), or if the Compatibility format interrupts are disabled (CFIS field in the Global Status Register is Clear), the Compatibility format interrupts are blocked.
- Else, Compatibility format interrupts are processed as pass-through (bypasses interrupt-remapping).

— Interrupt requests in the Remappable format (i.e. request with Interrupt Format field Set) are processed as follows:

- The reserved fields in the Remappable interrupt requests are checked to be zero. If the reserved field checking fails, the interrupt request is blocked. Else, the Source-id, Handle, SHV, and Subhandle fields are retrieved from the interrupt request.
- Hardware computes the interrupt_index per the algorithm described in Section 5.1.3. The computed interrupt_index is validated to be less than the interrupt-remapping table size configured in the Interrupt Remap Table Address Register. If the bounds check fails, the interrupt request is blocked.
- If the above bounds check succeeds, the IRTE corresponding to the interrupt_index value is either retrieved from the Interrupt Entry Cache, or fetched from the interrupt-remapping table. If the Coherent (C) field is reported as Clear in the Extended Capability Register, the IRTE fetch from memory will not snoop the processor caches. If the Present (P) field in the IRTE is Clear, the interrupt request is blocked and treated as fault.
- If IRTE is present (P=1), hardware performs verification of the interrupt requester per the programming of the SVT, SID and SQ fields in the IRTE as described in Section 9.10. If the source-id checking fails, the interrupt request is blocked.

— If IRTE has Mode field clear (IM=0):[1]

- Hardware interprets the IRTE in remappable format (as described in Section 9.10). If invalid programming of remappable-format IRTE is detected, the interrupt request is blocked.
- If above checks succeed, a remapped interrupt request is generated per the programming of the IRTE fields[2].

- Any of the above checks that result in interrupt request to be blocked is treated as a interrupt-remapping fault condition. The interrupt-remapping fault conditions are enumerated in the following section.

1. If the IM field is 1, hardware interprets the IRTE in posted format (as described in Section 9.11). Refer to Section 5.2.3 for interrupt-posting hardware operation.
2. When forwarding the remapped interrupt request to the system bus, the ‘Trigger Mode Level’ field in the interrupt request on the system bus is always set to “asserted” (1b).

5.1.4.1 Interrupt Remapping Fault Conditions

The following table enumerates the various conditions resulting in faults when processing interrupt requests. A fault conditions is treated as 'qualified' if the fault is reported to software only when the Fault Processing Disable (FPD) field is Clear in the IRTE used to process the faulting interrupt request.

Table 8. Interrupt Remapping Fault Conditions

Interrupt Remapping Fault Conditions	Fault Reason	Qualified	Behavior
Decoding of the interrupt request per the Remappable request format detected one or more reserved fields as Set.	20h	No	Unsupported Request
The interrupt_index value computed for the Remappable interrupt request is greater than the maximum allowed for the interrupt-remapping table size configured by software, or hardware attempt to access the IRTE corresponding to the interrupt_index value referenced an address above Host Address Width (HAW).	21h	No	
The Present (P) field in the IRTE entry corresponding to the interrupt_index of the interrupt request is Clear.	22h	Yes	
Hardware attempt to access the interrupt-remapping table through the Interrupt-Remapping Table Address (IRTA) field in the Interrupt Remap Table Address Register resulted in error.	23h	No	
Hardware detected one ore more reserved fields that are not initialized to zero in an IRTE with Present (P) field Set. This also includes cases where software programmed various conditional reserved fields wrongly.	24h	Yes	
On Intel® 64 platforms, hardware blocked an interrupt request in Compatibility format either due to Extended Interrupt Mode Enabled (EIME field Set in Interrupt Remapping Table Address Register) or Compatibility format interrupts disabled (CFIS field Clear in Global Status Register).	25h	No	
Hardware blocked a Remappable interrupt request due to verification failure of the interrupt requester's source-id per the programming of SID, SVT and SQ fields in the corresponding IRTE with Present (P) field Set.	26h	Yes	
Hardware attempt to access the Posted Interrupt Descriptor (PID) through the Posted Descriptor Address High/Low fields of an IRTE for posted interrupts resulted in error.[1]	27h	Yes	
Hardware detected one ore more reserved fields that are not initialized to zero in an Posted Interrupt Descriptor (PID).[1]	28h	Yes	

1. Fault Reasons 27h and 28h are applicable only for interrupt requests processed through IRTEs programmed for Interrupt Posting as described in Section 9.11. Refer to Section 5.2 for details on Interrupt Posting.

5.1.5 Programming Interrupt Sources To Generate Remappable Interrupts

Software performs the following general steps to configure an interrupt source to generate remappable interrupts:

- Allocate a free interrupt remap table entry (IRTE) and program the remapped interrupt attributes per the IRTE format described in Section 9.10.
- Program the interrupt source to generate interrupts in remappable format with appropriate handle, subhandle and SHV fields that effectively encodes the index of the allocated IRTE as the interrupt_index defined in Section 5.1.3. The interrupt_index may be encoded using the handle, subhandle and SHV fields in one of the following ways:
  - SHV = 0; handle = interrupt_index;
  - SHV = 1; handle = interrupt_index; subhandle = 0;
  - SHV = 1; handle = 0; subhandle = interrupt_index;
  - SHV = 1; handle = interrupt_index - subhandle;

The following sub-sections describes example programming for I/OxAPIC, MSI and MSI-X interrupt sources to generate interrupts per the Remappable interrupt request format.

5.1.5.1 I/OxAPIC Programming

Software programs the Redirection Table Entries (RTEs) in I/OxAPICs as illustrated in Figure 5-16.

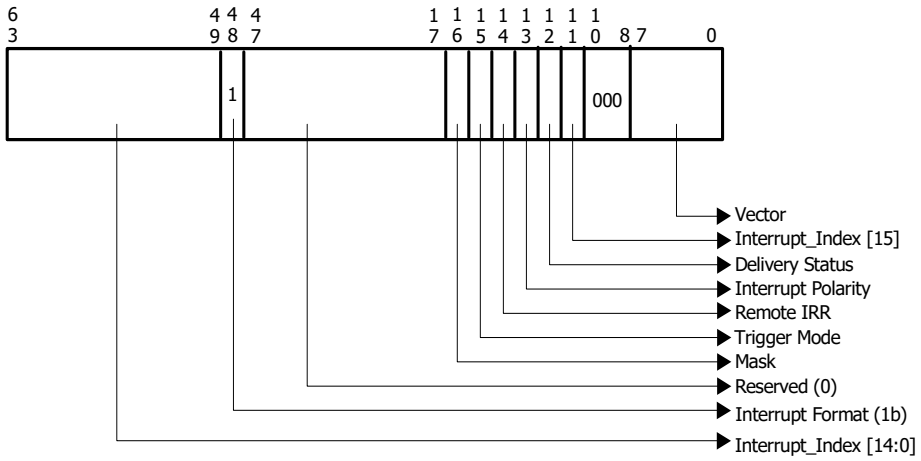


Figure 5-16. I/OxAPIC RTE Programming

- The Interrupt_Index[14:0] is programmed in bits 63:49 of the I/OxAPIC RTE. The most significant bit of the Interrupt_Index (Interrupt_Index[15]) is programmed in bit 11 of the I/OxAPIC RTE.
- Bit 48 in the I/OxAPIC RTE is Set to indicate the Interrupt is in Remappable format.
- RTE bits 10:8 is programmed to 000b (Fixed) to force the SHV (SubHandle Valid) field as Clear in the interrupt address generated.
- The Trigger Mode field (bit 15) in the I/OxAPIC RTE must match the Trigger Mode in the IRTE referenced by the I/OxAPIC RTE. This is required for proper functioning of level-triggered interrupts.
- For platforms using End-of-Interrupt (EOI) broadcasts, Vector field in the I/OxAPIC RTEs for level-triggered interrupts (i.e. Trigger Mode field in I/OxAPIC RTE is Set, and Trigger Mode field in the IRTE referenced by the I/OxAPIC RTE is Set), must match the Vector field programmed in the referenced IRTE. This is required for proper processing of End-Of-Interrupt (EOI) broadcast by the I/OxAPIC.
- Programing of all other fields in the I/OxAPIC RTE are not impacted by interrupt remapping.

5.1.5.2 MSI and MSI-X Register Programming

Figure 5-17 illustrates the programming of MSI/MSI-X address and data registers to support remapping of the message signalled interrupt.

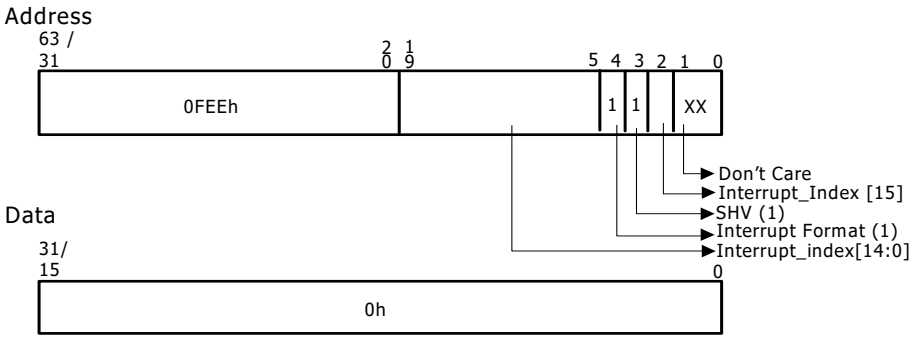


Figure 5-17. MSI-X Programming

Specifically, each address and data registers must be programmed as follows:

- Address register bits 63/31: 20 must be programmed with the interrupt address identifier value of 0FEEh.
- Address register bits 19:5 is programmed with Interrupt_Index[14:0] and address register bit 2 must be programmed with Interrupt_Index[15]. The Interrupt_Index is the index of the Interrupt Remapping Table Entry (IRTE) that remaps the corresponding interrupt requests.
  - Devices supporting MSI allows software to enable multiple vectors (up to 32) in powers of 2. For such multiple-vector MSI usages, software must allocate N contiguous IRTE entries (where N is the number of vectors enabled on the MSI device) and the interrupt_index value programmed to the Handle field must be the index of the first IRTE out of the N contiguous IRTEs allocated. The device owns the least significant log-N bits of the data register, and encodes the relative interrupt number (0 to N-1) in these bits of the interrupt request payload.
- Address register bit 4 must be Set to indicate the interrupt is in Remappable format.
- Address register bit 3 is Set so as to set the SubHandle Valid (SHV) field in the generated interrupt request.
- Data register is programmed to 0h.

5.1.6 Remapping Hardware - Interrupt Programming

Interrupts generated by the remapping hardware itself (Fault Event and Invalidation Completion Events) are not subject to interrupt remapping. The following sections describe the programming of the Fault Event and Invalidation Completion Event data/address registers on Intel® 64 platforms.

5.1.7 Programming in Intel® 64 xAPIC Mode

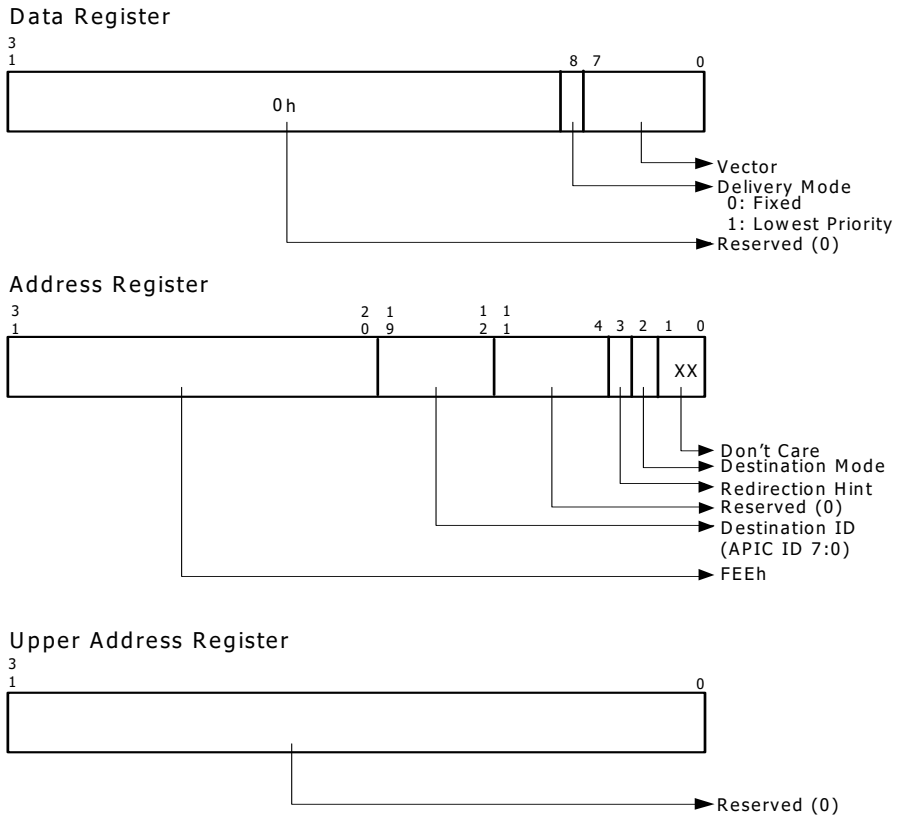


Figure 5-18. Remapping Hardware Interrupt Programming in Intel® 64 xAPIC Mode

5.1.8 Programming in Intel® 64 x2APIC Mode[1]

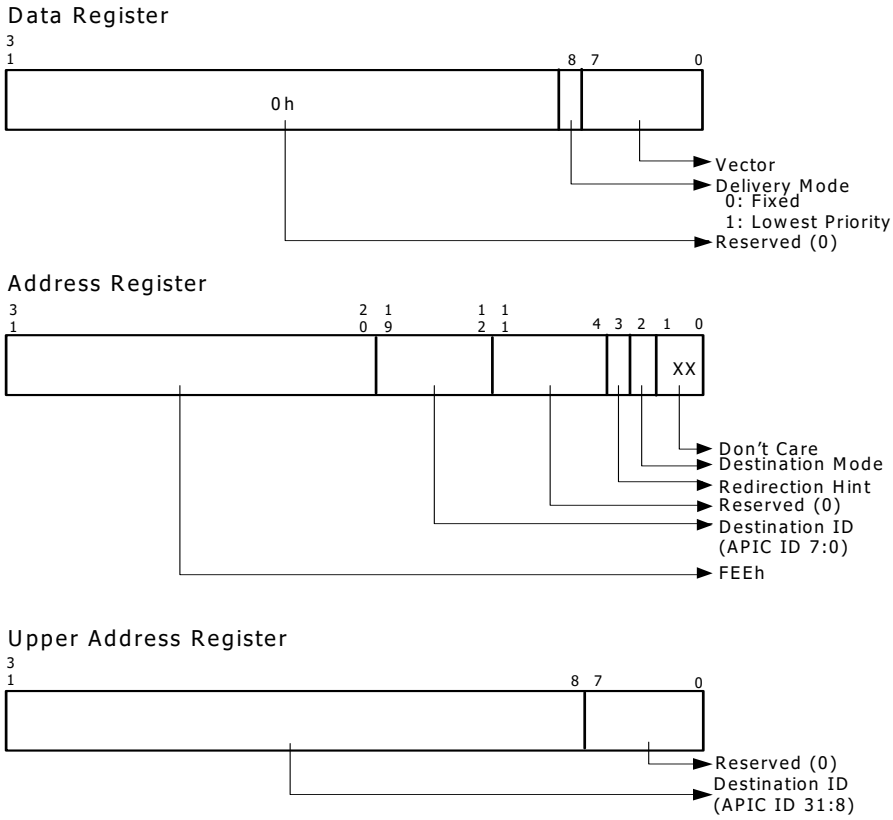


Figure 5-19. Remapping Hardware Interrupt Programming in Intel® 64 x2APIC Mode

5.1.9 Handling of Platform Events

Platforms supporting interrupt remapping are highly recommended to use side-band mechanisms (such as dedicated pins between chipset/board-logic and CPU), or in-band methods (such as platform/vendor defined messages) to deliver platforms events such as SMI/PMI/NMI/INIT/MCA. This is to avoid the dependence on system software to deliver these critical platform events.

Some existing platforms are known to use I/OxAPIC RTEs (Redirection Table Entries) to deliver SMI, PMI and NMI events. There are at least two existing initialization approaches for such platform events delivered through I/OxAPIC RTEs.

- Some existing platforms report to system software the I/OxAPIC RTEs connected to platform event sources through ACPI, enabling system software to explicitly program/enable these RTEs. Example for this include, the 'NMI Source Reporting' structure in ACPI MADT (for reporting NMI source).

1. Hardware support for x2APIC mode is reported through the EIM field in the Extended Capability Register. x2APIC mode is enabled through the Interrupt Remapping Table Address Register.

- Alternatively, some existing platforms program the I/OxAPIC RTEs connected to specific platform event sources during BIOS initialization, and depend on system software to explicitly preserve these RTEs in the BIOS initialized state. (For example, some platforms are known to program specific I/OxAPIC RTE for SMI generation through BIOS before handing control to system software, and depend on system software preserving the RTEs pre-programmed with SMI delivery mode).

On platforms supporting interrupt-remapping, delivery of SMI, PMI and NMI events through I/OxAPIC RTEs require system software programming the respective RTEs to be properly remapped through the Interrupt Remapping Table. To avoid this management burden on system software, platforms supporting interrupt remapping are highly recommended to avoid delivering platform events through I/OxAPIC RTEs, and instead deliver them through dedicated pins (such as the processor's xAPIC LINTn input) or through alternative platform-specific messages.

5.2 Interrupt Posting

Interrupt-posting capability is an extension of interrupt-remapping hardware for extended processing of remappable format interrupt requests. Interrupt-posting enables a remappable format interrupt request to be posted (recorded) in a coherent main memory resident data-structure, with an optional notification event to the CPU complex to signal pending posted interrupt.

Interrupt-posting capability (along with the support in Intel® 64 processors for posted-interrupt processing and APIC Virtualization) enables a Virtual Machine Monitor (VMM) software to efficiently process interrupts from devices assigned to virtual machines. Section 2.5.3 describes high-level usages and benefits of interrupt-posting. Refer to '*Intel® 64 Architecture Software Developer's Manual, Volume 3B: System Programming*' for details on Intel® 64 processor support for APIC virtualization and posted-interrupt processing.

Remapping hardware support for interrupt-posting capability is reported through the Posted Interrupt Support (PI) field in the Capability register (CAP_REG). Section 10.4.2 describes interrupt-posting capability reporting.

5.2.1 Interrupt Remapping Table Support for Interrupt Posting

All remappable interrupt requests are processed through the Interrupt Remapping Table as described in Section 5.1.3. The IRTE Mode (IM) field in an Interrupt Remapping Table Entry (IRTE) specifies if remappable interrupt requests processed through that IRTE is subject to interrupt-remapping or interrupt-posting.

- If the IM field is 0 in an IRTE, the IRTE is interpreted in remappable format (described in Section 9.10) to remap interrupt requests processed through it. The interrupt-remapping hardware operation is described in Section 5.1.4.
- If the IM field is 1 in an IRTE, the IRTE is interpreted in posted format (described in Section 9.11) to post interrupt requests processed through it. The interrupt-posting hardware operation is described in Section 5.2.3.

IRTE entries in posted format support following new fields:

- Address of the Posted Interrupt Descriptor data structure to post (record) the interrupt to. Section 5.2.2 describes the Posted Interrupt Descriptor.
- Urgent (URG) qualification to indicate if interrupt requests processed through this IRTE require real-time processing or not. Section 5.2.3 describes the hardware operation with this field.
- Vector field specifies the vector to use when posting interrupts processed through an IRTE. Unlike remappable-format (where the Vector field is used when generating the remapped interrupt request), the Vector field for posted-format IRTEs is used to determine which bit to Set when posting the interrupt to the Posted Interrupt Descriptor referenced by the IRTE.

As with interrupt remapping, interrupts generated by the remapping hardware itself are not subject to interrupt posting.

5.2.2 Posted Interrupt Descriptor

Posted Interrupt Descriptor is a 64-byte aligned and sized structure in memory used by interrupt-posting hardware to post (record) interrupt requests subject to posting. Section 9.12 describes the Posted Interrupt Descriptor Format. System software must allocate the Posted Interrupt Descriptors in coherent (write-back) main memory.

The Posted Interrupt Descriptor hosts the following fields:

- Posted Interrupt Request (PIR) field provides storage for posting (recording) interrupts (one bit per vector, for up to 256 vectors).
- Outstanding Notification (ON) field indicates if there is a notification event outstanding (not processed by processor or software) for this Posted Interrupt Descriptor. When this field is 0, hardware modifies it from 0 to 1 when generating a notification event, and the entity receiving the notification event (processor or software) resets it as part of posted interrupt processing.
- Suppress Notification (SN) field indicates if a notification event is to be suppressed (not generated) for non-urgent interrupt requests (interrupts processed through an IRTE with URG=0).
- Notification Vector (NV) field specifies the vector for notification event (interrupt).
- Notification Destination (NDST) field specifies the physical APIC-ID of the destination logical processor for the notification event.

5.2.3 Interrupt-Posting Hardware Operation

Interrupt requests in remappable format are processed by hardware as described in Section 5.1.4. When such processing encounters a IRTE entry in posted format (IM=1), the interrupt request is processed through posting (instead of remapping). The following provides a functional overview of the interrupt-posting hardware operation:

- If IRTE retrieved has Mode field as set (IM=1)[1]
  - — Hardware interprets the IRTE in posted format (as described in Section 9.11). If invalid programming of posted-format IRTE is detected, the interrupt request is blocked.
  - — If above checks succeed, the IRTE provides the pointer to the Posted Interrupt Descriptor (PDA-L/PDA-H), the vector value (Vector) to be posted, and if the interrupt request is qualified as urgent (URG) or not.
- Hardware performs a coherent atomic read-modify-write operation of the posted-interrupt descriptor as follows:
  - — Read contents of the Posted Interrupt Descriptor, claiming exclusive ownership of its hosting cache-line. If invalid programming (e.g., non-zero reserved fields) of Posted Interrupt Descriptor is detected, release ownership of the cache-line, and block the interrupt request.
  - — If above checks succeed, retrieve current values of Posted Interrupt Requests (PIR bits 255:0), Outstanding Notification (ON), Suppress Notification (SN), Notification Vector (NV), and Notification Destination (NDST) fields in the Posted Interrupt Descriptor.
  - — Modify the following descriptor field values atomically:
    - Set bit in PIR corresponding to the Vector field value from the IRTE
    - Compute X = ((ON == 0) & (URG | (SN == 0)))
    - If (X == 1), Set ON field.

1. If the IM field is 0, hardware interprets the IRTE in remapped format (described in Section 9.10). Refer to Section 5.1.4 for interrupt-remapping hardware operation.

— Promote the cache-line to be globally observable, so that the modifications are visible to other caching agents. Hardware may write-back the cache-line anytime after this step.

— If (X == 1) in previous step, generate a notification event (interrupt) with attributes as follows:

- NSDT field specifies the physical APIC-ID of destination logical CPU. Refer to Section 9.12 on how this field is interpreted for xAPIC and x2APIC modes.
- NV field specifies the vector to be used for the notification interrupt to signal the destination CPU about pending posted interrupt.
- Delivery mode field for notification interrupt is forced to Fixed (000b)
- Re-direction Hint field for notification interrupt is forced to Clear (0b)
- Trigger Mode field for notification interrupt is forced to Edge (0b)
- Trigger Mode Level field for notification interrupt is forced to Asserted (1b).

- Any of the above checks that result in interrupt request to be blocked is treated as a interrupt-remapping fault condition as enumerated in Section 5.1.4.1.

5.2.4 Ordering Requirements for Interrupt Posting

This section summarizes the ordering requirements to be met by interrupt-posting hardware when posting interrupts.

- Interrupt requests are posted transactions and follow PCI-Express posted ordering rules. This ensures that an interrupt request will not be observed by software until all prior inbound posted requests (writes) are committed to their destinations.
  — This requirement needs to be maintained even if the interrupt requests are posted. i.e., before an interrupt is posted (recorded) in the posted-interrupt descriptor and made visible to software, all preceding posted requests must be completed.
- Since interrupt requests are posted transactions, upstream read completions must push preceding interrupt requests.
  — This requirement needs to be maintained even if one or more of the preceding interrupt requests are posted. i.e., An upstream read completion must wait until all preceding interrupts (irrespective of if they are remapped or posted) are completed. In case of an interrupt that is posted, 'completion' of the interrupt means, both the atomic update of the posted interrupt descriptor and the associated notification event are completed.
- In the interrupt-posting operation, hardware must make sure that modifications to a posted-interrupt descriptor is observable to software before issuing the notification event for that descriptor.

5.2.5 Using Interrupt Posting for Virtual Interrupt Delivery

This section is informative and intended to illustrate a simplified example[1] usage of how a Virtual Machine Monitor (VMM) software may use interrupt-posting hardware to support efficient delivery of virtual interrupts from assigned devices to virtual machines.

VMM software may enable interrupt-posting for a virtual machine as follows:

- For each virtual processor in the virtual machine, the VMM software may allocate a Posted Interrupt Descriptor. Each such descriptor is used for posting all interrupts that are to be delivered to the respective virtual processor.
- The VMM software allocates two physical interrupt vectors (across all logical CPUs in the platform) for notification events.

1. This simplified usage example assumes the VMM software typically runs with interrupts masked, except perhaps when placing the logical CPUs in low power states. The example illustrated here may be extended to cover other usage scenarios.

  - — One of this physical vectors may be used as the 'Active Notification Vector' (ANV) for posted interrupt notifications to any virtual processor that is active (executing) at the time of posting an interrupt to it.
  - — The other physical vector allocated may be used as the 'Wake-up Notification Vector' (WNV) for posted interrupt notifications to any virtual processor that is blocked (halted) at the time of posting an interrupt to it.
- For each interrupt source from any assigned device(s) to this virtual machine, the VMM software may intercept and virtualize the guest software programming of respective interrupt resources (IOxAPIC entries and/or MSI/MSI-X registers). Through this virtualization, the VMM software detects the target virtual processor and virtual vector assigned by guest software.
- For each such interrupt source, the VMM software allocates a posted-format IRTE.
  - — The vector field in each such IRTE is programmed by the VMM software with the respective virtual vector value assigned for the interrupt source by guest software.
  - — The posted descriptor address field in each such IRTE is programmed by the VMM software to reference the posted descriptor allocated for the virtual processor assigned by guest software for the interrupt source.
  - — The urgent (URG) field in an IRTE is Set by the VMM software if the respective interrupt source is designated as requiring immediate (non-deferred) processing.
- The VMM software configures the processor hardware to enable APIC virtualization (including 'virtual-interrupt delivery' and 'process posted interrupts' capabilities) for the virtual processors.
  - — The 'posted-interrupt notification vector' for the virtual processors are configured with the 'Active Notification Vector' (ANV) value described earlier in this section.
  - — The 'posted-interrupt descriptor' for the virtual processors are configured with the address of the Posted Interrupt Descriptor allocated for respective virtual processors.
- The VMM software scheduler may manage a virtual processor's scheduling state as follows:
  - — When a virtual processor is selected for execution, the virtual processor state is designated as 'active' before entering/resuming it. This state is specified in its Posted Interrupt Descriptor by programming its Notification Vector (NV) field with the ANV vector value[1]. This allows all interrupts for this virtual processor that are received while it is active (running) are processed by the processor hardware without transferring control to the VMM software. The processor hardware processes these notification events (with ANV vector value) by transferring any posted interrupts in the Posted Interrupt Descriptor to the Virtual-APIC page of the virtual processor and directly delivering it (without VMM software intervention) to the virtual processor. Refer to '*Intel® 64 Architecture Software Developer's Manual, Volume 3: System Programming Guide*' for details on Intel®64 processor support for APIC Virtualization and Posted-Interrupt Processing.
  - — When a virtual processor is preempted (e.g., on quantum expiry), the virtual processor state is designated as 'ready-to-run'. This state is specified in its Posted Interrupt Descriptor by programming the Suppress Notification (SN) field to 1. This allows all non-urgent interrupts for this virtual processor received while it is in preempted state to be posted to its Posted Interrupts Descriptor without generating a notification interrupt (thereby avoiding disruption of currently running virtual processors). If there are interrupt sources qualified as urgent for targeting this virtual processor, the VMM software may also modify the NV field in the Posted Interrupt Descriptor to WNV vector value. This enables the VMM software to receive notifications (with WNV vector value) when urgent interrupts are posted when virtual processor is not running, allowing appropriate software actions (such as preempting the current running virtual processor and immediately scheduling this virtual processor).

1. There may be varying approaches for VMM software to manage notification vectors. For example, an alternate approach may be for VMM software to allocate unique Activation Notification Vectors (ANV) for each virtual processor (as opposed to sharing the same ANV for all virtual processors). This approach may enable such VMM software to avoid switching between active and wake-up vector values in the Posted Interrupt Descriptor on virtual processor scheduling state changes, and instead update them only on virtual processor migrations across logical processors.

  - When a virtual processor halts (e.g., on execution of HLT instruction), the VMM software may get control, blocks further execution of the virtual processor, and designate the virtual processor state as 'halted'. This state is specified in its Posted Interrupt Descriptor by programming its Notification Vector (NV) field with the WNV vector value. This enables the VMM software to receive notifications (with WNV vector value) when any interrupt (urgent or non-urgent) is posted for this virtual processor, allowing appropriate software action (such as to schedule the virtual processor for future or immediate activation).

- When entering/resuming a virtual processor, the VMM software may process any pending posted interrupts in its posted descriptor as follows:
  - VMM first transitions the virtual CPU to 'active' state by programming the notification vector in the Posted Interrupt Descriptor to ANV vector value.
  - VMM may check if there are pending interrupts in the posted descriptor (e.g. by scanning PIR field for non-zero value).
  - If there are pending posted interrupts, VMM may generate a self-IPI[1] (Inter Processor Interrupt to the same logical CPU) with vector value of ANV, through the Local xAPIC. This interrupt is recognized by the processor as soon as interrupts are enabled in the virtual processor enter/resume path. Since the virtual processor is configured with ANV vector value as the 'posted-interrupt notification vector', this results in processor hardware processing it same as any notification event it may receive while the virtual processor is active. This approach enables the VMM software to 'off-load' the posted interrupt processing (such as delivering the interrupt to the virtual processor through the Virtual-APIC) to the processor hardware, irrespective of the scheduling state of the virtual processor when the interrupt was posted by remapping hardware to the Posted Interrupt Descriptor.
- The VMM software may also apply the 'posted-interrupt processing' capability of the processor to inject virtual interrupts generated by VMM software to a virtual machine (in addition to interrupts from direct assigned devices to the virtual machine). This may be done by the VMM software atomically 'posting' a virtual interrupt to the Posted Interrupt Descriptor (using atomic/LOCK instructions that enforces cache-line update atomicity) and generating a notification event (as IPI) to the logical processor identified as notify destination in the Posted interrupt Descriptor.
- The VMM software may handle virtual processor migrations across logical processors by atomically updating the Notification Destination (NDST) field in the respective Posted Interrupt Descriptor to the physical APIC-ID of the logical processor to which the virtual processor is migrated to. This enables all new notification events from the posted descriptor of this virtual processor to be routed to the new logical processor.

5.2.6 Interrupt Posting for Level Triggered Interrupts

Level-triggered interrupts generated through IOxAPICs Redirection Table Entries (illustrated in Figure 5-16) can be processed through Interrupt Remap Table Entries (IRTE) for Posted Interrupts (illustrated in Section 9.11). However, unlike with interrupt-remapping, all interrupts (including level interrupts) processed by the posted interrupt processing hardware are treated as edge-triggered interrupts. Thus VMM software enabling posting of Level-triggered interrupts must take special care to properly virtualize the End of Interrupt (EOI) processing by the virtual processor. For example, the VMM software may set up the virtual processor execution controls to gain control on EOI operation to the Virtual APIC controller by guest software, and virtualize the operation by performing a Directed-EOI to the IOxAPIC that generated the level-triggered interrupt. A Directed-EOI is performed by software writing directly to the IOxAPIC EOI register. Refer to the IOxAPIC specification for details on IOxAPIC EOI register.

1. The usage illustrated in this section assumes the VMM software is executing with interrupts disables, and interrupts are enabled by the processor hardware as part of entering/resuming the virtual processor. For VMM software implementations that have interrupt enabled in the VMM, precaution must be taken by the VMM software to disable interrupt on the logical processor before generating the self-IPI and resuming the virtual processor.

This Page Is Left Intentionally Blank

6 Caching Translation Information

Remapping hardware may accelerate the address-translation process by caching data from the memory-resident paging structures. Because the hardware does not ensure that the data that it caches are always consistent with the structures in memory, it is important for software to comprehend how and when the hardware may cache such data, what actions can be taken to remove cached data that may be inconsistent, and when it should do so.

6.1 Caching Mode

The Caching Mode (CM) field in Capability Register indicates if the hardware implementation caches not-present or erroneous translation-structure entries. When the CM field is reported as Set, any software updates to any remapping structures (including updates to not-present entries or present entries whose programming resulted in translation faults) requires explicit invalidation of the caches.

Hardware implementations of this architecture must support operation corresponding to CM=0. Operation corresponding to CM=1 may be supported by software implementations (emulation) of this architecture for efficient virtualization of remapping hardware. Software managing remapping hardware should be written to handle both caching modes.

Software implementations virtualizing the remapping architecture (such as a VMM emulating remapping hardware to an operating system running within a guest partition) may report CM=1 to efficiently virtualize the hardware. Software virtualization typically requires the guest remapping structures to be shadowed in the host. Reporting the Caching Mode as Set for the virtual hardware requires the guest software to explicitly issue invalidation operations on the virtual hardware for any/all updates to the guest remapping structures. The virtualizing software may trap these guest invalidation operations to keep the shadow translation structures consistent to guest translation structure modifications, without resorting to other less efficient techniques (such as write-protecting the guest translation structures through the processor's paging facility).

6.2 Address Translation Caches

This section provides architectural behavior of following remapping hardware address translation caches:

- Context-cache
  - — Caches context-entry (or extended-context-entry) encountered on a page-walk.
- PASID-cache
  - — Caches PASID entries used for processing requests-with-PASID.
- I/O Translation Look-aside Buffer (IOTLB)
  - — Caches the effective translation for a request. This is the result of the second-level page-walk for requests-without-PASID, and result of first-level page-walk (or nested page-walk, if nesting is enabled) for requests-with-PASID.
- Paging-structure Caches
  - — Caches the intermediate paging-structure entries (i.e., entries referencing a paging-structure entry) encountered on a first-level or second-level page-walk.

6.2.1 Tagging of Cached Translations

Remapping architecture supports tagging of various translation caches as follows:

- Source-ID tagging:
  - Context-cache, IOTLB, and interrupt-entry-cache are architecturally tagged by the Source-id of the request that resulted in allocation of the respective cache entry.
- Domain-ID tagging:
  - Context-entries (and extended-context-entries) allow software to specify a domain-identifier that can be used by hardware to allocate or look-up cached information when processing DMA requests targeting a domain. Paging-structure caches and PASID-cache are architecturally tagged by domain-id. Context-cache and IOTLB can also be tagged by domain-ID to facilitate batch invalidation of cached entries associated with a given domain.
- Request-type tagging:
  - Since IOTLB caches effective translations for both requests-without-PASID and requests-with-PASID (possibly from the same endpoint device), hardware may either tag IOTLB entries with a bit to indicate if a translation is for request-without-PASID or for request-with-PASID, or manage them in separate dedicated caches. Similar treatment may be applied to paging-structure caches to differentiate first-level and second-level cached entries.
- Address tagging:
  - IOTLB entries are tagged by the upper bits of the input-address (called the page number) in the request that resulted in allocation of the respective cache entry. For first-level translation, if the translation does not use a PDE (because the PS flag is 1 in the PDPE used), the page size is 1-GBytes and the page number comprise of bits N:30 (where N=56 for 5-level paging, and N=47 for 4-level paging) of the input-address; If the translation does use a PDE but does not use a PTE (because the PS flag is 1 in the PDE used), the page size is 2-MBytes and the page number comprise of bits N:21 of the input-address; If the translation does use a PTE, the page size is 4-KBytes and the page number comprise of bits N:12 of the input-address. For second-level translation, similar address-bit tagging applies, based on the last second-level paging-structure entry used to form the translation (MGAW:30 for 1-GByte page, MGAW:21 for 2-MByte page, and MGAW:12 for 4-KByte page).
  - Paging-structure caches are tagged by the respective bits of the input-address. For first-level translation, input-address bits 56:48 are used to tag cached PML5E with 5-level paging, input-address bits N:39 are used to tag cached PML4E, bits N:30 are used to tag cached PDPE, and bits N:21 are used to tag cached PDE (where N=56 with 5-level paging, and N=47 with 4-level paging). Similarly, for second-level translation, input-address bits MGAW:47 are used to tag cached SL-PML5E, input-address bits MGAW:39 are used to tag cached SL-PML4E, bits MGAW:30 are used to tag cached SL-PDPE, and bits MGAW:21 are used to tag cached SL-PDE.
- PASID tagging:
  - IOTLB, paging-structure cache, and PASID-cache are architecturally tagged by the PASID value in the request-with-PASID that resulted in allocation of the respective cache entry.
- Interrupt-index tagging:
  - Interrupt-remapping cache is architecturally tagged by the interrupt-index of remappable-format interrupt requests that resulted in allocation of the interrupt-entry-cache entry.

Tagging of cached translations enable remapping hardware to cache information to process requests from multiple endpoint devices targeting multiple address-spaces. Tagging also enable software to efficiently invalidate groups of cached translations that are associated with the same tag value.

6.2.2 Context-cache

Context-cache is used to cache context-entries (or extended-context entries) used to address translate requests. Each entry in the context-cache is an individual context-entry (or extended-context-entry). Each cached entry is referenced by the source-id in the request. Each context-cache entry architecturally contains the following information:

- Context-cache entries hosting context-entries (see Section 9.3):
  - The physical address of the second-level translation structure (SLPTPTR) from the context-entry.
  - Attributes from the context-entry:
    - The Translation Type
    - The Fault Processing Disable flag.
    - The Domain-ID
    - The Address Width of the domain
- Context-cache entries hosting extended-context-entries (see Section 9.4):
  - The physical address of the second-level translation structure (SLPTPTR) from the extended-context-entry.
  - Attributes from the extended-context-entry:
    - The Translation Type
    - The Fault Processing Disable flag.
    - The Domain-ID
    - The Address Width of the domain
  - Additional attributes from the extended-context-entry:
    - The physical address of the PASID-table (PASIDPTR) from extended-context-entry
    - The physical address of the PASID-state-table (PASIDSTPTR) from extended-context-entry
    - The size of the PASID-table/PASID-state-table (PTS) from the extended-context-entry
    - The PASID Enable flag
    - The Nesting Enable flag
    - The Page Request Enable flag
    - The Deferred Invalidate Enable flag
    - The request-with-PASID control flags (ERE, SLEE)
    - The first-level translation control flags (PGE, NXE, WPE, SMEP)
    - The memory-type attributes and flags (CD, PAT, EMTE, EMT)

When nesting-enable is 1 in an extended-context-entry, instead of caching the PASIDPTR and PASIDSTPTR fields from the extended-context-entry, hardware may cache the physical address from the second-level translation of the guest-physical address in the PASIDPTR and PASIDSTPTR fields.

For implementations reporting Caching Mode (CM) as 0 in the Capability Register, if any of the following fault conditions are encountered as part of accessing a context-entry (or extended-context-entry), the resulting entry is not cached in the context-cache (and hence do not require software to invalidate the context-cache on modifications to such entries).

- Hardware attempt to access a root-entry (extended-root-entry), or context-entry (extended-context-entry) resulted in error.
- Present (P) field of the root-entry (extended-root-entry) is 0.
- Invalid programming of one or more fields in the present root-entry (extended-root-entry).

- Present (P) field of the context-entry (extended-context-entry) is 0.
- Invalid programming of one or more fields in present context-entry (extended-context-entry).
- One or more non-zero reserved fields in the present root-entry (extended-root-entry), or context-entry (extended-context-entry).
- When nesting is enabled, second-level translation of the PASIDPTR, or second-level translation of the PASIDSTPTR (if Deferred Invalidate Enable is 1), in PASID-entry, resulted in a translation fault.

For implementations reporting Caching Mode (CM) as 1 in the Capability Register, above conditions may cause caching of the entry that resulted in the fault, and require explicit invalidation by software to invalidate such cached entries. When CM=1, if the fault was detected without a present context-entry (extended-context-entry), the reserved domain-id value of 0 is used to tag the cached entry that caused the fault.

6.2.2.1 Context-Entry Programming Considerations

Software must ensure that, if multiple context-entries (or extended-context-entries) are programmed with the same Domain-id (DID), such entries must be programmed with same value for the second-level page-table pointer (SLPTPTR) field, and same value for the PASID Table Pointer (PASIDTPTR) field. This is required since hardware implementations tag the various translation caches with DID (see Chapter 6). Context-entries (or extended-context-entries) with the same value in these table address pointer fields are recommended to use the same DID value for best hardware efficiency.

When modifying root-entries (extended-root-entries) or context-entries (extended-context-entries):

- When modifying fields in present (P=1) root/extended-root/context/extended-context entries, software must ensure that at any point of time during the modification (performed through single or multiple write operations), the before and after state of the entry being modified is individually self-consistent. For example, software performing such updates must factor in the guaranteed write atomicity of processor hardware (8-Byte aligned writes for Intel® 64 processors). This is required as remapping hardware may be fetching these entries at any point of time while they are being modified by software. Software modifying these present (P=1) entries are also responsible to ensure these does not impact in-flight transactions from the affected endpoint devices[1].
- Software must serially invalidate the context-cache, PASID-cache (if applicable), and the IOTLB when modifying present root-entries (extended-root-entries) or context-entries (extended-context-entries). The serialization is required since hardware may utilize information from the context-caches (e.g., Domain-ID) to tag new entries inserted to the PASID-cache and IOTLB for processing in-flight requests. Section 6.5 describe the invalidation operations.

Software must not use domain-id value of 0 on when programming context-entries (or extended-context-entries) on implementations reporting CM=1 in the Capability register.

6.2.3 PASID-cache

PASID-cache is used to cache PASID-entries that are used to translate requests-with-PASID. Each entry in a PASID-cache is an individual PASID-entry. Each cached entry is referenced by the PASID number in the request and the Domain-ID from the extended-context-entry used to process the request. Each PASID-cache entry architecturally contains the following information (see Section 9.5):

- The physical address of the first-level translation structure (FLPTPTR) from the PASID-entry.
- The values of PCD and PWT flags of the PASID-entry.
- The value of Supervisor Request Enable (SRE) field of the PASID-entry.

1. Example usages for modifying present context/extended-context entries may include modifying the translation-type (TT) field to transition between pass-through and non-pass-through modes, or modifying the PASIDE/PRE/NESTE fields. Refer to Section 9.4 for details on these fields.

When nesting-enable is 1 in the extended-context-entry used to process a request-with-PASID, instead of caching the FLPTPTR from the PASID-entry, hardware may cache the physical address from the second-level translation of the guest-physical address in the FLPTPTR field.

For implementations reporting Caching Mode (CM) as 0 in the Capability Register, if any of the following translation fault conditions are encountered leading up to or as part of accessing a PASID-entry, the entry is not cached in the PASID-cache.

- Translation fault conditions described in Section 6.2.2.
- Present (P) field of the PASID-entry is 0.
- Invalid programming of one or more fields in the present PASID-entry.
- One or more non-zero reserved fields in the present PASID-entry.
- When nesting is enabled, second-level translation of the FLPTPTR in PASID-entry resulted in a translation fault.

For implementations reporting Caching Mode (CM) as 1 in the Capability Register, above conditions may cause caching of the PASID-entry that resulted in the fault, and require explicit invalidation by software to invalidate such cached entries. The caching of such faulted translations in PASID-entry cache follows same tagging as if there was no faults (i.e., PASID value from the request-with-PASID, domain-id from the context/extended-context entry that led to the PASID-entry, etc.).

Since information from the present PASID-entries (e.g., PML5 table pointer with 5-level paging or PML4 table pointer with 4-level paging) can be utilized to fill other caches (e.g., IOTLB, Paging-structure caches), to ensure updates to PASID-entries are visible to hardware, software must invalidate the PASID-cache followed by invalidation of IOTLB and paging-structure caches, in that order. Section 6.5 describe the invalidation operations.

6.2.4 IOTLB

Remapping hardware caches information about the translation of input-addresses in the IOTLB. IOTLB may cache information with different functionality as below:

- First-level mappings:
  - Each of these is a mapping from a input page number in a request-with-PASID to the physical page frame to which it translates (derived from first-level translation), along with information about access privileges and memory typing (if applicable).
- Second-level mappings:
  - Each of these is a mapping from a input page number in a request-without-PASID to the physical page frame to which it translates (derived from second-level translation), along with information about access privileges and memory typing (if applicable).
- Combined mappings:
  - Each of these is a mapping from a input page number in a request-with-PASID to the physical page frame to which it translates (derived from both first-level and second-level translation), along with information about access privileges and memory typing (if applicable).

Each entry in a IOTLB is an individual translation. Each translation is referenced by a page number. Each entry architecturally contains the following information:

- IOTLB entries hosting first-level mappings:
  - The physical address corresponding to the page number (the page frame).
  - The access rights from the first-level paging-structure entries used to translate input-addresses with the page number (see Section 3.6.2)
    - The logical-AND of the R/W flags.
    - The logical-AND of the U/S flags.
    - The logical-OR of the XD flags (necessary only if NXE=1 in extended-context-entry).

  - Attributes from a first-level paging-structure entry that identifies the final page frame for the page number (either a PTE or a first-level paging-structure entry with PS=1):
    - The dirty flag (see Section 3.6.3).
    - The memory type (see Section 3.6.5).
- IOTLB entries hosting second-level mappings:
  - The physical address corresponding to the page number (the page frame).
  - The access rights from the second-level paging-structure entries used to translate input-addresses with the page number (see Section 3.7.2)
    - The logical-AND of the R flags.
    - The logical-AND of the W flags.
  - Attributes from a second-level paging-structure entry that identifies the final page frame for the page number (either a SL-PTE or a second-level paging-structure entry with PS=1):
    - The memory type (see Section 3.7.4 and Section 3.8.4).
    - The snoop (SNP) bit (see Section 3.7.3 and Section 4.2.3).
    - The Transient-Mapping (TM) bit (see Section 4.2.3).
- IOTLB entries hosting nested mappings:
  - The physical address corresponding to the page number (the page frame).
  - The combined access rights from the first-level paging-structure and second-level paging-structure entries used to translate input-addresses with the page number (see Section 3.8.2)
    - The logical-AND of the R/W flags (from first-level translation) and W flags (from second-level translation of the result of first-level translation).
    - The logical-AND of the U/S flags (from first-level translation).
    - The logical-OR of the XD flags from first-level translation (necessary only if NXE=1 in extended-context-entry) with the logical-NAND of the X flags from second-level translation (necessary only if SLEE=1 in extended-context-entry).
  - Attributes from a first-level paging-structure entry that identifies the final page frame for the page number (either a PTE or a paging-structure entry with PS=1):
    - The dirty flag (see Section 3.6.3).
  - Combined attributes from first-level and second-level paging-structure entries that identifies the final page frame for the page number (either a page-table-entry or a paging-structure entry with PS=1):
    - The memory type (see Section 3.8.4).
    - The Transient-Mapping (TM) bit from second-level paging-entry that identifies the final page frame for the page number (see Section 4.2.3).

IOTLB entries may contain other information as well. A remapping hardware may implement multiple IOTLBs, and some of these may be for special purposes, e.g., only for instruction fetches. Such special-purpose IOTLBs may not contain some of this information if it is not necessary. For example, a IOTLB used only for instruction fetches need not contain information about the R/W and dirty flags.)

As noted in Section 6.2.1, any IOTLB entries created by hardware are associated with appropriate tags (e.g., source-id of request that allocated the entry, PASID value if request had a PASID, domain-id from the context/extended-context entry that led to the translation, etc.).

Remapping hardware need not implement any IOTLBs. Remapping hardware that do implement IOTLBs may evict or invalidate any IOTLB entry at any time. Software should not rely on the existence of IOTLBs or on the retention of IOTLB entries.

6.2.4.1 Details of IOTLB Use

For implementations reporting Caching Mode (CM) as 0 in the Capability Register, IOTLB caches only valid mappings (i.e. results of successful page-walks that did not result in a translation fault). Specifically, if any of the translation fault conditions described in Section 3.6.1 (for first-level translation), Section 3.7.1 (for second-level translation), Section 3.8.1 (for nested translation), and Section 4.2.3 (for Device-TLB translation requests) are encountered, the results are not cached in the IOTLB.

For implementations reporting Caching Mode (CM) as Set in the Capability Register, these translation fault conditions may cause caching of the faulted translation in the IOTLB. The caching of such faulted translations in IOTLB follows same tagging as if there was no faults (i.e., source-id of request that allocated the entry, PASID value if request had a PASID, domain-id from the context/extended-context entry that led to the translation, etc.).

With first-level translation, hardware does not cache a translation for a page number unless the accessed flag is 1 (and extended-accessed flag is 1, if the Extended-context-entry used specifies EAFE=1) in each of the first-level paging-structure entries used during translation; before caching a translation, the hardware sets any of these accessed and extended-accessed flags that is not already 1; with nested translation, setting of accessed and extended-accessed flags are subject to write permission checks at second-level translation.

If the page number of a input-address corresponds to a IOTLB entry tagged with the right source-id (and PASID for requests-with-PASID), the hardware may use that IOTLB entry to determine the page frame, access rights, and other attributes for accesses to that input-address. In this case, the hardware may not actually consult the paging structures in memory. The hardware may retain a IOTLB entry unmodified even if software subsequently modifies the relevant paging-structure entries in memory. See Section 6.5 for how software can ensure that the hardware uses the modified paging-structure entries.

If the paging structures specify a translation using a page larger than 4-KBytes, some hardware implementations may choose to cache multiple smaller-page IOTLB entries for that translation. Each such IOTLB entry would be associated with a page number corresponding to the smaller page size (e.g., bits N:12 of a input-address with first-level translation, where N is 56 bits with 5-level paging or 47 bits with 4-level paging), even though part of that page number (e.g., bits 20:12) is part of the offset with respect to the page specified by the paging structures. The upper bits of the physical address in such a IOTLB entry are derived from the physical address in the PDE used to create the translation, while the lower bits come from the input-address of the access for which the translation is created.

There is no way for software to be aware that multiple translations for smaller pages have been used for a large page. If software modifies the paging structures so that the page size used for a 4-KByte range of input-addresses changes, the IOTLBs may subsequently contain multiple translations for the address range (one for each page size). A reference to a input- address in the address range may use any of these translations. Which translation is used may vary from one execution to another, and the choice may be implementation-specific.

6.2.4.2 Global Pages

The first-level translation allow for global pages. If the G flag (bit 8) is 1 in a first-level paging-structure entry that maps a page (either a PTE or a first-level paging-structure entry in which the PS flag is 1), any IOTLB entry cached for a input-address using that paging-structure entry is considered to be global. Because the G flag is used only in first-level paging-structure entries that map a page, and because information from such entries are not cached in the paging-structure caches, the global-page feature does not affect the behavior of the paging-structure caches.

Hardware may use a global IOTLB entry to translate input-address in a request-with-PASID, even if the IOTLB entry is associated with a PASID different from the PASID value in the request, as long as IOTLB entry is associated with the right source-id.

6.2.5 Caches for Paging Structures

Remapping hardware may cache frequently used paging-structure entries that reference other paging-structure entries (as opposed to page frames). Depending on the type of the paging-structure entry cached, the paging-structure caches may be classified as PML5-cache, PML4-cache, PDPE-cache, and PDE-cache. These may cache information with different functionality as below:

- First-level-paging-structure entries:
  - Each of these is a mapping from the upper portion of a input-address in a request-with-PASID to the physical address of the first-level paging structure used to translate the corresponding region of the input-address space, along with information about access privileges. For example: With 5-level paging, bits 56:48 of the input-address would map to the address of the relevant first-level PML4 table; With 4-level paging, bits 47:39 of the input-address would map to the address of the relevant first-level page-directory-pointer table.
- Second-level-paging-structure entries:
  - Each of these is a mapping from the upper portion of a guest-physical address to the physical address of the second-level paging structure used to translate the corresponding region of the guest-physical address space, along with information about access privileges. The guest-physical address can be the input-address in a request-without-PASID, or can be the second-level address of a first-level paging-structure entry or PASID-table entry (accessed as part of a nested translation). For example, bits MGAW:39 of the input-address would map to the address of the relevant second-level page-directory-pointer table.
- Combined-paging-structure entries:
  - Each of these is a mapping from the upper portion of a input-address in a request-with-PASID to the physical address of the first-level paging structure (after nesting through second-level translation) used to translate the corresponding region of the input-address space, along with information about access privileges.

Hardware implementations may implement none or any of these paging-structure-caches, and may use separate caches in implementation specific ways to manage different types of cached mappings (e.g., first-level and nested mappings may be held in one cache and second-level in a different cache, or any other formations).

6.2.5.1 PML5-cache

When 5-level paging is effective, each entry in a PML5-cache holds the following information:

- PML5-cache entries hosting first-level PML5Es:
  - Each PML5-cache entry caching a first-level PML5E is referenced by a 9-bit value and is used for input-addresses for which bits 56:48 have that value.
  - The entry contains information from the PML5E used to translated such input-addresses:
    - The physical address from the PML5E (address of first-level PML4 table).
    - The value of R/W flag of the PML5E.
    - The value of U/S flag of the PML5E.
    - The value of XD flag of the PML5E (necessary only if NXE=1 in extended-context-entry).
    - The values of PCD and PWT flags of the PML5E.
- PML5-cache entries hosting SL-PML5Es:
  - Each PML5-cache entry caching a second-level mapping is referenced by a N-bit value and is used for input-addresses for which bits MGAW:48 have that value.
  - The entry contains information from the SL-PML5E used to translate such input-addresses:
    - The physical address from the SL-PML5E (address of second-level PML4 table).
    - The value of R flag of the SL-PML5E.

    - The value of W flag of the SL-PML5E.
    - The value of X flag of the SL-PML5E (necessary only if SLEE=1 in extended-context-entry).
- PML5-cache entries hosting nested PML5Es:
  — Each PML5-cache entry caching a nested mapping is referenced by a 9-bit value and is used for input-addresses for which bits 56:48 have that value.
  — The entry contains information from the first-level PML5E used to translate such input-addresses, combined with information from the nested second-level translation of the physical address from that PML5E:
    - The physical address from the second-level translation of the address in the PML5E (physical-address of first-level PML4 table).
    - The logical-AND of the R/W flag from the PML5E with the W flags from second-level translation of the address in PML5E.
    - The value of U/S flag of the PML5E.
    - The logical-OR of the XD flag of the PML5E (necessary only if NXE=1 in extended-context-entry) with the logical-NAND of the X flags from second-level translation of the address in PML5E (necessary only if SLEE=1 in extended-context-entry).

The following describes how a hardware implementation may use the PML5-cache:

- If the hardware has a PML5-cache entry for a input-address, it may use that entry when translating the input-address (instead of the PML5E in memory).
- For first-level mappings, hardware does not create a PML5-cache entry unless the P flag is 1 and all reserved bits are 0 in the PML5E in memory. For nested mappings, hardware also does not create a PML5-cache entry unless there is a second-level translation with read permission for the address in PML5E. For second-level mappings, hardware does not create a PML5E-cache entry unless at least one of R and W flags is 1 and all reserved bits are 0 in the SL-PML5E in memory[1].
- For first-level mappings, before creating a PML5-cache entry, hardware sets the accessed (A) flag to 1 in the PML5E in memory, if it is not already 1. Hardware also sets the extended-accessed (EA) flag to 1, if EAFE=1 in the relevant Extended-context-entry. With nested translation, setting of accessed and extended-accessed flags are subject to write permission checks at second-level translation.
- The hardware may create a PML5-cache entry even if there are no translations for any input-address that might use that entry (e.g., because the P flags are 0 in all entries in the referenced PML4 table).

If the hardware creates a PML5-cache entry, the hardware may retain it unmodified even if software subsequently modifies the corresponding PML5E (SL-PML5E) in memory.

6.2.5.2 PML4-cache

Each entry in a PML4-cache holds the following information:

- PML4-cache entries hosting first-level PML4Es:
  — Each PML4-cache entry caching a first-level PML4E is referenced by a 9-bit (or 18-bit with 5-level paging) value and is used for input-addresses for which bits N:39 (N=56 with 5-level paging and N=47 with 4-level paging) have that value.
  — The entry contains information from the PML5E and PML4E used to translated such input-addresses:
    - The physical address from the PML4E (address of first-level page-directory-pointer table).

1. This behavior applies for implementations reporting Caching Mode (CM) as 0 in the Capability register. See Section 6.1 for caching behavior on implementations reporting CM=1.

- The logical-AND of the R/W flags in the PML5E and PML4E (with 5-level paging), or the value of R/W flag of the PML4E (with 4-level paging).
- The logical-AND of the U/S flags in the PML5E and PML4E (with 5-level paging), or the value of U/S flag of the PML4E (with 4-level paging).
- The logical-OR of the XD flags in the PML5E and PML4E (with 5-level paging), or the value of XD flag of the PML4E (with 4-level paging); This is necessary only if NXE=1 in extended-context-entry.
- The values of PCD and PWT flags of the PML4E.

- PML4-cache entries hosting SL-PML4Es:
  - Each PML4-cache entry caching a second-level mapping is referenced by a N-bit value and is used for input-addresses for which bits MGAW:39 have that value.
  - The entry contains information from the SL-PML5E and SL-PML4E used to translate such input-addresses:
    - The physical address from the SL-PML4E (address of second-level page-directory-pointer table).
    - The logical-AND of the R flags in the SL-PML5E and SL-PML4E (with 5-level translation), or the value of R flag of the SL-PML4E (with 4-level translation).
    - The logical-AND of the W flags in the SL-PML5E and SL-PML4E with 5-level translation), or the value of W flag of the SL-PML4E (with 4-level translation).
    - The logical-NAND of the X flag in the SL-PML5E and SL-PML4E (with 5-level translation), or the value of X flag of the SL-PML4E (with 4-level translation); This is necessary only if SLEE=1 in extended-context-entry).
- PML4-cache entries hosting nested PML4Es:
  - Each PML4-cache entry caching a nested mapping is referenced by a 9-bit (or 18-bit with 5-level paging) value and is used for input-addresses for which bits N:39 (where N=56 with 5-level paging and N=47 with 4-level paging) have that value.
  - The entry contains information from the first-level PML5E and PML4E used to translate such input-addresses, combined with information from the nested second-level translation of the physical address from that PML4E:
    - The physical address from the second-level translation of the address in the PML4E (physical-address of first-level page-directory-pointer table).
    - With 5-level paging: The logical-AND of the R/W flags in the PML5E and PML4E, with the W flags from second-level translation of the address in the PML4E; With 4-level paging: The logical-AND of the R/W flag from the PML4E with the W flags from second-level translation of the address in PML4E.
    - With 5-level paging: The logical-AND of the U/S flags in the PML5E and PML4E; With 4-level paging: The value of U/S flag of the PML4E.
    - With 5-level paging: The logical-OR of the XD flags in the PML5E and PML4E (necessary only if NXE=1 in extended-context-entry) with the logical-NAND of the X flags from second-level translation of the address in PML4E (necessary only if SLEE=1 in extended-context-entry); With 4-level paging: The logical-OR of the XD flag of the PML4E (necessary only if NXE=1 in extended-context-entry) with the logical-NAND of the X flags from second-level translation of the address in PML4E (necessary only if SLEE=1 in extended-context-entry).

The following items detail how a hardware implementation may use the PML4-cache:

- If the hardware has a PML4-cache entry for a input-address, it may use that entry when translating the input-address (instead of the PML5E and PML4E in memory).
- For first-level mappings, hardware does not create a PML4-cache entry unless the P flag is 1 and all reserved bits are 0 in the PML5E and PML4E in memory. For nested mappings, hardware also does not create a PML4-cache entry unless there is a second-level translation with read

permission for the address in PML5E and PML4E. For second-level mappings, hardware does not create a PML4E-cache entry unless at least one of R and W flags is 1 and all reserved bits are 0 in the SL-PML5E and SL-PML4E in memory[1].

- For first-level mappings, before creating a PML4-cache entry, hardware sets the accessed (A) flag to 1 in the relevant PML5E and PML4E in memory, if it is not already 1. Hardware also sets the extended-accessed (EA) flag to 1 in these entries, if EAFE=1 in the relevant Extended-context-entry. With nested translation, setting of accessed and extended-accessed flags are subject to write permission checks at second-level translation[1].
- The hardware may create a PML4-cache entry even if there are no translations for any input-address that might use that entry (e.g., because the P flags are 0 in all entries in the referenced page-directory-pointer table).
- If the hardware creates a PML4-cache entry, the hardware may retain it unmodified even if software subsequently modifies the corresponding PML5E (SL-PML5E) or PML4E (SL-PML4E) in memory.

6.2.5.3 PDPE-cache

Each entry in a PDPE-cache holds the following information:

- PDPE-cache entries hosting first-level PDPEs:
  - Each PDPE-cache entry caching a first-level PDPE is referenced by an 18-bit (27-bit with 5-level paging) value and is used for input-addresses for which bits N:30 (where N=56 with 5-level paging and N=47 with 4-level paging) have that value.
  - The entry contains information from the PML5E (with 5-level paging), PML4E and PDPE used to translate such input-addresses:
    - The physical address from the PDPE (address of first-level page-directory). (No PDPE-cache entry is created for a PDPE that maps a page.)
    - The logical-AND of the R/W flags in the PML5E (with 5-level paging), PML4E and PDPE.
    - The logical-AND of the U/S flags in the PML5E (with 5-level paging), PML4E and PDPE.
    - The logical-OR of the XD flags in the PML5E (with 5-level paging), PML4E and PDPE (necessary only if NXE=1 in extended-context-entry).
    - The values of PCD and PWT flags of the PDPE.
- PDPE-cache entries hosting SL-PDPEs:
  - Each PDPE-cache entry caching a SL-PDPE is referenced by a N-bit value and is used for input-addresses for which bits MGAW:30 have that value.
  - The entry contains information from the SL-PML4E and SL-PDPE used to translated such input-addresses:
    - The physical address from the SL-PDPE (address of second-level page-directory). (No PDPE-cache entry is created for a SL-PDPE that maps a page.)
    - The logical-AND of the R flags in the SL-PML5E (with 5-level translation), SL-PML4E and SL-PDPE.
    - The logical-AND of the W flags in the SL-PML5E (with 5-level translation), SL-PML4E and SL-PDPE.
    - The logical-NAND of the X flag in the SL-PML5E (with 5-level translation), SL-PML4E and SL-PDPE (necessary only if SLEE=1 in extended-context-entry).
- PDPE-cache entries hosting nested PDPEs:

1. This behavior applies for implementations reporting Caching Mode (CM) as 0 in the Capability register. See Section 6.1 for caching behavior on implementations reporting CM=1.

— Each PDPE-cache entry caching a nested mapping is referenced by a 18-bit (27-bit with 5-level paging) value and is used for input-addresses for which bits N:30 (where N=56 with 5-level paging and N=47 with 4-level paging) have that value.

— The entry contains information from the PML5E (with 5-level paging), PML4E and PDPE used to translated such input-addresses, combined with information from the nested second-level translation of the physical address from that PDPE:

- The physical address from the second-level translation of the address in the PDPE (physical-address of first-level page-directory). (No PDPE-cache entry is created for a PDPE that maps a page.)
- The logical-AND of the R/W flags in the PML5E (with 5-level paging), PML4E and PDPE, with the W flags from second-level translation of the address in the PDPE.
- The logical-AND of the U/S flags in the PML5E (with 5-level paging), PML4E and PDPE.
- The logical-OR of the XD flags in the PML5E (with 5-level paging), PML4E and PDPE (necessary only if NXE=1 in extended-context-entry) with the logical-NAND of the X flags from second-level translation of the address in PDPE (necessary only if SLEE=1 in extended-context-entry).

The following items detail how a hardware implementation may use the PDPE-cache:

- If the hardware has a PDPE-cache entry for a input-address, it may use that entry when translating the input-address (instead of the PML5E, PML4E and PDPE in memory).
- For first-level mappings, hardware does not create a PDPE-cache entry unless the P flag is 1 and all reserved bits are 0 in the PML5E, PML4E and the PDPE in memory. For nested mappings, hardware also does not create a PDPE-cache entry unless there is a second-level translation with read permission for the address in the PML5E, PML4E and the PDPE. For second-level mappings, hardware does not create a PDPE-cache entry unless at least one of R and W flags is 1 and all reserved bits are 0 in the SL-PML5E, SL-PML4E and the SL-PDPE in memory[1].
- For first-level mappings, before creating a PDPE-cache entry, hardware sets the accessed (A) flag to 1 in the relevant PML5E, PML4E and PDPE in memory, if it is not already 1. Hardware also sets the extended-accessed (EA) flag to 1 in these entries, if EAFE=1 in the relevant Extended-context-entry. With nested translation, setting of accessed and extended-accessed flags are subject to write permission checks at second-level translation.
- The hardware may create a PDPE-cache entry even if there are no translations for any input-address that might use that entry (e.g., because the P flags are 0 in all entries in the referenced page-directory)
- If the hardware creates a PDPE-cache entry, the hardware may retain it unmodified even if software subsequently modifies the corresponding PML5E (SL-PML5E), PML4E (SL-PML4E) or PDPE (SL-PDPE) in memory.

6.2.5.4 PDE-cache

Each entry in a PDE-cache holds the following information:

- PDE-cache entries hosting first-level PDEs:

  — Each PDE-cache entry caching a first-level PDE is referenced by an 27-bit (36-bit with 5-level paging) value and is used for input-addresses for which bits N:21 (where N=56 with 5-level paging and N=47 with 4-level paging) have that value.

  — The entry contains information from the PML5E (with 5-level paging), PML4E, PDPE and PDE used to translate such input-addresses:

  - The physical address from the PDE (address of first-level page-table). (No PDE-cache entry is created for a PDE that maps a page.)

1. This behavior applies for implementations reporting Caching Mode (CM) as 0 in the Capability register. See Section 6.1 for caching behavior on implementations reporting CM=1.

- The logical-AND of the R/W flags in the PML5E (with 5-level paging), PML4E, PDPE and PDE.
- The logical-AND of the U/S flags in the PML5E (with 5-level paging), PML4E, PDPE and PDE.
- The logical-OR of the XD flags in the PML5E (with 5-level paging), PML4E, PDPE and PDE (necessary only if NXE=1 in extended-context-entry).
- The values of PCD and PWT flags of the PDE.

- PDE-cache entries hosting SL-PDEs:
  - — Each PDE-cache entry caching a SL-PDE is referenced by a N-bit value and is used for input-addresses for which bits MGAW:21 have that value.
  - — The entry contains information from the SL-PML5E (with 5-level translation), SL-PML4E, SL-PDPE and SL-PDE used to translated such input-addresses:
    - The physical address from the SL-PDE (address of second-level page-table). (No PDE-cache entry is created for a SL-PDE that maps a page.)
    - The logical-AND of the R flags in the SL-PML5E (with 5-level translation), SL-PML4E, SL-PDPE and SL-PDE.
    - The logical-AND of the W flags in the SL-PML5E (with 5-level translation), SL-PML4E, SL-PDPE and SL-PDE.
    - The logical-NAND of the X flag in the SL-PML5E (with 5-level translation), SL-PML4E, SL-PDPE and SL-PDE (necessary only if SLEE=1 in extended-context-entry).
- PDE-cache entries hosting nested PDEs:
  - — Each PDE-cache entry caching a nested mapping is referenced by a 27-bit (36-bit with 5-level paging) value and is used for input-addresses for which bits N:21 (where N=56 with 5-level paging and N=47 with 4-level paging) have that value.
  - — The entry contains information from the PML5E (with 5-level paging), PML4E, PDPE and PDE used to translated such input-addresses, combined with information from the nested second-level translation of the physical address from that PDE:
    - The physical address from the second-level translation of the address in the PDE (physical-address of first-level page-table). (No PDE-cache entry is created for a PDE that maps a page.)
    - The logical-AND of the R/W flags in the PML5E (with 5-level paging), PML4E, PDPE and PDE, with the W flags (from second-level translation of the address in the PDE).
    - The logical-AND of the U/S flags in the PML5E (with 5-level paging), PML4E, PDPE and PDE.
    - The logical-OR of the XD flags in the PML5E (with 5-level paging), PML4E, PDPE and PDE (necessary only if NXE=1 in extended-context-entry) with the logical-NAND of the X flags from second-level translation of the address in PDE (necessary only if SLEE=1 in extended-context-entry).

The following items detail how a hardware implementation may use the PDE-cache:

- If the hardware has a PDE-cache entry for a input-address, it may use that entry when translating the input-address (instead of the PML5E, PML4E, the PDPE, and the PDE in memory).
- For first-level mappings, hardware does not create a PDE-cache entry unless the P flag is 1 and all reserved bits are 0 in the PML5E, PML4E, the PDPTE, and the PDE in memory. For nested mappings, hardware also does not create a PDE-cache entry unless there is a second-level translation with read permission for the address in the PML5E, PML4E, the PDPE, and the PDE. For second-level mappings, hardware does not create a PDE-cache entry unless at least one of R and W flags is 1 and all reserved bits are 0 in the SL-PML5E, SL-PML4E, the SL-PDPE, and the SL-PDE in memory[1].

- For first-level mappings, before creating a PDE-cache entry, hardware sets the accessed (A) flag to 1 in the relevant PML5E, PML4E, PDPE and PDE in memory, if it is not already 1. Hardware also sets the extended-accessed (EA) flag to 1 in these entries, if EAFE=1 in the relevant Extended-context-entry. With nested translation, setting of accessed and extended-accessed flags are subject to write permission checks at second-level translation.
- The hardware may create a PDE-cache entry even if there are no translations for any input-address that might use that entry (e.g., because the P flags are 0 in all entries in the referenced page-table).
- If the hardware creates a PDE-cache entry, the hardware may retain it unmodified even if software subsequently modifies the corresponding PML5E (SL-PML5E), PML4E (SL-PML4E), PDPE (SL-PDPE), or PDE (SL-PDE) in memory.

6.2.5.5 Details of Paging-Structure Cache Use

For implementations reporting Caching Mode (CM) as Clear in the Capability Register, paging-structure-caches host only valid mappings (i.e. results of successful page-walks up to the cached paging-structure entry that did not result in a translation fault). Specifically, if any of the translation fault conditions described in Section 3.6.1 (for first-level translation), Section 3.7.1 (for second-level translation), Section 3.8.1 (for nested translation), and Section 4.2.3 (for Device-TLB translation requests) are encountered, the results are not cached in the paging-structure caches.

For implementations reporting Caching Mode (CM) as Set in the Capability Register, these translation fault conditions may cause caching of the faulted translation in the paging-structure caches. The caching of such faulted translations in paging-structure caches follows same tagging as if there was no faults (i.e., PASID value if request had a PASID, domain-id from the context/extended-context entry that led to the translation, etc.).

Information from a paging-structure entry can be included in entries in the paging-structure-caches for other paging-structure entries referenced by the original entry. For example, with 4-level paging, if the R/W flag is 0 in a PML4E, then the R/W flag will be 0 in any PDPTE-cache entry for a PDPTE from the page-directory-pointer table referenced by that PML4E. This is because the R/W flag of each such PDPTE-cache entry is the logical-AND of the R/W flags in the appropriate PML4E and PDPTE.

The paging-structure caches contain information only from paging-structure entries that reference other paging structures (and not those that map pages). For first-level-paging-structure cache entries, because the G flag is not used in such paging-structure entries, the global-page feature does not affect the behavior of the paging-structure caches.

As noted in Section 6.2.1, any entries created in paging-structure caches are associated with the target domain-ID (and PASID for entries hosting first-level and nested entries).

A remapping hardware implementation may or may not implement any of the paging-structure caches. Software should rely on neither their presence nor their absence. The hardware may invalidate entries in these caches at any time. Because the hardware may create the cache entries at the time of translation and not update them following subsequent modifications to the paging structures in memory, software should take care to invalidate the cache entries appropriately when causing such modifications. The invalidation of IOTLBs and the paging-structure caches is described in Section 6.5.

6.2.6 Using the Paging-Structure Caches to Translate Requests

When a input-address is accessed, the remapping hardware uses a procedure such as the following to determine the physical address to which it translates and whether the access should be allowed:

- For requests-with-PASID:

1. This behavior applies for implementations reporting Caching Mode (CM) as 0 in the Capability register. See Section 6.1 for caching behavior on implementations reporting CM=1.

— If the hardware finds an IOTLB entry that is for the page number of the input-address and that is associated with both the Source-ID in the request and the PASID value in the request (or which is global), it may use the physical address, access rights, and other attributes from that entry.

— Hardware may use the source-ID of the request to select a context-cache entry. It can use that entry to qualify the request based on the attributes in the entry, and obtain the Domain-ID specified by software. If the hardware does not find a matching context-cache entry, it can traverse the extended-root-table and extended-context-table to obtain and cache the extended-context-entry. Hardware may use the PASID value in the request and the Domain-ID from the context-cache entry to select a PASID-cache entry. If the hardware does not find a matching PASID-cache entry, it can traverse the PASID-table to obtain the PASID-entry.

— If the hardware does not find a relevant IOTLB entry, it may use the bits N:21 (where N=56 with 5-level paging and N=47 with 4-level paging) of the input-address to select an entry from the PDE-cache that is associated with the PASID in the request and the Domain-ID from the context-cache entry. It can then use that entry to complete the translation process (locating a PTE, etc., as described in Section 3.6 for first-level translation, or Section 3.8 for nested translation) as if it had traversed the PML5E (with 5-level paging), PML4E, PDPE and PDE corresponding to the PDE-cache entry.

— If the hardware does not find a relevant IOTLB entry or a relevant PDE-cache entry, it may use bits N:30 (where N=56 with 5-level paging and N=47 with 4-level paging) of the input-address to select an entry from the PDPE cache that is associated with the PASID in the request and the Domain-ID from the context-cache entry. It can then use that entry to complete the translation process (locating a PDE, etc., as described in Section 3.6 for first-level translation, or Section 3.8 for nested translation) as if it had traversed the PML5E (with 5-level paging), the PML4E, and the PDPE corresponding to the PDPE-cache entry.

— If the hardware does not find a relevant IOTLB entry, a relevant PDE-cache entry, or a relevant PDPE-cache entry, it may use bits N:39 (where N=56 with 5-level paging and N=47 with 4-level paging) of the input-address to select an entry from the PML4E-cache that is associated with the PASID in the request and the Domain-ID from the context-cache entry. It can then use that entry to complete the translation process (locating a PDPE, etc., as described in Section 3.6 for first-level translation, or Section 3.8 for nested translation) as if it had traversed the corresponding PML5E (with 5-level paging) and PML4E.

— With 5-level paging, if the hardware does not find a relevant IOTLB entry, a relevant PDE-cache entry, a relevant PDPE-cache entry, or a relevant PML4E-cache entry, it may use bits 56:48 of the input-address to select an entry from the PML5E-cache that is associated with the PASID in the request and the Domain-ID from the context-cache entry. It can then use that entry to complete the translation process (locating a PML4E, PDPE, etc., as described in Section 3.6 for first-level translation, or Section 3.8 for nested translation) as if it had traversed the corresponding PML5E.

— If the hardware does not find an IOTLB or paging-structure-cache entry for the input-address, it uses the PASID in the request and the Domain-ID from the context-cache entry to select a PASID-cache entry. It can use that entry to complete the translation process (locating a PML5E (with 5-level paging), PML4E, PDPE etc., as described in Section 3.6 for first-level translation, or Section 3.8 for nested translation) as if it has traversed the corresponding PASID-entry.

(Any of the above steps would be skipped if the hardware does not support the cache in question.)

- For requests-without-PASID:

— If the hardware finds an IOTLB entry that is for the page number of the input-address and that is associated with the Source-ID in the request, it may use the physical address, access rights, and other attributes from that entry.

— Hardware may use the source-ID of the request to select a context-cache entry. It can use that entry to qualify the request based on the attributes in the entry, and obtain the Domain-ID specified by software. If the hardware does not find a matching context-cache entry, it can traverse the root/extended-root tables and context/extended-context tables to obtain and

cache the context/extended-context entry. If the context-cache entry indicates pass-through access, the request is processed as if it found a IOTLB entry with a matching unity translation. Else, it continues the translation process as follows.

— If the hardware does not find a relevant IOTLB entry, it may use the bits MGAW:21 of the input-address to select an entry from the PDE-cache that is associated with the Domain-ID from the context-cache entry. It can then use that entry to complete the translation process (locating a SL-PTE, etc., as described in Section 3.7 for second-level translation) as if it had traversed the SL-PML5E (with 5-level translation), SL-PML4E, SL-PDPE and SL-PDE corresponding to the PDE-cache entry.

— If the hardware does not find a relevant IOTLB entry or a relevant PDE-cache entry, it may use bits MGAW:30 of the input-address to select an entry from the PDPE cache that is associated with the Domain-ID from the context-cache entry. It can then use that entry to complete the translation process (locating a SL-PDE, etc., as described in Section 3.7 for second-level translation) as if it had traversed the SL-PML5E (with 5-level translation), SL-PML4E and the SL-PDPE corresponding to the PDPE-cache entry.

— If the hardware does not find a relevant IOTLB entry, a relevant PDE-cache entry, or a relevant PDPE-cache entry, it may use bits MGAW:39 of the input-address to select an entry from the PML4E-cache that is associated with the Domain-ID from the context-cache entry. It can then use that entry to complete the translation process (locating a SL-PDPE, etc., as described in Section 3.7 for second-level translation) as if it had traversed the corresponding SL-PML5E (with 5-level translation) and SL-PML4E.

— With 5-level translation, if the hardware does not find a relevant IOTLB entry, a relevant PDE-cache entry, a relevant PDPE-cache entry, or a relevant PML4E-cache entry, it may use bits MGAW:48 of the input-address to select an entry from the PML5E-cache that is associated with the Domain-ID from the context-cache entry. It can then use that entry to complete the translation process (locating a SL-PML4E, SL-PDPE, etc., as described in Section 3.7 for second-level translation) as if it had traversed the corresponding SL-PML5E.

— If the hardware does not find an IOTLB, or paging-structure-cache entry for the input-address in the request-without-PASID, it uses the source-ID, and input-address in the request to traverse the entire second-level paging-structure hierarchy.

(Any of the above steps would be skipped if the hardware does not support the cache in question.)

6.2.7 Multiple Cached Entries for a Single Paging-Structure Entry

The paging-structure caches and IOTLBs and paging-structure caches may contain multiple entries associated with a single PASID and domain-ID) and with information derived from a single paging-structure entry. For illustration, following are some examples for first-level translation with 4-level paging (similar scenarios are possible with 5-level paging):

- Suppose that two PML4Es contain the same physical address and thus reference the same page-directory-pointer table. Any PDPTE in that table may result in two PDPTE-cache entries, each associated with a different set of input-addresses. Specifically, suppose that the $n1^{th}$ and $n2^{th}$ entries in the PML4 table contain the same physical address. This implies that the physical address in the m^{th} PDPTE in the page-directory-pointer table would appear in the PDPTE-cache entries associated with both p1 and p2, where (p1 » 9) = n1, (p2 » 9) = n2, and (p1 & 1FFH) = (p2 & 1FFH) = m. This is because both PDPTE-cache entries use the same PDPTE, one resulting from a reference from the $n1^{th}$ PML4E and one from the $n2^{th}$ PML4E.
- Suppose that the first PML4E (i.e., the one in position 0) contains the physical address X in PASID-entry (the physical address of the PML4 table). This implies the following:
  — Any PML4-cache entry associated with input-address with 0 in bits 47:39 contains address X.
  — Any PDPTE-cache entry associated with input-addresses with 0 in bits 47:30 contains address X. This is because the translation for a input-address for which the value of bits 47:30 is 0 uses the value of bits 47:39 (0) to locate a page-directory-pointer table at address X (the address of the PML4 table). It then uses the value of bits 38:30 (also 0) to find address X again and to store that address in the PDPTE-cache entry.

— Any PDE-cache entry associated with input-addresses with 0 in bits 47:21 contains address X for similar reasons.

— Any IOTLB entry for page number 0 (associated with input-addresses with 0 in bits 47:12) translates to page frame X » 12 for similar reasons.

The same PML4E contributes its address X to all these cache entries because the self-referencing nature of the entry causes it to be used as a PML4E, a PDPTE, a PDE, and a PTE.

Similar examples can be constructed with other paging structures (e.g., PDPE, PDE) and with PML5E (with 5-level paging). Multiple cached entries for a single paging-structure entry are also possible with second-level translation (involving SL-PML5Es (with 5-level translation), SL-PML4Es, SL-PDPEs, SL-PDEs, and SL-PTEs).

6.3 Translation Caching at Endpoint Device

Chapter 4 described support for endpoint devices to request translations from remapping hardware and cache on Device-TLBs that are local to the endpoint device. Device-TLBs may be utilized to improve address-translation performance and/or to support recoverable translation faults (see Chapter 7). Translation requests from endpoint devices are address translated by the remapping hardware using its translation caches as described in previous sections, and the resulting translation is returned to the endpoint device in a Translation Completion. Refer to Section 4.1.2 for attributes returned in the Translation-Completion. The endpoint device may cache the information returned in the Translation-Completion locally in its Device-TLBs.

6.4 Interrupt Entry Cache

Remapping hardware supporting interrupt remapping may cache frequently used interrupt remapping table entries in the interrupt-entry-cache (IEC). Each entry in a interrupt-entry-cache is an individual interrupt-remap-table-entry. Each cached entry is referenced by the interrupt_index number computed from attributes in the interrupt request (see Section 5.1.3). Each interrupt-entry-cache entry architecturally contains the following information (see Section 9.10):

- Attributes of the remapped interrupt from the IRTE:
  - — Interrupt Vector
  - — Destination ID
  - — Delivery Mode
  - — Trigger Mode
  - — Redirection Hint
- The Fault Processing Disable (FPD) flag from the IRTE
- The interrupt source validation attributes (SID, SQ, SVT fields) from the IRTE.

For implementations reporting Caching Mode (CM) as Clear in the Capability Register, if any of the interrupt-remapping fault conditions described in Section 5.1.4.1 is encountered, the resulting entry is not cached in the IEC. For implementations reporting Caching Mode (CM) as Set in the Capability Register, interrupt-remapping fault conditions may cause caching of the corresponding interrupt remapping entries.

Remapping hardware utilize the interrupt-entry cache as follows:

- If the hardware finds an IEC entry that is for the interrupt_index number of the request, it may use the interrupt attributes from the IEC entry (subject to the interrupt-source validation checks as described in Section 9.10).
- If the hardware does not find a matching IEC entry, it uses the interrupt_index computed for the request to fetch the interrupt-remap-table-entry from the interrupt-remap-table.

6.5 Invalidation of Translation Caches

As noted in Section 6.2, the remapping hardware may create entries in the various translation caches when requests are translated, and it may retain these entries even after the translation structures used to create them have been modified by software. To ensure that address translation uses the modified translation structures, software should take action to invalidate any cached entries that may contain information that has since been modified.

For software to invalidate the various caching structures, the architecture supports the following two types of invalidation interfaces:

- **Register-based invalidation interface**: A legacy invalidation interface with limited capabilities, supported by all implementations of this architecture.
- **Queued invalidation interface**: An expanded invalidation interface with extended capabilities, supported by later implementations of this architecture. Hardware implementations report support for queued invalidation interface through the Extended Capability Register (see Section 10.4.3).

The following sections provides more details on these hardware interfaces.

6.5.1 Register-based Invalidation Interface

The register-based invalidations provides a synchronous hardware interface for invalidations. Software writes to the invalidation command registers to submit invalidation command and may poll on these registers to check for invalidation completion.

Hardware implementations must process commands submitted through the invalidation registers irrespective of the remapping hardware enable status (i.e irrespective of TES and IES status in the Global Status Register. See Section 10.4.5).

Register-based invalidation has the following limitations:

- Register-based invalidation can be used only when queued-invalidations are not enabled.
- Register-based invalidation can target only invalidation of second-level translations. Invalidation of first-level and nested translations are not supported (which are supported only through queued-invalidations).
- Register-based invalidation cannot invalidate Device-TLBs on endpoint devices.

The following sub-sections describe the register-based invalidation command registers.

6.5.1.1 Context Command Register

Context Command Register (see Section 10.4.7) supports invalidating the context-cache. The architecture defines the following types of context-cache invalidation requests. Hardware implementations may perform the actual invalidation at a coarser granularity if the requested invalidation granularity is not supported.

- *Global Invalidation*: All context-cache entries cached at the remapping hardware are invalidated.
- *Domain-Selective Invalidation*: Context-cache entries associated with the specified domain-id are invalidated.
- *Device-Selective Invalidation*: Context-cache entries associated with the specified device source-id and domain-id are invalidated.

When modifying root-entries or context-entries referenced by more than one remapping hardware units in a platform, software is responsible to explicitly invalidate the context-cache at each of these hardware units.

6.5.1.2 IOTLB Registers

IOTLB invalidation is supported through two 64-bit registers; (a) IOTLB Invalidate Register (see Section 10.4.8.1) and (b) Invalidation Address Register (see Section 10.4.8.2).

The architecture defines the following types of IOTLB invalidation requests. Hardware implementations may perform the actual invalidation at a coarser granularity if the requested invalidation granularity is not supported.

- *Global Invalidation*:
  - — All IOTLB entries are invalidated.
  - — All PASID-cache entries are invalidated.
  - — All paging-structure-cache entries are invalidated.
- *Domain-Selective Invalidation*:
  - — IOTLB entries caching mappings (first-level, second-level, and nested) associated with the specified domain-id are invalidated.
  - — PASID-cache entries associated with the specified domain-id are invalidated.
  - — Paging-structure-cache entries caching mappings (first-level, second-level and nested) associated with the specified domain-id are invalidated.
- *Page-Selective-within-Domain Invalidation*:
  - — IOTLB entries caching second-level mappings associated with the specified domain-id and the second-level-input-address range are invalidated.
  - — IOTLB entries caching first-level and nested mappings associated with the specified domain-id are invalidated.
  - — PASID-cache entries associated with the specified domain-id are invalidated.
  - — Paging-structure-cache entries caching first-level and nested mappings associated with the specified domain-id tare invalidated.
  - — Paging-structure-cache entries caching second-level mappings associated with the specified domain-id and the second-level-input-address range are invalidated, if the Invalidation Hint (IH) field is Clear. Else, the paging-structure-cache entries caching second-level mappings are preserved.

For any of the above operations, hardware may perform coarser invalidation. The actual invalidation granularity reported by hardware in the IOTLB Invalidate Register is always the granularity at which the invalidation was performed on the IOTLB.

When modifying page-table entries referenced by more than one remapping hardware units in a platform, software is responsible to explicitly invalidate the IOTLB at each of these hardware units.

6.5.2 Queued Invalidation Interface

The queued invalidation provides an advanced interface for software to submit invalidation requests to hardware and to synchronize invalidation completions with hardware. Hardware implementations report queued invalidation support through the Extended Capability Register.

Queued-invalidation supports invalidation of first-level, second-level and nested translations. Queued invalidations may be beneficial for the following software usages:

- Usages that frequently map and un-map pages in the translation structures (causing frequent invalidations).
- Usages where page-selective invalidation requests frequently span pages that are not virtually contiguous.
- Usages where software can do useful work while an invalidation operation is pending in hardware.

- Invalidation operations that are latency prone (such as invalidating Device-TLBs on a endpoint device across an I/O interconnect).
- Usages of VMM virtualizing remapping hardware, where VMM may improve the virtualization performance by allowing guests to queue invalidation requests (instead of intercepting guest MMIO accesses for each invalidation request as required by the register based interface).

The queued invalidation interface uses an Invalidation Queue (IQ), which is a circular buffer in system memory. Software submits commands by writing Invalidation Descriptors to the IQ. The following registers are defined to configure and manage the IQ:

- *Invalidation Queue Address Register*: Software programs this register to configure the base physical address and size of the contiguous memory region in system memory hosting the Invalidation Queue.
- *Invalidation Queue Head Register*: This register points to the invalidation descriptor in the IQ that hardware will process next. The Invalidation Queue Head register is incremented by hardware after fetching a valid descriptor from the IQ. Hardware interprets the IQ as empty when the head and tail registers are equal.
- *Invalidation Queue Tail Register*: This register points to the invalidation descriptor in the IQ to be written next by software. Software increments this register after writing one or more invalidation descriptors to the IQ.

To enable queued invalidations, software must:

- Ensure all invalidation requests submitted to hardware through the register-based invalidation registers are completed. (i.e. no pending invalidation requests in hardware).
- Initialize the Invalidation Queue Tail Register (see Section 10.4.22) to zero.
- Setup the IQ address and size through the Invalidation Queue Address Register (see Section 10.4.23).
- Enable the queued invalidation interface through the Global Command Register (see Section 10.4.4). When enabled, hardware sets the QIES field in the Global Status Register (see Section 10.4.5).

When the queued invalidation is enabled, software must submit invalidation commands only through the IQ (and not through any register-based invalidation command registers).

Hardware fetches descriptors from the IQ in FIFO order starting from the Head Register if all of the following conditions are true. This is independent of the remapping hardware enable status (state of TES and IES fields in Global Status Register).

- QIES field in the Global Status Register is Set (indicating queued invalidation is enabled)
- IQ is not empty (i.e. Head and Tail pointer Registers are not equal)
- There is no pending Invalidation Queue Error or Invalidation Time-out Error (IQE and ITE fields in the Fault Status Register are both Clear)

Hardware implementations may fetch one or more descriptors together. However, hardware must increment the Invalidation Queue Head Register only after verifying the fetched descriptor to be valid. Hardware handling of invalidation queue errors are described in Section 6.5.2.10.

Once enabled, to disable the queued invalidation interface, software must:

- *Quiesce the invalidation queue*. The invalidation queue is considered quiesced when the queue is empty (head and tail registers equal) and the last descriptor completed is an Invalidation Wait Descriptor (which indicates no invalidation requests are pending in hardware).
- *Disable queued invalidation.* The queued invalidation interface is disabled through the Global Command Register. When disabled, hardware resets the Invalidation Queue Head Register to zero, and clears the QIES field in the Global Status Register.

The following subsections describe the various Invalidation Descriptors. All descriptors are 128-bit sized. Type field (bits 3:0) of each descriptor identifies the descriptor type. Software must program the reserved fields in the descriptors as zero.

6.5.2.1 Context-cache Invalidate Descriptor

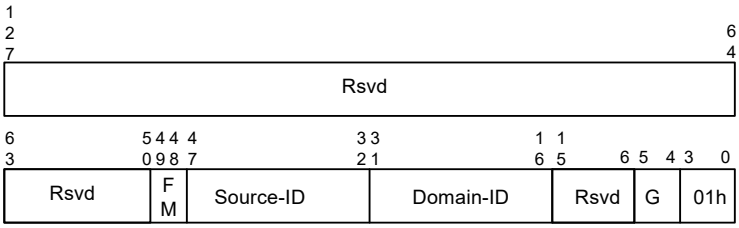


Figure 6-20. Context-cache Invalidate Descriptor

The Context-cache Invalidate Descriptor (*cc_inv_dsc*) allows software to invalidate the context-cache, there by forcing hardware to use the entries from root (extended-root) and context (extended-context) tables in system memory. The context-cache invalidate descriptor includes the following parameters:

- *Granularity (G)*: The G field indicates the requested invalidation granularity. The encoding of the G field is same as the CIRG field in the Context Command Register (described in Section 10.4.7). Hardware implementations may perform coarser invalidation than the granularity requested.
  - *Global Invalidation (01b)*: All context-cache entries cached at the remapping hardware are invalidated.
  - *Domain-Selective Invalidation (10b)*: Context-cache entries associated with the specified domain-id are invalidated.
  - *Device-Selective Invalidation* (11b): Context-cache entries associated with the specified device source-id and domain-id are invalidated.*Domain-ID (DID)*: For domain-selective and device-selective invalidations, the DID field indicates the target domain-id.
- *Source-ID (SID)*: For device-selective invalidations, the SID field indicates the device source-id.
- *Function Mask (FM)*: The Function Mask field indicates the bits of the SID field to be masked for device-selective invalidations. The usage and encoding of the FM field is same as the FM field encoding in the Context Command Register (see Section 10.4.7).

Hardware implementations reporting a write-buffer flushing requirement (RWBF=1 in Capability Register) must implicitly perform a write buffer flushing before invalidating the context-cache. Refer to Section 6.8 for write buffer flushing requirements.

Since information from the context-cache may be used to tag entries in the PASID-cache, IOTLB and paging-structure caches, software must always follow a context-cache invalidation with a PASID-cache invalidation (if context-cache entry supports requests-with-PASID), followed by an IOTLB invalidation. The granularity of the PASID-cache and IOTLB invalidation must be equal or greater than the preceding context-cache invalidation (e.g., A global context-cache invalidation must be followed by all-PASIDs PASID-cache invalidation and global IOTLB invalidation; A domain/device selective context-cache invalidation must be followed by all-PASIDs PASID-cache invalidation and domain-selective or global IOTLB invalidation).

6.5.2.2 PASID-cache Invalidate Descriptor

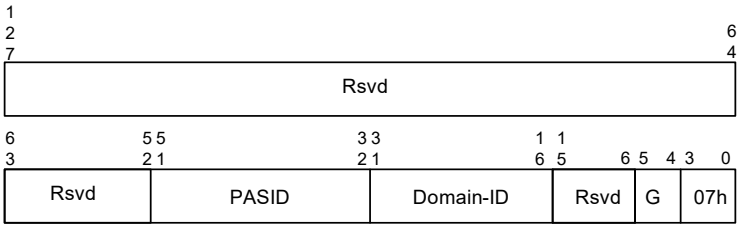


Figure 6-21. PASID-cache Invalidate Descriptor

The PASID-cache Invalidate Descriptor (*pc_inv_dsc*) allows software to invalidate the PASID-cache, there by forcing hardware to use entries from the PASID table in system memory for translating requests-with-PASID. The PASID-cache invalidate descriptor includes the following parameters:

- *Granularity (G)*: The G field indicates the requested invalidation granularity. Hardware implementations may perform coarser invalidation than the granularity requested. The encoding of the G field is as follows:
  - *All-PASIDs Invalidation (00b)*: All PASID-cache entries associated with the specified domain-id are invalidated.
  - *PASID-Selective Invalidation* (01b): PASID-cache entries associated with the specified PASID value and the domain-id are invalidated.
- *Domain-ID (DID)*: The DID field indicates the target domain-id. Hardware ignores bits 31:(16+N), where N is the domain-id width reported in the Capability Register.
- *PASID*: The PASID value indicates the target process-address-space to be invalidated. This field is ignored by hardware for all-PASIDs invalidation granularity.

Hardware implementations reporting a write-buffer flushing requirement (RWBF=1 in Capability Register) must implicitly perform a write buffer flushing before invalidating the PASID-cache. Refer to Section 6.8 for write buffer flushing requirements.

Since information from the PASID-cache may be used to tag the IOTLB and paging-structure caches, software must always follow a PASID-cache invalidation with a IOTLB invalidation. *All-PASIDs* granularity PASID-cache invalidation must be followed by *All-mappings-within-all-PASIDs* extended IOTLB invalidation; A *PASIDs-selective* granularity PASID-cache invalidation must be followed by *Non-globals-within-PASID* extended IOTLB invalidation).

6.5.2.3 IOTLB Invalidate Descriptor

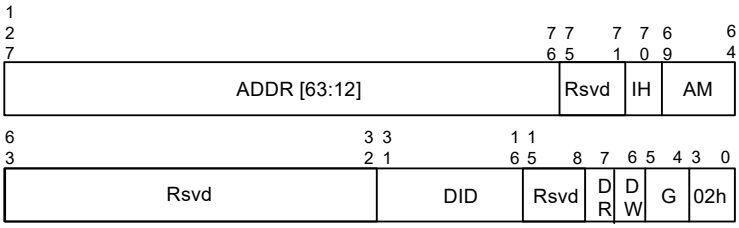


Figure 6-22. IOTLB Invalidate Descriptor

The IOTLB Invalidate Descriptor (*iotlb_inv_dsc*) allows software to invalidate the IOTLB, paging-structure-caches, and PASID-cache. This descriptor is expected to be used when invalidating cached entries that utilized second-level translations. The descriptor includes the following parameters:

- *Granularity (G)*: The G field indicates the requested invalidation granularity. Hardware implementations may perform coarser invalidation than the granularity requested. The encoding of the G field is same as the IIRG field in the IOTLB Invalidate Register (see Section 10.4.8):
  - *Global Invalidation (01b)*:
    - All IOTLB entries are invalidated.
    - All paging-structure-cache entries are invalidated.
    - All PASID-cache entries are invalidated.
  - *Domain-Selective Invalidation (10b)*:
    - IOTLB entries caching mappings (first-level, second-level, and nested) associated with the specified domain-id are invalidated.
    - Paging-structure-cache entries caching mappings (first-level, second-level and nested) associated with the specified domain-id are invalidated.
    - PASID-cache entries associated with the specified domain-id are invalidated.
  - *Page-Selective-within-Domain Invalidation (11b)*:
    - IOTLB entries caching second-level mappings associated with the specified domain-id and the second-level-input-address range are invalidated.
    - IOTLB entries caching first-level and nested mappings associated with the specified domain-id are invalidated.
    - Paging-structure-cache entries caching first-level and nested mappings associated with the specified domain-id are invalidated.
    - Paging-structure-cache entries caching second-level mappings associated with the specified domain-id and the second-level-input-address range are invalidated, if the Invalidation Hint (IH) field has value of 0. If the IH value is 1, the paging-structure-cache entries caching second-level mappings are preserved.
    - PASID-cache entries associated with the specified domain-id are invalidated.
- *Drain Reads (DR)*: Software sets this flag to indicate hardware must drain read requests that are already processed by the remapping hardware, but queued within the Root-Complex to be completed. When the value of this flag is 1, hardware must perform the read drain before the next Invalidation Wait Descriptor (see Section 6.5.2.8) is completed. Section 6.5.5 describes hardware support for draining.
- *Drain Writes (DW)*: Software sets this flag to indicate hardware must drain relevant write requests that are already processed by the remapping hardware, but queued within the Root-

Complex to be completed. When the value of this flag is 1, hardware must drain the relevant writes before the next Invalidation Wait Descriptor is completed. Section 6.5.5 describes hardware support for draining.

- *Domain-ID (DID)*: For domain-selective and page-selective invalidations, the DID field indicates the target domain-id. Hardware ignores bits 31:(16+N), where N is the domain-id width reported in the Capability Register. This field is ignored by hardware for global invalidations.
- *Invalidation Hint (IH)*: For page-selective-within-domain invalidations, the Invalidation Hint specifies if the second-level mappings cached in the paging-structure-caches that controls the specified address/mask range needs to be invalidated or not. For software usages that updates only the leaf SL-PTEs, the second-level mappings in the paging-structure-caches can be preserved by specifying the Invalidation Hint field value of 1. This field is ignored by hardware for global and domain-selective invalidations.
- *Address (ADDR)*: For page-selective-within-domain invalidations, the Address field indicates the starting second-level page address of the mappings that needs to be invalidated. Hardware ignores bits 127:(64+N), where N is the maximum guest address width (MGAW) supported. This field is ignored by hardware for global and domain-selective invalidations.
- *Address Mask (AM)*: For page-selective-within-domain invalidations, the Address Mask specifies the number of contiguous second-level pages that needs to be invalidated. The encoding for the AM field is same as the AM field encoding in the Invalidate Address Register (see Section 10.4.8.2). When invalidating a large-page translation, software must use the appropriate Address Mask value (0 for 4KByte page, 9 for 2-MByte page, and 18 for 1-GByte page).

Hardware implementations reporting a write-buffer flushing requirement (RWBF=1 in Capability Register) must implicitly perform a write buffer flushing before invalidating the IOTLB.

6.5.2.4 Extended IOTLB Invalidate Descriptor

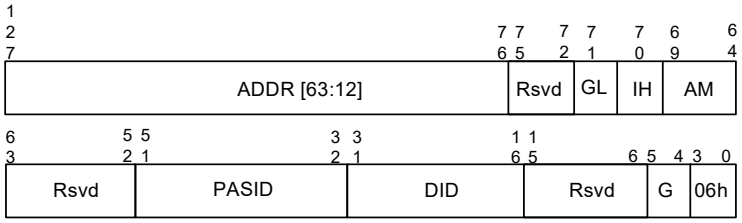


Figure 6-23. Extended IOTLB Invalidate Descriptor

The Extended IOTLB Invalidate Descriptor (*ext_iotlb_inv_dsc*) allows software to invalidate first-level and nested mappings from the IOTLB and the paging-structure-caches. The descriptor includes the following parameters:

- *Granularity (G)*: The G field indicates the requested invalidation granularity. Hardware implementations may perform coarser invalidation than the granularity requested. The encoding of the G field is as follows:
  - *All-mappings-within-all-PASIDs (00b):*
    - IOTLB entries caching first-level and nested mappings (including mappings to global pages) that are associated with the specified domain-id are invalidated. The global pages are invalidated, independent of the programming of the GL field.
    - Paging-structure-cache entries caching first-level and nested mappings that are associated with the specified domain-id are invalidated.
  - *Non-globals-within-all-PASIDs (01b)*:

- IOTLB entries caching first-level and nested mappings to non-global pages (i.e. pages that are translated by first-level paging entries with the Global field value of 0) that are associated with the specified domain-id are invalidated.
- Paging-structure-cache entries caching first-level and nested mappings that are associated with the specified domain-id are invalidated.

— *Non-globals-within-PASID (10b)*:

- IOTLB entries caching first-level and nested mappings to non-global pages (i.e. pages that are translated by first-level paging entries with the Global field value of 0) that are associated with the specified PASID and domain-id are invalidated.
- Paging-structure-cache entries caching first-level and nested mappings that are associated with the specified PASID and domain-id are invalidated.

— *Page-Selective-within-PASID (11b):*

- IOTLB entries caching first-level and nested mappings that are associated with the specified PASID, domain-id, and the first-level-input-address range are invalidated. If the Global Hint (GL) field has value of 1, the mappings specified by the first-level-input-address range are invalidated even if any of these mappings are global pages (and hence not private to the specified PASID). If the GL field has value of 0, any of the specified mappings that are cached as global pages may be preserved.
- Paging-structure-cache entries caching first-level and nested mappings that are associated with the specified PASID, domain-id, and the first-level-input-address range are invalidated, if the Invalidation Hint (IH) field has value of 0. If the IH value is 1, the paging-structure-cache entries are preserved.

- *Domain-ID (DID)*: The DID field indicates the target domain-id. Hardware ignores bits 31:(16+N), where N is the domain-id width reported in the Capability Register.
- *PASID*: The PASID value indicates the target process-address-space to be invalidated. This field is ignored by hardware for *all-mappings-within-all-PASIDs* and *non-globals-within-all-PASIDs* invalidations.
- *Invalidation Hint (IH)*: For *page-selective-within-PASID* invalidations, the Invalidation Hint specifies if the first-level and nested mappings cached in the paging-structure-caches that controls the specified address/mask range needs to be invalidated or not. For software usages that updates only the leaf PTEs, the first-level and nested mappings in the paging-structure-caches can be preserved by specifying the Invalidation Hint field value of 1. This field is ignored by hardware for other invalidation granularities.
- *Global Hint (GL)*: For *page-selective-within-PASID* invalidations, the Global Hint (GL) field specifies if mappings to global pages needs to be invalidated or not. For software usages that updates only non-global mappings, the global mappings in the IOTLB can be preserved by specifying the Global Hint field value of 0. This field is ignored by hardware for other invalidation granularities.
- *Address (ADDR)*: For *page-selective-within-PASID* invalidations, the Address field indicates the starting first-level page address of the mappings that needs to be invalidated. This field is ignored by hardware for all-mappings-within-all-PASIDs and non-globals-within-all-PASIDs invalidations.
- *Address Mask (AM)*: For *page-selective-within-PASID* invalidations, the Address Mask specifies the number of contiguous first-level 4-KByte pages that needs to be invalidated. The encoding for the AM field is same as the AM field encoding in the Invalidate Address Register (see Section 10.4.8.2). When invalidating a large-page translation, software must use the appropriate Address Mask value (0 for 4KByte page, 9 for 2-MByte page, and 18 for 1-GByte page).

Extended IOTLB invalidations are not required to invalidate PASID-cache entries, and second-level mappings cached in paging-structure-caches.

Extended IOTLB invalidations must always drain read and write requests that are already processed by the remapping hardware, but queued within the Root-Complex to be completed. Hardware must drain such outstanding read and write requests (to make them globally observable) before the next Invalidation Wait Descriptor (see Section 6.5.2.8) is completed. Section 6.5.5 further describes hardware support for draining.

6.5.2.5 Device-TLB Invalidate Descriptor

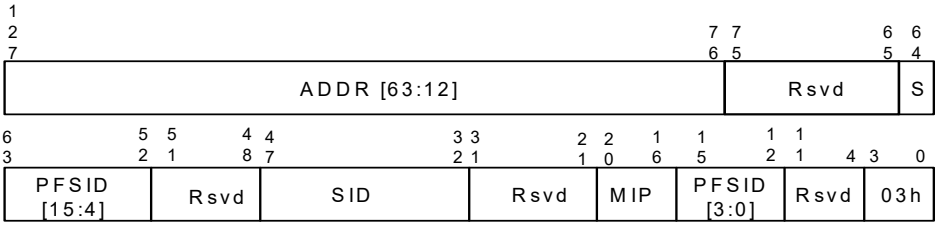


Figure 6-24. Device-TLB Invalidate Descriptor

The Device-TLB Invalidate Descriptor (*dev_tlb_inv_dsc)* allows software to invalidate cached mappings used by requests-without-PASID from the Device-TLB on a endpoint device. The descriptor includes the following parameters:

- *Source-ID (SID)*: The SID field indicates the source-id of the endpoint device whose Device-TLB needs to be invalidated.
- *Address (ADDR)*: The address field indicates the starting second-level-input-address for the mappings that needs to be invalidated. The Address field is qualified by the S field.
- *Size (S)*: The size field indicates the number of consecutive pages targeted by this invalidation request. If S field is zero, a single page at page address specified by Address [63:12] is requested to be invalidated. If S field is Set, the least significant bit in the Address field with value 0b indicates the invalidation address range. For example, if S field is Set and Address[12] is Clear, it indicates an 8KB invalidation address range with base address in Address [63:13]. If S field and Address[12] is Set and bit 13 is Clear, it indicates a 16KB invalidation address range with base address in Address [63:14], etc.
- *Max Invalidations Pending (MIP)*: This field is a hint to hardware to indicate the maximum number of pending invalidation requests the specified PCI-Express endpoint device (Physical Function) can handle optimally. Endpoint devices are required to accept up to 32 pending invalidation requests, but the device may put back pressure on the I/O interconnect (E.g., PCI-Express link) for multiple pending invalidations beyond Max Invalidations Pending. A value of 0h in MIP field indicates the device is capable of handling maximum (32) pending invalidation requests without throttling the link. Hardware implementations may utilize this field to throttle the number of pending invalidation requests issued to the specified device. Remapping hardware implementations reporting Pending Invalidation Throttling (DIT=1 in ECAP_REG) utilize this field to throttle the number of pending invalidation requests issued to the physical function specified in PFSID.
- *Physical Function Source-ID (PFSID)*: Remapping hardware implementations reporting Device-TLB Invalidation Throttling as not supported (DIT = 0 in ECAP_REG) treats this field as reserved. For implementations reporting Device-TLB Invalidation Throttling as supported (DIT=1 in ECAP_REG), if the Source-ID (SID) field specifies a Physical Function (PF), PFSID field specifies same value as the SID field; If the Source-ID (SID) field specifies a SR-IOV Virtual Function (VF), PFSID field specifies the Source-ID of the Physical Function (PF) associated with the Virtual Function (VF).

Since translation requests from a device may be serviced by hardware from the IOTLB, software must always request IOTLB invalidation (*iotlb_inv_dsc)* before requesting corresponding Device-TLB (*dev_tlb_inv_dsc)* invalidation.

6.5.2.6 Extended Device-TLB Invalidate Descriptor

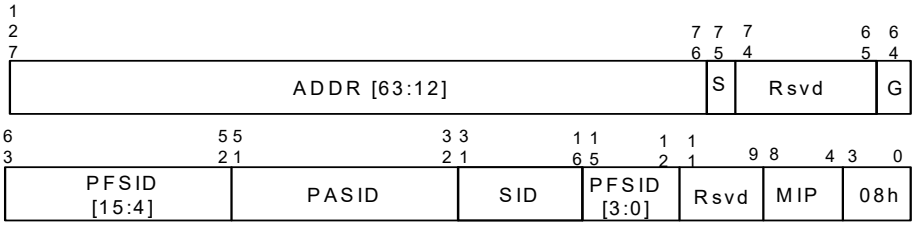


Figure 6-25. Extended Device-TLB Invalidate Descriptor

The Extended Device-TLB Invalidate Descriptor (*ext_dev_tlb_inv_dsc)* allows software to invalidate cached mappings used by requests-with-PASID from the Device-TLB on an endpoint device. The descriptor includes the following parameters:

- *Source-ID (SID)*: The SID field indicates the source-id of the endpoint device whose Device-TLB needs to be invalidated.
- *Global (G)*: The G field value is 1, the extended device-TLB invalidation request applies across all PASIDs at the endpoint device-TLB. If G field value is 0, the invalidated is targeted to a specific PASID specified by the PASID value field.
- *PASID*: The PASID value indicates the target process-address-space to be invalidated. This field is ignored by hardware if the value of G field is 1.
- *Address (ADDR)*: The address field indicates the starting first-level-input-address for the mappings that needs to be invalidated. The Address field is qualified by the S field.
- *Size (S)*: The size field indicates the number of consecutive pages targeted by this invalidation request. If S field is zero, a single page at page address specified by Address [63:12] is requested to be invalidated. If S field is Set, the least significant bit in the Address field with value 0b indicates the invalidation address range. For example, if S field is Set and Address[12] is Clear, it indicates an 8KB invalidation address range with base address in Address [63:13]. If S field and Address[12] is Set and bit 13 is Clear, it indicates a 16KB invalidation address range with base address in Address [63:14], etc.
- *Max Invalidations Pending (MIP)*: This field is a hint to hardware to indicate the maximum number of pending invalidation requests the specified PCI-Express endpoint device (Physical Function) can handle optimally. Endpoint devices are required to accept up to 32 pending invalidation requests, but the device may put back pressure on the I/O interconnect (E.g., PCI-Express link) for multiple pending invalidations beyond Max Invalidations Pending. A value of 0h in MIP field indicates the device is capable of handling maximum (32) pending invalidation requests without throttling the link. Hardware implementations may utilize this field to throttle the number of pending invalidation requests issued to the specified device. Remapping hardware implementations reporting Pending Invalidation Throttling (DIT=1 in ECAP_REG) utilize this field to throttle the number of pending invalidation requests issued to the physical function specified in PFSID.
- *Physical Function Source-ID (PFSID)*: Remapping hardware implementations reporting Device-TLB Invalidation Throttling as not supported (DIT = 0 in ECAP_REG) treats this field as reserved. For implementations reporting Device-TLB Invalidation Throttling as supported (DIT=1 in ECAP_REG), if the Source-ID (SID) field specifies a Physical Function (PF), PFSID field specifies same value as the SID field; If the Source-ID (SID) field specifies a SR-IOV Virtual Function (VF), PFSID field specifies the Source-ID of the Physical Function (PF) associated with the Virtual Function (VF).

Since translation requests from a device may be serviced by hardware from the IOTLB, software must always request an extended IOTLB invalidation (ext_*iotlb_inv_dsc)* before requesting corresponding Device-TLB (ext_*dev_tlb_inv_dsc)* invalidation.

6.5.2.7 Interrupt Entry Cache Invalidate Descriptor

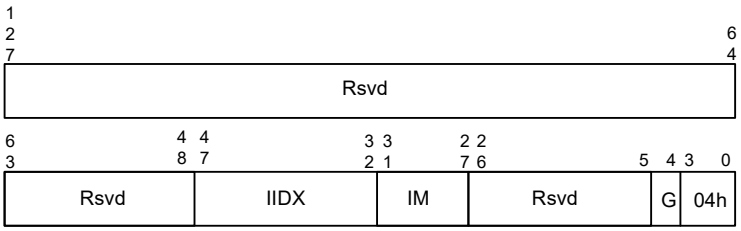


Figure 6-26. Interrupt Entry Cache Invalidate Descriptor

The Interrupt Entry Cache Invalidate Descriptor (*iec_inv_dsc*) allows software to invalidate the Interrupt Entry Cache. The descriptor includes the following parameters:

- *Granularity (G):* This field indicates the granularity of the invalidation request. If Clear, a global invalidation of the interrupt-remapping cache is requested. If Set, a index-selective invalidation is requested.
- *Interrupt Index (IIDX):* This field specifies the index of the interrupt remapping entry that needs to be invalidated through a index-selective invalidation.
- *Index Mask (IM):* For index-selective invalidations, the index-mask specifies the number of contiguous interrupt indexes that needs to be invalidated. The encoding for the IM field is described below in Table 9).

Table 9. Index Mask Programming

Index Mask Value	Index bits Masked	Mappings Invalidated
0	None	1
1	0	2
2	1:0	4
3	2:0	8
4	3:0	16
…	…	…

As part of IEC invalidation, hardware must drain interrupt requests that are already processed by the remapping hardware, but queued within the Root-Complex to be delivered to the processor. Section 6.5.6 describes hardware support for interrupt draining.

Hardware implementations reporting a write-buffer flushing requirement (RWBF=1 in Capability Register) must implicitly perform a write buffer flushing before invalidating the Interrupt Entry Cache.

6.5.2.8 Invalidation Wait Descriptor

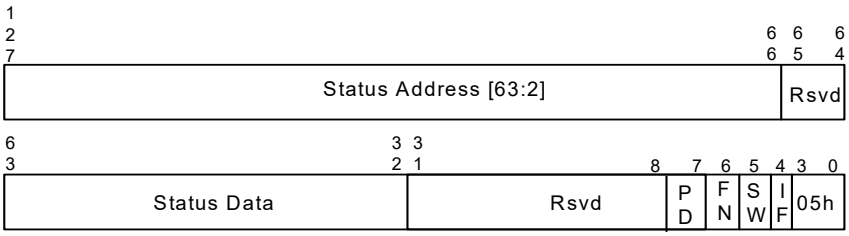


Figure 6-27. Invalidation Wait Descriptor

The Invalidation Wait Descriptor (*inv_wait_dsc*) descriptor allows software to synchronize with hardware for the invalidation request descriptors submitted before the wait descriptor. The descriptor includes the following parameters:

- *Status Write (SW):* Indicate the invalidation wait descriptor completion by performing a coherent DWORD write of the value in the Status Data field to the address specified in the Status Address field.
- *Status Address and Data:* Status address and data is used by hardware to perform wait descriptor completion status write when the SW field is Set. Hardware behavior is undefined if the Status Address specified is not an address route-able to memory (such as peer address, interrupt address range of 0xFEEX_XXXX etc.). The Status Address and Data fields are ignored by hardware when the Status Write (SW) field is Clear.
- *Interrupt Flag (IF):* Indicate the invalidation wait descriptor completion by generating an invalidation completion event per the programming of the Invalidation Completion Event Registers. Section 6.5.2.9 describes details on invalidation event generation.
- *Fence Flag (FN):* When Set, indicates descriptors following the invalidation wait descriptor must be processed by hardware only after the invalidation wait descriptor completes.
- *Page-request Drain (PD):* Remapping hardware implementations reporting Page-request draining as not supported (PDS = 0 in ECAP_REG) treats this field as reserved. For implementations reporting Page-request draining as supported (PDS=1 in ECAP_REG), value of 1 in this field specifies the Invalidation wait completion status write (if SW=1) and Invalidation wait completion interrupt (if IF=1) must be ordered (visible to software) behind page-request descriptor (*page_req_dsc*) writes for all page requests received by remapping hardware before invalidation wait descriptor completion. For optimal performance, software must Set this field only if Page Request draining is required. Refer to Section 7.8 for remapping hardware behavior for page request draining.

Section 6.5.2.11 describes queued invalidation ordering considerations. Hardware completes an invalidation wait command as follows:

- If a status write is specified in the wait descriptor (SW=1), hardware performs a coherent write of the status data to the status address.
- If an interrupt is requested in the wait descriptor (IF=1), hardware sets the IWC field in the Invalidation Completion Status Register. An invalidation completion interrupt may be generated as described in the following section.

6.5.2.9 Hardware Generation of Invalidation Completion Events

The invalidation event interrupt generation logic functions as follows:

- At the time hardware sets the IWC field, it checks if the IWC field is already Set to determine if there is a previously reported invalidation completion interrupt condition that is yet to be serviced by software. If IWC field is already Set, the invalidation event interrupt is not generated.
- If the IWC field is not already Set, the Interrupt Pending (IP) field in the Invalidation Event Control Register is Set. The Interrupt Mask (IM) field is then checked and one of the following conditions is applied:
  - — If IM field is Clear, invalidation completion event interrupt is generated along with clearing the IP field.
  - — If IM field is Set, the invalidation completion event interrupt is not generated.

The following logic applies for interrupts held pending by hardware in the IP field:

- If IP field was Set when software clears the IM field, the invalidation completion event interrupt is generated along with clearing the IP field.
- If IP field was Set when software services the pending interrupt condition (indicated by IWC field in the Invalidation Completion Status Register being Clear), the IP field is cleared.

At the time an invalidation wait descriptor is completed by remapping hardware, if PD=1 in the wait descriptor, the invalidation completion status write (and/or invalidation completion event interrupt) that signal wait descriptor completion to software must push *page_req_desc* writes for all page requests already received by the remapping hardware. Refer to Section 7.8 for details on page request draining.

The invalidation completion event interrupt must push any in-flight invalidation completion status writes, including status writes that may have originated from the same *inv_wait_dsc* for which the interrupt was generated. Similarly, read completions due to software reading the Invalidation Completion Status Register (ICS_REG) or Invalidation Event Control Register (IECTL_REG) must push (commit) any in-flight invalidation completion event interrupts and status writes generated by the respective hardware unit.

The invalidation completion event interrupts are never subject to interrupt remapping.

6.5.2.10 Hardware Handling of Queued Invalidation Interface Errors

Hardware handles the various queued invalidation interface error conditions as follows:

- *Invalidation Queue Errors*: If hardware detects an invalid Tail pointer at the time of fetching a descriptor, or detects an error when fetching a descriptor from the invalidation queue, or detects that the fetched descriptor fetched is invalid, hardware sets the IQE (Invalidation Queue Error) field in the Fault Status Register. A fault event may be generated based on the programming of the Fault Event Control Register. The Head pointer Register is not incremented, and references the descriptor associated with the queue error. No new descriptors are fetched from the Invalidation Queue until software clears the IQE field in the Fault Status Register. Tail pointer Register updates by software while the IQE field is Set does not cause descriptor fetches by hardware. Any invalidation commands ahead of the invalid descriptor that are already fetched and pending in hardware at the time of detecting the invalid descriptor error are completed by hardware as normal.
- *Invalid Device-TLB Invalidation Response*: If hardware receives an invalid Device-TLB invalidation response, hardware sets the Invalidation Completion Error (ICE) field in the Fault Status Register. A fault event may be generated based on the programming of the Fault Event Control Register. Hardware continues with processing of descriptors from the Invalidation Queue as normal.
- *Device-TLB Invalidation Response Time-out*: If hardware detects a Device-TLB invalidation response time-out, hardware frees the corresponding ITag and sets the ITE (Invalidation Time-out Error) field in the Fault Status Register. A fault event may be generated based on the programming of the Fault Event Control Register. No new descriptors are fetched from the Invalidation Queue until software clears the ITE field in the Fault Status Register. Tail pointer Register updates by software while the ITE field is Set does not cause descriptor fetches by hardware. At the time ITE field is Set, hardware aborts any *inv_wait_dsc* commands pending in

hardware. Any invalidation responses received while ITE field is Set are processed as normal (as described in Section 4.3). Since the time-out could be for any (one or more) of the pending *dev_iotlb_inv_dsc* commands, execution of all descriptors including and behind the oldest pending *dev_iotlb_inv_dsc* is not guaranteed.

6.5.2.11 Queued Invalidation Ordering Considerations

Hardware must support the following ordering considerations when processing descriptors fetched from the invalidation queue:

- Hardware must execute an IOTLB invalidation descriptor (*iotlb_inv_dsc*) or extended IOTLB invalidation descriptor (*ext_iotlb_inv_dsc)* only after all Context-cache invalidation descriptors (*cc_inv_dsc*) and PASID-cache invalidation descriptors (*pc_inv_dsc*) ahead of it in the Invalidation Queue are completed.
- Hardware must execute a PASID-cache invalidation descriptors (*pc_inv_dsc*) only after all Context-cache invalidation descriptors (*cc_inv_dsc*) (*pc_inv_dsc*) ahead of it in the Invalidation Queue are completed.
- Hardware must execute a Device-TLB invalidation descriptor (*dev_tlb_inv_dsc)* only after all IOTLB invalidation descriptors (*iotlb_inv_dsc)* and Interrupt Entry Cache invalidation descriptors (*iec_inv_dsc)* ahead of it in the Invalidation Queue are completed.
- Hardware must execute an extended Device-TLB Invalidation descriptor (*ext_dev_tlb_inv_dsc)* only after all extended IOTLB invalidation descriptors (*ext_iotlb_inv_dsc)* and Interrupt Entry Cache invalidation descriptors (*iec_inv_dsc)* ahead of it in the Invalidation Queue are completed.
- Hardware must report completion of an Invalidation Wait Descriptor (*inv_wait_dsc)* only after at least all the descriptors ahead of it in the Invalidation Queue and behind the previous *inv_wait_dsc* are completed.
- If the Fence (FN) flag is 0 in a *inv_wait_dsc,* hardware may execute descriptors following the *inv_wait_dsc* before the wait command is completed. If the Fence (FN) flag is 1 in a *inv_wait_dsc,* hardware must execute descriptors following the *inv_wait_dsc* only after the wait command is completed.
- When a Device-TLB invalidation or extended Device-TLB invalidation time-out is detected, hardware must not complete any pending *inv_wait_dsc* commands.

6.5.3 IOTLB Invalidation Considerations

The following subsections describes additional details and considerations on IOTLB invalidations with use of first-level translations.

6.5.3.1 Implicit Invalidation on Page Requests

In addition to the explicit invalidation through invalidation commands (see Section 6.5.1 and Section 6.5.2) or through deferred invalidation messages (see Section 6.5.4), identified above, Page Requests from endpoint devices invalidate entries in the IOTLBs and paging-structure caches.

In particular, as part of reporting a Page Request from a endpoint device (to report a recoverable fault detected at the Device-TLB), the remapping hardware will invalidate any IOTLB entries that are for a page number corresponding to the address (and PASID, if present) in the Page Request. It also invalidates all entries in the paging-structure caches that would be used for that address (and that are associated with the PASID, if PASID present in the Page Request). These invalidations ensure that this recoverable translation fault will not recur (if the faulting operation is re-executed at the device) if it would not be caused by the contents of the paging structures in memory (and if, therefore, it resulted from cached entries that were not invalidated after the paging structures were modified in memory).

6.5.3.2 Caching Fractured Translations

Some implementations may choose to cache multiple smaller-page IOTLB entries (fractured translations) for a translation specified by the paging structures to use a page larger than 4 KBytes. There is no way for software to be aware that multiple translations for smaller pages have been used for a large page. Since software is required to always specify the appropriate Address Mask value to cover the address range to be invalidated (Address Mask value of 0 for invalidating a 4-KByte page, 9 for invalidating a 2-MByte page, and 18 for invalidating a 1-GByte page) in the IOTLB invalidation commands, these commands naturally invalidate all IOTLB entries corresponding a large-page translation.

6.5.3.3 Recommended Invalidation

The following items provide some recommendations regarding when software should perform IOTLB invalidations when modifying first-level paging entries.

- If software modifies a paging-structure entry that identifies the final page frame for a page number (either a PTE or a paging-structure entry in which the PS flag is 1), it should execute *page-selective-within-PASID* IOTLB invalidation command for any address with a page number whose translation uses that paging-structure entry, with an address-mask matching the page frame size. (Address Mask value of 0 for 4-KByte page, 9 for 2-Mbyte page, and 18 for 1-GByte page). If no intermediate paging-structures entries with PS=0 is modified, the invalidation command can specify Invalidation Hint (IH) as 1, if software chooses to not invalidate the paging-structure caches.

  If the same paging-structure entry may be used in the translation of different page numbers — see Section 6.2.7 — software should perform *page-selective-within-PASID* IOTLB invalidation for addresses with each of those page numbers, with Invalidation Hint (IH) value of 0; alternatively, software could use a coarser-grained IOTLB invalidation command (see Invalidation Granularity description in Section 6.5.2.4).

- If software modifies a paging-structure entry that references another paging structure, it may use one of the following approaches depending upon the types and number of translations controlled by the modified entry:

  — Execute *page-selective-within-PASID* IOTLB invalidation command for any addresses with each of the page numbers with translations that would use the entry. These invalidations must specify Invalidation Hint (IH) value of 0 (so that it invalidates the paging-structure caches). However, if no page numbers that would use the entry have translations (e.g., because the P flags are 0 in all entries in the paging structure referenced by the modified entry), it remains necessary to execute the *page-selective-within-PASID* IOTLB invalidation command at least once.

  — Execute *non-globals-within-PASID* (or *non-globals-within-all-PASIDs)* IOTLB invalidation command, if the modified entry controls no global pages.

  — Execute *all-mappings-within-all-PASIDs* IOTLB invalidation command

- If the nature of the paging structures is such that a single entry may be used for multiple purposes (see Section 6.2.7), software should perform invalidations for all of these purposes. For example, if a single entry might serve as both a PDE and PTE, it may be necessary to execute *page-selective-within-PASID* IOTLB invalidation command with two (or more) input-addresses, one that uses the entry as a PDE and one that uses it as a PTE. (Alternatively, software could use *non-globals-within-PASID* or *all-mappings-within-all-PASIDs* IOTLB invalidation granularities)

- As noted in Section 6.2.4, the IOTLB may subsequently contain multiple translations for the address range if software modifies the paging structures so that the page size used for a 4-KByte range of input-addresses changes. A reference to an input-address in the address range may use any of these translations.

  Software wishing to prevent this uncertainty should not write to a paging structure entry in a way that would change, for any input-address, both the page size and either the page frame, access rights, or other attributes. It can instead use the following algorithm: first clear the P flag in the relevant paging-structure entry (e.g., PDE); then invalidate any translations for the affected

input-addresses (see above); and then modify the relevant paging-structure entry to set the P flag and establish modified translation(s) for the new page size.

- When establishing a previously used PASID value for a different process address space, software must execute a *non-globals-within-PASID* IOTLB invalidation command (with Invalidation Hint (IH) value of 0). This ensures invalidation of any information that may have been cached in the IOTLB and paging-structure caches for the previous address space that was associated with the PASID value.

  The use of *non-globals-within-PASID* IOTLB invalidation command assumes that both address spaces use the same global pages and that it is thus not necessary to invalidate any global mapping cached in the IOTLB. If that is not the case, software must include those entries in the invalidation by execution invalidate those entries by executing *all-mappings-within-all-PASIDs* IOTLB Invalidation Command.

6.5.3.4 Optional Invalidation

The following items describe cases in which software may choose not to invalidate the IOTLB when modifying first-level paging entries, and the potential consequences of that choice:

- If a paging-structure entry is modified to change the P flag from 0 to 1, no invalidation is necessary. This is because no IOTLB entry or paging-structure cache entry is created with information from a paging-structure entry in which the P flag is 0[1].
- If a paging-structure entry is modified to change the accessed flag from 0 to 1, no invalidation is necessary (assuming that an invalidation was performed the last time the accessed flag was changed from 1 to 0). This is because no IOTLB entry or paging-structure cache entry is created with information from a paging-structure entry in which the accessed flag is 0.
- If a paging-structure entry is modified to change the R/W flag from 0 to 1, and the entry is used only by requests that can tolerate recoverable translation faults (see Section 7.5), failure to perform an invalidation may result in a recoverable address translation fault detected at the Device-TLB (e.g., in response to an attempted write access), generating a “spurious” Page Request (see Section 7.7). If requests that does not support recoverable page-faults (see Section 7.3) is using such translation, the result is a non-recoverable translation fault (and hence software cannot consider such paging-structure entry modification for optional IOTLB invalidation).
- If SMEP=0 in the extended-context-entry, and a paging-structure entry is modified to change the U/S flag from 0 to 1, and the entry is used only by requests that can tolerate recoverable translation faults (see Section 7.5), failure to perform an invalidation may result in a recoverable address translation fault detected at the Device-TLB (e.g., in response to an attempted user-mode access), generating a “spurious” Page Request (see Section 7.7). If requests that does not support recoverable page-faults (see Section 7.3) is using such translation, the result is a non-recoverable translation fault (and hence software cannot consider such paging-structure entry modification for optional IOTLB invalidation).
- If a paging-structure entry is modified to change the XD flag from 1 to 0, and the entry is used only by requests that can tolerate recoverable translation faults (see Section 7.5), failure to perform an invalidation may result in a recoverable address translation fault detected at the Device-TLB (e.g., in response to an attempted instruction fetch), generating a “spurious” Page Request (see Section 7.7). If requests that does not support recoverable page-faults (see Section 7.3) is using such translation, the result is a non-recoverable translation fault (and hence software cannot consider such paging-structure entry modification for optional IOTLB invalidation).
- If a paging-structure entry is modified to change the accessed flag from 1 to 0, failure to perform an invalidation may result in the hardware not setting that bit in response to a subsequent access to a address whose translation uses the entry. Software cannot interpret the bit being clear as an indication that such an access has not occurred.

1. If it is also the case that no IOTLB invalidation was performed the last time the P flag was changed from 1 to 0, hardware may use a IOTLB entry or paging-structure cache entry that was created when the P flag had earlier been 1.

- If software modifies a PTE or a paging-structure entry in which the PS flag is 1, to change the dirty flag from 1 to 0, failure to perform an invalidation may result in the hardware not setting that bit in response to a subsequent write to a input-address whose translation uses the entry. Software cannot interpret the bit being clear as an indication that such a write has not occurred.

6.5.3.5 Delayed Invalidation

Required invalidations may be delayed under some circumstances with first-level paging. Software developers should understand that, between the modification of a paging-structure entry and execution of the IOTLB invalidation command, the hardware may use translations based on either the old value or the new value of the paging-structure entry. The following items describe some of the potential consequences of delayed invalidation:

- If a paging-structure entry is modified to change the P flag from 1 to 0, an access to an input-address whose translation is controlled by this entry may or may not cause a translation fault.
- If a paging-structure entry is modified to change the R/W flag from 0 to 1, write accesses to input-addresses whose translation is controlled by this entry may or may not cause a translation fault.
- If a paging-structure entry is modified to change the U/S flag from 0 to 1, user-mode accesses to input-addresses whose translation is controlled by this entry may or may not cause a translation fault.
- If a paging-structure entry is modified to change the XD flag from 1 to 0, instruction fetches from input-addresses whose translation is controlled by this entry may or may not cause a translation fault.

In some cases, the consequences of delayed invalidation may not affect software adversely. For example, when freeing a portion of the process address space (by marking paging-structure entries “not present”), IOTLB invalidation command may be delayed if software does not re-allocate that portion of the process address space or the memory that had been associated with it. However, because of speculative execution by devices (or errant software), there may be accesses to the freed portion of the process address space before the invalidations occur. In this case, the following can happen:

- Reads can occur to the freed portion of the process address space. Therefore, invalidation should not be delayed for an address range that has side effects for reads from devices (e.g., mapped to MMIO).
- The hardware may retain entries in the IOTLBs and paging-structure caches for an extended period of time. Software should not assume that the hardware will not use entries associated with a input-address simply because time has passed.
- As noted in Section 6.2.5, the hardware may create an entry in a paging-structure cache even if there are no translations for any input-address that might use that entry. Thus, if software has marked “not present” all entries in the page table, the hardware may subsequently create a PDE-cache entry for the PDE that references that page table (assuming that the PDE itself is marked “present”).
- If software attempts to write to the freed portion of the input-address space, the hardware might not generate a translation fault. (Such an attempt would likely be the result of a software error.) For that reason, the page frames previously associated with the freed portion of the process address space should not be reallocated for another purpose until the appropriate invalidations have been performed.

6.5.4 TLB Shootdown Optimization for Root-Complex Integrated Devices

The process of propagating the changes to a paging-structure entry across multiple agents is common referred to as “TLB shootdown”. In a multi-processor system, Operating Systems (OSs) typically implement optimized algorithms to minimize overhead of TLB shootdowns across processors. With one such optimization, OS tracks what all processors are active in a given virtual address space at any

given point of time. When modifying present paging-entries for a given virtual address space, OS may use the tracking information to optimize shootdown of processors that are not active in that virtual-address space.

For example, if the processor-TLBs are not tagged (i.e., Processor-TLBs cache mappings for at most one virtual-address space at any time), OS may skip shootdown of processors where the virtual address space is currently not active. If the processor-TLBs are tagged (i.e., Processor-TLB may cache mappings for recently run, but currently in-active, virtual address spaces), OS can defer the TLB invalidation on processors where the address-space is not active, until the target virtual address space is scheduled on that processor. These optimizations improve the overheads that are otherwise incurred by always shooting down TLBs on all processors.

With Shared Virtual Memory (SVM) usage (see Section 2.5.1.1), OSs may share first-level translation structures across processors and remapping hardware so as to share an application's virtual address space across CPUs and accelerator devices (such as Intel Processor-Graphics). For performance reasons, such accelerator devices may support an operating mode where applications are allowed to queue work to the device, and a programmable scheduler on the device schedules such queued work, time sharing the device resources across multiple applications. Since the OS is not directly involved in task scheduling on such devices, OS cannot track if a virtual address space is active on the device or not. Therefore, the OS TLB shootdown optimizations that depend on virtual address space tracking cannot be applied for IOTLB and Device-TLB shootdown.

The following section describes extension to remapping hardware to track active virtual address spaces on Root-Complex integrated devices (such as Intel processor-graphics device) that supports SVM, and how OSs may use this tracking information to support the TLB shootdown optimizations.

6.5.4.1 Deferred Invalidation

Tracking of active virtual address spaces on Root-Complex integrated endpoint devices are supported through a PASID state-update request and response protocol between the accelerator device and the remapping hardware. The protocol consists of the following transactions:

- *PASID State-update Request*
  - — PASID state-update request for a virtual address space (PASID) is issued by the endpoint device whenever a task associated with a PASID is scheduled or preempted on the device. The device performs the scheduling or pre-emption operation only after it receives the PASID state-update response from the remapping hardware.
  - — PASID State-update requests specify the following attributes:
    - *Source-ID*:
      - — Identify (bus/device/function) of the device sending the request
    - *PASID Value*:
      - — Identity of the virtual address space (PASID) whose state is updated
    - *Type*:
      - — Indicates the type of the state-update request; If the state-update request is due to scheduling of a task associated with the PASID on the device, the type field indicates value of 1 (active). If the state-update request is due to pre-emption of task associated with the PASID on the device, the type field indicates value of 0 (inactive).
- *PASID State-update Response*
  - — PASID state-update response is issued by the remapping hardware in response to a PASID state-update request.
  - — PASID state-update responses specify the following attributes:
    - *Device-ID*:

  - Identity of the device (bus/device/function) for which the PASID state-update response is targeted. This field contains the value of Source-ID field from the corresponding PASID state-update request.

- *PASID Value*:
  - Identity of the virtual address space (PASID) for this response. This field contains the PASID Value field from the corresponding PASID state-update request.
- *Type*:
  - Indicates the type of the state-update response; This field contains the value of the Type field from the corresponding PASID state-update request.
- *Synchronize Flag*:
  - The Synchronize-flag is derived by the remapping hardware processing of the PASID state-update request. When the Type flag is 1 (active), the Synchronize-flag indicates if the IOTLB and Device-TLB needs to be invalidated (to be consistent with any modifications made to translations structures in memory), before the PASID can transition to active state on the device.
  - If the Synchronize-flag is 1 and the Type field is 1 (active) in a PASID state-update response with Response Code of 0 (Success), an extended IOTLB invalidation with granularity of Non-globals-within-PASID (011b) is performed by the remapping hardware before sending the state-update response to the device. When the device receives a successful PASID state-update response with both Synchronize-flag and Type field as 1, device invalidates all entries in the Device-TLB tagged with the PASID value in the response, before activating the task that led to the PASID state-update response.
- *Response Code*:
  - Value of 0 (Success) in the response code indicates the PASID state-update request was successfully processed by the remapping hardware.
  - Value of 1 (Invalid Request) indicates PASID state-update request processing failed at the remapping hardware. This can be either due to the programming of the extended-context-entry used to process the request (e.g., PASIDE=0, or DINVE=0), or due to invalid PASID value in the request (e.g., PASID value in the request is out of bound for the PASID-table-size specified in the extended-context-entry), or other errors (e.g., hardware access to PASID-state entry resulted in error).

6.5.4.2 PASID-State Table

The PASID state-update requests are processed by the remapping hardware through entries in a PASID-state table. Extended-context-entries enabled to support deferred invalidation (DINVE=1 in the extended-context-entry), hosts the pointer to the PASID-state table (PASIDSTPTR). The size of the PASID-state table matches the side of the PASID-table, and is specified through the PASID table size (PTS) field in the extended-context-entry. Section 9.4 describes the extended-context-entry fields in detail.

The PASID value in the PASID state-update requests are used to index the respective PASID-state-entry in the PASID-state table. Each PASID-state table entry contains an active-reference-count field and a Deferred-Invalidate flag. A non-zero active-reference-count value in a PASID-state entry indicates one or more agents on the device are actively executing tasks on the respective PASID. The Deferred-Invalidate (DINV) flag in a PASID-state entry is used by software to indicate pending invalidation operation for that PASID, and is used by hardware to compose the Synchronize-Flag in the PASID state-update responses. Section 9.6 describes the PASID-state entry field in detail.

6.5.4.3 Remapping Hardware Handling of PASID State-Update Requests

Remapping hardware implementations report support for deferred invalidations through the Deferred-Invalidation-Support field (DIS) in the Extended Capability Register (see Section 10.4.3). When supported, software enables deferred invalidations for a device through the Deferred Invalidation Enable (DINVE) field in the respective extended-context-entry.

PASID state-update requests are processed by remapping hardware as follows:

- The Source-ID field in the request is used to identify the extended-context-entry controlling requests from the device that issued the request.
- If any errors are detected at the extended-context-entry used to process the request, or if deferred invalidations are explicitly disabled (DINVE=0 in the extended-context-entry), or, if the PASID value in the request is out-of-bound for the PASID table size in the extended-context-entry, a PASID state-update response is returned with Response Code of Invalid Request.
- If above errors are not detected, the PASID value in the request is used to locate the PASID-state entry in the PASID-state table.
- Perform an atomic read-modify-write of the PASID-state entry as follows:
  - Read the PASID-state entry, locking the cache-line hosting the PASID-state entry
  - If the Type field in the request is 1 (Active), increment the Active-Refcount field and clear the Deferred-Invalidate field.
  - Else if the Type field in the request is 0 (Inactive), decrement the Active-Refcount field.
  - Unlock the cache-line
- If the Type field in the request is 1 (Active), and, the Deferred-Invalidate field read as part of above read-modify-write operation is 1 (i.e., there is a pending invalidate operation for this PASID), perform an extended IOTLB invalidation with granularity of non-globals-within-PASID (see Section 6.5.2.4) and set the Synchronize-flag in the PASID state-update response to 1.
- Else if the Type field in the request is 0 (Inactive), or, if the Deferred-Invalidate field read as part of above read-modify-write operation is 0, clear the Synchronize-flag in the PASID state-update response to 0.
- Send the PASID state-update response to the device with Response Code of Success (0), and the Synchronize-flag computed as above.

6.5.4.4 Root-Complex Integrated Device Handling of PASID State-Update Responses

Root-Complex integrated endpoint device support for deferred invalidations are reported by respective endpoint device driver through OS specific software interfaces. One example of such Root-Complex integrated device is Intel® Processor Graphics device.

A PASID state-update request for a PASID is sent by the endpoint device whenever a task that operates within the PASID is scheduled or preempted on any agent on the device. The scheduling or pre-emption operation is completed only after the respective PASID state-response is received.

If the PASID state-response is received with a non-successful Response Code, the tasks operating within that PASID are terminated by the device and the associated device context is reported as in error through the device-specific driver.

If the PASID state-update response received has Response Code of Success (0), Type field value of 1 (Active), and Synchronize-flag value of 1, the device performs an extended Device-TLB invalidation for all translations cached for this PASID (i.e., an invalidation operation equivalent to receiving an extended Device-TLB invalidation request with Global (G) field value of 0, Size (S) field value of 1, and Address (ADDR) bits 62:12 are all 1's and address bit 63 is 0; See Section 6.5.2.6 for extended Device-TLB invalidations). The scheduling operation on the device that resulted in the PASID state-update request/response that led to the Device-TLB invalidation, is completed only after the Device-TLB invalidation is completed.

6.5.4.5 Ordering of PASID State-Update Requests and Responses

The PASID state-update request and response messages follow PCI-Express posted ordering rules. Specifically, it cannot pass other posted requests (such as memory writes, Device-TLB invalidation messages, and PASID state-update messages) ahead or it.

The Root-Complex integrated devices follows the below ordering rules at the device:

- Any time a scheduling or pre-emption operation is initiated for tasks within a PASID on any agent on the device, a PASID state-update request must be issued, and the scheduling or pre-emption operation can be completed only after receiving the PASID state-update response.
- PASID state-update requests received are processed in the order at which they are received.
- On pre-emption operation on an agent on the device, the device ensures there are no pending non-posted requests for that agent (i.e., it has received completions for all pending non-posted requests from that agent - including memory read requests, or translation-requests) before issuing the PASID state-update request.
- If the device supports non-centralized Device-TLBs (i.e., independent Device-TLB for each processing element on the device), any extended Device-TLB invalidations (explicitly requested through extended Device-TLB invalidation requests, or performed implicitly on PASID state-update response as described in Section 6.5.4.5), must apply to all Device-TLBs on the device.

6.5.4.6 Example TLB Shootdown using Deferred Invalidations

Software usages such as Shared Virtual Memory (SVM) described in Section 2.5.1.1, may require sharing of the first-level translation structures across CPUs and SVM-capable devices. For such usages, when software modifies the shared paging entries, any resulting TLB shootdown (that currently only invalidates TLBs on CPU agents) needs to also invalidate IOTLB in the remapping hardware and Device-TLBs on the device.

This section provides an example TLB shootdown software flow using the Deferred Invalidation support. The example flow is simplified for clarity, and is provided for illustration purposes only.

- If paging entries modified maps to global pages (i.e., not specific to a PASID), software explicitly invalidates TLBs on CPU agents (using TLB shootdown IPIs), and IOTLB and Device-TLBs (using extended IOTLB invalidate and extended Device-TLB invalidate commands through the remapping hardware invalidation-queue).
- Else if the paging entries modified maps to non-global pages, software may perform the steps:
  - Read the PASID-state entry (quad-word) for the PASID whose non-global page mappings were modified.
  - If the Active-Refcount field in the PASID-state entry is non-zero (indicating PASID is active on one or more agents on the device), explicitly invalidate the IOTLB and Device-TLB (using extended IOTLB invalidate and extended Device-TLB invalidate commands through the remapping hardware invalidation-queue).
  - Else, if the Active-Refcount field in the PASID-state entry is 0 (indicating PASID is not active on any agents on the device), perform an atomic compare-exchange (LOCK CMPXCHG8) to set the Deferred-Invalidate field in the PASID-state entry to 1. If the compare-exchange fails (indicating the PASID-state entry was modified by remapping hardware as part of processing a PASID state-update request, between the time software read the PASID-state entry and performed the locked compare-exchange), redo the above steps.

6.5.5 Draining of Requests to Memory

Requests from devices that are already processed by the remapping hardware, but queued within the Root-Complex to be completed to memory are referred as non-committed requests. Draining refers to hardware pushing (committing) these requests to the global ordering point. Hardware implementations report support for draining through the Capability Registers.

A write request to system memory is considered drained when the effects of the write are visible to processor accesses to addresses targeted by the write request. A read request to system memory is considered drained when the Root-Complex has finished fetching all of its read response data from memory.

Requirements for draining are described below:

- Draining applies only to requests to memory and do not guarantee draining of requests to peer destinations.
- Draining applies only for untranslated requests (AT=00b), including those processed as pass-through by the remapping hardware.
- Draining of translated requests (AT=10b) requires issuing a Device-TLB invalidation command to the endpoint device. Endpoint devices supporting Address Translation Services (ATS) are required to wait for pending translated read responses (or keep track of pending translated read requests and discard their read responses when they arrive) before issuing the ATS invalidation completion message. This effectively guarantees draining of translated read requests. The ATS invalidation completion message is issued on the posted channel and pushes all writes from the device (including any translated writes) ahead of it. To ensure proper write draining of translated requests, remapping hardware must process ATS invalidation completion messages per the PCI-Express ordering rules (i.e., after processing all posted requests ahead of it).
- Read and write draining of untranslated requests are required when remapping hardware status changes from disabled to enabled. The draining must be completed before hardware sets the TES field in Global Status Register (which indicates remapping hardware is enabled). Hardware implementations may perform draining of untranslated requests when remapping hardware status changes from enabled to disabled.
- Read and write draining of untranslated requests are performed on IOTLB invalidation requests specifying Drain Read (DR) and Drain Write (DW) flags respectively. For IOTLB invalidations submitted through the IOTLB Invalidate Register (IOTLB_REG), draining must be completed before hardware clears the IVT field in the register (which indicates invalidation completed). For IOTLB invalidations submitted through the queued invalidation interface, draining must be completed before the next Invalidation Wait Descriptor (*inv_wait_dsc*) is completed by hardware.
  - For global IOTLB invalidation requests specifying DMA read/write draining, all non-committed DMA read/write requests queued within the Root-Complex are drained.
  - For domain-selective IOTLB invalidation requests specifying read/write draining, hardware only guarantees draining of non-committed read/write requests to the domain specified in the invalidation request.
  - For page-selective IOTLB invalidation requests specifying read/write draining, hardware only guarantees draining of non-committed read/write requests with untranslated address overlapping the address range specified in the invalidation request and to the specified domain.
- Read and write draining of untranslated requests are performed on all extended IOTLB invalidation requests, where draining is completed before the next Invalidation Wait Descriptor (*inv_wait_dsc*) is completed by hardware.

6.5.6 Interrupt Draining

Interrupt requests that are already processed by the remapping hardware, but queued within the Root-Complex to be completed are referred as non-committed interrupt requests. Interrupt draining refers to hardware pushing (committing) these interrupt requests to the appropriate processor's interrupt controller (Local xAPIC). An interrupt request is considered drained when the interrupt is accepted by the processor Local xAPIC (for fixed and lowest priority delivery mode interrupts this means the interrupt is at least recorded in the Local xAPIC Interrupt Request Register (IRR)).

Requirements for interrupt draining are described below:

- Interrupt draining applies to all non-committed interrupt requests, except Compatibility format interrupt requests processed as pass-through on Intel® 64 platforms.

- Interrupt draining is required when interrupt-remapping hardware status changes from disabled to enabled. The draining must be completed before hardware sets the IES field in Global Status Register (indicating interrupt-remapping hardware is enabled). Hardware implementations may perform interrupt draining when interrupt-remapping hardware status changes from enabled to disabled.
- Interrupt draining is performed on Interrupt Entry Cache (IEC) invalidation requests. For IEC invalidations submitted through the queued invalidation interface, interrupt draining must be completed before the next Invalidation Wait Descriptor is completed by hardware.
  - For global IEC invalidation requests, all non-committed interrupt requests queued within the Root-Complex are drained.
  - For index-selective IEC invalidation requests, hardware only guarantees draining of non-committed interrupt requests referencing interrupt indexes specified in the invalidation request.
- The Root-Complex considers an interrupt request as drained when it receives acknowledgement from the processor complex. Interrupt draining requires processor complex to ensure the interrupt request received is accepted by the Local xAPIC (for fixed interrupts, at least recorded in the IRR) before acknowledging the request to the Root-Complex.

6.6 Set Root Table Pointer Operation

Software must always perform a Set Root-Table Pointer operation before enabling or re-enabling (after disabling) remapping hardware.

On a root-table pointer set operation, software must perform an ordered global invalidate of the context-cache, PASID-cache (if applicable), and IOTLB to ensure hardware references only the new structures for further remapping.

If software sets the root-table pointer while remapping hardware is active (TES=1 in Global Status register), software must ensure the structures referenced by the new root-table pointer provide identical remapping results as the structures referenced by the previous root-table pointer so that in-flight requests are properly translated. This is required since hardware may utilize the cached old paging structure entries or the new paging structure entries in memory to translate in-flight requests, until the Context -cache, PASID-cache, and IOTLB invalidations are completed. Software must not modify the Root-Table-Type (RTT) field in the Root-table Address register (i.e., switch from using root/context entries to extended root/context entries), while remapping hardware is active (TES=1 in Global Status register).

6.7 Set Interrupt Remapping Table Pointer Operation

Software must always set the interrupt-remapping table pointer before enabling or re-enabling (after disabling) interrupt-remapping hardware.

Software must always follow the interrupt-remapping table pointer set operation with a global invalidate of the IEC to ensure hardware references the new structures before enabling interrupt remapping.

If software updates the interrupt-remapping table pointer while interrupt-remap hardware is active, software must ensure the structures referenced by the new interrupt-remapping table pointer provide identical remapping results as the structures referenced by the previous interrupt-remapping table pointer to ensure any valid in-flight interrupt requests are properly remapped. This is required since hardware may utilize the old structures or the new structures to remap in-flight interrupt requests, until the IEC invalidation is completed.

6.8 Write Buffer Flushing

On remapping hardware page-table walk, earlier implementations of this architecture did not flush or snoop the write buffers at the memory controller that buffers writes to DRAM, and required explicit software flushing of these write buffers on paging structure modifications. These earlier hardware implementations reported this restriction to software by reporting the Required Write Buffer Flushing (RWBF) field in the Capability Register to 1.

For such hardware implementations requiring write buffer flushing (RWBF=1 in the Capability register), software updates to memory-resident remapping structures may be held in Root-Complex internal hardware write-buffers, and not implicitly visible to remapping hardware. For such implementations, software must explicitly make these updates visible to hardware through one of two methods below:

- For updates to remapping hardware structures that require context-cache, PASID-cache, IOTLB or IEC invalidation operations to flush stale entries from the hardware caches, no additional action is required to make the modifications visible to hardware. This is because, hardware performs an implicit write-buffer-flushing as a pre-condition to context-cache, PASID-cache, IOTLB and IEC invalidation operations.
- For updates to remapping hardware structures (such as modifying a currently not-present entry) that do not require context-cache, PASID-cache, IOTLB or IEC invalidations, software must explicitly perform write-buffer-flushing to ensure the updated structures are visible to hardware.

Newer hardware implementations are expected to NOT require explicit software flushing of write buffers and report RWBF=0 in the Capability register.

6.9 Hardware Register Programming Considerations

A register used to submit a command to a remapping unit is owned by hardware while the command is pending in hardware. Software must not update the associated register until hardware indicates the command processing is complete through appropriate status registers.

For each remapping hardware unit, software must serialize commands submitted through the Global Command register, Context Command register, IOTLB registers and Protected Memory Enable register.

For platforms supporting more than one remapping hardware unit, there are no hardware serialization requirements for operations across remapping hardware units.

6.10 Sharing Remapping Structures Across Hardware Units

Software may share[1] (fully or partially) the various remapping structures across multiple remapping hardware units. When the remapping structures are shared across hardware units, software must explicitly perform the invalidation operations on each remapping hardware unit sharing the modified entries. The software requirements described in this section must be individually applied for each such invalidation operation.

1. Sharing of Extended Root and Extended Context tables across remapping hardware units are possible only across remapping hardware units that report Extended Context Support (ECS) field as Set in the Extended Capability register.

This Page Is Left Intentionally Blank

7 Translation Faults

This chapter describes the hardware handling of translation faults. Translation faults are broadly categorized as follows:

- ***Interrupt Translation Faults****:* Faults detected when remapping interrupt requests are categorized as Interrupt translation faults. Interrupt translation faults are non-recoverable. *Section 7.1* describes interrupt translation fault conditions in detail.
- ***Address Translation Faults****:* Faults detected when remapping memory requests (or translation requests from Device-TLBs) are referred to as address translation faults. *Section 7.2* describes address translation fault conditions in detail.

7.1 Interrupt Translation Faults

The following table enumerates the various interrupt translation fault conditions. An interrupt translation fault condition is treated as 'qualified' if the fault is reported to software only when the Fault Processing Disable (FPD) field is 0 in the Interrupt-Remap-Table-Entry (IRTE) used to process the faulting interrupt request. Interrupt translation faults are non-recoverable and faulting interrupt request is treated as Unsupported Request by the remapping hardware.

Table 10. Interrupt Remapping Fault Conditions

Interrupt Remapping Fault Conditions	Fault Reason	Qualified	Behavior
Decoding of the interrupt request per the Remappable request format detected one or more reserved fields as Set.	20h	No	Unsupported Request
The interrupt_index value computed for the Remappable interrupt request is greater than the maximum allowed for the interrupt-remapping table size configured by software.	21h	No	
The Present (P) field in the IRTE corresponding to the interrupt_index of the interrupt request is Clear.	22h	Yes	
Hardware attempt to access the interrupt-remapping table through the Interrupt-Remapping Table Address (IRTA) field in the Interrupt Remap Table Address Register resulted in error.	23h	No	
Hardware detected one ore more reserved fields that are not initialized to zero in an IRTE with Present (P) field Set.	24h	Yes	
On Intel® 64 platforms, hardware blocked an interrupt request in Compatibility format either due to Extended Interrupt Mode Enabled (EIME field Set in Interrupt Remapping Table Address Register) or Compatibility format interrupts disabled (CFIS field Clear in Global Status Register).	25h	No	
Hardware blocked a Remappable interrupt request due to verification failure of the interrupt requester's source-id per the programming of SID, SVT and SQ fields in the corresponding IRTE with Present (P) field Set.	26h	Yes	

7.2 Address Translation Faults

Address translation faults are classified as follows:

- ***Non-recoverable Faults***: Requests that encounter non-recoverable address translation faults are aborted by the remapping hardware, and typically require a reset of the device (such as through a function-level-reset) to recover and re-initialize the device to put it back into service. Section 7.2.1 describe the non-recoverable fault conditions in detail.
- ***Recoverable Faults***: Requests that encounter recoverable address translation faults can be retried by the requesting device after the condition causing the recoverable fault is handled by software. Recoverable translation faults are detected at the Device-TLB on the device and require the device to support Address Translation Services (ATS) capability. Refer to the PCI-Express ATS specification for details. Section 7.2.2 describe recoverable fault conditions in detail.

7.2.1 Non-Recoverable Address Translation Faults

Non-recoverable address translation faults can be detected by remapping hardware for requests-without-PASID or for requests-with-PASID. A non-recoverable fault condition is considered 'qualified' if it is reported to software only if the Fault Processing Disable (FPD) field in the context-entry (or extended-context-entry) used to process the faulting request is 0. Memory requests that result in non-recoverable address translation faults are blocked by hardware. The exact method for blocking such requests are implementation-specific. For example:

- Faulting write requests may be handled in much the same way as hardware handles write requests to non-existent memory. For example, the write request is discarded in a manner convenient for implementations (such as by dropping the cycle, completing the write request to memory with all byte enables masked off, re-directing to a catch-all memory location, etc.).
- Faulting read requests may be handled in much the same way as hardware handles read requests to non-existent memory. For example, the request may be redirected to a catch-all memory location, returned as all 0's or 1's in a manner convenient to the implementation, or the request may be completed with an explicit error indication (recommended). For faulting read requests from PCI-Express devices, hardware indicates "Unsupported Request" (UR) in the completion status field of the PCI-Express read completion.

7.2.1.1 Non-Recoverable Faults for Untranslated Requests Without PASID

Table 11 enumerates the non-recoverable address translation fault conditions that can be encountered when processing requests-without-PASID specifying an untranslated address (Address Type (AT) field value of 00b in the transaction header).

Table 11. Non-Recoverable Faults for Untranslated Requests Without PASID

Fault Conditions	Fault Reason	Qualified	Behavior
The Present (P) field in root-entry (or UP/LP fields in extended-root-entry) used to process the untranslated request without PASID is 0.	1h	No	Unsupported Request
The Present (P) field in the context-entry (or extended-context-entry) used to process the untranslated request without PASID is 0.	2h	Yes	
Invalid programming of a context-entry (or extended-context-entry) used to process an untranslated request without PASID. For example: • The Address-Width (AW) field is programmed with a value not supported by hardware or inconsistent with Translation-Type (T) field. • The Translation-Type (T) field is programmed to indicate a translation type not supported by the hardware implementation. • Hardware attempt to access the second-level paging entry referenced through the SLPTPTR field resulted in error.	3h		

Table 11. Non-Recoverable Faults for Untranslated Requests Without PASID

Fault Conditions	Fault Reason	Qualified	Behavior
Input-address in the untranslated request without PASID is above (2^X - 1), where X is the minimum of MGAW reported in Capability Register and value in the Address-Width (AW) field of context-entry (or extended-context-entry) used to process the request.	4h	Yes	Unsupported Request
An untranslated Write (or AtomicOp) request without PASID is blocked due to lack of write permission. Refer to Section 3.7.2 for access rights checking with second-level translation for requests-without-PASID.	5h		
An untranslated Read (or AtomicOp) request without PASID is blocked due to lack of read permission. Refer to Section 3.7.2 for access rights checking with second-level translation for requests-without-PASID. For implementations reporting ZLR field as 1 in the Capability Register, this fault condition is not applicable for zero-length read requests without PASID to write-only mapped pages.	6h		
Hardware attempt to access a second-level paging entry (SL-PML4E, SL-PDPE, SL-PDE, or SL-PTE) referenced through the Address (ADDR) field in a preceding second-level paging entry (SL-PML5E, SL-PML4E, SL-PDPE, SL-PDE) resulted in error.	7h		
Hardware attempt to access a root-entry (or extended-root-entry) referenced through the Root-Table Address (RTA) field in the Root-entry Table Address Register resulted in error.	8h	No	
Hardware attempt to access a context-entry (or extended-context-entry) referenced through the CTP field in a root-entry (or UCTP/LCTP field in an extended-root-entry) resulted in error.	9h		
Non-zero reserved field in a root-entry with Present (P) field Set (or an extended-root-entry with UP/LP field Set).	Ah		
Non-zero reserved field in a context-entry (or extended-context-entry) with Present (P) field Set.	Bh	Yes	
Non-zero reserved field in a second-level paging entry (SL-PML5E, SL-PML4E, SL-PDPE, SL-PDE, or SL-PTE) with at least one of the Read (R) or Write (W) fields Set.	Ch		

7.2.1.2 Non-Recoverable Faults for Untranslated Requests With PASID

Table 12 enumerates the non-recoverable address translation fault conditions that can be encountered when processing requests-with-PASID specifying an untranslated address (Address Type (AT) field value of 00b in the transaction header).

Table 12. Non-Recoverable Faults for Untranslated Requests With PASID

Fault Conditions	Fault Reason	Qualified	Behavior
The present (UP/LP) field in the extended root-entry used to process the untranslated request with PASID is 0.	1h	No	Unsupported Request
The Present (P) field in the extended-context-entry used to process the untranslated request with PASID is 0.	2h	Yes	

Table 12. Non-Recoverable Faults for Untranslated Requests With PASID

Fault Conditions	Fault Reason	Qualified	Behavior
Invalid programming of the extended-context-entry used to process an untranslated request with PASID. For example: • The Address width (AW) field is programmed with a value either not supported by hardware or inconsistent with Translation-Type (T) field. • The Translation-Type (T) field is programmed to indicate a translation type not supported by the hardware implementation. • When nested translation is enabled in the extended-context-entry, hardware attempt to access the second-level paging entry referenced through the SLPTPTR field resulted in error. • Hardware attempt to access the PASID-entry referenced through the PASIDTPTR field referenced an address above HAW or resulted in hardware error.	3h	Yes	Unsupported Request
When nested translation is enabled, the intermediate-address presented to nested second-level translation is beyond (2^X - 1), where X is the minimum of the MGAW reported in the Capability Register and the value in the Address-Width (AW) field of the extended-context-entry used to process the request. The intermediate-address can be the intermediate-address of a PASID-entry, intermediate-address of a first-level paging entry (FL-PML5E, FL-PML4E, FL-PDPE, FL-PDE, or FL-PTE), or intermediate-address of the page frame.	4h		
An untranslated Write (or AtomicOp) request with PASID is blocked due to lack of write permission. For nested translations the lack of write permission can be at first-level translation or at the nested second-level translation. Also, for nested translations, this fault condition can be encountered on nested second-level translation performed for setting the Accessed/Dirty flag of a first-level paging entry. Refer to Section 3.6.2 for access rights checking with first-level translation, and Section 3.8.2 for access rights checking with nested translation.	5h		
An untranslated Read (or AtomicOp) request with PASID is blocked due to lack of read permission. For nested translations the lack of read permission can be at first-level translation or at the nested second-level translation. Also, for nested translations, this fault condition can be encountered on the nested second-level translation performed to access a first-level paging entry. Refer to Section 3.6.2 for access rights checking with first-level translation, and Section 3.8.2 for access rights checking with nested translation. For implementations reporting ZLR field as 1 in the Capability Register, this fault condition is not applicable for zero-length read requests with PASID to write-only mapped pages in second-level translation.	6h		
Hardware attempt to access a second-level paging entry (SL-PML4E, SL-PDPE, SL-PDE, or SL-PTE) referenced through the Address (ADDR) field in a preceding second-level paging entry (SL-PML5E, SL-PML4E, SL-PDPE, SL-PDE) resulted in error.	7h		
Hardware attempt to access an extended-root-entry referenced through the Root-Table Address (RTA) field in the Root-entry Table Address Register resulted in error.	8h	No	
Hardware attempt to access an extended-context-entry referenced through the UCTP/LCTP field in an extended-root-entry resulted in error.	9h		
Non-zero reserved field in lower 64-bits of the extended-root-entry with LP field Set, and/or non-zero reserved fields in the upper 64-bits of the extended-root-entry with UP field Set.	Ah		

Table 12. Non-Recoverable Faults for Untranslated Requests With PASID

Fault Conditions	Fault Reason	Qualified	Behavior
Non-zero reserved field in extended-context-entry with Present (P) field Set.	Bh	Yes	Unsupported Request
Non-zero reserved field in a second-level paging entry (SL-PML5E, SL-PML4E, SL-PDPE, SL-PDE, or SL-PTE) with at least one of the Read (R), Execute (X), or Write (W) fields Set. Execute (X) field in second-level paging entries are applicable only when supported by hardware and enabled by software.	Ch		
Translation Type (T) field in present extended-context-entry (E.g., 010b) specifies blocking of untranslated requests with PASID.	Dh		
The PASID Enable (PASIDE) field in extended-context-entry (with P=1) used to process the untranslated request with PASID is 0.	10h		
The PASID value in the untranslated request with PASID is larger than the maximum PASID-value supported by the PASID-Table-Size (PTS) field in the extended-context-entry (with P=PASIDE=1) used to process the request.	11h		
The Present (P) field in the PASID-entry used to process the untranslated request with PASID is 0.	12h		
Non-zero reserved field in a PASID-entry with the Present (P) field Set.	13h		
Input-address in the request with PASID is not Canonical (i.e., address bits 63:N not same value as address bit [N-1], where N is 48-bits with 4-level paging and 57-bits with 5-level paging).	14h		
Hardware attempt to access the FL-PML4 (FL-PML5 with 5-level paging) entry referenced through the FLPTPTR field in the PASID-entry resulted in error.	15h		
Non-zero reserved field in first-level paging entry (FL-PML5E, FL-PML4E, FL-PDPE, FL-PDE, or FL-PTE) with Present (P) field Set.	16h		
Hardware attempt to access a first-level paging entry (FL-PML4E, FL-PDPE, FL-PDE, or FL-PTE) referenced through the Address (ADDR) field in a preceding first-level paging entry (FL-PML5E, FL-PML4E, FL-PDPE, or FL-PDE) resulted in error. (For nested translations, second-level nested translation faults encountered when accessing first-level paging entries are treated as fault conditions 4h, 5h or 6h. See description of these fault conditions above).	17h		
An untranslated request with PASID with Execute-Requested (ER) field Set is blocked due to lack of execute permission. For nested translations, the lack of execute permission can be at first-level translation, or at the second-level translation for the final page. Refer to Section 3.6.2 for access rights checking with first-level translation, and Section 3.8.2 for access rights checking with nested translation.	18h		
The Execute Requests Enable (ERE) field is 0 in extended-context-entry (with P=1) used to process the request with PASID with Execute-Requested (ER) field Set.	19h		
The Supervisor Requests Enable (SRE) field is 0 in PASID-entry (with P=1) used to process the request-with-PASID with Privileged-mode-Requested (PR) field Set.	1Ah		
Root Table Type (RTT) field is 0 in Root-table Address register (RTADDR_REG) used to process the request with PASID.	1Bh	No	
Privilege checks by first-level translation blocked an untranslated request with PASID with user-request privilege	1Ch	Yes	

7.2.1.3 Non-Recoverable Faults for Translation Requests Without PASID

Table 13 enumerates the non-recoverable address translation fault conditions that can be encountered when processing translation-requests (Address Type (AT) field value of 01b in the transaction header) without PASID.

Table 13. Non-Recoverable Faults For Translation Requests Without PASID

Fault Conditions	Fault Reason	Qualified	Translation Completion Status
Conditions that explicitly block translation requests without PASID:			
The Present (P) field in root-entry (or UP/LP fields in extended root-entry) used to process the translation request without PASID is 0.	1h	No	Unsupported Request
The Present (P) field in the context-entry (or extended-context-entry) used to process the translation request without PASID is 0.	2h	Yes	
The present context-entry (or extended-context-entry) specifies blocking of translation requests without PASID. (i.e., Translation Type (T) field value not equal to 01b in context-entry, or Translation Type (T) field value not equal to 001b in extended-context-entry).	Dh		
Hardware or programming error conditions that block translation requests without PASID:			
Invalid programming of a context-entry (or extended-context-entry) used to process an translation request without PASID. For example: • The Address width (AW) field is programmed with a value either not supported by hardware or inconsistent with Translation-Type (T) field. • The Translation-Type (T) field is programmed to indicate a translation type not supported by the hardware implementation. • Hardware attempt to access the second-level paging entry referenced through the SLPTPTR field resulted in error.	3h	Yes	Completer Abort
Hardware attempt to access a second-level paging entry (SL-PML4E, SL-PDPE, SL-PDE, or SL-PTE) referenced through the Address (ADDR) field in a preceding second-level paging entry (SL-PML5E, SL-PML4E, SL-PDPE, SL-PDE) resulted in error.	7h		
Hardware attempt to access a root-entry (or extended-root-entry) referenced through the Root-Table Address (RTA) field in the Root-entry Table Address Register resulted in error.	8h	No	
Hardware attempt to access a context-entry (or extended-context-entry) referenced through the CTP field in a root-entry (or UCTP/LCTP field in an extended-root-entry) resulted in error.	9h		
Non-zero reserved field in a root-entry with Present (P) field Set (or an extended-root-entry with UP/LP field Set).	Ah		
Non-zero reserved field in a context-entry (or extended-context-entry) with Present (P) field Set.	Bh	Yes	
Non-zero reserved field in a second-level paging entry (SL-PML5E, SL-PML4E, SL-PDPE, SL-PDE, or SL-PTE) with at least one of the Read (R) or Write (W) permission fields Set.	Ch		

7.2.1.4 Non-Recoverable Faults for Translation Requests With PASID

Table 14 enumerates the non-recoverable address translation fault conditions that can be encountered when processing translation-requests (Address Type (AT) field value of 01b in the transaction header) with PASID.

Table 14. Non-Recoverable Faults For Translation Requests With PASID

Fault Conditions	Fault Reason	Qualified	Translation Completion Status
Conditions that explicitly block translation requests with PASID:			
The present (UP/LP) field in the extended root-entry used to process the translation request with PASID is 0.	1h	No	Unsupported Request
Root Table Type (RTT) field is 0 in Root-table Address register (RTADDR_REG) used to process the request with PASID.	1Bh		
The Present (P) field in the extended-context-entry used to process the translation request with PASID is 0.	2h	Yes	
The present extended-context-entry specifies blocking of translation requests with PASID. i.e., Translation Type (T) field value not equal to 001b or 101b in extended-context-entry.	Dh		
The PASID-Enable (PASIDE) field in extended-context-entry (with P=1) used to process the translation request with PASID is 0.	10h		
The PASID value in the translation request with PASID is larger than the maximum PASID-value supported by the PASID-Table-Size (PTS) field in the extended-context-entry (with P=PASIDE=1) used to process the request.	11h		
Hardware or programming error conditions that block translation requests with PASID:			
Invalid programming of the extended-context-entry used to process a translation request with PASID. For example: • The Address-Width (AW) field is programmed with a value not supported by hardware or inconsistent with Translation-Type (T) field. • The Translation-Type (T) field is programmed to indicate a translation type not supported by the hardware implementation. • When nested translation is enabled in the extended-context-entry, hardware attempt to access the second-level paging entry referenced through the SLPTPTR field resulted in error. • Hardware attempt to access the PASID-entry referenced through the PASIDTPTR field referenced address above HAW resulting in error.	3h	Yes	Completer Abort
Hardware attempt to access a second-level paging entry (SL-PDPE, SL-PDE, or SL-PTE) referenced through Address (ADDR) field in a preceding second-level paging entry resulted in error.	7h		
Hardware attempt to access an extended-root-entry referenced through the Root-Table Address (RTA) field in the Root-entry Table Address Register resulted in error.	8h	No	
Hardware attempt to access an extended-context-entry referenced through the UCTP/LCTP field in an extended-root-entry resulted in error.	9h		
Non-zero reserved field in lower 64-bits of the extended-root-entry with LP field Set, and/or non-zero reserved fields in the upper 64-bits of the extended-root-entry with UP field Set.	Ah		
Non-zero reserved field in extended-context-entry with P=1	Bh	Yes	
Non-zero reserved field in a second-level paging entry (SL-PML5E, SL-PML4E, SL-PDPE, SL-PDE, or SL-PTE) with at least one of the Read (R), Execute (X), or Write (W) permission fields Set. Execute (X) field in second-level paging entries are applicable only when supported by hardware and enabled by software.	Ch		
Non-zero reserved field in PASID-entry with Present (P) field Set.	13h		
Hardware attempt to access the FL-PML4 (FL-PML5 with 5-level paging) entry referenced through the FLPTPTR field in the PASID-entry resulted in error.	15h		
Non-zero reserved field in first-level paging entry (FL-PML5E, FL-PML4E, FL-PDPE, FL-PDE, or FL-PTE) with Present (P) field Set.	16h		
Hardware attempt to access a first-level paging entry (FL-PDPE, FL-PDE, or FL-PTE) referenced through the Address (ADDR) field in a preceding first-level paging entry resulted in error.	17h		

7.2.1.5 Non-Recoverable Faults for Translated Requests

Table 15 enumerates the non-recoverable address translation fault conditions that can be encountered when processing translated-requests (Address Type (AT) field value of 10b in the transaction header). Translated requests specify the translated (host-physical) address and hence has no PASID.

Table 15. Non-Recoverable Faults For Translated Requests

Unsuccessful Translated Request Conditions	Fault Reason	Qualified	Behavior
The Present (P) field in root-entry (or UP/LP fields in extended root-entry) used to process the translated request is 0.	1h	No	Unsupported Request
The Present (P) field in the context-entry (or extended-context-entry) used to process the translated request is 0.	2h	Yes	
Invalid programming of a context-entry (or extended-context-entry) used to process a translated request. For example: • The Address-Width (AW) field is programmed with a value not supported by hardware or inconsistent with Translation-Type (T) field. • The Translation-Type (T) field is programmed to indicate a translation type not supported by the hardware implementation.	3h		
Hardware attempt to access a root-entry (or extended-root-entry) referenced through the Root-Table Address (RTA) field in the Root-entry Table Address Register resulted in error.	8h	No	
Hardware attempt to access a context-entry (or extended-context-entry) referenced through the CTP field in a root-entry (or UCTP/LCTP field in an extended-root-entry) resulted in error.	9h		
Non-zero reserved field in a root-entry with Present (P) field Set (or an extended-root-entry with UP/LP field Set).	Ah		
Non-zero reserved field in a context-entry (or extended-context-entry) with Present (P) field Set.	Bh	Yes	
The present context-entry (or extended-context-entry) specifies blocking of translation requests without PASID. (i.e., Translation Type (T) field value not equal to 01b in context-entry, or Translation Type (T) field value not equal to 001b or 101b in extended-context-entry).	Dh		

7.2.2 Recoverable Address Translation Faults

Devices supporting Device-TLBs can support recoverable address translation faults for translations obtained by the Device-TLB (by issuing a Translation request to the remapping hardware, and receiving a Translation Completion with Successful response code). What device accesses can tolerate and recover from Device-TLB detected faults and what device accesses cannot tolerate Device-TLB detected faults is specific to the device. Device-specific software (e.g., driver) is expected to make sure translations with appropriate permissions and privileges are present before initiating device accesses that cannot tolerate faults. Device operations that can recover from such Device-TLB faults typically involves two steps:

- Report the recoverable fault to host software; This may be done in a device-specific manner (e.g., through the device-specific driver), or if the device supports PCI-Express Page Request Services (PRS) Capability, by issuing a page-request message to the remapping hardware. Section 7.5 describe the page-request interface through the remapping hardware.
- After the recoverable fault is serviced by software, the device operation that originally resulted in the recoverable fault may be replayed, in a device-specific manner.

Table 16 enumerates the recoverable address translation fault conditions detected by the remapping hardware when processing translation-requests. These fault conditions are not reported by the remapping hardware as non-recoverable faults, and instead, result in sending a successful translation completion for the faulting translation request with limited or no permission/privileges. When such a translation completion is received by the Device-TLB, a translation fault is detected at the Device-TLB, and handled as either recoverable or non-recoverable, depending on the device operation using the returned translation.

Table 16. Recoverable Fault Conditions For Translation Requests

Successful Translation Request Conditions	Translation Completion Data Entry
Hardware detected address in the translation request is to the interrupt address range (0xFEEx_xxxx). The special handling to interrupt address range is to comprehend potential endpoint device behavior of issuing translation requests to all of its memory transactions including its message signalled interrupt (MSI) posted writes.	R=0, W=1, U=1, S=0
Conditions where hardware could not find a translation for address specified in translation request without PASID, or the requested translation lacked both read and write permissions. • When performing second-level translation for translation request without PASID, or when performing any second-level step of nested translation for translation-request-with-PASID, hardware detected input address above the minimum of MGAW (reported in Capability Register) and (2X - 1), where X is the AGAW corresponding to address-width programmed in context-entry or extended-context-entry used to process the request. • The address in the translation request with PASID is not canonical (i.e., address bits 63:N not same value as address bit [N-1], where N is 48-bits with 4-level paging and 57-bits with 5-level paging). • When performing second-level translation for translation-request-without-PASID, or when performing any second-level step of nested translation for translation-request-with-PASID, hardware found a not-present (R=W=0) second-level-paging-entry (SL-PML5E, SL-PML4E, SL-PDPE, SL-PDE, or SL-PTE), and hence could not complete the page-walk. • Hardware detected that the logical-AND of the Read (R) permission bits and logical-AND of Write (W) permission bits from the result of the second-level page-walk to be both 0. • When performing first-level translation for translation request with PASID, hardware encountered not-present (P=0) PASID-entry. • When performing first-level translation for translation request with PASID, hardware encountered a not-present (P=0) first-level-paging-entry along the page-walk, and hence could not complete the page-walk. • Execute Requests Enable (ERE) is 0 in extended-context-entry used to process a translation request with PASID that has Execute-Requested (ER) field Set. • Supervisor Requests Enable (SRE) is 0 in PASID-entry used to process a translation request with PASID that has Privileged-mode-Requested (PR) field Set. • When performing first-level translation for translation-request-with-PASID with user privilege (value of 0 in Privilege-mode-requested (PR) field), hardware encountered a present first-level-paging-entry with U/S field value of 0, causing a privilege violation.	R=W=U=S=0
If hardware successfully fetched the requested translation and the translation has at least one of Read and Write permissions, a successful translation completion with the applicable translation attributes is returned.	See Section 4.2.3 for attributes retuned in translation completion

Device-TLB implementations are required (per PCI-Express ATS specification) to implicitly invalidate faulting translations from the Device-TLB. Also, the IOTLB and paging-structure caches at the remapping hardware are invalidated when a recoverable page-fault is reported through the remapping hardware (see Section 7.5.1 for details). Thus, when replaying the faulted device operation after the recoverable fault is serviced, the old translation that caused the fault is no longer cached in the Device-TLB or IOTLB, and the resulting translation request from the device obtains the up to date translation.

7.3 Non-Recoverable Fault Reporting

Processing of non-recoverable address translation faults (and interrupt translation faults) involve logging the fault information, and reporting to software through a fault event (interrupt). Remapping architecture defines two types of fault logging facilities: (a) Primary Fault Logging; and (b) Advanced Fault Logging. The primary fault logging method must be supported by all implementations of this architecture. Support for advanced fault logging is optional. Software must not change the fault logging method while hardware is enabled (i.e., when TES or IRES fields are Set in the Global Status Register).

7.3.1 Primary Fault Logging

The primary method for logging non-recoverable faults is through Fault Recording Registers. The number of Fault Recording Registers supported is reported through the Capability Register (see Section 10.4.2). Section 10.4.14 describes the Fault Recording Registers.

Hardware maintains an internal index to reference the Fault Recording Register in which the next non-recoverable fault can be recorded. The index is reset to zero when both address and interrupt translations are disabled (i.e., TES and IES fields Clear in Global Status Register), and increments whenever a fault is recorded in a Fault Recording Register. The index wraps around from N-1 to 0, where N is the number of fault recording registers supported by the remapping hardware unit.

Hardware maintains the Primary Pending Fault (PPF) field in the Fault Status Register as the logical "OR" of the Fault (F) fields across all the Fault Recording Registers. The PPF field is re-computed by hardware whenever hardware or software updates the F field in any of the Fault Recording Registers.

When primary fault recording is active, hardware functions as follows upon detecting a non-recoverable address translation or interrupt translation fault:

- Hardware checks the current value of the Primary Fault Overflow (PFO) field in the Fault Status Register. If it is already Set, the new fault is not recorded.
- If hardware supports compression[1] of multiple faults from the same requester, it compares the source-id (SID) field of each Fault Recording Register with Fault (F) field Set, to the source-id of the currently faulted request. If the check yields a match, the fault information is not recorded.
- If the above check does not yield a match (or if hardware does not support compression of faults), hardware checks the Fault (F) field of the Fault Recording Register referenced by the internal index. If that field is already Set, hardware sets the Primary Fault Overflow (PFO) field in the Fault Status Register, and the fault information is not recorded.
- If the above check indicates there is no overflow condition, hardware records the current fault information in the Fault Recording Register referenced by the internal index. Depending on the current value of the PPF field in the Fault Status Register, hardware performs one of the following steps:
  - If the PPF field is currently Set (implying there are one or more pending faults), hardware sets the F field of the current Fault Recording Register and increments the internal index.
  - Else, hardware records the internal index in the Fault Register Index (FRI) field of the Fault Status Register and sets the F field of the current Fault Recording Register (causing the PPF field also to be Set). Hardware increments the internal index, and an interrupt may be generated based on the hardware interrupt generation logic described in Section 7.4.

Software is expected to process the non-recoverable faults reported through the Fault Recording Registers in a circular FIFO fashion starting from the Fault Recording Register referenced by the Fault Recording Index (FRI) field, until it finds a Fault Recording Register with no faults (F field Clear).

1. Hardware implementations supporting only a limited number of fault recording registers are recommended to collapse multiple pending faults from the same requester.

To recover from a primary fault overflow condition, software must first process the pending faults in each of the Fault Recording Registers, Clear the Fault (F) field in all those registers, and Clear the overflow status by writing a 1 to the Primary Fault Overflow (PFO) field. Once the PFO field is cleared by software, hardware continues to record new faults starting from the Fault Recording Register referenced by the current internal index.

7.3.2 Advanced Fault Logging

Advanced fault logging is an optional hardware feature. Hardware implementations supporting advanced fault logging report the feature through the Capability Register (see Section 10.4.2).

Advanced fault logging uses a memory-resident fault log to record non-recoverable fault information. The base and size of the memory-resident fault log region is programmed by software through the Advanced Fault Log Register. Advanced fault logging must be enabled by software through the Global Command Register before enabling the remapping hardware. Section 9.2 illustrates the format of the fault record.

When advanced fault recording is active, hardware maintains an internal index into the memory-resident fault log where the next non-recoverable fault can be recorded. The index is reset to zero whenever software programs hardware with a new fault log region through the Global Command Register, and increments whenever a non-recoverable fault is logged in the fault log. Whenever the internal index increments, hardware checks for internal index wrap-around condition based on the size of the current fault log. Any internal state used to track the index wrap condition is reset whenever software programs hardware with a new fault log region.

Hardware may compress multiple back-to-back faults from the same requester by maintaining internally the source-id of the last fault record written to the fault log. This internal “source-id from previous fault” state is reset whenever software programs hardware with a new fault log region.

Read completions due to software reading the remapping hardware registers must push (commit) any in-flight fault record writes to the fault log by the respective remapping hardware unit.

When a non-recoverable fault is detected, advanced fault logging functions in hardware as follows:

- Hardware checks the current value of the Advanced Fault Overflow (AFO) field in the Fault Status Register. If it is already Set, the new fault is not recorded.
- If hardware supports compressing multiple back-to-back faults from same requester, it compares the source-id of the currently faulted request to the internally maintained “source-id from previous fault”. If a match is detected, the fault information is not recorded.
- Otherwise, if the internal index wrap-around condition is Set (implying the fault log is full), hardware sets the AFO field in the Advanced Fault Log Register, and the fault information is not recorded.
- If the above step indicates no overflow condition, hardware records the current fault information to the fault record referenced by the internal index. Depending on the current value of the Advanced Pending Fault (APF) field in the Fault Status Register and the value of the internal index, hardware performs one of the following steps:
  - — If APF field is currently Set, or if the current internal index value is not zero (implying there are one or more pending faults in the current fault log), hardware simply increments the internal index (along with the wrap-around condition check).
  - — Otherwise, hardware sets the APF field and increments the internal index. An interrupt may be generated based on the hardware interrupt generation logic described in Section 7.4.

7.4 Non-Recoverable Fault Event

Non-recoverable faults are reported to software using a message-signalled interrupt controlled through the Fault Event Control Register. The non-recoverable fault event information (such as interrupt vector, delivery mode, address, etc.) is programmed through the Fault Event Data and Fault Event Address Registers.

A Fault Event may be generated under the following conditions:

- When primary fault logging is active, recording a non-recoverable fault to a Fault Recording Register causing the Primary Pending Fault (PPF) field in Fault Status Register to be Set.
- When advanced fault logging is active, logging a non-recoverable fault in the advanced fault log that causes the Advanced Pending Fault (APF) field in the Fault Status Register to be Set.
- When queued invalidation interface is active, an invalidation queue error causing the Invalidation Queue Error (IQE) field in the Fault Status Register to be Set.
- Invalid Device-TLB invalidation completion response received causing the Invalidation Completion Error (ICE) field in the Fault Status Register to be Set.
- Device-TLB invalidation completion time-out detected causing the Invalidation Time-out Error (ITE) field in the Fault Status Register to be Set.
- Recording a recoverable fault in the Page Request Queue that cause the Page Request Overflow (PRO) field in the Fault Status Register to be Set.

For these conditions, the Fault Event interrupt generation hardware logic functions as follows:

- Hardware checks if there are any previously reported interrupt conditions that are yet to be serviced by software. Hardware performs this check by evaluating if any of the PPF[1], PFO, (APF, AFO if advanced fault logging is active), IQE, ICE, ITE and PRO fields in the Fault Status Register is Set. If hardware detects any interrupt condition yet to be serviced by software, the Fault Event interrupt is not generated.
- If the above check indicates no interrupt condition yet to be serviced by software, the Interrupt Pending (IP) field in the Fault Event Control Register is Set. The Interrupt Mask (IM) field is then checked and one of the following conditions is applied:
  - If IM field is Clear, the fault event is generated along with clearing the IP field.
  - If IM field is Set, the interrupt is not generated.

The following logic applies for interrupts held pending by hardware in the IP field:

- If IP field was Set when software clears the IM field, the fault event interrupt is generated along with clearing the IP field.
- If IP field was Set when software services all the pending interrupt conditions (indicated by all status fields in the Fault Status Register being Clear), the IP field is cleared.

Read completions due to software reading the Fault Status Register (FSTS_REG) or Fault Event Control Register (FECTL_REG) must push (commit) any in-flight Fault Event interrupt messages generated by the respective hardware unit.

The fault event interrupts are never subject to interrupt remapping.

1. The PPF field is computed by hardware as the logical OR of Fault (F) fields across all the Fault Recording Registers of a hardware unit.

7.5 Recoverable Fault Reporting

Recoverable faults are detected at the Device-TLB on the endpoint device. Devices supporting Page Request Services (PRS) Capability reports the recoverable faults as page-requests to software through the remapping hardware. Software informs the servicing of the page-requests by sending page-responses to the device through the remapping hardware. Refer to the PCI-Express Address Translation Services (ATS) specification for details on the page-request and page-response messages.

The following sections describe the remapping hardware processing of page-requests from endpoint devices and page-responses from software. Remapping hardware indicates support for page-requests through the Extended Capability Register (see Section 10.4.3).

7.5.1 Handling of Page Requests

When PRS Capability is enabled at an endpoint device, recoverable faults detected at its Device-TLB cause the device issuing page-request messages to the remapping hardware.

Remapping hardware supports a page-request queue, as a circular buffer in system memory to record page request messages received. The following registers are defined to configure and manage the page-request-queue:

- *Page Request Queue Address Register*: Software programs this register to configure the base physical address and size of the contiguous memory region in system memory hosting the Page Request Queue.
- *Page Request Queue Head Register*: This register points to the page-request descriptor in the page request queue that software will process next. Software increments this register after processing one or more page-request descriptors in the page request queue.
- *Page Request Queue Tail Register*: This register points to the page-request descriptor in the page request queue to be written next by hardware. The Page Request Queue Head Register is incremented by hardware after writing a page-request descriptor to the page request queue.

Hardware interprets the page request queue as empty when the Head and Tail Registers are equal. Hardware interprets the page request queue as full when the Head Register is one behind the Tail Register (i.e., when all entries but one in the queue are used). This way, hardware will write at most only N-1 page-requests in a N entry page request queue.

To enable page requests from an endpoint device, software must:

- Initialize the Page Request Queue Head and Tail Registers (see Section 10.4.30 and Section 10.4.31) to zero.
- Configure the extended-context-entry used to process requests from the device, such that both the Present (P) and Page Request Enable (PRE) fields are Set.
- Setup the page request queue address and size through the Page Request Queue Address Register (see Section 10.4.32).
- Configure and enable page requests at the device through the PRS Capability Registers. (Refer to the PCI-Express ATS specification for PRS Capability Register details).

A page request message with both Last Page In Group (LPIG) and Stream Response Requested (SRR) fields clear (indicates no response is required for this request) received by remapping hardware is discarded if any of the following conditions are true:

- The Present (P) field or the Page Request Enable (PRE) field in the extended-context-entry used to process the page request is 0. Hardware may report this as non-recoverable fault (Refer to Fault Reason 30h in Appendix A).
- The Page Request Overflow (PRO) field in the Fault Status Register is 1
- The Page Request Queue is already full (i.e., the current value of the Head Register is one behind the value of the Tail Register), causing hardware to Set the Page Request Overflow (PRO) field in

the Fault Status Register (see Section 10.4.9). Setting the PRO field can cause a fault event to be generated depending on the programming of the Fault Event Registers (see Section 7.4).

A page request message with the Last Page In Group (LPIG) field clear and the Stream Response Requested (SRR) field set received by the remapping hardware results in hardware returning a Invalid Request Page Stream Response message, if:

- The Present (P) field or the Page Request Enable (PRE) field in the extended-context-entry used to process the page request is 0. Hardware may report this as non-recoverable fault (Refer to Fault Reason 30h in Appendix A).

A page request message with the Last Page In Group (LPIG) field clear and the Stream Response Requested (SRR) field set received by the remapping hardware results in hardware returning a successful Page Stream Response message, if one of the following is true:

- The Page Request Overflow (PRO) field in the Fault Status Register is 1
- The Page Request Queue is already full (i.e., the current value of the Head Register is one behind the value of the Tail Register), causing hardware to Set the Page Request Overflow (PRO) field in the Fault Status Register (see Section 10.4.9). Setting the PRO field can cause a fault event to be generated depending on the programming of the Fault Event Registers (see Section 7.4).

A page request message with the Last Page In Group (LPIG) field set received by the remapping hardware results in hardware returning a Invalid Request Page Group Response message, if:

- The Present (P) field or the Page Request Enable (PRE) field in the extended-context-entry used to process the page request is 0. Hardware may report this as non-recoverable fault (Refer to Fault Reason 30h in Appendix A).

A page request message with the Last Page In Group field set received by the remapping hardware results in hardware returning a successful Page Group Response message[1], if one of the following is true:

- The Page Request Overflow (PRO) field in the Fault Status Register is 1
- The Page Request Queue is already full (i.e., the current value of the Head Register is one behind the value of the Tail Register), causing hardware to Set the Page Request Overflow (PRO) field in the Fault Status Register (see Section 10.4.9). Setting the PRO field can cause a fault event to be generated depending on the programming of the Fault Event Registers (see Section 7.4).

If none of above conditions are true on receiving a page request message, the remapping hardware:

- Performs an implicit invalidation to invalidate any translations cached in the IOTLB and Paging Structure caches that controls the address specified in the Page Request.
- Writes a Page Request Descriptor to the Page Request Queue entry at offset specified by the Tail Register, and increments the value in the Tail Register. Depending on the type of the Page Request Descriptor written to Page Request Queue and programming of the Page Request Event Registers, a recoverable fault event may be generated (see Section 7.6).

The implicit invalidation of IOTLB and Paging Structure caches by the remapping hardware before a Page Request is reported to software, along with the endpoint device requirement to invalidate faulting translation from its Device-TLB before sending the Page Request, enforces there are no cached translations for a faulted page address before the Page Request is reported to software. This allows software to service a recoverable fault by making necessary modifications to the paging entries and send a Page Response to restart the faulted operation at the device, without performing any explicit invalidation operations.

1. Hardware generating a successful Page Group Response for last page request in a group that encounters a page request queue full/overflow condition can result in a Page Stream Response for one of the older page requests in this group to be generated later (by software). Receiving a page stream response at the endpoint after the page group response for the same group is received must not affect functioning of endpoint device.

The following sections describe the Page Request Descriptor types written by hardware. All descriptors are 128-bit sized. The Type field (bits 1:0) of each page request descriptor identifies the descriptor type.

7.5.1.1 Page Request Descriptor

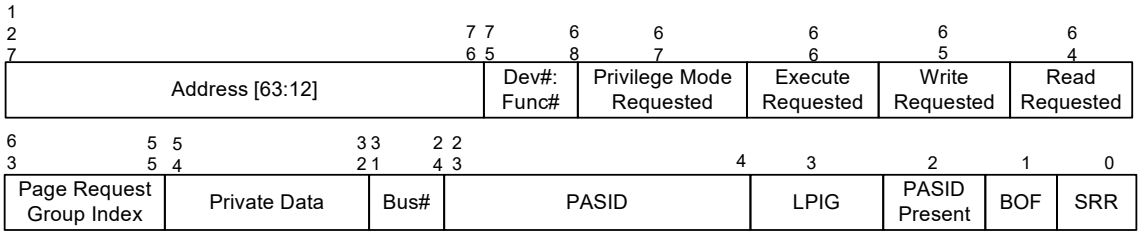


Figure 7-28. Page Request Descriptor

A Page Request Descriptor (*page_req_dsc*) is used to report Page Request messages received by the remapping hardware.

Page Request Messages: Page request messages[1] are sent by endpoint devices to report one or more page requests that are part of a page group (i.e., with same value in Page Request Group Index field), for which a **Page Group Response** is expected by the device after software has serviced all requests that are part of the page group. A page group can be composed of as small as single page request. Page requests with PASID Present field value of 1 are considered as page-requests-with-PASID. Page requests with PASID Present field value of 0 are considered as page-requests-without-PASID.

For Root-Complex integrated devices, any page-request-with-PASID in a page group, except the last page request (i.e., requests with Last Page in Group (LPIG) field value of 0), can request a **Page Stream Response** when that individual page request is serviced, by setting the Streaming Response Requested (SRR) field. Intel® Processor Graphics device requires use of this page stream response capability.

The Page Request Descriptor (*page_req_dsc*) includes the following fields:

- *Bus Number:* The bus number field contains the upper 8-bits of the source-id of the endpoint device that sent the Page Request. Refer to Section 3.4.1 for format of source-id.
- *Device and Function Numbers:* The Dev#:Func# field contains the lower 8-bits of the source-id of the endpoint device that sent the Page Request.
- *PASID Present:* If the PASID Present field is 1, the Page Request is due to a recoverable fault by a request-with-PASID. If PASID Present field is 0, the page request is due to a recoverable fault by a request-without-PASID.
- *PASID:* If the PASID Present field is 1, this field provides the PASID value of the request-with-PASID that encountered the recoverable fault that resulted in this Page Request. If PASID Present field is 0, this field is undefined.
- *Address (ADDR)*: If both the Read Requested and Write Requested fields are 0, this field is reserved. Else, this field indicates the faulted page address. If the PASID Present field is 1, the address field specifies an input-address for first-level translation. If the PASID Present field is 0, the address field specifies an input-address for second-level translation.
- *Page Request Group Index (PRGI):* The 9-bit Page Request Group Index field identifies the page group to which this request is part of. Software is expected to return the Page Request Group

1. Refer to PCI Express Address Translation Services (ATS) specification for details on page request and response messages.

Index in the respective page response. This field is undefined if both the Read Requested and Write Requested fields are 0.

  - Multiple page-requests-with-PASID (PASID Present field value of 1) from a device with same PASID value can contain any Page Request Group Index value (0-511). However, for a given PASID value, there can at most be one page-request-with-PASID outstanding from a device, with Last Page in Group (LPIG) field Set and same Page Request Group Index value.
  - Multiple page-requests-without-PASID (PASID Present field value of 0) from a device can contain any Page Request Group Index value (0-511). However, there can at most be one page-request-without-PASID outstanding from a device, with Last Page in Group field Set and same Page Request Group Index value.

- *Last Page in Group (LPIG):* If the Last Page in Group field is 1, this is the last request in the page group identified by the value in the Page Request Group Index field.
- *Streaming Response Requested (SRR):* If the Last Page in Group (LPIG) field is 0, a value of 1 in the Streaming Response Requested (SRR) field indicates a Page Stream Response is requested for this individual page request after it is serviced. If Last Page in Group (LPIG) field is 1, this field is reserved(0).
- *Blocked on Fault (BOF):* If the Last Page in Group (LPIG) field is 0 and Streaming Response Requested (SRR) field is 1, a value of 1 in the Blocked on Fault (BOF) field indicates the fault that resulted in this page request resulted in a blocking condition on the Root-Complex integrated endpoint device. This field is informational and may be used by software to prioritize processing of such blocking page requests over normal (non-blocking) page requests for improved endpoint device performance or quality of service. If Last Page in Group (LPIG) field is 1 or Streaming Response Requested (SRR) field is 0, this field is reserved(0).
- *Read Requested:* If the Read Requested field is 1, the request that encountered the recoverable fault (that resulted in this page request), requires read access to the page.
- *Write Requested:* If the Write Requested field is 1, the request that encountered the recoverable fault (that resulted in this page request), requires write access to the page.
- *Execute Requested:* If the PASID Present, Read Requested and Execute Requested fields are all 1, the request-with-PASID that encountered the recoverable fault that resulted in this page request, requires execute access to the page.
- *Privilege Mode Requested:* If the PASID Present is 1, and at least one of the Read Requested or the Write Requested field is 1, the Privilege Mode Requested field indicates the privilege of the request-with-PASID that encountered the recoverable fault (that resulted in this page request). A value of 1 for this field indicates supervisor privilege, and value of 0 indicates user privilege.
- *Private Data:* The Private Data field can be used by Root-Complex integrated endpoints to uniquely identify device-specific private information associated with an individual page request. For Intel® Processor Graphics device, the Private Data field specifies the identity of the GPU advanced-context (see Section 3.10) sending the page request.
  - For page requests requesting a page stream response (SRR=1 and LPIG =0), software is expected to return the Private Data in the respective Page Stream Response.
  - For page requests that identifies as the last request in a page group (LPIG=1), software is expected to return the Private Data in the respective Page Group Response.

For page-requests-with-PASID indicating page stream response (SRR=1 and LPIG = 0), software must respond with a Page Stream response after the respective page request is serviced. For page requests indicating last request in group (LPIG = 1), software must respond with a Page Group Response after servicing all page requests that are part of that page group. Section 7.7.1 describes the Page Group Response. Section 7.7.2 describes the Page Stream Response.

7.6 Recoverable Fault Event

Remapping hardware supports notifying pending recoverable faults to software through a Page Request Event interrupt.

When a page request descriptor (*page_req_dsc*) is written to the page request queue, hardware Sets the Pending Page Request (PPR) field in the Page Request Status Register (see Section 10.4.33). Setting of the PPR field can result in hardware generating a Page Request Event as follows:

- If hardware detects the PPR field in the Page Request Event was already 1, the Page Request Event is not generated per the programming of the Page Request Event Registers.
- Else, the Interrupt Pending (IP) field in the Page Request Event Control Register (see Section 10.4.34) is Set. The Interrupt Mask (IM) field in this register is then checked and one of the following conditions is applied:
  - — If IM field is Clear, the fault event is generated along with clearing the IP field.
  - — If IM field is Set, the interrupt is not generated.

The following logic applies for interrupts held pending by hardware in the IP field in the Page Request Event Control Register:

- If IP field was 1 when software clears the IM field, the page request event interrupt is generated along with clearing the IP field.
- If IP field was 1 when software services (writes 1 to Clear) the PPR field in the Page Request Status Register, the IP field is cleared

A page request from an endpoint is considered ‘received’ by remapping hardware when it arrives at the remapping hardware ingress. A received page request is considered ‘accepted’ to the page request queue by remapping hardware when the corresponding page request descriptor write (*page_req_dsc)* is issued. An ‘accepted’ page request is considered ‘delivered’ by remapping hardware when the respective page request descriptor write (*page_req_dsc*) to page request queue becomes visible to software followed by increment of the Page Request Queue Tail register.

For producer consumer ordering of page request processing, the following ordering requirements must be met by remapping hardware:

- A Page Request Event Interrupt must ensure all ‘accepted’ page requests (including the accepted page request that led to generation of this interrupt) are ‘delivered’ (become software visible), before the page request event interrupt is delivered to software.
- Read completions due to software reading Page Request Queue Tail Register (PQT_REG) must ensure all ‘accepted’ page requests are ‘delivered’.
- Read completions due to software reading Page Request Status Register (PRS_REG), Page Request Queue Tail Register (PQT_REG) or Page Request Event Control Register (PECTL_REG) must push (commit) any in-flight Page Request Event Interrupt generated by the respective remapping hardware unit.

The page request event interrupts are never subject to interrupt remapping.

7.7 Servicing Recoverable Faults

Software processes page request descriptors written to the Page Request Queue by remapping hardware. Processing the descriptor involves, resolving the page-fault condition, creating the translation with appropriate permission and privilege (if the Page requested is legitimate), and issuing a response back to the device through the remapping hardware.

For page requests indicating last request in a group (LPIG=1), a single Page Group Response is sent by software after servicing all page requests for respective Page Group. For page requests requesting stream response, a Page Stream Response is sent by software after servicing the respective page request. The responses are sent by software to the remapping hardware by submitting Page Response Descriptors through the Invalidation Queue (IQ). The remapping hardware processes each response descriptor by formatting and sending the appropriate Page Request Response to the endpoint device specified in the response descriptor. Refer to Section 6.5.2 for details on Invalidation Queue operation.

Servicing of a page request by software may determine that the request is spurious. i.e., the page reported in the page request already has a translation with the requested permissions and privilege in the page tables. Spurious page requests can result if software upgraded a paging entry (e.g., not present to present, read-only to read-write, etc.), and the faulting request used the translation before the upgrade that was cached in the IOTLB or Device-TLB. Irrespective of how a Page Request was serviced by software (i.e., successfully processed by creating the translation, identified as a spurious page request that did not require any update to translation, identified as invalid request due to invalid page/permission/privilege requested), software must send page response with appropriate Response Code (for page requests that are last in a group, a Page Group Response is sent; for other page requests that requests a stream response, a Page Stream Response is sent).

The following sections describe the Page Response Descriptor types written by software to the Invalidation Queue. All descriptors are 128-bit sized. The Type field (bits 3:0) of each page request descriptor identifies the descriptor type (similar to other invalidation descriptors submitted through the Invalidation Queue).

7.7.1 Page Group Response Descriptor

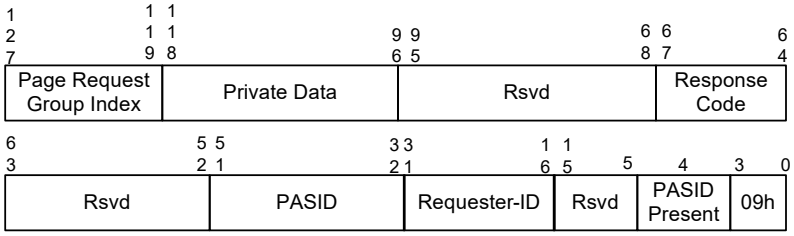


Figure 7-29. Page Group Response Descriptor

A Page Group Response Descriptor is issued by software in response to a page request indicating last request in a group. The Page Group Response must be issued after servicing all page requests with the same Page Request Group Index value.

The Page Group Request Descriptor (*page_grp_resp_dsc*) includes the following fields:

- *Requester-ID:* The Requester-ID field identifies the endpoint device function targeted by the Page Request Group Response. The upper 8-bits of the Requester-ID field specifies the Bus number and the lower 8-bits specifies the Device number and Function number. Software copies the bus number, device number, and function number fields from the respective Page Request Descriptor to form the Requester-ID field in the Page Group Response Descriptor. Refer to Section 3.4.1 for format of this field.
- *PASID Present:* If the PASID Present field is 1, the Page Group Response carries a PASID. The value in this field must match the value in the PASID Present field of the respective Page Request Descriptor.
- *PASID:* If the PASID Present field is 1, this field provides the PASID value for the Page Group Response. The value in this field must match the value in the PASID field of the respective Page Request Descriptor.
- *Page Request Group Index:* The Page Request Group Index identifies the page group of this Page Group Response. The value in this field must match the value in the Page Request Group Index field of the respective Page Request Descriptor.
- *Response Code:* The Response Code indicates the Page Group Response status. The field follows the Response Code (see Table 17) in Page Group Response message as specified in the PCI Express Address Translation Services (ATS) specification. Refer to the PCI Express ATS

specification for endpoint device behavior with these Response Codes. If all page Requests that are part of a Page Group serviced successfully, Response Status code of Success is returned.

- *Private Data:* The Private Data field is used to convey device-specific private information associated with the page request and response. The value in this field must match the value in the Private Data field of the respective Page Request Descriptor.

Table 17. Response Codes

Value	Status	Description
0h	Success	All Page Requests in the Page Request Group were successfully serviced.
1h	Invalid Request	One ore more Page Requests within the Page Request Group was not successfully serviced.
2h - Eh	Reserved	Not used.
Fh	Response Failure	Servicing of one or more Page Requests within the Page Request Group encountered a non-recoverable error.

7.7.2 Page Stream Response Descriptor

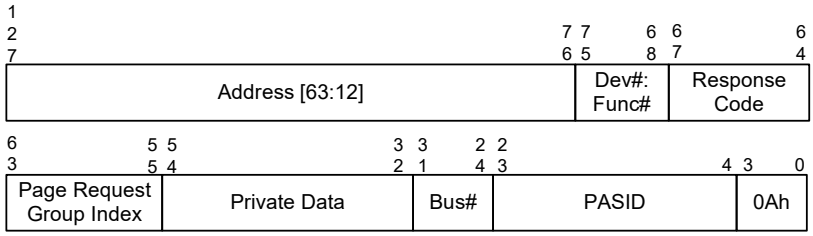


Figure 7-30. Page Stream Response Descriptor

A Page Stream Response Descriptor is used to indicate servicing of an individual page-request-with-PASID requesting a stream response (from Root-Complex integrated devices such as Intel® Processor Graphics).

The Page Stream Response Descriptor (*page_strm_resp_dsc*) includes the following fields:

- *Requester-ID:* The Requester-ID field identifies the endpoint device function targeted by the Page Stream Response. The upper 8-bits of the Requester-ID field specifies the Bus number and the lower 8-bits specifies the Device number and Function number. Software copies the bus number, device number, and function number fields from the respective Page Request Descriptor to form the Requester-ID field in the Page Stream Response Descriptor. Refer to Section 3.4.1 for format of this field.
- *PASID:* This field provides the PASID value for the Page Stream Response. The value in this field must match the value in the PASID field of the respective Page Request Descriptor.
- *Address (ADDR)*: The address field indicates the page address serviced. The value in this field must match the value in the Address field of the respective Page Request Descriptor.
- *Page Request Group Index:* The Page Request Group Index identifies the page group of the page request for which this Page Stream Response is issued. The value in this field must match the value in the Page Request Group Index field of the respective Page Request Descriptor.

- *Private Data:* The Private Data field is used to uniquely identify device-specific private information associated with the Page request/response. The value in this field must match the value in the Private Data field of the respective Page Request Descriptor.
- *Response Code:* The Response Code indicates the Page Stream Response status. The field supports the same Response Code values as for the Page Group Response (see Table 17), except applied to the individual page request.

7.8 Page Request Ordering and Draining

This section describes the expected endpoint device behavior and remapping hardware behavior on ordering and draining of page-requests.

- A recoverable fault encountered by an endpoint device is considered 'dispatched' when the corresponding page request message is posted to its egress to the interconnect. On a Device-TLB invalidation, the endpoint device must ensure the respective Device-TLB invalidation completion message is posted to its egress to the interconnect, ordered behind 'dispatched' page requests.
- Page requests and Device-TLB invalidation completion messages follow strict posted ordering on the interconnect fabric.
- A Page request is considered 'received' by remapping hardware when it arrives at the remapping hardware ingress. A 'received' page request is considered 'accepted' to the Page Request Queue by remapping hardware when the corresponding page request descriptor write (*page_req_dsc)* is issued. An 'accepted' page request is considered 'delivered' by remapping hardware when the respective page request descriptor write (*page_req_dsc*) to page request queue becomes visible to software followed by increment of the Page Request Queue Tail register.
- Remapping hardware must ensure that, page requests and Device-TLB invalidation completion messages received in its ingress are processed in order.
- If remapping hardware processing of a Device-TLB invalidation completion message results in a pending Invalidation Wait Descriptor (*inv_wait_dsc*) to complete, and if the Page-request Drain (PD=1) flag is Set in the *inv_wait_dsc*, the respective invalidation wait completion status write (if SW=1) and invalidation wait completion interrupt (if IF=1) must be ordered (made visible to software) behind page-request descriptor (*page_req_dsc*) writes for all page requests 'received' ahead of the Device-TLB invalidation completion message.

With above ordering, to drain in-flight page requests from an endpoint device, software can issue a *dev_tlb_inv_dsc* (or *ext_dev_tlb_inv_dsc*) of any invalidation granularity targeting the device, followed by an *inv_wait_dsc* with PD flag Set and SW/IF flag Set, and wait for the *inv_wait_dsc* to complete.

7.9 Page Response Ordering and Draining

This section describes the remapping hardware behavior and expected endpoint device behavior on ordering and draining of page-request responses.

- Remapping hardware must ensure that page responses (*page_grp_resp_dsc* or *page_strm_resp_dsc)* and Device-TLB invalidation requests (*dev_tlb_inv_dsc* or *ext_dev_tlb_inv_dsc*) submitted by software through the Invalidation Queue are processed in order and respective messages are posted in order to the interconnect.
- Page responses and Device-TLB invalidation requests follow strict posted ordering on the interconnect fabric.
- An endpoint device is expected to 'accept' valid page responses and Device-TLB invalidation requests in the order they are received. This implies that, before a valid Device-TLB invalidation request is acted on by the endpoint device (leading to posting an invalidation response message), the endpoint device has 'accepted' all valid page responses received ahead of the Device-TLB invalidation request.

With above ordering, to drain in-flight page responses issued by software to an endpoint device, software can issue a *dev_tlb_inv_dsc* (or *ext_dev_tlb_inv_dsc*) of any invalidation granularity targeting the device, followed by an *inv_wait_dsc* with SW/IF flag Set, and wait for the *inv_wait_dsc* to complete.

7.10 Pending Page Request Handling on Terminal Conditions

This section describes the expected endpoint device behavior for handling pending page requests on terminal conditions. A page request is considered pending at an endpoint device, if the devices has issued the page request and have not yet received the respective page response.

Terminal conditions are defined as software initiated conditions that can result in an endpoint device resetting its internal state used to match page responses to pending page requests. Examples of terminal conditions encountered by a device while it has pending page requests may include:

- Device specific software stack/driver detecting a unresponsive device and performing any form of partial device reset (E.g., GPU Engine Reset) leading to termination of one or more work-items (that may have pending page requests).
- Device specific software stack detecting a unresponsive device and performing a full device reset (E.g., GPU Timeout Detection and Recovery Reset) leading to termination of all work-items (some or all with pending page requests) active on that device.
- System software performing Function Level Reset (FLR) of a Physical Function (or SR-IOV Virtual Function) leading to termination of all work-items (some or all with pending page requests) submitted through respective function.

On above terminal conditions[1], the endpoint device is expected to handle pending page requests as follows:

- Ensure that recoverable faults encountered by the device before the terminal event are 'dispatched' (i.e., corresponding page requests are posted to device egress to the interconnect), so that any subsequent Device-TLB invalidation request completions from the device are ordered behind such page requests (i.e., follow same ordering described in Section 7.8 as with normal operation if there was no terminal condition).
- Cancel tracking of pending page requests affected by the terminal condition and replenish page request credits consumed by such page requests.
- Continue normal behavior of discarding any page responses (without adverse side effects) received that has no matching pending page request.

To handle in-flight page requests affected by terminal conditions, software initiating the terminal condition must follow below steps:

- After completing the terminal condition and before putting the device function back in service, request system software to drain (and discard) in-flight page requests/responses from the endpoint device (as described in Section 7.11), and only after completion of such page request/response draining resume normal operation of the device. For terminal conditions (such as partial device reset) where only a subset of PASIDs active on the device are affected, the drain and discard request may specify the affected PASID(s), in which case, only page requests from the device with specified PASID(s) are discarded after draining (and page requests from other PASIDs are handled normally with page responses).

The following section describes the steps system software may follow to drain all in-flight and pending page requests and page responses from/to an endpoint device.

1. For devices that may be subject to device specific software stack/driver initiated terminal conditions while in operation, such devices are expected to normally receive, process and respond to Device-TLB invalidation requests during such terminal conditions. This is required as these device specific terminal conditions may be initiated transparent to system software operation that can be issuing Device-TLB invalidations as part of TLB shootdown operations.

7.11 Software Steps to Drain Page Requests & Responses

System Software may follow below steps to drain in-flight page requests and page responses between remapping hardware queues (Page Request Queue for page requests and Invalidation Queue for page responses) and an endpoint device.

a. Submit Invalidation Wait Descriptor (*inv_wait_dsc*) with Fence flag (FN=1) Set to Invalidation Queue. This ensures that all requests submitted to the Invalidation Queue ahead of this wait descriptor are processed and completed by remapping hardware before processing requests after the Invalidation Wait Descriptor. It is not required to specify SW flag (or IF flag) in this descriptor or for software to wait on its completion, as its function is to only act as a barrier.

b. Submit an IOTLB invalidate descriptor (*iotlb_inv_dsc* or *ext_iotlb_inv_dsc*) followed by Device-TLB invalidation descriptor (*dev_tlb_inv_dsc* or *ext_dev_tlb_inv_dsc*) targeting the endpoint device. These invalidation requests can be of any granularity. Per the ordering requirements described in Section 7.9, older page responses issued by software to the endpoint device before step (a) are guaranteed to be received by the endpoint before the endpoint receives this Device-TLB invalidation request.

c. Submit Invalidation Wait Descriptor (*inv_wait_dsc*) with Page-request Drain (PD=1) flag Set, along with Invalidation Wait Completion status write flag (SW=1), and wait on Invalidation Wait Descriptor completion. Per the ordering requirements described in Section 7.8, the Device-TLB invalidation completion from the device is guaranteed to be ordered behind already issued page requests from the device. Also, per the ordering requirements in Section 7.8, the remapping hardware ensures that the Invalidation Wait Descriptor status write (that signals completion of invalidation descriptors submitted in step (b)) is ordered (with respect to software visibility) behind the page request descriptor (*page_req_dsc*) writes for page requests received before the Device-TLB invalidation completion.

d. If there are no page request queue overflow condition encountered by remapping hardware during above steps, software can be guaranteed that all page requests and page responses are drained between the remapping hardware and the target endpoint device. However, if a page-request queue full condition was detected by remapping hardware when processing a page request with Last Page In Group (LPIG) field Set or with Stream Response Requested (SRR) field Set during steps (a) through (c) above, the remapping hardware generates a successful auto page response (see Section 7.5.1 for remapping hardware auto page response behavior). To drain such potential auto page responses generated by remapping hardware, software must repeat steps (b) and (c).

7.12 Revoking PASIDs with Pending Page Faults

At any time of operation, system software resource management actions (E.g., on host application process termination) can result in system software requesting the endpoint device specific driver to revoke the PASID that it has previously allocated and is actively being used by the endpoint device. To service such system software request, it is the responsibility of the endpoint device and the driver to revoke use of this PASID by the device and ensure all outstanding page-requests for this PASID are serviced by system software and page-request responses received, before returning success to system software for the PASID revocation request.

After de-allocating a PASID, system software may follow the same steps for in-flight page request/response draining described in Section 7.11 to ensure any in-flight page requests/responses for the de-allocated PASID are drained before re-allocating that PASID to a new client.

8 BIOS Considerations

The system BIOS is responsible for detecting the remapping hardware functions in the platform and for locating the memory-mapped remapping hardware registers in the host system address space. The BIOS reports the remapping hardware units in a platform to system software through the DMA Remapping Reporting (DMAR) ACPI table described below[1].

8.1 DMA Remapping Reporting Structure

Field	Byte Length	Byte Offset	Description
Signature	4	0	"DMAR". Signature for the DMA Remapping Description table.
Length	4	4	Length, in bytes, of the description table including the length of the associated remapping structures.
Revision	1	8	1
Checksum	1	9	Entire table must sum to zero.
OEMID	6	10	OEM ID
OEM Table ID	8	16	For DMAR description table, the Table ID is the manufacturer model ID.
OEM Revision	4	24	OEM Revision of DMAR Table for OEM Table ID.
Creator ID	4	28	Vendor ID of utility that created the table.
Creator Revision	4	32	Revision of utility that created the table.
Host Address Width	1	36	This field indicates the maximum DMA physical addressability supported by this platform. The system address map reported by the BIOS indicates what portions of this addresses are populated. The Host Address Width (HAW) of the platform is computed as (N+1), where N is the value reported in this field. For example, for a platform supporting 40 bits of physical addressability, the value of 100111b is reported in this field.

1. All Reserved fields in DMAR remapping structures must be initialized to 0.

Field	Byte Length	Byte Offset	Description
Flags	1	37	• Bit 0: INTR_REMAP - If Clear, the platform does not support interrupt remapping. If Set, the platform supports interrupt remapping. • Bit 1: X2APIC_OPT_OUT - For firmware compatibility reasons, platform firmware may Set this field to request system software to opt out of enabling Extended xAPIC (X2APIC) mode. This field is valid only when the INTR_REMAP field (bit 0) is Set. Since firmware is permitted to hand off platform to system software in legacy xAPIC mode, system software is required to check this field as Clear as part of detecting X2APIC mode support in the platform. • Bit 2: DMA_CTRL_PLATFORM_OPT_IN_FLAG: Platform firmware is recommended to Set this field to report any platform initiated DMA is restricted to only reserved memory regions (reported in RMRR structures) when transferring control to system software such as on *ExitBootServices()*. System software may program DMA remapping hardware to block DMA outside of RMRR, except for memory explicitly registered by device drivers with system software. • Bits 3-7: Reserved (0).
Reserved	10	38	Reserved (0).
Remapping Structures[]	-	48	A list of structures. The list will contain one or more DMA Remapping Hardware Unit Definition (DRHD) structures, and zero or more Reserved Memory Region Reporting (RMRR) and Root Port ATS Capability Reporting (ATSR) structures. These structures are described below.

8.2 Remapping Structure Types

The following types of remapping structures are defined. All remapping structures start with a ‘Type’ field followed by a ‘Length’ field indicating the size in bytes of the structure.

Value	Description
0	DMA Remapping Hardware Unit Definition (DRHD) Structure
1	Reserved Memory Region Reporting (RMRR) Structure
2	Root Port ATS Capability Reporting (ATSR) Structure
3	Remapping Hardware Static Affinity (RHSA) Structure
4	ACPI Name-space Device Declaration (ANDD) Structure
>4	Reserved for future use. For forward compatibility, software skips structures it does not comprehend by skipping the appropriate number of bytes indicated by the Length field.

BIOS implementations must report these remapping structure types in numerical order. i.e., All remapping structures of type 0 (DRHD) enumerated before remapping structures of type 1 (RMRR), and so forth.

8.3 DMA Remapping Hardware Unit Definition Structure

A DMA-remapping hardware unit definition (DRHD) structure uniquely represents a remapping hardware unit present in the platform. There must be at least one instance of this structure for each PCI segment in the platform.

Field	Byte Length	Byte Offset	Description
Type	2	0	0 - DMA Remapping Hardware Unit Definition (DRHD) structure
Length	2	2	Varies (16 + size of Device Scope Structure)
Flags	1	4	Bit 0: INCLUDE_PCI_ALL • If Set, this remapping hardware unit has under its scope all PCI compatible devices in the specified Segment, except devices reported under the scope of other remapping hardware units for the same Segment. If a DRHD structure with INCLUDE_PCI_ALL flag Set is reported for a Segment, it must be enumerated by BIOS after all other DRHD structures for the same Segment[1]. A DRHD structure with INCLUDE_PCI_ALL flag Set may use the 'Device Scope' field to enumerate I/OxAPIC and HPET devices under its scope. • If Clear, this remapping hardware unit has under its scope only devices in the specified Segment that are explicitly identified through the 'Device Scope' field. Bits 1-7: Reserved.
Reserved	1	5	Reserved (0).
Segment Number	2	6	The PCI Segment associated with this unit.
Register Base Address	8	8	Base address of remapping hardware register-set for this unit.
Device Scope []	-	16	The Device Scope structure contains zero or more Device Scope Entries that identify devices in the specified segment and under the scope of this remapping hardware unit. The Device Scope structure is described below.

1. On platforms with multiple PCI segments, any of the segments can have a DRHD structure with INCLUDE_PCI_ALL flag Set.

8.3.1 Device Scope Structure

The Device Scope Structure is made up of Device Scope Entries. Each Device Scope Entry may be used to indicate a PCI endpoint device, a PCI sub-hierarchy, or devices such as I/OxAPICs or HPET (High Precision Event Timer).

In this section, the generic term 'PCI' is used to describe conventional PCI, PCI-X, and PCI-Express devices. Similarly, the term 'PCI-PCI bridge' is used to refer to conventional PCI bridges, PCI-X bridges, PCI-Express root ports, or downstream ports of a PCI-Express switch.

A PCI sub-hierarchy is defined as the collection of PCI controllers that are downstream to a specific PCI-PCI bridge. To identify a PCI sub-hierarchy, the Device Scope Entry needs to identify only the parent PCI-PCI bridge of the sub-hierarchy.

Field	Byte Length	Byte Offset	Description
Type	1	0	The following values are defined for this field. • 0x01: PCI Endpoint Device - The device identified by the 'Path' field is a PCI endpoint device. This type must not be used in Device Scope of DRHD structures with INCLUDE_PCI_ALL flag Set. • 0x02: PCI Sub-hierarchy - The device identified by the 'Path' field is a PCI-PCI bridge. In this case, the specified bridge device and all its downstream devices are included in the scope. This type must not be in Device Scope of DRHD structures with INCLUDE_PCI_ALL flag Set. • 0x03: IOAPIC - The device identified by the 'Path' field is an I/O APIC (or I/O SAPIC) device, enumerated through the ACPI MADT I/O APIC (or I/O SAPIC) structure. • 0x04: MSI_CAPABLE_HPET[1] - The device identified by the 'Path' field is an HPET Timer Block capable of generating MSI (Message Signaled interrupts). HPET hardware is reported through ACPI HPET structure. • 0x05: ACPI_NAMESPACE_DEVICE - The device identified by the 'Path' field is an ACPI name-space enumerated device capable of generating DMA requests. Other values for this field are reserved for future use.
Length	1	1	Length of this Entry in Bytes. (6 + X), where X is the size in bytes of the "Path" field.
Reserved	2	2	Reserved (0).
Enumeration ID	1	4	When the 'Type' field indicates 'IOAPIC', this field provides the I/O APICID as provided in the I/O APIC (or I/O SAPIC) structure in the ACPI MADT (Multiple APIC Descriptor Table). When the 'Type' field indicates 'MSI_CAPABLE_HPET', this field provides the 'HPET Number' as provided in the ACPI HPET structure for the corresponding Timer Block. When the 'Type' field indicates 'ACPI_NAMESPACE_DEVICE', this field provides the "ACPI Device Number" as provided in the ACPI Name-space Device Declaration (ANDD) structure for the corresponding ACPI device. This field is treated reserved (0) for all other 'Type' fields.

Field	Byte Length	Byte Offset	Description
Start Bus Number	1	5	This field describes the bus number (bus number of the first PCI Bus produced by the PCI Host Bridge) under which the device identified by this Device Scope resides.
Path	2 * N	6	Describes the hierarchical path from the Host Bridge to the device specified by the Device Scope Entry. For example, a device in a N-deep hierarchy is identified by N {PCI Device Number, PCI Function Number} pairs, where N is a positive integer. Even offsets contain the Device numbers, and odd offsets contain the Function numbers. The first {Device, Function} pair resides on the bus identified by the 'Start Bus Number' field. Each subsequent pair resides on the bus directly behind the bus of the device identified by the previous pair. The identity (Bus, Device, Function) of the target device is obtained by recursively walking down these N {Device, Function} pairs. If the 'Path' field length is 2 bytes (N=1), the Device Scope Entry identifies a 'Root-Complex Integrated Device'. The requester-id of 'Root-Complex Integrated Devices' are static and not impacted by system software bus rebalancing actions. If the 'Path' field length is more than 2 bytes (N > 1), the Device Scope Entry identifies a device behind one or more system software visible PCI-PCI bridges. Bus rebalancing actions by system software modifying bus assignments of the device's parent bridge impacts the bus number portion of device's requester-id.

1. An HPTE Timer Block is capable of MSI interrupt generation if any of the Timers in the Timer Block reports FSB_INTERRUPT_DELIVERY capability in the Timer Configuration and Capability Registers. HPET Timer Blocks not capable of MSI interrupt generation (and instead have their interrupts routed through I/OxAPIC) are not reported in the Device Scope.

The following pseudocode describes how to identify the device specified through a Device Scope structure:

```
n = (DevScope.Length - 6) / 2;              // number of entries in the 'Path' field
type = DevScope.Type;                       // type of device
bus = DevScope.StartBusNum;                 // starting bus number
dev = DevScope.Path[0].Device;              // starting device number
func = DevScope.Path[0].Function;           // starting function number
i = 1;
while (--n) {
    bus = read_secondary_bus_reg(bus, dev, func);// secondary bus# from config reg.
    dev = DevScope.Path[i].Device;          // read next device number
    func = DevScope.Path[i].Function;       // read next function number
    i++;
}
source_id = {bus, dev, func};
```

```
target_device = {type, source_id};      // if 'type' indicates 'IOAPIC', DevScope.EnumID
                                        // provides the I/O APICID as reported in the ACPI MADT
```

8.3.1.1 Reporting Scope for I/OxAPICs

Interrupts from devices that only support (or are only enabled for) legacy interrupts are routed through the I/OxAPICs in the platform. Each I/OxAPIC in the platform is reported to system software through ACPI MADT (Multiple APIC Descriptor Tables). Some platforms may also expose I/OxAPICs as PCI-discoverable devices.

For platforms reporting interrupt remapping capability (INTR_REMAP flag Set in the DMAR structure), each I/OxAPIC in the platform reported through ACPI MADT must be explicitly enumerated under the Device Scope of the appropriate remapping hardware units (even for remapping hardware unit reported with INCLUDE_PCI_ALL flag Set in DRHD structure).

- For I/OxAPICs that are PCI-discoverable, the source-id for such I/OxAPICs (computed using the above pseudocode from its Device Scope structure) must match its PCI requester-id effective at the time of boot.
- For I/OxAPICs that are not PCI-discoverable:
  - If the 'Path' field in Device Scope has a size of 2 bytes, the corresponding I/OxAPIC is a Root-Complex integrated device. The 'Start Bus Number' and 'Path' field in the Device Scope structure together provides the unique 16-bit source-id allocated by the platform for the I/OxAPIC. Examples are I/OxAPICs integrated to the IOH and south bridge (ICH) components.
  - If the 'Path' field in Device Scope has a size greater than 2 bytes, the corresponding I/OxAPIC is behind some software visible PCI-PCI bridge. In this case, the 'Start Bus Number' and 'Path' field in the Device Scope structure together identifies the PCI-path to the I/OxAPIC device. Bus rebalancing actions by system software modifying bus assignments of the device's parent bridge impacts the bus number portion of device's source-id. Examples are I/OxAPICs in PCI-Express-to-PCI-X bridge components in the platform.

8.3.1.2 Reporting Scope for MSI Capable HPET Timer Block

High Precision Event Timer (HPET) Timer Block supporting Message Signaled Interrupt (MSI) interrupts may generate interrupt requests directly to the Root-Complex (instead of routing through I/OxAPIC). Platforms supporting interrupt remapping must explicitly enumerate any MSI-capable HPET Timer Block in the platform through the Device Scope of the appropriate remapping hardware unit. In this case, the 'Start Bus Number' and 'Path' field in the Device Scope structure together provides the unique 16-bit source-id allocated by the platform for the MSI-capable HPET Timer Block.

8.3.1.3 Reporting Scope for ACPI Name-space Devices

Some platforms may support ACPI name-space enumerated devices that are capable of generating DMA requests. Platforms supporting DMA remapping must explicitly declare any such DMA-capable ACPI name-space devices in the platform through ACPI Name-space Device Declaration (ANDD) structure and enumerate them through the Device Scope of the appropriate remapping hardware unit. In this case, the 'Start Bus Number' and 'Path' field in the Device Scope structure together provides the unique 16-bit source-id allocated by the platform for the ACPI name-space device. Multiple ACPI name-space devices that share common bus-mastering hardware resources may share a common source-id. For example, some Intel® SoC platforms supports a Low Power Sub System (LPSS) in the southbridge, that shares a common DMA resource across multiple ACPI name-space devices such as I2C, SPI, UART, and SDIO.

8.3.1.4 Device Scope Example

This section provides an example platform configuration with multiple remapping hardware units. The configurations described are hypothetical examples, only intended to illustrate the Device Scope structures.

Figure 8-31 illustrates a platform configuration with a single PCI segment and host bridge (with a starting bus number of 0), and supporting four remapping hardware units as follows:

1. Remapping hardware unit #1 has under its scope all devices downstream to the PCI-Express root port located at (dev:func) of (14:0).
2. Remapping hardware unit #2 has under its scope all devices downstream to the PCI-Express root port located at (dev:func) of (14:1).
3. Remapping hardware unit #3 has under its scope a Root-Complex integrated endpoint device located at (dev:func) of (29:0).
4. Remapping hardware unit #4 has under its scope all other PCI compatible devices in the platform not explicitly under the scope of the other remapping hardware units. In this example, this includes the integrated device at (dev:func) at (30:0), and all the devices attached to the south bridge component. The I/OxAPIC in the platform (I/O APICID = 0) is under the scope of this remapping hardware unit, and has a BIOS assigned bus/dev/function number of (0,12,0).

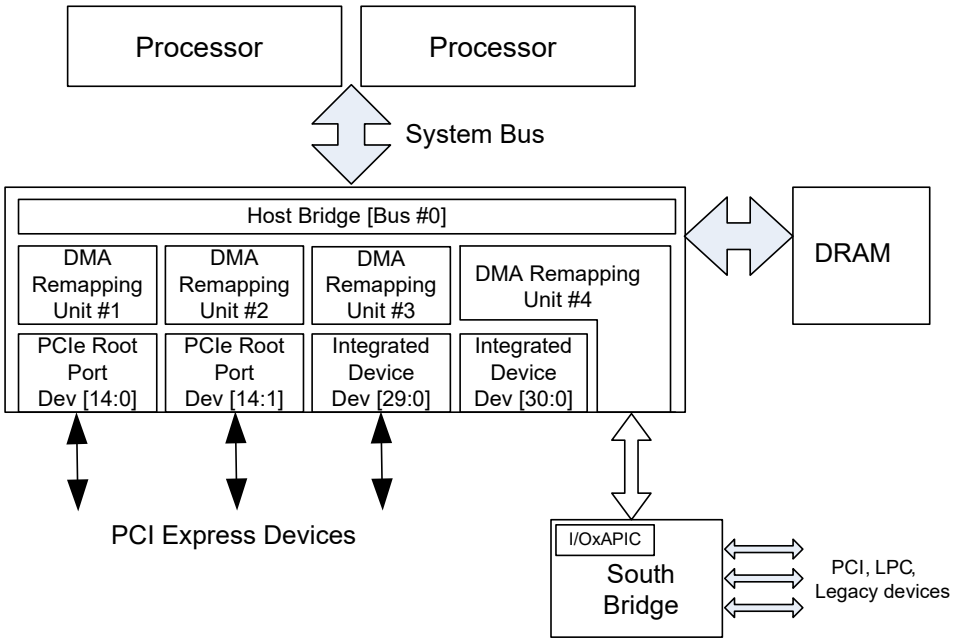


Figure 8-31. Hypothetical Platform Configuration

This platform requires 4 DRHD structures. The Device Scope fields in each DRHD structure are described as below.

- Device Scope for remapping hardware unit #1 contains only one Device Scope Entry, identified as [2, 8, 0, 0, 0, 14, 0].
  - — System Software uses the Entry Type field value of 0x02 to conclude that all devices downstream of the PCI-PCI bridge device at PCI Segment 0, Bus 0, Device 14, and Function 0 are within the scope of this remapping hardware unit.
- Device Scope for remapping hardware unit #2 contains only one Device Scope Entry, identified as [2, 8, 0, 0, 0, 14, 1].
  - — System Software uses the Entry Type field value of 0x02 to conclude that all devices downstream of the PCI-PCI bridge device at PCI Segment 0, Bus 0, Device 14, and Function 1 are within the scope of this remapping hardware unit.

- Device Scope for remapping hardware unit #3 contains only one Device Scope Entry, identified as [1, 8, 0, 0, 0, 29, 0].
  - System software uses the Type field value of 0x1 to conclude that the scope of remapping hardware unit #3 includes only the endpoint device at PCI Segment 0, Bus 0, Device 29 and Function 0.
- Device Scope for remapping hardware unit #4 contains only one Device Scope Entry, identified as [3, 8, 0, 1, 0, 12, 0]. Also, the DHRD structure for remapping hardware unit #4 indicates the INCLUDE_PCI_ALL flag. This hardware unit must be the last in the list of hardware unit definition structures reported.
  - System software uses the INCLUDE_PCI_ALL flag to conclude that all PCI compatible devices that are not explicitly enumerated under other remapping hardware units are in the scope of remapping unit #4. Also, the Device Scope Entry with Type field value of 0x3 is used to conclude that the I/OxAPIC (with I/O APICID=0 and source-id of [0,12,0]) is under the scope of remapping hardware unit #4.

8.3.2 Implications for ARI

The PCI-Express Alternate Routing-ID Interpretation (ARI) Extended Capability enables endpoint devices behind ARI-capable PCI-Express Root/Switch ports to support 'Extended Functions', beyond the limit of 8 'Traditional Functions'. When ARI is enabled, 'Extended Functions' on an endpoint are under the scope of the same remapping unit as the 'Traditional Functions' on the endpoint.

8.3.3 Implications for SR-IOV

The PCI-Express Single-Root I/O Virtualization (SR-IOV) Capability enables a 'Physical Function' on an endpoint device to support multiple 'Virtual Functions' (VFs). A 'Physical Function' can be a 'Traditional Function' or an ARI 'Extended Function'. When SR-IOV is enabled, 'Virtual Functions' of a 'Physical Function' are under the scope of the same remapping unit as the 'Physical Function'.

8.3.4 Implications for PCI/PCI-Express Hot Plug

Conventional PCI and PCI-Express defines support for hot plug. Devices hot plugged behind a parent device (PCI* bridge or PCI-Express root/switch port) are under the scope of the same remapping unit as the parent device.

8.3.5 Implications with PCI Resource Rebalancing

System software may perform PCI resource rebalancing to dynamically reconfigure the PCI sub-system (such as on PCI or PCI-Express hot-plug). Resource rebalancing can result in system software changing the bus number allocated for a device. Such rebalancing only changes the device's identity (Source-ID). The device will continue to be under the scope of the same remapping unit as it was before rebalancing. System software is responsible for tracking device identity changes and resultant impact to Device Scope.

8.3.6 Implications with Provisioning PCI BAR Resources

System BIOS typically provisions the initial PCI BAR resources for devices present at time of boot. To conserve physical address space (especially below 4GB) consumed by PCI BAR resources, BIOS implementations traditionally use compact allocation policies resulting in BARs of multiple devices/functions residing within the same system-base-page-sized region (4KB for Intel® 64 platforms). However, allocating BARs of multiple devices in the same system-page-size region imposes challenges to system software using remapping hardware to assign these devices to isolated domains.

For platforms supporting remapping hardware, BIOS implementations should avoid allocating BARs of otherwise independent devices/functions in the same system-base-page-sized region.

8.4 Reserved Memory Region Reporting Structure

Section 3.14 described the details of BIOS allocated reserved memory ranges that may be DMA targets. BIOS may report each such reserved memory region through the RMRR structures, along with the devices that requires access to the specified reserved memory region. Reserved memory ranges that are either not DMA targets, or memory ranges that may be target of BIOS initiated DMA only during pre-boot phase (such as from a boot disk drive) must not be included in the reserved memory region reporting. The base address of each RMRR region must be 4KB aligned and the size must be an integer multiple of 4KB. BIOS must report the RMRR reported memory addresses as reserved in the system memory map returned through methods such as INT15, EFI GetMemoryMap etc. The reserved memory region reporting structures are optional. If there are no RMRR structures, the system software concludes that the platform does not have any reserved memory ranges that are DMA targets.

The RMRR regions are expected to be used for legacy usages (such as USB, UMA Graphics, etc.) requiring reserved memory. Platform designers should avoid or limit use of reserved memory regions since these require system software to create holes in the DMA virtual address range available to system software and its drivers.

Field	Byte Length	Byte Offset	Description
Type	2	0	1 - Reserved Memory Region Reporting Structure
Length	2	2	Varies (24 + size of Device Scope structure)
Reserved	2	4	Reserved (0).
Segment Number	2	6	PCI Segment Number associated with devices identified through the Device Scope field.
Reserved Memory Region Base Address	8	8	Base address of 4KB-aligned reserved memory region.
Reserved Memory Region Limit Address	8	16	Last address of the reserved memory region. Value in this field must be greater than the value in Reserved Memory Region Base Address field. The reserved memory region size (Limit - Base + 1) must be an integer multiple of 4KB.
Device Scope[]	-	24	The Device Scope structure contains one or more Device Scope entries that identify devices requiring access to the specified reserved memory region. The devices identified in this structure must be devices under the scope of one of the remapping hardware units reported in DRHD.

8.5 Root Port ATS Capability Reporting Structure

This structure is applicable only for platforms supporting Device-TLBs as reported through the Extended Capability Register. For each PCI Segment in the platform that supports Device-TLBs, BIOS provides an ATSR structure. The ATSR structures identifies PCI-Express Root-Ports supporting Address Translation Services (ATS) transactions. Software must enable ATS on endpoint devices behind a Root Port only if the Root Port is reported as supporting ATS transactions.

Field	Byte Length	Byte Offset	Description
Type	2	0	2 - Root Port ATS Capability Reporting Structure
Length	2	2	Varies (8 + size of Device Scope Structure)
Flags	1	4	• Bit 0: ALL_PORTS: If Set, indicates all PCI-Express Root Ports in the specified PCI Segment supports ATS transactions. If Clear, indicates ATS transactions are supported only on Root Ports identified through the Device Scope field. • Bits 1-7: Reserved.
Reserved	1	5	Reserved (0).
Segment Number	2	6	The PCI Segment associated with this ATSR structure.
Device Scope []	-	8	If the ALL_PORTS flag is Set, the Device Scope structure is omitted. If ALL_PORTS flag is Clear, the Device Scope structure contains Device Scope Entries that identifies Root Ports supporting ATS transactions. The Device Scope structure is described in Section 8.3.1. All Device Scope Entries in this structure must have a Device Scope Entry Type of 02h.

8.6 Remapping Hardware Static Affinity Structure

Remapping Hardware Status Affinity (RHSA) structure is applicable for platforms supporting non-uniform memory (NUMA), where Remapping hardware units spans across nodes. This optional structure provides the association between each Remapping hardware unit (identified by its respective Base Address) and the proximity domain to which that hardware unit belongs. Such platforms, report the proximity of processor and memory resources using ACPI Static Resource Affinity (SRAT) structure. To optimize remapping hardware performance, software may allocate translation structures referenced by a remapping hardware unit from memory in the same proximity domain. Similar to SRAT, the information in the RHSA structure is expected to be used by system software during early initialization, when evaluation of objects in the ACPI name-space is not yet possible.

Field	Byte Length	Byte Offset	Description
Type	2	0	3 - Remapping Hardware Static Affinity Structure. This is an optional structure and intended to be used only on NUMA platforms with Remapping hardware units and memory spanned across multiple nodes. When used, there must be a Remapping Hardware Static Affinity structure for each Remapping hardware unit reported through DRHD structure.
Length	2	2	Length is 20 bytes
Reserved	4	4	Reserved (0).
Register Base Address	8	8	Register Base Address of this Remap hardware unit reported in the corresponding DRHD structure.
Proximity Domain [31:0]	4	16	Proximity Domain to which the Remap hardware unit identified by the Register Base Address field belongs.

8.7 ACPI Name-space Device Declaration Structure

An ACPI Name-space Device Declaration (ANDD) structure uniquely represents an ACPI name-space enumerated device capable of issuing DMA requests in the platform. ANDD structures are used in conjunction with Device-Scope entries of type 'ACPI_NAMESPACE_DEVICE'. Refer to Section 8.3.1 for details on Device-Scope entries.

Field	Byte Length	Byte Offset	Description
Type	2	0	4 - ACPI Name-space Device Declaration (ANDD) structure
Length	2	2	Length of this Entry in Bytes. (8 + N), where N is the size in bytes of the "ACPI Object Name" field.
Reserved	3	4	Reserved (0).
ACPI Device Number	1	7	Each ACPI device enumerated through an ANDD structure must have a unique value for this field. To report an ACPI device with 'ACPI Device Number' value of X, under the scope of a DRHD unit, a Device-Scope entry of type 'ACPI_NAMESPACE_DEVICE' is used with value of X in the Enumeration ID field. The 'Start Bus Number' and 'Path' fields in the Device-Scope together provides the 16-bit source-id allocated by the platform for the ACPI device.
ACPI Object Name	N	8	ASCII, null terminated, string that contains a fully qualified reference to the ACPI name-space object that is this device. (For example, "_SB.I2C0" represents the ACPI object name for an embedded I2C controller in southbridge; Quotes are omitted in the data field). Refer to ACPI specification for fully qualified references for ACPI name-space objects.

8.8 Remapping Hardware Unit Hot Plug

Remapping hardware units are implemented in Root-Complex components such as the I/O Hub (IOH). Such Root-Complex components may support hot-plug capabilities within the context of the interconnect technology supported by the platform. These hot-pluggable entities consist of an I/O subsystem rooted in a ACPI host bridge. The I/O subsystem may include Remapping hardware units, in addition to I/O devices directly attached to the host bridge, PCI/PCI-Express sub-hierarchies, and I/OxAPICs.

The ACPI DMAR static tables and sub-tables defined in previous sections enumerate the remapping hardware units present at platform boot-time. Following sections illustrates the ACPI methods for dynamic updates to remapping hardware resources, such as on I/O hub hot-plug. Following sections assume familiarity with ACPI 3.0 specification and system software support for host-bridge hot-plug.

8.8.1 ACPI Name Space Mapping

ACPI defines Device Specific Method (_DSM) as a method that enables ACPI devices to provide device specific functions without name-space conflicts. A Device Specific Method (_DSM) with the following GUID is used for dynamic enumeration of remapping hardware units.

GUID
D8C1A3A6-BE9B-4C9B-91BF-C3CB81FC5DAF

The _DSM method would be located under the ACPI device scope where the platform wants to expose the remapping hardware units. For example, ACPI name-space includes representation for hot-pluggable I/O hubs in the system as a ACPI host bridges. For Remapping hardware units implemented in I/O hub component, the _DSM method would be under the respective ACPI host bridge device.

The _DSM method supports the following function indexes.

Function Index	Description
0	Query function as specified in ACPI 3.0 specification. Returns which of the below function indexes are supported.
1	Return DMA Remapping Hardware Definition (DRHD) Structures[1]
2	Return Root Port ATS Capability Reporting (ATSR) Structure
3	Return Remapping Hardware Static Affinity (RHSA) Structure

1. Reserved Memory Region Reporting (RMRR) structures are not reported via _DSM, since use of reserved memory regions are limited to legacy devices (USB, iGFX etc.) that are not applicable for hot-plug.

8.8.2 ACPI Sample Code

This section illustrates sample ASL code for enumerating remapping resources in an I/O hub.

```
Scope \_SB {
    ....
    Device (IOHn) {                         // host bridge representation for I/O Hub n
        Name (_HID, EISAID("PNP0A08"))
        Name (_CID, EISAID("PNP0A03"))
        ...
        Method (_DSM, 0, NotSerialized) {   // Device specific method
            Switch(Arg0) {
                case (ToUUID("D8C1A3A6-BE9B-4C9B-91BF-C3CB81FC5DAF")) {
                    Switch (Arg2) {         // No switch for Arg1, since only one version of this method is supported
                        case(0): {Return (Buffer() {0x1F})}     // function indexes 1-4 supported
                        case(1): {Return DRHDT} // DRHDT is a buffer containing relavent DRHD structures for I/O Hub n
                        case(2): {Return ATSRT}  // ATSRT is a buffer containing relavent ATSR structure for I/O Hub n
                        case(3): {Return RHSAT} // RHSAT is a buffer containing relavent RHSAT structure for I/O Hub n
                    }
                }
            }
        }
        ...
    }
} // end of Scope SB
```

8.8.3 Example Remapping Hardware Reporting Sequence

The following sequence may be practiced for enumerating remapping hardware resources at boot time.

- Platform prepares name space and populates the ACPI DMAR static reporting tables to be reported to system software. These DMAR static tables report only the remapping hardware units that are present at time of boot, and accessible by system software.

The following sequence may be practiced on I/O hub hot-add:

- Platform notifies system software via ACPI the presence of new resources.
- System software evaluates the handle to identify the object of the notify as ACPI host bridge (I/O hub)
- If System software is able to support the hot-add of host bridge, it calls _OST to indicate success.
- System software evaluates _DSM method to obtain the remapping hardware resources associated with this host bridge (I/O hub)[1].
- System software initializes and prepares the remapping hardware for use.
- System software continues with host-bridge hot-add processing, including discovery and configuration of I/O hierarchy below the hot-added host-bridge.

1. Invoking the _DSM method does not modify the static DMAR tables. System software must maintain the effective DMAR information comprehending the initial DMAR table reported by the platform, and any remapping hardware units added or removed via _DSM upon host bridge hot-add or hot-remove.

9 Translation Structure Formats

This chapter describes the memory-resident structures for DMA and interrupt remapping.

9.1 Root Entry

The following figure and table describe the root-entry. The Root Table Address Register points to table of root-entries, when Root-table-type (RTT) field in the register is Clear.

.

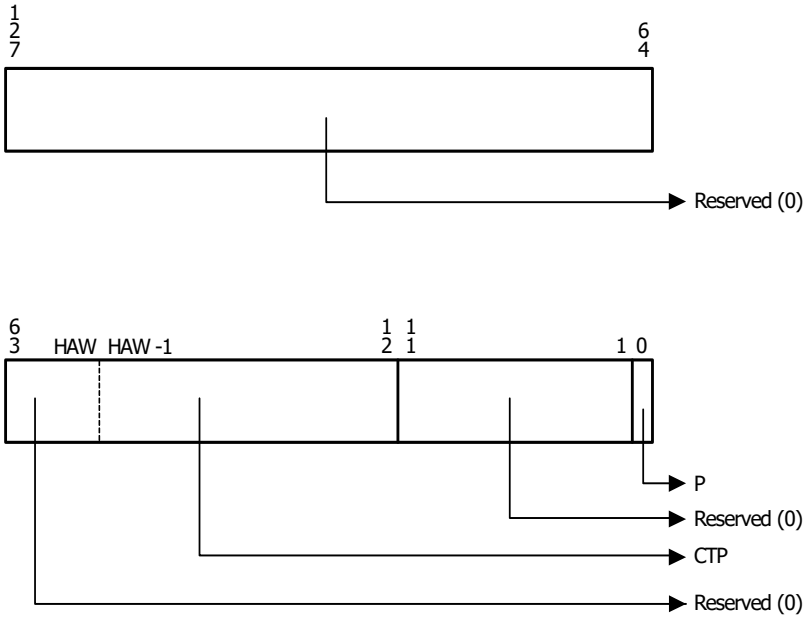


Figure 9-32. Root-Entry Format

Bits	Field	Description
127:64	R: Reserved	Reserved. Must be 0.
63:12	CTP: Context-table Pointer	Pointer to Context-table for this bus. The Context-table is 4KB in size and size-aligned. Hardware treats bits 63:HAW as reserved (0), where HAW is the host address width of the platform.
11:1	R: Reserved	Reserved. Must be 0.
0	P: Present	This field indicates whether the root-entry is present. • 0: Indicates the root-entry is not present. All other fields are ignored by hardware. • 1: Indicates the root-entry is present.

9.2 Extended Root Entry

The following figure and table describe the extended-root-entry. The Root Table Address Register points to table of extended-root-entries, when Root-table-type (RTT) field in the register is Set.

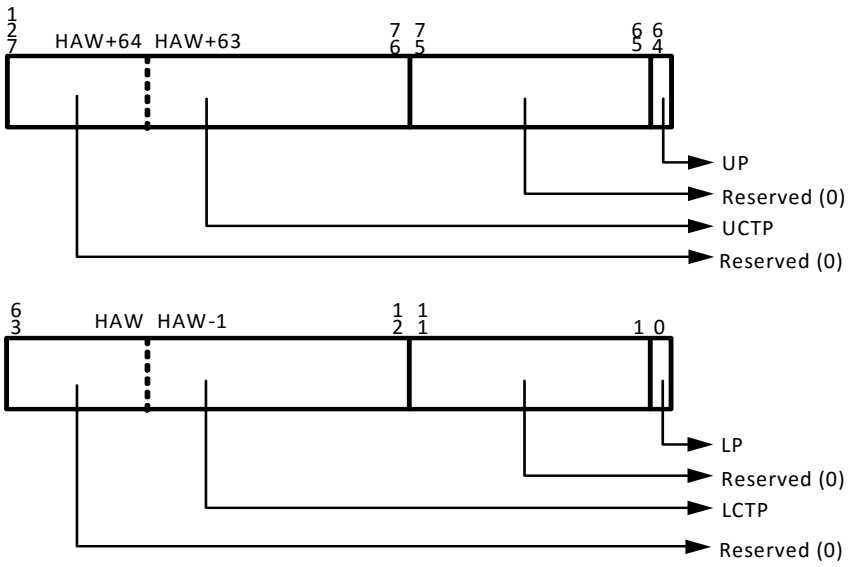


Figure 9-33. Extended-Root-Entry Format

Bits	Field	Description
127:76	UCTP: Upper Context Table Pointer	Pointer to upper extended-context-table for this bus. The upper extended-context-table is 4KB in size and size-aligned. Hardware treats bits 127:(HAW+64) as reserved (0), where HAW is the host address width of the platform.
75:65	R: Reserved	Reserved. Must be 0.
64	UP: Upper Present	This field indicates whether the upper-half of the extended root-entry is present. • 0: Indicates upper half of the extended-root-entry is not present. Bits 127:65 are ignored by hardware. • 1: Indicates the upper-half of the extended root-entry is present.
63:12	LCTP: Lower Context Table Pointer	Pointer to lower extended-context-table for this bus. The lower extended-context-table is 4KB in size and size-aligned. Hardware treats bits 63:HAW as reserved (0), where HAW is the host address width of the platform.
11:1	R: Reserved	Reserved. Must be 0.
0	LP: Lower Present	This field indicates whether the lower-half of the extended-root-entry is present. • 0: Indicates lower half of the extended root-entry is not present. Bits 63:1 are ignored by hardware. • 1: Indicates the lower-half of the extended root-entry is present.

9.3 Context Entry

The following figure and table describe the context-entry. Context-entries support translation of requests-without-PASID. Context-entries are referenced through root-entries described in Section 9.1.

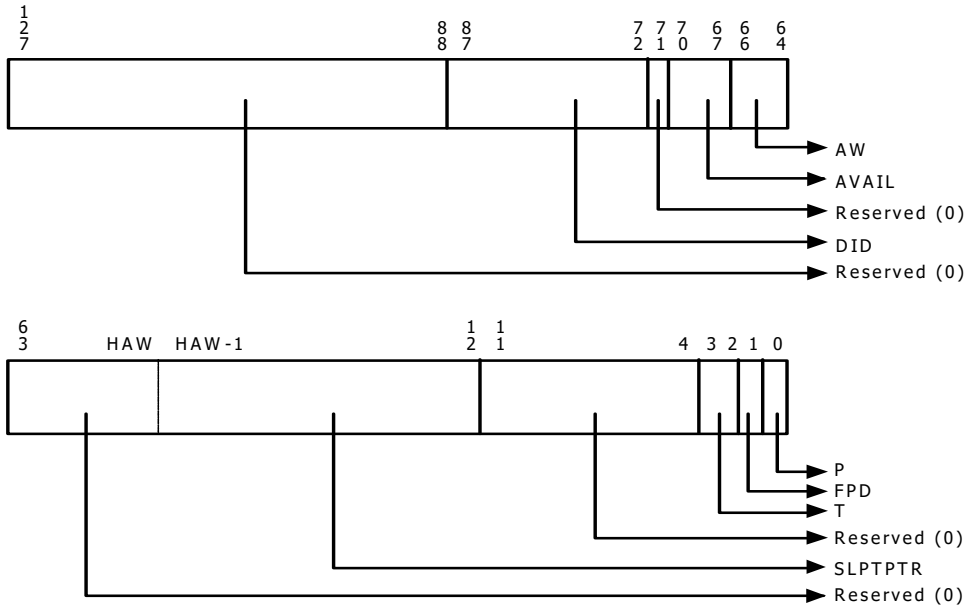


Figure 9-34. Context-Entry Format

Bits	Field	Description
127:88	R: Reserved	Reserved. Must be 0.
87:72	DID: Domain Identifier	Identifier for the domain to which this context-entry maps. Hardware may use the domain identifier to tag its internal caches. The Capability Register reports the domain-id width supported by hardware. For implementations supporting less than 16-bit domain-ids, unused bits of this field are treated as reserved by hardware. For example, for implementation supporting 8-bit domain-ids, bits 87:80 of this field are treated as reserved. Context-entries programmed with the same domain identifier must always reference same address translation (SLPTPTR field). Context-entries referencing same address translation are recommended to be programmed with same domain id for hardware efficiency. When Caching Mode (CM) field in Capability Register is reported as Set, the domain-id value of zero is architecturally reserved. Software must not use domain-id value of zero when CM is Set.
71	R: Reserved	Reserved. Must be 0.
70:67	IGN: Ignored	Hardware ignores the programming of this field.
66:64	AW: Address Width	When the Translation-type (T) field is 00b or 01b, this field indicates the adjusted guest-address-width (AGAW) to be used by hardware for the second-level page-table walk. The following encodings are defined for this field: • 000b: Reserved • 001b: 39-bit AGAW (3-level page table) • 010b: 48-bit AGAW (4-level page table) • 011b: 57-bit AGAW (5-level page table) • 100b-111b: Reserved The value specified in this field must match an AGAW value supported by hardware (as reported in the SAGAW field in the Capability Register). When the Translation-type (T) field indicates pass-through processing (10b), this field must be programmed to indicate the largest AGAW value supported by hardware. Untranslated requests-without-PASID processed through this context-entry and accessing addresses above 2^X-1 (where X is the AGAW value indicated by this field) are blocked and treated as translation faults.
63:12	SLPTPTR: Second Level Page Translation Pointer	When the Translation-Type (T) field is 00b or 01b, this field points to the base of second-level paging entries (described in Section 9.8). Hardware treats bits 63:HAW as reserved (0), where HAW is the host address width of the platform. This field is ignored by hardware when Translation-Type (T) field is 10b (pass-through).
11:4	R: Reserved	Reserved. Must be 0.

Bits	Field	Description
3:2	T: Translation Type	• 00b: Untranslated requests are translated using second-level paging structures referenced through SLPTPTR field. Translated requests and Translation Requests are blocked. • 01b: Untranslated, Translated and Translation Requests are supported. This encoding is treated as reserved by hardware implementations not supporting Device-TLBs (DT=0 in Extended Capability Register). • 10b: Untranslated requests are processed as pass-through. SLPTPTR field is ignored by hardware. Translated and Translation Requests are blocked. This encoding is treated by hardware as reserved for hardware implementations not supporting Pass Through (PT=0 in Extended Capability Register). • 11b: Reserved. This field is applicable only for requests-without-PASID. Refer to Translation Type (T) field in extended-context-entry in Section 9.4 for handling of requests-with-PASID.
1	FPD: Fault Processing Disable	Enables or disables recording/reporting of faults caused by DMA requests processed through this context-entry: • 0: Indicates fault recording/reporting is enabled for DMA requests processed through this context-entry. • 1: Indicates fault recording/reporting is disabled for DMA requests processed through this context-entry. This field is evaluated by hardware irrespective of the setting of the present (P) field.
0	P: Present	• 0: Indicates the context-entry is not present. All other fields except Fault Processing Disable (FPD) field are ignored by hardware. • 1: Indicates the context-entry is present.

9.4 Extended-Context-Entry

The following figure and table describe the extended-context-entry. Extended-context-entries support translation of requests without and with PASID. Extended-context-entries are referenced through extended-root-entries described in Section 9.2.

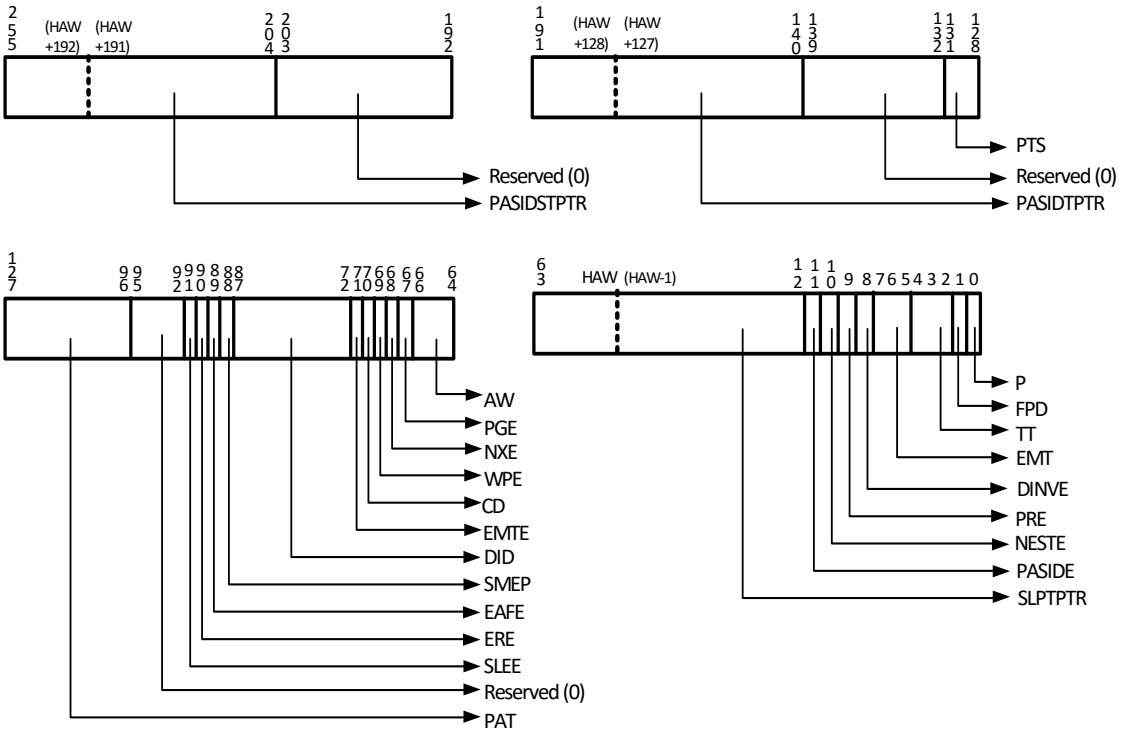
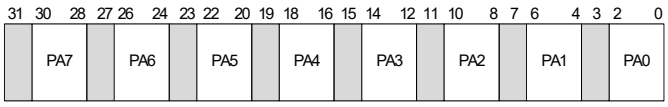


Figure 9-35. Extended-Context-Entry Format

Bits	Field	Description
255:204	PASIDSTPTR: PASID State Table Pointer	This field is ignored when Deferred Invalidate Enable (DINVE) field is Clear. This field points to the base of the PASID-state table. Hardware treats upper (52-HAW) bits in this field as reserved (0), where HAW is the host address width of the platform. When Nested Translation Enable (NESTE) field is Set, this field is treated as Guest Physical Address (GPA) and translated through the second-level translation. Section 9.6 describes format of entries in PASID-state table.
203:192	R: Reserved	Reserved. Must be 0.
191:140	PASIDPTR: PASID Table Pointer	This field is ignored when PASID Enable (PASIDE) field is Clear. This field points to the base of the PASID-table. Hardware treats upper (52-HAW) bits in this field as reserved (0), where HAW is the host address width of the platform. When Nested Translation Enable (NESTE) field is Set, this field is treated as Guest Physical Address (GPA) and translated through the second-level translation. Section 9.5 describes format of entries in PASID-table.
139:132	R: Reserved	Reserved. Must be 0.
131:128	PTS: PASID Table Size	This field is ignored when PASID Enable (PASIDE) field is Clear. Value of X in this field indicates PASID-table with $2^{(X+5)}$ entries. When Deferred Invalidate Enable (DINVE) field is Set, this field also specifies size of the PASID-state table.
127:96	PAT: Page Attribute Table	This field is treated as Reserved(0) for implementations not supporting Memory Type (MTS=0 in Extended Capability Register). This field is ignored when PASID Enable (PASIDE) field is Clear. When PASIDE is Set, this field is used to compute memory-type for requests-with-PASID from devices that operate in processor coherency domain. Refer to Section 3.6.5, and Section 3.8.4 for hardware handling of memory-type for requests-with-PASID with first-level and nested translation respectively. The format of this field is specified below.  Following encodings are specified for the value programmed in sub-fields PA0 through PA7. **Encoding** / **Mnemonic** 00h — Uncacheable (UC) 01h — Write Combining (WC) 02h — Reserved 03h — Reserved 04h — Write Through (WT) 05h — Write Protected (WP) 06h — Write Back (WB) 07h — Uncached (UC-) 08h-0Fh — Reserved

Bits	Field	Description
95:92	R: Reserved	Reserved. Must be 0.
91	SLEE: Second-Level Execute Enable	This field is ignored when Nested Translation Enable (NESTE) or Execute Requests Enable (ERE) field is Clear. • 0: Instruction fetch allowed from any guest-physical-address with read permissions. • 1: Instruction fetch can be prevented from specified guest-physical-addresses through the X flag, bit 2, in second-level paging entries (even if data reads from such addresses are allowed).
90	ERE: Execute Requests Enable	This field is treated as Reserved(0) for implementations not supporting Execute Requests (ERS=0 in the Extended Capability Register). This field is ignored when PASID Enable (PASIDE) is Clear. If Clear, requests-with-PASID requesting execute permission are blocked and treated as DMA remapping faults.
89	EAFE: Extended Accessed Flag Enable	This field is treated as Reserved(0) for implementations not supporting Extended Accessed flag (EAFS=0 in the Extended Capability Register). This field is ignored when PASID Enable (PASIDE) field is Clear. If 1, Extended-Accessed (EA) flag is atomically set in any first-level paging-entries referenced by remapping hardware through this Extended-context-entry.
88	SMEP: Supervisor Mode Execute Protection	This field is ignored when PASID Enable (PASIDE) or Execute Requests Enable (ERE) field is Clear, or when Supervisor Requests Enable (SRE) in PASID-entry used to process request-with-PASID is Clear. If Set, supervisor-level execute requests from a user-mode page (a page that has every first-level translation entry leading up to the page has the U/S flag Set) are blocked and treated as DMA remapping faults.
87:72	DID: Domain Identifier	Identifier for the domain to which this extended-context-entry maps. Hardware may use the domain identifier to tag its internal caches. The Capability Register reports the domain-id width supported by hardware. For implementations supporting less than 16-bit domain-ids, unused bits of this field are treated as reserved by hardware. For example, for an implementation supporting 8-bit domain-ids, bits 87:80 of this field are treated as reserved. Extended-context-entries programmed with same domain identifier must always reference same second-level translation (SLPTPTR field) and same PASID table (PASIDPTR field, when PASIDE=1). Extended-context-entries referencing same second-level translation (and referencing same PASID table, when PASIDE=1) are recommended to be programmed with the same domain id for hardware efficiency. When Caching Mode (CM) field in Capability Register is reported as Set, the domain-id value of zero is architecturally reserved. Software must not use domain-id value of zero when CM is Set.
71	EMTE: Extended Memory Type Enable	This field is treated as Reserved(0) for implementations not supporting Nested Translations or Memory Type (NEST=0 or MTS=0 in Extended Capability Register). This field is ignored when Nested Translation Enable (NESTE) field is Clear. • 0: Extended Memory Type (EMT) field in extended-context-entry and in second-level leaf paging-entries are ignored. • 1: Extended Memory Type (EMT) field in extended-context-entry and in second-level leaf paging-entries are used for memory type determination for requests-with-PASID from devices operating in processor coherency domain. Refer to Section 3.8.4 for hardware handling of memory-type for requests-with-PAID with nested translation.

Bits	Field	Description

70	CD: Cache Disable	This field is treated as Reserved(0) for implementations not supporting Memory Type (MTS=0 in Extended Capability Register).
		This field is only applicable for requests from devices that operate in processor coherency domain.
		• 0: Normal Cache Mode.
		  • Read hits access cache; Read misses may cause replacement.
		  • Write hits update cache; Write misses cause cache line fill.
		  • Writes to shared lined and write misses update system memory.
		  • Write hits can change shared lines to modified under control of MTRR registers or EMT field, with associated read invalidation cycle.
		• 1: Cache is disabled.
		  • Effective memory-type forced to Uncacheabe (UC), irrespective of programming of MTRR registers and PAT/EMT fields.
		  • Cache continues to respond to snoop traffic.

69	WPE: Write Protect Enable	This field is ignored when PASID Enable (PASIDE) field is Clear or when Supervisor Requests Enable (SRE) field in PASID-entry used to process request-with-PASID is Clear. • 0: Allows supervisor-level accesses to write into read-only pages, regardless of the U/S flag setting in the first-level paging entries. • 1: Inhibits supervisor-level accesses from writing into read-only pages.
68	NXE: No Execute Enable	This field is ignored when PASID Enable (PASIDE) or Execute Requests Enable (ERE) field is Clear. • 0: Instruction fetch allowed from any linear address with read permissions. • 1: Instruction fetch can be prevented from specified linear addresses through the XD flag, bit 63, in first-level paging entries (even if data reads from such addresses are allowed).
67	PGE: Page Global Enable	This field is ignored when PASID Enable (PASIDE) is Clear. • 0: Disable global page feature. • 1: Enable global page feature. Global page feature allows frequently used or shared pages to be marked as global across PASIDs (done with setting Global (G) flag, bit 8, in first-level leaf paging-entries).
66:64	AW: Address Width	This field indicates the adjusted guest-address-width (AGAW) to be used by hardware for the second-level page-table walk. The following encodings are defined for this field: • 000b: Reserved • 001b: 39-bit AGAW (3-level page table) • 010b: 48-bit AGAW (4-level page table) • 011b: 57-bit AGAW (5-level page table) • 100b-111b: Reserved The value specified in this field must match an AGAW value supported by hardware (as reported in the SAGAW field in the Capability Register). When the Translation-type (T) field indicates pass-through (010b) or guest-mode (100b or 101b), this field must be programmed to indicate the largest AGAW value supported by hardware. Untranslated requests-without-PASID to addresses above 2^X-1 (where X is the AGAW value indicated by this field) are blocked and treated as translation faults.
63:12	SLPTPTR: Second-Level Page Translation Pointer	This field points to the base of the second-level page-tables (described in Section 9.8). Hardware treats bits 63:HAW as reserved (0), where HAW is the host address width of the platform.

Bits	Field	Description
11	PASIDE: PASID Enable	This field is treated as Reserved(0) for implementations not supporting PASID (PASID=0 in Extended Capability Register). • 0: Requests with PASID are blocked. • 1: Requests with PASID are processed per programming of Translation Type (T) and Nested Translation Enable (NESTE) fields.
10	NESTE: Nested Translation Enable	This field is treated as Reserved(0) for implementations not supporting Nested Translations (NEST=0 in Extended Capability Register). This field is ignored when PASID Enable (PASIDE) is Clear. • 0: Requests remapped through PASID table (referenced through PASIDPTPTR field) are subject to first-level translation only. First-level page-tables are referenced through FLPTPTR field in the PASID-table. • 1: Requests remapped through PASID table referenced through PASIDPTPTR field) are subject to nested first-level and second-level translation. first-level page-tables are referenced through FLPTPTR field in the PASID-table, and second-level page-tables are referenced through SLPTPTR field in the extended-context-entry. Refer to Section 3.8 for hardware behavior for nested translations.
9	PRE: Page Request Enable	This field is treated as Reserved(0) for implementations not supporting Page Requests (PRS=0 in Extended Capability Register) and when the Translation-Type (T) field value does not support Device-TLBs (000b, 010b,or 100b). • 0: Page Requests to report recoverable address translations faults are not enabled. • 1: Page Requests to report recoverable address translation faults are enabled.
8	DINVE: Deferred Invalidate Enable	This field is treated as Reserved(0) for implementations not supporting Deferred Invalidations (DIS=0 in Extended Capability Register). This field is ignored when PASID Enable (PASIDE) is Clear. • 0: PASID-state update messages are blocked. • 1: PASID-state update messages are processed through the PASID-state table referenced through the PASIDSTPTR field.
7:5	EMT: Extended Memory Type	This field is ignored when Extended Memory Type Enable (EMTE) field is Clear. When EMTE field is Set, this field is used to compute effective memory-type for nested translation of requests-with-PASID from devices operating in the processor coherency domain. Refer to Section 3.8.4 for hardware handling of memory-type with nested translations. The encodings defined for this field are 0h for Uncacheable (UC), and 6h for Write Back (WB). All other values are Reserved. (contd.)

Bits	Field	Description

4:2	T: Translation Type	• 000b: Host mode with Device-TLBs disabled
		  • Untranslated requests-without-PASID are remapped using the second-level page-table referenced through SLPTPTR field.
		  • If PASIDE field is Set, Untranslated requests-with-PASID are remapped using the PASID Table referenced through PASIDPTPTR field.
		  • Translation requests (with or without PASID), and Translated Requests are blocked.
		• 001b: Host mode with Device-TLBs enabled
		  • Untranslated requests-without-PASID and Translation requests-without-PASID are remapped using the second-level page-table referenced through SLPTPTR field.
		  • If PASIDE field is Set, Untranslated requests-with-PASID and Translation requests-with-PASID are remapped using the PASID Table referenced through PASIDPTPTR field.
		  • Translated requests bypass address translation.
		• 010b: Pass-through mode
		  • This encoding is treated by hardware as Reserved for hardware implementations not supporting Pass-through (PT=0 in the Extended Capability Register).
		  • Untranslated requests-without-PASID bypass address translation and are processed as passthrough. SLPTPTR field is ignored by hardware.
		  • Untranslated requests-with-PASID, Translation requests (with or without PASID), and Translated requests are blocked.
		• 011b: Reserved
		• 100b: Guest mode with Device-TLBs disabled
		  • This encoding is treated by hardware as Reserved for hardware implementations not supporting PASID (PASID=0 in Extended Capability Register), or when PASID Enable (PASIDE) field in this extended-context-entry is Clear.
		  • Untranslated requests-without-PASID bypass address translation and are processed as passthrough. SLPTPTR field is ignored by hardware.
		  • Untranslated requests-with-PASID are remapped using the PASID Table referenced through PASIDPTPTR field.
		  • Translation requests (with or without PASID) and Translated requests are blocked.
		• 101b: Guest mode with Device-TLBs enabled
		  • This encoding is treated by hardware as Reserved for hardware implementations not supporting PASID (PASID=0 in Extended Capability Register), or when PASID Enable (PASIDE) field in this extended-context-entry is Clear.
		  • Untranslated requests-without-PASID bypass address translation and are processed as passthrough. SLPTPTR field is ignored by hardware.
		  • Translation requests-without-PASID are responded with Untranslated access only bit Set (U=1) along with read and write permissions (R=W=1). SLPTPTR field is ignored by hardware.
		  • Untranslated requests-with-PASID, and Translation requests-with-PASID are remapped using the PASID Table referenced through PASIDPTPTR field.
		  • Translated requests bypass address translation.
		• 110b: Reserved
		• 111b: Reserved

Bits	Field	Description
1	FPD: Fault Processing Disable	Enables or disables recording/reporting of non-recoverable faults caused by requests processed through this extended-context-entry: • 0: Non-recoverable faults are recorded/reported for requests processed through this extended-context-entry. • 1: Non-recoverable faults are not recorded/reported for requests processed through this extended-context-entry. This field is evaluated by hardware irrespective of the setting of the present (P) field.
0	P: Present	• 0: Indicates the extended-context-entry is not present. All other fields except Fault Processing Disable (FPD) field are ignored by hardware. • 1: Indicates the extended-context-entry is present.

9.5 PASID Entry

The following figure and table describe entries in PASID-table used to translate requests-with-PASID.

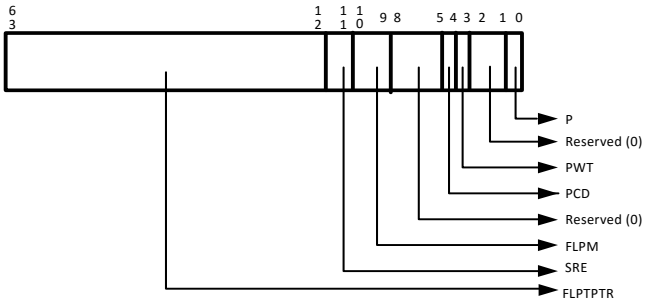


Figure 9-36. PASID Entry Format

Bits	Field	Description
63:12	FLPTPTR: First Level Page Translation Pointer	Pointer to root of first-level paging structures (base of FL-PML4 table if FLPM=00b; base of FL-PML5 table if FLPM=01b). Refer to Section 9.7 for first-level paging structure details. Hardware treats bits 63:HAW as reserved (0), where HAW is the host address width of the platform. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context-entry.
11	SRE: Supervisor Requests Enable	This field is treated as Reserved(0) for implementations not supporting supervisor-mode privilege (SRS=0 in the Extended Capability Register). If Clear, requests-with-PASID requesting supervisor-level privilege level are blocked and treated as DMA remapping faults.
10:9	FLPM: First-level Paging Mode	This field specifies the paging mode for first-level translation. • 00: 4-level paging (FLPTPR is base of FL-PML4) • 01: 5-level paging (FLPTPR is base of FL-PML5) • 10-11: Reserved For implementations reporting 5-level Paging as not supported (5LP = 0) in Capability Register, this field must be programmed as 00b.
8:5	R: Reserved	Reserved (0).
4	PCD: Page-level Cache Disable	This field is treated as Reserved(0) for implementations reporting Memory Type Support (MTS) as Clear in Extended Capability Register. This field is used along with PWT field to determine the memory-type for accessing PML4 (if FLPM=00b) or PML5 (if FLPM=01b) entries of first-level paging structures, when processing requests-with-PASID from devices operating in the processor coherency domain.
3	PWT: Page-level Write Through	This field is treated as Reserved(0) for implementations reporting Memory Type Support (MTS) as Clear in Extended Capability Register. This field is used along with PCD field to determine the memory-type for accessing PML4 (if FLPM=00b) or PML5 (if FLPM=01b) entries of first-level paging structures, when processing requests-with-PASID from devices operating in the processor coherency domain.
2:1	R: Reserved	Reserved (0).
0	P: Present	• 0: Indicates this PASID-entry is not valid. H/W ignores programming of other fields. • 1: Indicates PASID-entry is valid.

9.6 PASID-State Entry

The following figure and table describe the PASID-state entry structures used to process PASID-state update requests. PASID-state entries enable software to implement IOTLB and Device-TLB invalidation optimizations. Refer Section Section 6.5.4 for details on deferred TLB invalidation support.

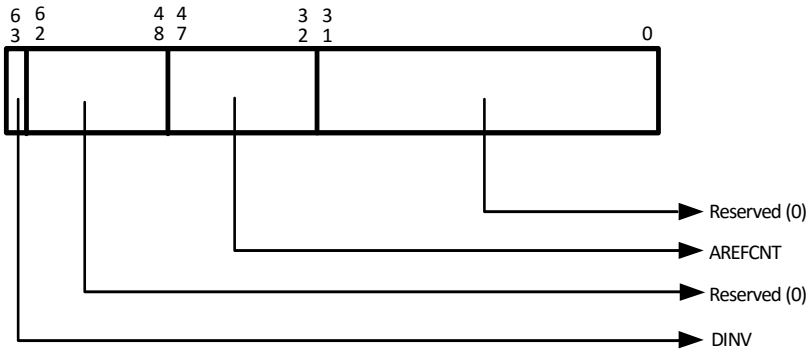


Figure 9-37. PASID-State Entry Format

Bits	Field	Description
63	DINV: Deferred Invalidate	Software Sets this field to request Deferred invalidation of IOTLB and Device-TLB entries for the corresponding PASID. Hardware atomically reads and Clears this field when processing PASID-state update requests for respective PASID activation, and returns the value read as the 'Synchronize-Flag' in the PASID-state update response.
62:48	R: Reserved	Reserved (0).
47:32	AREFCNT: Active Reference Count	Non-zero value in this field indicates one or more agents at the endpoint device are actively accessing the address space of PASID corresponding to this PASID-state entry. Hardware atomically increments this field when processing PASID-state update requests for respective PASID activation. Hardware atomically decrements this field when processing PASID-state update requests for respective PASID de-activation.
31:0	R: Reserved	Reserved (0).

9.7 First-Level Paging Entries

The following figure and table describe the first-level paging structures used to process requests-with-PASID. First-level paging entries follows Intel® 64 processor's 64-bit paging entry format, with the following additions:

- **Extended-Accessed flag**: First-level translation supports an Extended-Accessed (EA) flag in the paging entries. The EA flag works similar to the Accessed (A) flag in the paging entries. EA flags can be enabled through the Extended-Accessed-Flag-Enable (EAFE) field in the Extended-Context-entry (see Section 9.4). When enabled, bit 10 in the first-level paging entries are treated by hardware as the EA flag, and are atomically set whenever a paging-entry is accessed by hardware. For software usages where the first-level paging structures are shared across heterogeneous agents (e.g., CPUs and GPUs), EA flag may be used by software to identify pages accessed by non-CPU agent(s) (as opposed to the A flag which indicates access by any agent sharing the paging structures).

63	62:61	60:59	58:57	56:55	54:53	52:51	HAW	HAW-1:32	31:30	29:28	27:26	25:24	23:22	21:20	19:18	17:16	15:14	13:12	11	10	9	8	7	6	5	4	3	2	1	0	
XD	Ignored						Rsvd.	Address of PML4 table											Ign	EA[1]	Ign		Rsvd	Ign	A	PCD	PWT	U/S	R/W	1	PML5E
XD	Ignored						Rsvd.	Address of page-directory-pointer table											Ign	EA	Ign		Rsvd	Ign	A	PCD	PWT	U/S	R/W	1	PML4E
XD	Ignored						Rsvd.	Address of 1GB page frame									Reserved		PAT	Ign	EA	Ign	G	1	D	A	PCD	PWT	U/S	R/W	1	PDPE: 1GB page[2]
XD	Ignored						Rsvd.	Address of page directory											Ign	EA	Ign		0	Ign	A	PCD	PWT	U/S	R/W	1	PDPE: page directory
XD	Ignored						Rsvd.	Address of 2MB page frame									Reserved		PAT	Ign	EA	Ign	G	1	D	A	PCD	PWT	U/S	R/W	1	PDE: 2MB page
XD	Ignored						Rsvd.	Address of page table											Ign	EA	Ign		0	Ign	A	PCD	PWT	U/S	R/W	1	PDE: page table
XD	Ignored						Rsvd.	Address of 4KB page frame											Ign	EA	Ign	G	PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page

1. EA field is ignored by hardware if EAFS is reported as Clear in the Extended Capability Register of if EAFE=0 in the Extended-Context-entry referencing the first-level paging entries.
2. 1-GByte page support is reported through First-level 1-GByte Page Support (FL1GPS) field in the Capability Register.

Figure 9-38. Format for First-Level Paging Entries

Table 18. Format of PML5E that references a PML4 Table

Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the 512-GByte region controlled by this entry when XD=1.
62:52	IGN: Ignored	Ignored by hardware.
51:HAW	R: Reserved	Reserved (0).
(HAW-1) :12	ADDR: Address	Physical address of 4-KByte aligned PML4 table referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context-entry.
11	IGN: Ignored	Ignored by hardware.
10	EA: Extended Accessed	If EAFE=1 in the relevant Extended-context-entry, this bit indicates whether this entry has been used for address translation. Refer to Section 3.6.3 for extended-accessed bit handling. If EAFE=0 in the relevant Extended-context-entry, this bit is ignored.
9:8	IGN: Ignored	Ignored by hardware.
7	R: Reserved	Reserved (0).
6	IGN: Ignored	Ignored by hardware.
5	A: Accessed	Indicates whether this entry has been used for address translation. Refer to Section 3.6.3 for accessed bit handling.
4	PCD: Page-level Cache Disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the PML4 table referenced by this entry. For other devices, this field is ignored. Refer to Section 3.6.5 for memory-type handling.
3	PWT: Page-level Write Through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the PML4 table referenced by this entry. For other devices, this field is ignored. Refer to Section 3.6.5 for memory-type handling.
2	U/S: User/Supervisor	If 0, requests with user-level privilege are not allowed to the 256-TByte region controlled by this entry. Refer to Section 3.6.2 for access rights.
1	R/W: Read/Write	If 0, write permission not granted for requests with user-level privilege (and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry) to the 256-TByte region controlled by this entry. Refer to Section 3.6.2 for access rights.
0	P: Present	Must be 1 to reference a PML4 table.

Table 19. Format of PML4E that references a Page-Directory-Pointer Table

Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the 512-GByte region controlled by this entry when XD=1.
62:52	IGN: Ignored	Ignored by hardware.
51:HAW	R: Reserved	Reserved (0).
(HAW-1) :12	ADDR: Address	Physical address of 4-KByte aligned page-directory-pointer table referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context-entry.
11	IGN: Ignored	Ignored by hardware.
10	EA: Extended Accessed	If EAFE=1 in the relevant Extended-context-entry, this bit indicates whether this entry has been used for address translation. Refer to Section 3.6.3 for extended-accessed bit handling. If EAFE=0 in the relevant Extended-context-entry, this bit is ignored.
9:8	IGN: Ignored	Ignored by hardware.
7	R: Reserved	Reserved (0).
6	IGN: Ignored	Ignored by hardware.
5	A: Accessed	Indicates whether this entry has been used for address translation. Refer to Section 3.6.3 for accessed bit handling.
4	PCD: Page-level Cache Disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page-directory-pointer table referenced by this entry. For other devices, this field is ignored. Refer to Section 3.6.5 for memory-type handling.
3	PWT: Page-level Write Through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page-directory-pointer table referenced by this entry. For other devices, this field is ignored. Refer to Section 3.6.5 for memory-type handling.
2	U/S: User/Supervisor	If 0, requests with user-level privilege are not allowed to the 512-GByte region controlled by this entry. Refer to Section 3.6.2 for access rights.
1	R/W: Read/Write	If 0, write permission not granted for requests with user-level privilege (and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry) to the 512-GByte region controlled by this entry. Refer to Section 3.6.2 for access rights.
0	P: Present	Must be 1 to reference a page-directory-pointer table.

Table 20. Format of PDPE that maps a 1-GByte Page

Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the 1-GByte page referenced by this entry when XD=1.
62:52	IGN: Ignored	Ignored by hardware.
51:HAW	R: Reserved	Reserved (0).
(HAW-1):30	ADDR: Address	Physical address of 1-GByte page referenced by this entry. This field is treated as Guest Physical Address (GPA) when nested translations are enabled (NESTE=1) in the relevant extended-context-entry.
29:13	R: Reserved	Reserved (0).
12	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the 1-GByte page referenced by this entry. For other devices, this field is ignored. Refer to Section 3.6.5 for memory-type handling.
11	IGN: Ignored	Ignored by hardware.
10	EA: Extended Accessed	If EAFE=1 in the relevant Extended-context-entry, this bit indicates whether this entry has been used for address translation. Refer to Section 3.6.3 for extended-accessed bit handling. If EAFE=0 in the relevant Extended-context-entry, this bit is ignored.
9	IGN: Ignored	Ignored by hardware.
8	G: Global	If PGE=1 in the relevant extended-context-entry, this field can be Set by software to indicate the 1-GByte page translation is global.
7	PS: Page Size	Must be 1 (otherwise this entry references a page directory. Refer to Table 21).
6	D: Dirty	If 1, indicates one or more requests seeking write permission was successfully translated to the 1-GByte page referenced by this entry.
5	A: Accessed	Indicates whether this entry has been used for address translation. Refer to Section 3.6.3 for accessed bit handling.
4	PCD: Page-level Cache Disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the 1-GByte page referenced by this entry. For other devices, this field is ignored. Refer to Section 3.6.5 for memory-type handling.
3	PWT: Page-level Write Through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the 1-GByte page referenced by this entry. For other devices, this field is ignored. Refer to Section 3.6.5 for memory-type handling.
2	U/S: User/Supervisor	If 0, requests with user-level privilege are not allowed to the 1-GByte page referenced by this entry. Refer to Section 3.6.5 for access rights.
1	R/W: Read/Write	If 0, write permission not granted for requests with user-level privilege (and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry) to the 1-GByte page referenced by this entry. Refer to Section 3.6.5 for access rights.
0	P: Present	Must be 1 to map a 1-GByte page.

Table 21. Format of PDPE that references a Page-Directory Table

Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the 1-GByte region controlled by this entry when XD=1.
62:52	IGN: Ignored	Ignored by hardware.
51:HAW	R: Reserved	Reserved (0).
(HAW-1):12	ADDR: Address	Physical address of 4-KByte aligned page directory referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context-entry.
11	IGN: Ignored	Ignored by hardware.
10	EA: Extended Accessed	If EAFE=1 in the relevant Extended-context-entry, this bit indicates whether this entry has been used for address translation. Refer to Section 3.6.3 for extended-accessed bit handling. If EAFE=0 in the relevant Extended-context-entry, this bit is ignored.
9:8	IGN: Ignored	Ignored by hardware.
7	PS: Page Size	Must be 0 (otherwise this entry maps to a 1-GByte page. Refer to Table 20).
6	IGN: Ignored	Ignored by hardware.
5	A: Accessed	Indicates whether this entry has been used for address translation. Refer to Section 3.6.3 for accessed bit handling.
4	PCD: Page-level Cache Disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory referenced by this entry. For other devices, this field is ignored. Refer to Section 3.6.5 for memory-type handling.
3	PWT: Page-level Write Through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory referenced by this entry. For other devices, this field is ignored. Refer to Section 3.6.5 for memory-type handling.
2	U/S: User/Supervisor	If 0, requests with user-level privilege are not allowed to the 1-GByte region controlled by this entry. Refer to Section 3.6.2 for access rights.
1	R/W: Read/Write	If 0, write permission not granted for requests with user-level privilege (and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry) to the 1-GByte region controlled by this entry. Refer to Section 3.6.2 for access rights. for access rights.
0	P: Present	Must be 1 to reference a page directory.

Table 22. Format of PDE that maps a 2-MByte Page

Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the 2-MByte page referenced by this entry when XD=1.
62:52	IGN: Ignored	Ignored by hardware.
51:HAW	R: Reserved	Reserved (0).
(HAW-1):21	ADDR: Address	Physical address of 2-MByte page referenced by this entry. This field is treated as Guest Physical Address (GPA) when nested translations are enabled (NESTE=1) in the relevant extended-context-entry.
20:13	R: Reserved	Reserved (0).
12	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the 2-MByte page referenced by this entry. For other devices, this field is ignored. Refer to Section 3.6.5 for access rights. for memory-type handling.
11	IGN: Ignored	Ignored by hardware.
10	EA: Extended Accessed	If EAFE=1 in the relevant Extended-context-entry, this bit indicates whether this entry has been used for address translation. Refer to Section 3.6.3 for access rights. for extended-accessed bit handling. If EAFE=0 in the relevant Extended-context-entry, this bit is ignored.
9	IGN: Ignored	Ignored by hardware.
8	G: Global	If PGE=1 in the relevant extended-context-entry, this field can be Set by software to indicate the 2-MByte page translation is global.
7	PS: Page Size	Must be 1 (otherwise this entry references a page table. Refer to Table 23).
6	D: Dirty	If 1, indicates one or more requests seeking write permission was successfully translated to the 2-MByte page referenced by this entry. Refer to Section 3.6.3 for dirty bit handling.
5	A: Accessed	Indicates whether this entry has been used for address translation. Refer to Section 3.6.3 for accessed bit handling.
4	PCD: Page-level Cache Disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the 2-MByte page referenced by this entry. For other devices, this field is ignored. Refer to Section 3.6.5 for memory-type handling.
3	PWT: Page-level Write Through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the 2-MByte page referenced by this entry. For other devices, this field is ignored. Refer to Section 3.6.5 for memory-type handling.
2	U/S: User/Supervisor	If 0, requests with user-level privilege are not allowed to the 2-MByte page referenced by this entry. Refer to Section 3.6.2 for access rights.
1	R/W: Read/Write	If 0, write permission not granted for requests with user-level privilege (and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry) to the 2-MByte page referenced by this entry. Refer to Section 3.6.2 for access rights.
0	P: Present	Must be 1 to map a 2-MByte page.

Table 23. Format of PDE that references a Page Table

Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the 2-MByte region controlled by this entry when XD=1.
62:52	IGN: Ignored	Ignored by hardware.
51:HAW	R: Reserved	Reserved (0).
(HAW-1):12	ADDR: Address	Physical address of 4-KByte aligned page table referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context-entry.
11	IGN: Ignored	Ignored by hardware.
10	EA: Extended Accessed	If EAFE=1 in the relevant Extended-context-entry, this bit indicates whether this entry has been used for address translation. Refer to Section 3.6.3 for extended-accessed bit handling. If EAFE=0 in the relevant Extended-context-entry, this bit is ignored.
9:8	IGN: Ignored	Ignored by hardware.
7	PS: Page Size	Must be 0 (otherwise this entry maps to a 2-MByte page. Refer to Table 22).
6	IGN: Ignored	Ignored by hardware.
5	A: Accessed	Indicates whether this entry has been used for address translation. Refer to Section 3.6.3 for accessed bit handling.
4	PCD: Page-level Cache Disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page table referenced by this entry. For other devices, this field is ignored. Refer to Section 3.6.5 for memory-type handling.
3	PWT: Page-level Write Through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page table referenced by this entry. For other devices, this field is ignored. Refer to Section 3.6.5 for memory-type handling.
2	U/S: User/Supervisor	If 0, requests with user-level privilege are not allowed to the 2-MByte region controlled by this entry. Refer to Section 3.6.2 for access rights.
1	R/W: Read/Write	If 0, write permission not granted for requests with user-level privilege (and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry) to the 2-MByte region controlled by this entry. Refer to Section 3.6.2 for access rights.
0	P: Present	Must be 1 to reference a page table.

Table 24. Format of PTE that maps a 4-KByte Page

Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the 4-KByte page referenced by this entry when XD=1.
62:52	IGN: Ignored	Ignored by hardware.
51:HAW	R: Reserved	Reserved (0).
(HAW-1):12	ADDR: Address	Physical address of 4-KByte page referenced by this entry. This field is treated as Guest Physical Address (GPA) when nested translations are enabled (NESTE=1) in the relevant extended-context-entry.
11	IGN: Ignored	Ignored by hardware.
10	EA: Extended Accessed	If EAFE=1 in the relevant Extended-context-entry, this bit indicates whether this entry has been used for address translation. Refer to Section 3.6.3 for extended-accessed bit handling. If EAFE=0 in the relevant Extended-context-entry, this bit is ignored.
9	IGN: Ignored	Ignored by hardware.
8	G: Global	If PGE=1 in the relevant extended-context-entry, this field can be Set by software to indicate the 4-KByte page translation is global.
7	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the 4-KByte page referenced by this entry. For other devices, this field is ignored. Refer to Section 3.6.5 for memory-type handling.
6	D: Dirty	If 1, indicates one or more requests seeking write permission was successfully translated to the 4-KByte page referenced by this entry. Refer to Section 3.6.3 for dirty bit handling
5	A: Accessed	Indicates whether this entry has been used for address translation. Refer to Section 3.6.3 for accessed bit handling.
4	PCD: Page-level Cache Disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the 4-KByte page referenced by this entry. For other devices, this field is ignored. Refer to Section 3.6.5 for memory-type handling.
3	PWT: Page-level Write Through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the 4-KByte page referenced by this entry. For other devices, this field is ignored. Refer to Section 3.6.5 for memory-type handling.
2	U/S: User/Supervisor	If 0, requests with user-level privilege are not allowed to the 4-KByte page referenced by this entry. Refer to Section 3.6.2 for access rights.
1	R/W: Read/Write	If 0, write permission not granted for requests with user-level privilege (and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry) to the 4-KByte page referenced by this entry. Refer to Section 3.6.2 for access rights.
0	P: Present	Must be 1 to map a 4-KByte page.

9.8 Second-Level Paging Entries

The following figure and table describe the page-table entry.

63	62	61	60	59	58	57	56	55	54	53	52	51	HAW	HAW-1	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Ign	Rsvd	Ignored											Rsvd.	Address of Second-level-PML4 table																							Rsvd	Ign		Rsvd	Ign					X[1]	W	R	SL-PML5E
Ign	Rsvd	Ignored											Rsvd.	Address of Second-level-page-directory-pointer table																							Rsvd	Ign		Rsvd	Ign					X	W	R	SL-PML4E
Ign	TM	Ignored											Rsvd.	Address of 1GB page frame										Reserved													SNP	Ign		1	IPAT[3]		EMT[2]			X	W	R	SL-PDPE: 1GB page[3]
Ign	Rsvd	Ignored											Rsvd.	Address of second-level-page-directory table																							Rsvd	Ign		0	Ign					X	W	R	SL-PDPE: page directory
Ign	TM	Ignored											Rsvd.	Address of 2MB page frame																Reserved							SNP	Ign.		1	IPAT		EMT			X	W	R	SL-PDE: 2MB page[5]
Ign	Rsvd	Ignored											Rsvd.	Address of second-level-page table																							Rsvd	Ign		0	Ign					X	W	R	SL-PDE: page table
Ign	TM	Ignored											Rsvd.	Address of 4KB page frame																							SNP	Ign			IPAT		EMT			X	W	R	SL-PTE: 4KB page

1. X field is ignored by hardware if Execute Request Support (ERS) is reported as Clear in the Extended Capability Register or if SLEE=0 in the extended-context-entry referencing the second-level paging entries.
2. EMT and IPAT fields are ignored by hardware if Memory Type Support (MTS) is reported as Clear in the Extended Capability Register or if EMTE=0 in the extended-context-entry referencing the second-level paging entries.
3. 1-GByte page and 2-MByte page support is reported through Second-level Large Page Support (SLLPS) in the Capability Register.

Figure 9-39. Format for Second-Level Paging Entries

Table 25. Format of SL-PML5E referencing a Second-Level-PML4 Table

Bits	Field	Description
63	IGN: Ignored	Ignored by hardware.
62	R: Reserved	Reserved (0).
61:52	IGN: Ignored	Ignored by hardware.
51:HAW	R: Reserved	Reserved (0).
(HAW-1):12	ADDR: Address	Host physical address of 4-KByte aligned second-level-PML4 table referenced by this entry.
11	R: Reserved	Reserved (0).
10:8	IGN: Ignored	Ignored by hardware.
7	R: Reserved	Reserved (0).
6:3	IGN: Ignored	Ignored by hardware.
2	X: Execute	If SLEE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the 256-TByte region referenced by this entry, when X=0.
1	W: Write	If 0, write permission not granted for requests to the 256-TByte region controlled by this entry.
0	R: Read	If 0, read permission not granted for requests to the 256-TByte region controlled by this entry. For implementations reporting Zero-Length-Read (ZLR) field as Set in the Capability Register, read permission is not applicable to zero-length read requests if Execute (X) or Write (W) permission fields are 1.

Table 26. Format of SL-PML4E referencing a Second-Level-Page-Directory-Pointer Table

Bits	Field	Description
63	IGN: Ignored	Ignored by hardware.
62	R: Reserved	Reserved (0).
61:52	IGN: Ignored	Ignored by hardware.
51:HAW	R: Reserved	Reserved (0).
(HAW-1):12	ADDR: Address	Host physical address of 4-KByte aligned second-level-page-directory-pointer table referenced by this entry.
11	R: Reserved	Reserved (0).
10:8	IGN: Ignored	Ignored by hardware.
7	R: Reserved	Reserved (0).
6:3	IGN: Ignored	Ignored by hardware.
2	X: Execute	If SLEE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the 512-GByte region referenced by this entry, when X=0.
1	W: Write	If 0, write permission not granted for requests to the 512-GByte region controlled by this entry.
0	R: Read	If 0, read permission not granted for requests to the 512-GByte region controlled by this entry. For implementations reporting Zero-Length-Read (ZLR) field as Set in the Capability Register, read permission is not applicable to zero-length read requests if Execute (X) or Write (W) permission fields are 1.

Table 27. Format of SL-PDPE that maps a 1-GByte Page

Bits	Field	Description
63	IGN: Ignored	Ignored by hardware.
62	TM: Transient Mapping	This field is treated as reserved(0) by hardware implementations not supporting Device-TLBs (DT=0 in Extended Capability Register). Translation requests processed through this entry returns this field as the U-field (*Untranslated access only*) in the Translation-Completion Data in the response.
61:52	IGN: Ignored	Ignored by hardware.
51:HAW	R: Reserved	Reserved (0).
(HAW-1):30	ADDR: Address	Host physical address of 1-GByte page referenced by this entry.
29:12	R: Reserved	Reserved (0).
11	SNP: Snoop	This field is treated as reserved(0) by hardware implementations not supporting Snoop Control (SC=0 in Extended Capability Register). If 1, Untranslated requests-without-PASID processed through this entry always snoop processor caches irrespective of attributes (such as Non-Snoop (NS) attribute) in the request. Translation requests-without-PASID processed through this entry returns this field as the 'N' (*Non-snooped accesses*) field in Translation-Completion Data in the response. This field is ignored for requests-with-PASID. Untranslated requests-with-PASID always snoop processor caches. Translation requests-with-PASID always return N=1 (i.e., Non-snooped accesses not allowed) in the Translation-completion Data in the response.
10:8	IGN: Ignored	Ignored by hardware.
7	PS: Page Size	Must be 1 (Otherwise this entry references a second-level-page-directory. Refer to Table 28).
6	IPAT: Ignore PAT	This field is ignored by hardware when Extended Memory Type Enable (EMTE) field is Clear in the relevant extended-context-entry. This field is applicable only for nested translation of requests-with-PASID from devices operating in the processor coherency domain. • 0: Page Attribute Table (PAT) in extended-context-entry is used for effective memory-type determination • 1: Page Attribute Table (PAT) in extended-context-entry is not used for effective memory-type determination. Refer to Section 3.8.4 for memory type handling.
5:3	EMT: Extended Memory Type	This field is ignored by hardware when Extended Memory Type Enable (EMTE) field is Clear in the relevant extended-context-entry. This field is applicable only for nested translation of requests-with-PASID from devices operating in the processor coherency domain. The encodings defined for this field are 0h for Uncacheable (UC), 1h for Write Combining (WC), 4h for Write Through (WT), 5h for Write Protected (WP), and 6h for Write Back (WB). All other values are Reserved. Refer to Section 3.8.4 for memory type handling.
2	X: Execute	If SLEE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the 1-GByte page referenced by this entry, when X=0.
1	W: Write	If 0, write permission not granted for requests to the 1-GByte page referenced by this entry.
0	R: Read	If 0, read permission not granted for requests to the 1-GByte page referenced by this entry. For implementations reporting Zero-Length-Read (ZLR) field as Set in the Capability Register, read permission is not applicable to zero-length read requests if Execute (X) or Write (W) permission fields are 1.

Table 28. Format of SL-PDPE that references a Second-Level-Page-Directory

Bits	Field	Description
63	IGN: Ignored	Ignored by hardware.
62	R: Reserved	Reserved (0).
61:52	IGN: Ignored	Ignored by hardware.
51:HAW	R: Reserved	Reserved (0).
(HAW-1):12	ADDR: Address	Host physical address of 4-KByte aligned second-level-page-directory referenced by this entry.
11	R: Reserved	Reserved (0).
10:8	IGN: Ignored	Ignored by hardware.
7	PS: Page Size	Must be 0 (Otherwise this entry references a 1-GByte page. Refer to Table 27).
6:3	IGN: Ignored	Ignored by hardware.
2	X: Execute	If SLEE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the 1-GByte region referenced by this entry, when X=0.
1	W: Write	If 0, write permission not granted for requests to the 1-GByte region controlled by this entry.
0	R: Read	If 0, read permission not granted for requests to the 1-GByte region controlled by this entry. For implementations reporting Zero-Length-Read (ZLR) field as Set in the Capability Register, read permission is not applicable to zero-length read requests if Execute (X) or Write (W) permission fields are 1.

Table 29. Format of SL-PDE that maps to a 2-MByte Page

Bits	Field	Description
63	IGN: Ignored	Ignored by hardware.
62	TM: Transient Mapping	This field is treated as reserved(0) by hardware implementations not supporting Device-TLBs (DT=0 in Extended Capability Register). Translation requests processed through this entry returns this field as the U-field (*Untranslated access only*) in the Translation-Completion Data in the response.
61:52	IGN: Ignored	Ignored by hardware.
51:HAW	R: Reserved	Reserved (0).
(HAW-1):21	ADDR: Address	Host physical address of 2-MByte page referenced by this entry.
20:12	R: Reserved	Reserved (0).
11	SNP: Snoop	This field is treated as reserved(0) by hardware implementations not supporting Snoop Control (SC=0 in Extended Capability Register). If 1, Untranslated requests-without-PASID processed through this entry always snoop processor caches irrespective of attributes (such as Non-Snoop (NS) attribute) in the request. Translation requests-without-PASID processed through this entry returns this field as the 'N' (*Non-snooped accesses*) field in Translation-Completion Data in the response. This field is ignored for requests-with-PASID. Untranslated requests-with-PASID always snoop processor caches. Translation requests-with-PASID always return N=1 (i.e., Non-snooped accesses not allowed) in the Translation-Completion Data in the response.
10:8	IGN: Ignored	Ignored by hardware.
7	PS: Page Size	Must be 1 (Otherwise this entry references a second-level-page table. Refer to Table 30).
6	IPAT: Ignore PAT	This field is ignored by hardware when Extended Memory Type Enable (EMTE) field is Clear in the relevant extended-context-entry. This field is applicable only for nested translation of requests-with-PASID from devices operating in the processor coherency domain. • 0: Page Attribute Table (PAT) in extended-context-entry is used for effective memory-type determination • 1: Page Attribute Table (PAT) in extended-context-entry is not used for effective memory-type determination. Refer to Section 3.8.4 for memory type handling.
5:3	EMT: Extended Memory Type	This field is ignored by hardware when Extended Memory Type Enable (EMTE) field is Clear in the relevant extended-context-entry. This field is applicable only for nested translation of requests-with-PASID from devices operating in the processor coherency domain. The encodings defined for this field are 0h for Uncacheable (UC), 1h for Write Combining (WC), 4h for Write Through (WT), 5h for Write Protected (WP), and 6h for Write Back (WB). All other values are Reserved. Refer to Section 3.8.4 for memory type handling.
2	X: Execute	If SLEE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the 2-MByte page referenced by this entry, when X=0.
1	W: Write	If 0, write permission not granted for requests to the 2-MByte page referenced by this entry.
0	R: Read	If 0, read permission not granted for requests to the 2-MByte page referenced by this entry. For implementations reporting Zero-Length-Read (ZLR) field as Set in the Capability Register, read permission is not applicable to zero-length read requests if Execute (X) or Write (W) permission fields are 1.

Table 30. Format of SL-PDE that references a Second-Level-Page Table

Bits	Field	Description
63	IGN: Ignored	Ignored by hardware.
62	R: Reserved	Reserved (0).
61:52	IGN: Ignored	Ignored by hardware.
51:HAW	R: Reserved	Reserved (0).
(HAW-1):12	ADDR: Address	Host physical address of 4-KByte aligned second-level-page-table referenced by this entry.
11	R: Reserved	Reserved (0).
10:8	IGN: Ignored	Ignored by hardware.
7	PS: Page Size	Must be 0 (Otherwise this entry references a 2-MByte page. Refer to Table 29).
6:3	IGN: Ignored	Ignored by hardware.
2	X: Execute	If SLEE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the 2-MByte region referenced by this entry, when X=0.
1	W: Write	If 0, write permission not granted for requests to the 2-MByte region controlled by this entry.
0	R: Read	If 0, read permission not granted for requests to the 2-MByte region controlled by this entry. For implementations reporting Zero-Length-Read (ZLR) field as Set in the Capability Register, read permission is not applicable to zero-length read requests if Execute (X) or Write (W) permission fields are 1.

Table 31. Format of SL-PTE that maps 4-KByte Page

Bits	Field	Description
63	IGN: Ignored	Ignored by hardware.
62	TM: Transient Mapping	This field is treated as reserved(0) by hardware implementations not supporting Device-TLBs (DT=0 in Extended Capability Register). Translation requests processed through this entry returns this field as the U-field (*Untranslated access only*) in the Translation-Completion Data in the response.
61:52	IGN: Ignored	Ignored by hardware.
51:HAW	R: Reserved	Reserved (0).
(HAW-1):12	ADDR: Address	Host physical address of 4-KByte page referenced by this entry.
11	SNP: Snoop	This field is treated as reserved(0) by hardware implementations not supporting Snoop Control (SC=0 in Extended Capability Register). If 1, Untranslated requests-without-PASID processed through this entry always snoop processor caches irrespective of attributes (such as Non-Snoop (NS) attribute) in the request. Translation requests-without-PASID processed through this entry returns this field as the 'N' (*Non-snooped accesses*) field in Translation-Completion Data in the response. This field is ignored for requests-with-PASID. Untranslated requests-with-PASID always snoop processor caches. Translation requests-with-PASID always return N=1 (i.e., Non-snooped accesses not allowed) in the Translation-Completion Data in the response.
10:7	IGN: Ignored	Ignored by hardware.
6	IPAT: Ignore PAT	This field is ignored by hardware when Extended Memory Type Enable (EMTE) field is Clear in the relevant extended-context-entry. This field is applicable only for nested translation of requests-with-PASID from devices operating in the processor coherency domain. • 0: Page Attribute Table (PAT) in extended-context-entry is used for effective memory-type determination • 1: Page Attribute Table (PAT) in extended-context-entry is not used for effective memory-type determination. Refer to Section 3.8.4 for memory type handling.
5:3	EMT: Extended Memory Type	This field is ignored by hardware when Extended Memory Type Enable (EMTE) field is Clear in the relevant extended-context-entry. This field is applicable only for nested translation of requests-with-PASID from devices operating in the processor coherency domain. The encodings defined for this field are 0h for Uncacheable (UC), 1h for Write Combining (WC), 4h for Write Through (WT), 5h for Write Protected (WP), and 6h for Write Back (WB). All other values are Reserved. Refer to Section 3.8.4 for memory type handling.
2	X: Execute	If SLEE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the 4-KByte page referenced by this entry, when X=0.
1	W: Write	If 0, write permission not granted for requests to the 4-KByte page referenced by this entry.
0	R: Read	If 0, read permission not granted for requests to the 4-KByte page referenced by this entry. For implementations reporting Zero-Length-Read (ZLR) field as Set in the Capability Register, read permission is not applicable to zero-length read requests if Execute (X) or Write (W) permission fields are 1.

9.9 Fault Record

The following figure and table describe the fault record format for advanced fault logging.

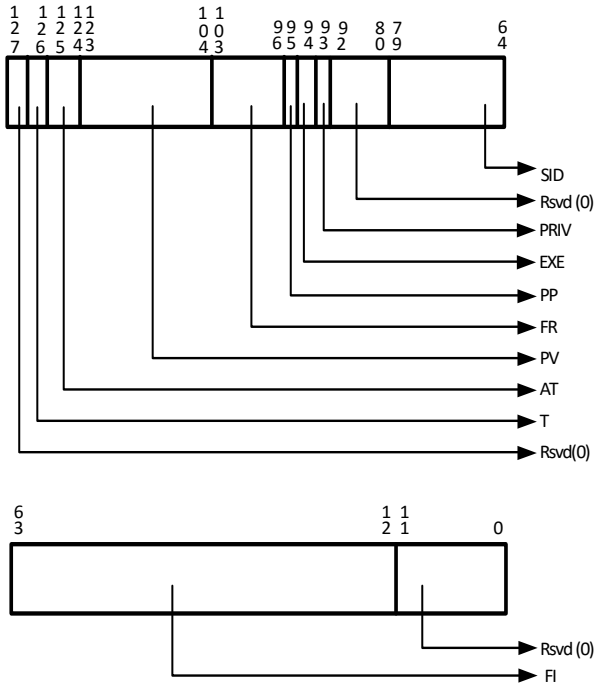


Figure 9-40. Fault-Record Format

Bits	Field	Description
127	R: Reserved	Reserved (0).
126	T: Type	Memory access type of faulted request. This field is valid only when the Fault Reason (FR) indicates one of the non-recoverable address translation fault conditions. • 0: Write request • 1: Read request or AtomicOp request This field is valid only when Fault Reason (FR) indicates one of the non-recoverable address translation fault conditions.
125:124	AT: Address Type	Address Type (AT) field in the faulted DMA request. This field is valid only when Fault Reason (FR) indicates one of the non-recoverable address translation fault conditions. Hardware implementations not suporting Device-TLBs (DT=0 in Extended Capability Register) treat this field as Reserved (0).
123:104	PV: PASID Value	PASID Value in the faulted request. This field is relevant only when the PASID Present (PP) field is Set. Hardware implementations not suporting PASID (PASID=0 in Extended Capability Register) treat this field as Reserved (0).
103:96	FR: Fault Reason	Reason for the non-recoverable fault.
95	PP: PASID Present	When Set, indicates faulted request has a PASID. The value of the PASID is reported in the PASID Value (PV) field. This field is relevant only when Fault Reason (FR) indicates one of the non-recoverable address translation fault conditions. Hardware implementations not suporting PASID (PASID=0 in Extended Capability Register) treat this field as Reserved (0).
94	EXE: Execute Permission Requested	When Set, indicates Execute permission was requested by the faulted read request. This field is relevant only when the PASID Present (PP) and Type (T) fields are both Set. Hardware implementations not suporting PASID (PASID=0 in Extended Capability Register) treat this field as Reserved (0).
93	PRIV: Privilege Mode Requested	When Set, indicates Supervisory privilege was requested by the faulted request. This field is relevant only when the PASID Present (PP) field is Set. Hardware implementations not suporting PASID (PASID=0 in Extended Capability Register) treat this field as Reserved (0).
92:80	R: Reserved	Reserved (0).
79:64	SID: Source Identifier	Requester-id associated with the fault condition.
63:12	FI: Fault Information	When the Fault Reason (FR) field indicates one of the non-recoverable address translation fault conditions, bits 63:12 of this field contains the page address in the faulted DMA request. When the Fault Reason (FR) field indicates one of the interrupt-remapping fault conditions, bits 63:48 of this field contains the interrupt_index computed for the faulted interrupt request, and bits 48:12 are cleared.
11:0	R: Reserved	Reserved (0).

9.10 Interrupt Remapping Table Entry (IRTE) for Remapped Interrupts

The following figure and table describe the interrupt remapping table entry for interrupt requests that are subject to remapping.

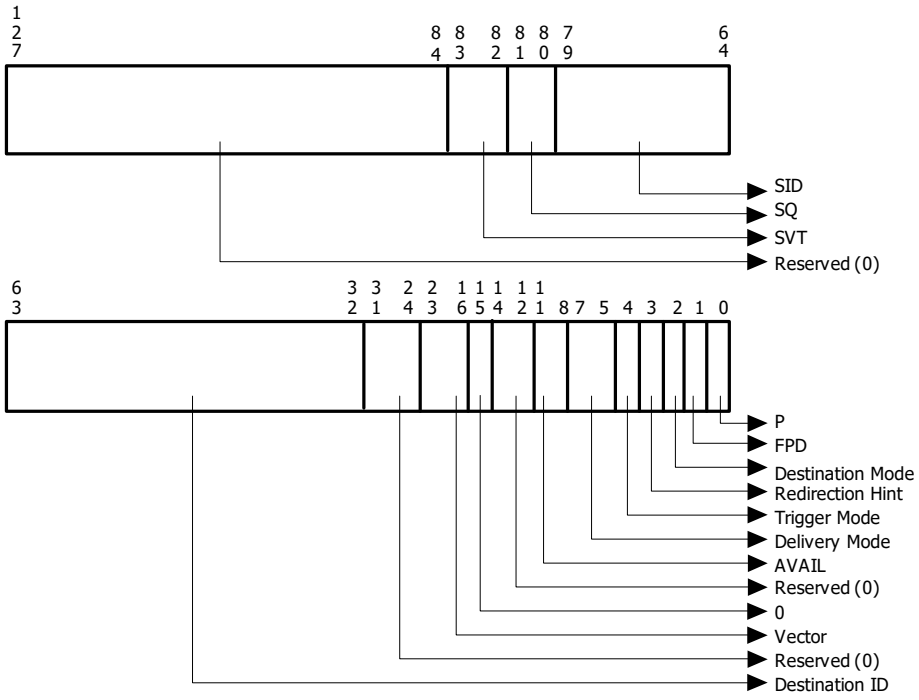


Figure 9-41. Interrupt Remap Table Entry Format for Remapped Interrupts

Bits	Field	Description
127:84	R: Reserved	Reserved. Software must program these bits to 0. This field is evaluated by hardware only when the Present (P) field is Set.
83:82	SVT: Source Validation Type	This field specifies the type of validation that must be performed by the interrupt-remapping hardware on the source-id of the interrupt requests referencing this IRTE. • 00b: No requester-id verification is required. • 01b: Verify requester-id in interrupt request using SID and SQ fields in the IRTE. • 10b: Verify the most significant 8-bits of the requester-id (Bus#) in the interrupt request is equal to or within the Startbus# and EndBus# specified through the upper and lower 8-bits of the SID field respectively. This encoding may be used to verify interrupts originated behind PCI-Express-to-PCI/PCI-X bridges. Refer Section 5.1.1 for more details. • 11b: Reserved. This field is evaluated by hardware only when the Present (P) field is Set.
81:80	SQ: Source-id Qualifier	The SVT field may be used to verify origination of interrupt requests generated by devices supporting phantom functions. If the SVT field is 01b, the following encodings are defined for the SQ field. • 00b: Verify the interrupt request by comparing all 16-bits of SID field with the 16-bit requester-id of the interrupt request. • 01b: Verify the interrupt request by comparing most significant 13 bits of the SID and requester-id of interrupt request, and comparing least significant two bits of the SID field and requester-id of interrupt request. (i.e., ignore the third least significant field of the SID field and requestor-id). • 10b: Verify the interrupt request by comparing most significant 13 bits of the SID and requester-id of interrupt request, and comparing least significant bit of the SID field and requester-id of interrupt request. (i.e., ignore the second and third least significant fields of the SID field and requestor-id). • 11b: Verify the interrupt request by comparing most significant 13 bits of the SID and requester-id of interrupt request. (i.e., ignore the least three significant fields of the SID field and requestor-id). This field is evaluated by hardware only when the Present (P) field is Set and SVT field is 01b.
79:64	SID: Source Identifier	This field specifies the originator (source) of the interrupt request that references this IRTE. The format of the SID field is determined by the programming of the SVT field. If the SVT field is: • 01b: The SID field contains the 16-bit requestor-id (Bus/Dev/Func #) of the device that is allowed to originate interrupt requests referencing this IRTE. The SQ field is used by hardware to determine which bits of the SID field must be considered for the interrupt request verification. • 10b: The most significant 8-bits of the SID field contains the startbus#, and the least significant 8-bits of the SID field contains the endbus#. Interrupt requests that reference this IRTE must have a requester-id whose bus# (most significant 8-bits of requester-id) has a value equal to or within the startbus# to endbus# range. This field is evaluated by hardware only when the Present (P) field is Set and SVT field is 01b or 10b.

Bits	Field	Description

63:32	DST: Destination ID	This field identifies the remapped interrupt request's target processor(s). It is evaluated by hardware only when the Present (P) field is Set.
		
		The format of this field in various processor and platform modes[1] is as follows:
		• Intel® 64 xAPIC Mode (Cluster):
		  • 63:48 - Reserved (0)
		  • 47:44 - APIC DestinationID[7:4]
		  • 43:40 - APIC DestinationID[3:0]
		  • 39:32 - Reserved (0)
		• Intel® 64 xAPIC Mode (Flat):
		  • 63:48 - Reserved (0)
		  • 47:40 - APIC DestinationID[7:0]
		  • 39:32 - Reserved (0)
		• Intel® 64 xAPIC Mode (Physical):
		  • 63:48 - Reserved (0)
		  • 47:40 - APIC DestinationID[7:0]
		  • 39:32 - Reserved (0)
		• Intel® 64 x2APIC Mode (Cluster):
		  • 63:32 - APIC DestinationID[31:0]
		• Intel® 64 x2APIC Mode (Physical):
		  • 63:32 - APIC DestinationID[31:0]

31:24	R: Reserved	Reserved. Software must program these bits to 0. This field is evaluated by hardware only when the Present (P) field is Set.
23:16	V: Vector	This 8-bit field contains the interrupt vector associated with the remapped interrupt request. This field is evaluated by hardware only when the Present (P) field is Set.
15	IM: IRTE Mode	Value of 0 in this field indicate interrupt requests processed through this IRTE are remapped. Value of 1 in this field indicate interrupt requests processed through this IRTE are posted. Refer to Section 9.11 for IRTE format for posted interrupts.
14:12	R: Reserved	Reserved. Software must program these bits to 0. This field is evaluated by hardware only when the Present (P) field is Set.
11:8	AVAIL: Available	This field is available to software. Hardware always ignores the programming of this field.

Bits	Field	Description
7:5	DLM: Delivery Mode	This 3-bit field specifies how the remapped interrupt is handled. Delivery Modes operate only in conjunction with specified Trigger Modes (TM). Correct Trigger Modes must be guaranteed by software. Restrictions are indicated below: • *000b (Fixed Mode)* – Deliver the interrupt to all the agents indicated by the Destination ID field. The Trigger Mode for fixed delivery mode can be edge or level. • *001b (Lowest Priority)* – Deliver the interrupt to one (and only one) of the agents indicated by the Destination ID field (the algorithm to pick the target agent is component specific and could include priority based algorithm). The Trigger Mode can be edge or level. • *010b (System Management Interrupt or SMI)*: SMI is an edge triggered interrupt regardless of the setting of the Trigger Mode (TM) field. For systems that rely on SMI semantics, the vector field is ignored, but must be programmed to all zeroes for future compatibility. (Support for this delivery mode is implementation specific. Platforms supporting interrupt remapping are expected to generate SMI through dedicated pin or platform-specific special messages)[2] • *100b (NMI)* – Deliver the signal to all the agents listed in the destination field. The vector information is ignored. NMI is an edge triggered interrupt regardless of the Trigger Mode (TM) setting. (Platforms supporting interrupt remapping are recommended to generate NMI through dedicated pin or platform-specific special messages)[2] • *101b (INIT)* – Deliver this signal to all the agents indicated by the Destination ID field. The vector information is ignored. INIT is an edge triggered interrupt regardless of the Trigger Mode (TM) setting. (Support for this delivery mode is implementation specific. Platforms supporting interrupt remapping are expected to generate INIT through dedicated pin or platform-specific special messages)[2] • *111b (ExtINT)* – Deliver the signal to the INTR signal of all agents indicated by the Destination ID field (as an interrupt that originated from an 8259A compatible interrupt controller). The vector is supplied by the INTA cycle issued by the activation of the ExtINT. ExtINT is an edge triggered interrupt regardless of the Trigger Mode (TM) setting. This field is evaluated by hardware only when the Present (P) field is Set.
4	TM: Trigger Mode	This field indicates the signal type of the interrupt that uses the IRTE. • 0: Indicates edge sensitive. • 1: Indicates level sensitive. This field is evaluated by hardware only when the Present (P) field is Set.
3	RH: Redirection Hint	This bit indicates whether the remapped interrupt request should be directed to one among N processors specified in Destination ID field, under hardware control. • 0: When RH is 0, the remapped interrupt is directed to the processor listed in the Destination ID field. • 1: When RH is 1, the remapped interrupt is directed to 1 of N processors specified in the Destination ID field. This field is evaluated by hardware only when the present (P) field is Set.
2	DM: Destination Mode	This field indicates whether the Destination ID field in an IRTE should be interpreted as logical or physical APIC ID. • 0: Physical • 1: Logical This field is evaluated by hardware only when the present (P) field is Set.

Bits	Field	Description
1	FPD: Fault Processing Disable	Enables or disables recording/reporting of faults caused by interrupt messages requests processed through this entry. • 0: Indicates fault recording/reporting is enabled for interrupt requests processed through this entry. • 1: Indicates fault recording/reporting is disabled for interrupt requests processed through this entry. This field is evaluated by hardware irrespective of the setting of the Present (P) field.
0	P: Present	The P field is used by software to indicate to hardware if the corresponding IRTE is present and initialized. • 0: Indicates the IRTE is not currently allocated to any interrupt sources. Block interrupt requests referencing this IRTE. • 1: Process interrupt requests referencing this IRTE per the programming of other fields in this IRTE.

1. The various processor and platform interrupt modes (like Intel® 64 xAPIC mode, Intel® 64 x2APIC mode) are determined by platform/processor specific mechanisms and are outside the scope of this specification.
2. Refer Section 5.1.9 for hardware considerations for handling platform events.

9.11 Interrupt Remapping Table Entry (IRTE) for Posted Interrupts

The following figure and table describe the interrupt remapping table entry for interrupt requests that are processed as posted.

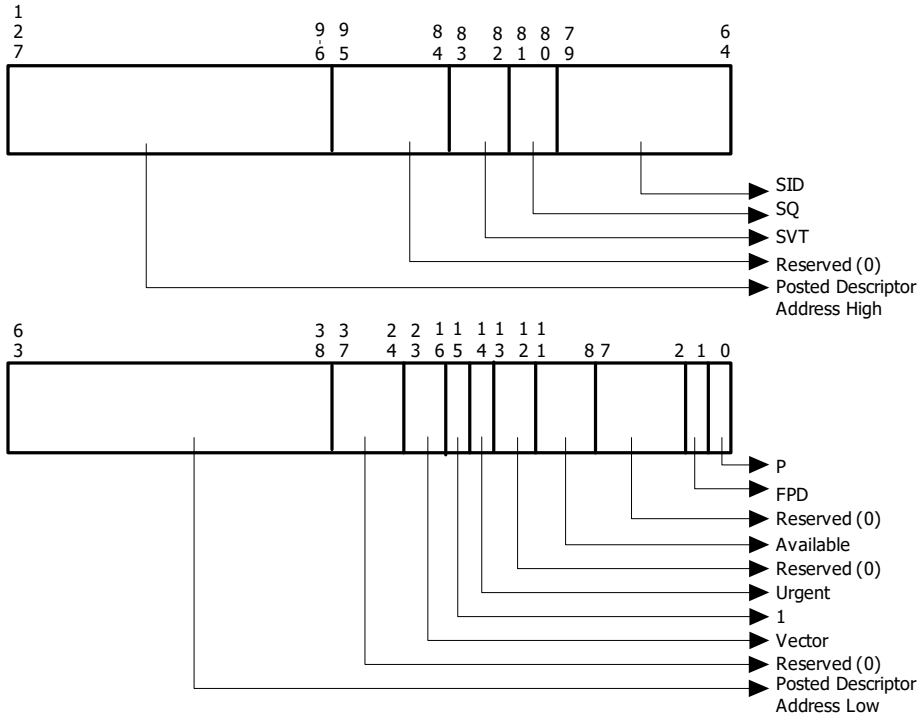


Figure 9-42. Interrupt Remap Table Entry Format for Posted Interrupts

Bits	Field	Description
127:96	PDAH: Posted Descriptor Address High	This field specifies address bits 63:32 of the 64-byte aligned Posted Interrupt Descriptor in memory used by hardware to post interrupt requests processed through this IRTE. Refer to Section 9.12 for Posted Interrupt Descriptor description. This field is evaluated by hardware only when the Present (P) field is Set. When evaluated, hardware treates upper address bits 63:HAW in this field as Reserved(0), where HAW is the Host Address Width of the platform.
95:84	R: Reserved	Reserved. Software must program these bits to 0. This field is evaluated by hardware only when the Present (P) field is Set.
83:82	SVT: Source Validation Type	This field specifies the type of validation that must be performed by the interrupt-remapping hardware on the source-id of the interrupt requests referencing this IRTE. • 00b: No requester-id verification is required. • 01b: Verify requester-id in interrupt request using SID and SQ fields in the IRTE. • 10b: Verify the most significant 8-bits of the requester-id (Bus#) in the interrupt request is equal to or within the Startbus# and EndBus# specified through the upper and lower 8-bits of the SID field respectively. This encoding may be used to verify interrupts originated behind PCI-Express-to-PCI/PCI-X bridges. Refer Section 5.1.1 for more details. • 11b: Reserved. This field is evaluated by hardware only when the Present (P) field is Set.
81:80	SQ: Source-id Qualifier	The SVT field may be used to verify origination of interrupt requests generated by devices supporting phantom functions. If the SVT field is 01b, the following encodings are defined for the SQ field. • 00b: Verify the interrupt request by comparing all 16-bits of SID field with the 16-bit requester-id of the interrupt request. • 01b: Verify the interrupt request by comparing most significant 13 bits of the SID and requester-id of interrupt request, and comparing least significant two bits of the SID field and requester-id of interrupt request. (i.e., ignore the third least significant field of the SID field and requestor-id). • 10b: Verify the interrupt request by comparing most significant 13 bits of the SID and requester-id of interrupt request, and comparing least significant bit of the SID field and requester-id of interrupt request. (i.e., ignore the second and third least significant fields of the SID field and requestor-id). • 11b: Verify the interrupt request by comparing most significant 13 bits of the SID and requester-id of interrupt request. (i.e., ignore the least three significant fields of the SID field and requestor-id). This field is evaluated by hardware only when the Present (P) field is Set and SVT field is 01b.
79:64	SID: Source Identifier	This field specifies the originator (source) of the interrupt request that references this IRTE. The format of the SID field is determined by the programming of the SVT field. If the SVT field is: • 01b: The SID field contains the 16-bit requestor-id (Bus/Dev/Func #) of the device that is allowed to originate interrupt requests referencing this IRTE. The SQ field is used by hardware to determine which bits of the SID field must be considered for the interrupt request verification. • 10b: The most significant 8-bits of the SID field contains the startbus#, and the least significant 8-bits of the SID field contains the endbus#. Interrupt requests that reference this IRTE must have a requester-id whose bus# (most significant 8-bits of requester-id) has a value equal to or within the startbus# to endbus# range. This field is evaluated by hardware only when the Present (P) field is Set and SVT field is 01b or 10b.

Bits	Field	Description
63:38	PDAL: Posted Descriptor Address Low	This field specifies address bits 31:6 of the 64-byte aligned Posted Interrupt Descriptor in memory sed by hardware to post interrupt requests processed through this IRTE. For optimal performance, software is recommended to reserve a full cacheline to host a Posted Interrupt Descriptor. Refer to Section 9.12 for Posted Interrupt Descriptor description. This field is evaluated by hardware only when the Present (P) field is Set.
37:24	R: Reserved	Reserved. Software must program these bits to 0. This field is evaluated by hardware only when the Present (P) field is Set.
23:16	VV: Virtual Vector	This 8-bit field contains the vector associated with the interrupt requests posted through this IRTE. This field is evaluated by hardware only when the Present (P) field is Set.
15	IM: IRTE Mode	Value of 1 in this field indicate interrupt requests processed through this IRTE are posted. Hardware implementations not suporting Posted Interrupts (PI=0 in Capability Register) treat this field as Reserved(0). Value of 0 in this field indicate interrupt requests processed through this IRTE are remapped. Refer to Section 9.10 for IRTE format for remapped interrupts.
14	URG: Urgent	This field indicates if the interrupt posted through this IRTE has latency sensitive processing requirements. • 0: Interrupt is not treated as urgent. • 1: Interrupt is treated as urgent. This field is evaluated by hardware only when the Present (P) field and IRTE Mode (IM) fields are both Set. Hardware implementations not suporting Posted Interrupts (PI=0 in Capability Register) treat this field as Reserved(0).
13:12	R: Reserved	Reserved. Software must program these bits to 0. This field is evaluated by hardware only when the Present (P) field is Set.
11:8	AVAIL: Available	This field is available to software. Hardware always ignores the programming of this field.
7:2	R: Reserved	Reserved. Software must program these bits to 0. This field is evaluated by hardware only when the Present (P) field is Set.
1	FPD: Fault Processing Disable	Enables or disables recording/reporting of faults caused by interrupt messages requests processed through this entry. • 0: Indicates fault recording/reporting is enabled for interrupt requests processed through this entry. • 1: Indicates fault recording/reporting is disabled for interrupt requests processed through this entry. This field is evaluated by hardware irrespective of the setting of the Present (P) field.
0	P: Present	The P field is used by software to indicate to hardware if the corresponding IRTE is present and initialized. • 0: Indicates the IRTE is not currently allocated to any interrupt sources. Block interrupt requests referencing this IRTE. • 1: Process interrupt requests referencing this IRTE per the programming of other fields in this IRTE.

9.12 Posted Interrupt Descriptor (PID)

The following figure and table describe the 64-byte aligned Posted Interrupt Descriptor. Software must allocate Posted Interrupt Descriptors is coherent (write-back) memory.

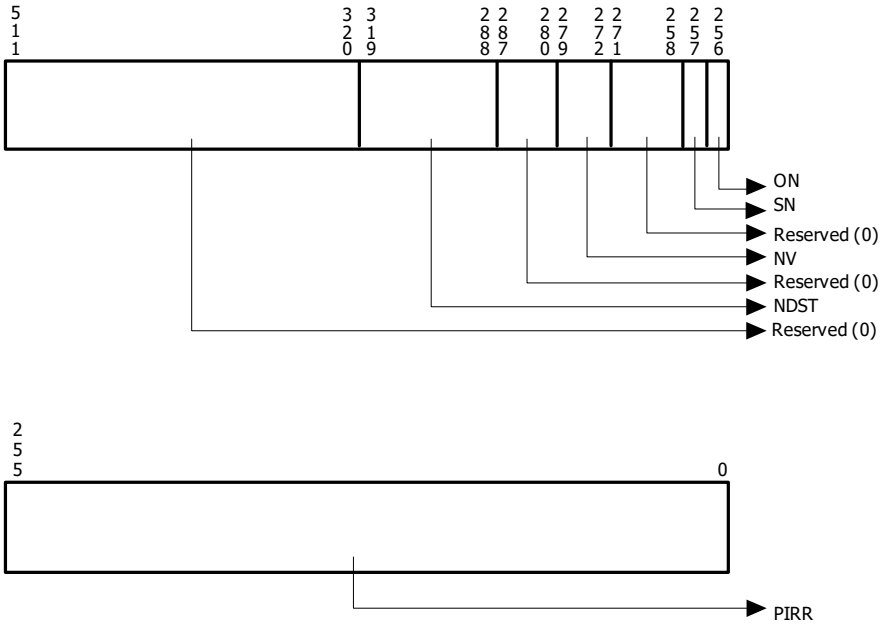


Figure 9-43. Posted Interrupt Descriptor Format

Bits	Field	Description
511:320	R: Reserved	Reserved. Software must program these bits to 0.

319:288	NDST: Notification Destination	This field specifies the Destination (Physical APIC-ID of the logical CPU) for the notification event. The format of this field is as follows:
		• Intel®64 xAPIC Mode (Physical):
		  • 319:304 - Reserved (0)
		  • 303:296 - APIC DestinationID[7:0]
		  • 295:288 - Reserved (0)
		• Intel®64 x2APIC Mode (Physical):
		  • 319:288 - APIC DestinationID[31:0]

287:280	R: Reserved	Reserved. Software must program these bits to 0.
279:272	NV: Notification Vector	This field specifies the physical vector used for the notification event. Notification events are issued as physical interrupts with trigger mode as Edge, and trigger mode level as Asserted.
271:258	R: Reserved	Reserved. Software must program these bits to 0.
257	SN: Suppress Notification	This field indicates if notification events must be suppressed when posting non-urgent interrupts to this descriptor. • 0: Do not suppress notification event. • 1: Suppress notification event. Non-urgent interrupts are interrupt requests processed through IRTE entries with IM field Set and URG field Clear. Refer to Section 9.11 for description of URG field in IRTE format for posted interrupts. If the value in this field is 1b at the time of hardware posting a non-urgent interrupt to PIR field, hardware functions as if the Outstanding Notification (ON) field value is 1b (i.e., hardware does not generate notification event nor modify the ON field). Refer to Section 5.2.3 on hardware operation for posting non-urgent interrupts.
256	ON: Outstanding Notification	This field indicates if a notification event is outstanding (pending processing by CPU or software) for this posted interrupt descriptor. • 0: No notification event outstanding for this descriptor. • 1: A notification event is outstanding for this descriptor. If this field is Clear at the time of hardware posting an interrupt to PIR field, hardware Sets it and generates a notification event. If this field is already Set at the time of hardware posting an interrupt to PIR field, notification event is not generated.
255:0	PIR: Posted Interrupt Requsts	This 256-bit field (one bit for each vector) provide storage for posted interrupts destined for a specific virtual processor. An interrupt request remapped through an IRTE for posted interrupt (IRTE with IM field Set) is considered posted by hardware when the bit corresponding to vector value in the IRTE is Set in this field. Refer to Section 5.2.3 on hardware operation for interrupt-posting.

This Page Is Left Intentionally Blank

10 Register Descriptions

This chapter describes the structure and use of the remapping registers.

10.1 Register Location

The register set for each remapping hardware unit in the platform is placed at a 4KB-aligned memory-mapped location. The exact location of the register region is implementation-dependent, and is communicated to system software by BIOS through the ACPI DMA-remapping hardware reporting structures (described in Chapter 8). For security, hardware implementations that support relocating these registers in the system address map must provide ability to lock its location by hardware specific secure initialization software.

10.2 Software Access to Registers

Software interacts with the remapping hardware by reading and writing its memory-mapped registers. The following requirements are defined for software access to these registers.

- Software is expected to access 32-bit registers as aligned doublewords. For example, to modify a field (e.g., bit or byte) in a 32-bit register, the entire doubleword is read, the appropriate field(s) are modified, and the entire doubleword is written back.
- Software must access 64-bit and 128-bit registers as either aligned quadwords or aligned doublewords. Hardware may disassemble a quadword register access as two double-word accesses. In such cases, hardware is required to complete the quad-word read or write request in a single clock in order (lower doubleword first, upper double-word second).
- When updating registers through multiple accesses (whether in software or due to hardware disassembly), certain registers may have specific requirements on how the accesses must be ordered for proper behavior. These are documented as part of the respective register descriptions.
- For compatibility with future extensions or enhancements, software must assign the last read value to all “Reserved and Preserved” (RsvdP) fields when written. In other words, any updates to a register must be read so that the appropriate merge between the RsvdP and updated fields will occur. Also, software must assign a value of zero for “Reserved and Zero” (RsvdZ) fields when written.
- Locked operations to remapping hardware registers are not supported. Software must not issue locked operations to access remapping hardware registers.

10.3 Register Attributes

The following table defines the attributes used in the remapping Registers. The registers are discussed in Section 10.4.

Attribute	Description
RW	Read-Write field that may be either set or cleared by software to the desired state.
RW1C	“Read-only status, Write-1-to-clear status” field. A read of the field indicates status. A set bit indicating a status may be cleared by writing a 1. Writing a 0 to an RW1C field has no effect.
RW1CS	“Sticky Read-only status, Write-1-to-clear status” field. A read of the field indicates status. A set bit indicating a status may be cleared by writing a 1. Writing a 0 to an RW1CS field has no effect. Not initialized or modified by hardware except on powergood reset.
RO	Read-only field that cannot be directly altered by software.
ROS	“Sticky Read-only” field that cannot be directly altered by software, and is not initialized or modified by hardware except on powergood reset.
WO	Write-only field. The value returned by hardware on read is undefined.
RsvdP	“Reserved and Preserved” field that is reserved for future RW implementations. Registers are read-only and must return 0 when read. Software must preserve the value read for writes.
RsvdZ	“Reserved and Zero” field that is reserved for future RW1C implementations. Registers are read-only and must return 0 when read. Software must use 0 for writes.

10.4 Register Descriptions

The following table summarizes the remapping hardware memory-mapped registers.

Offset	Register Name	Size	Description
000h	Version Register	32	Architecture version supported by the implementation.
004h	Reserved	32	Reserved
008h	Capability Register	64	Hardware reporting of capabilities.
010h	Extended Capability Register	64	Hardware reporting of extended capabilities.
018h	Global Command Register	32	Register controlling general functions.
01Ch	Global Status Register	32	Register reporting general status.
020h	Root Table Address Register	64	Register to set up location of root table.
028h	Context Command Register	64	Register to manage context-entry cache.
030h	Reserved	32	Reserved
034h	Fault Status Register	32	Register to report Fault/Error status
038h	Fault Event Control Register	32	Interrupt control register for fault events.
03Ch	Fault Event Data Register	32	Interrupt message data register for fault events.
040h	Fault Event Address Register	32	Interrupt message address register for fault event messages.
044h	Fault Event Upper Address Register	32	Interrupt message upper address register for fault event messages.
048h	Reserved	64	Reserved
050h	Reserved	64	Reserved
058h	Advanced Fault Log Register	64	Register to configure and manage advanced fault logging.
060h	Reserved	32	Reserved
064h	Protected Memory Enable Register	32	Register to enable DMA-protected memory region(s).
068h	Protected Low Memory Base Register	32	Register pointing to base of DMA-protected low memory region.
06Ch	Protected Low Memory Limit Register	32	Register pointing to last address (limit) of the DMA-protected low memory region.
070h	Protected High Memory Base Register	64	Register pointing to base of DMA-protected high memory region.
078h	Protected High Memory Limit Register	64	Register pointing to last address (limit) of the DMA-protected high memory region.
080h	Invalidation Queue Head	64	Offset to the invalidation queue entry that will be read next by hardware.
088h	Invalidation Queue Tail	64	Offset to the invalidation queue entry that will be written next by software.

Offset	Register Name	Size	Description
090h	Invalidation Queue Address Register	64	Base address of memory-resident invalidation queue.
098h	Reserved	32	Reserved
09Ch	Invalidation Completion Status Register	32	Register to indicate the completion of an Invalidation Wait Descriptor with IF=1.
0A0h	Invalidation Completion Event Control Register	32	Register to control Invalidation Queue Events
0A4h	Invalidation Completion Event Data Register	32	Invalidation Queue Event message data register for Invalidation Queue events.
0A8h	Invalidation Completion Event Address Register	32	Invalidation Queue Event message address register for Invalidation Queue events.
0ACh	Invalidation Completion Event Upper Address Register	32	Invalidation Queue Event message upper address register for Invalidation Queue events.
0B0h	Reserved	64	Reserved.
0B8h	Interrupt Remapping Table Address Register	64	Register indicating Base Address of Interrupt Remapping Table.
0C0h	Page Request Queue Head Register	64	Offset to the page request queue entry that will be processed next by software.
0C8h	Page Request Queue Tail Register	64	Offset to the page request queue entry that will be written next by hardware.
0D0h	Page Request Queue Address Register	64	Base address of memory-resident page request queue.
0D8h	Reserved	32	Reserved
0DCh	Page Request Status Register	32	Register to indicate one or more pending page requests in page request queue.
0E0h	Page Request Event Control Register	32	Register to control page request events.
0E4h	Page Request Event Data Register	32	Page request event message data register.
0E8h	Page Request Event Address Register	32	Page request event message address register
0ECh	Page Request Event Upper Address Register	32	Page request event message upper address register.
100h	MTRR Capability Register	64	Register for MTRR capability reporting.
108h	MTRR Default Type Register	64	Register to configure MTRR default type.
120h	Fixed-range MTRR Register for 64K_00000	64	Fixed-range memory type range register for 64K range starting at 00000h.
128h	Fixed-range MTRR Register for 16K_80000	64	Fixed-range memory type range register for 16K range starting at 80000h.
130h	Fixed-range MTRR Register for 16K_A0000	64	Fixed-range memory type range register for 16K range starting at A0000h.
138h	Fixed-range MTRR Register for 4K_C0000	64	Fixed-range memory type range register for 4K range starting at C0000h.
140h	Fixed-range MTRR Register for 4K_C8000	64	Fixed-range memory type range register for 4K range starting at C8000h.

Offset	Register Name	Size	Description
148h	Fixed-range MTRR Register for 4K_D0000	64	Fixed-range memory type range register for 4K range starting at D0000h.
150h	Fixed-range MTRR Register for 4K_D8000	64	Fixed-range memory type range register for 4K range starting at D8000h.
158h	Fixed-range MTRR Register for 4K_E0000	64	Fixed-range memory type range register for 4K range starting at E0000h.
160h	Fixed-range MTRR Register for 4K_E8000	64	Fixed-range memory type range register for 4K range starting at E8000h.
168h	Fixed-range MTRR Register for 4K_F0000	64	Fixed-range memory type range register for 4K range starting at F0000h.
170h	Fixed-range MTRR Register for 4K_F8000	64	Fixed-range memory type range register for 4K range starting at F8000h.
180h	Variable-range MTRR Base0	64	Variable-range memory type range0 base register.
188h	Variable-range MTRR Mask0	64	Variable-range memory type range0 mask register.
190h	Variable-range MTRR Base1	64	Variable-range memory type range1 base register.
198h	Variable-range MTRR Mask1	64	Variable-range memory type range1 mask register.
1A0h	Variable-range MTRR Base2	64	Variable-range memory type range2 base register.
1A8h	Variable-range MTRR Mask2	64	Variable-range memory type range2 mask register.
1B0h	Variable-range MTRR Base3	64	Variable-range memory type range3 base register.
1B8h	Variable-range MTRR Mask3	64	Variable-range memory type range3 mask register.
1C0h	Variable-range MTRR Base4	64	Variable-range memory type range4 base register.
1C8h	Variable-range MTRR Mask4	64	Variable-range memory type range4 mask register.
1D0h	Variable-range MTRR Base5	64	Variable-range memory type range5 base register.
1D8h	Variable-range MTRR Mask5	64	Variable-range memory type range5 mask register.
1E0h	Variable-range MTRR Base6	64	Variable-range memory type range6 base register.
1E8h	Variable-range MTRR Mask6	64	Variable-range memory type range6 mask register.
1F0h	Variable-range MTRR Base7	64	Variable-range memory type range7 base register.
1F8h	Variable-range MTRR Mask7	64	Variable-range memory type range7 mask register.
200h	Variable-range MTRR Base8	64	Variable-range memory type range8 base register.

Offset	Register Name	Size	Description
208h	Variable-range MTRR Mask8	64	Variable-range memory type range8 mask register.
210h	Variable-range MTRR Base9	64	Variable-range memory type range9 base register.
218h	Variable-range MTRR Mask9	64	Variable-range memory type range9 mask register.
XXXh	IOTLB Registers[1]	64	IOTLB registers consists of two 64-bit registers. Section 10.4.8 describes the format of the registers.
YYYh	Fault Recording Registers [n][1]	128	Registers to record the translation faults. The starting offset of the fault recording registers is reported through the Capability Register.

1. Hardware implementations may place IOTLB registers and fault recording registers in any unused or reserved addresses in the 4KB register space, or place them in adjoined 4KB regions. If one or more adjunct 4KB regions are used, unused addresses in those pages must be treated as reserved by hardware. Location of these registers is implementation dependent, and software must read the Capability Register to determine their offset location.

10.4.1 Version Register

Figure 10-44. Version Register

Abbreviation	VER_REG
General Description	Register to report the architecture version supported. Backward compatibility for the architecture is maintained with new revision numbers, allowing software to load remapping hardware drivers written for prior architecture versions.
Register Offset	000h

Bits	Access	Default	Field	Description
31:8	RsvdZ	0h	R: Reserved	Reserved.
7:4	RO	1h	MAX: Major Version number	Indicates supported architecture version.
3:0	RO	0h	MIN: Minor Version number	Indicates supported architecture minor version.

10.4.2 Capability Register

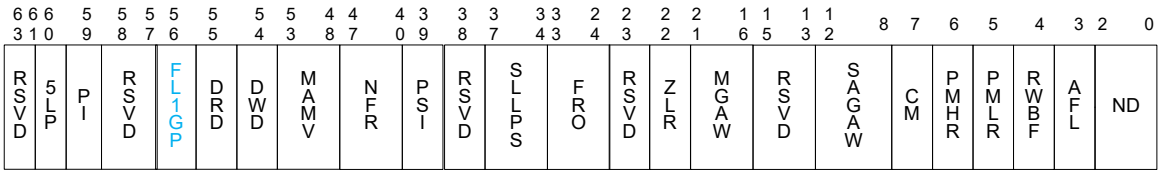


Figure 10-45. Capability Register

Abbreviation	CAP_REG
General Description	Register to report general remapping hardware capabilities
Register Offset	008h

Bits	Access	Default	Field	Description
63:61	RsvdZ	0h	R: Reserved	Reserved.
60	RO	X	5LP: 5-level Paging Support	• 0: Hardware does not support 5-level paging for requests-with-PASID subject to first-level translation. • 1: Hardware supports 5-level paging for requests-with-PASID subject to first-level translation.
59	RO	X	PI: Posted Interrupts Support	• 0: Hardware does not support Posting of Interrupts. • 1: Hardware supports Posting of Interrupts. Hardware implementations reporting this field as Set must also report Interrupt Remapping support (IR field in Extended Capability Register) field as Set. Refer Section 5.2 for description of interrupt posting.
58:57	RsvdZ	0h	R: Reserved	Reserved.
56	RO	X	FL1GP: First Level 1-GByte Page Support	A value of 1 in this field indicates 1-GByte page size is supported for first-level translation.
55	RO	X	DRD: Read Draining	• 0: Hardware does not support draining of read requests. • 1: Hardware supports draining of read requests. Refer Section 6.5.5 for description of read draining. Hardware implementations reporting support for process-address-spaces (PASID=1 in Extended Capability Register) must report this field as 1.

Bits	Access	Default	Field	Description
54	RO	X	DWD: Write Draining	• 0: Hardware does not support draining of write requests. • 1: Hardware supports draining of write requests. Refer Section 6.5.5 for description of DMA write draining. Hardware implementations reporting support for process-address-spaces (PASID=1 in Extended Capability Register) must report this field as 1.
53:48	RO	X	MAMV: Maximum Address Mask Value	The value in this field indicates the maximum supported value for the Address Mask (AM) field in the Invalidation Address register (IVA_REG), and IOTLB Invalidation Descriptor (*iotlb_inv_dsc*) used for invalidations of second-level translation. This field is valid when the PSI field in Capability register is reported as Set. Independent of value reported in this field, implementations supporting PASID must support address-selective extended IOTLB invalidations (*ext_iotlb_inv_dsc*) with any defined address mask.
47:40	RO	X	NFR: Number of Fault- recording Registers	Number of fault recording registers is computed as N+1, where N is the value reported in this field. Implementations must support at least one fault recording register (NFR = 0) for each remapping hardware unit in the platform. The maximum number of fault recording registers per remapping hardware unit is 256.
39	RO	X	PSI: Page Selective Invalidation	• 0: Hardware supports only global and domain-selective invalidates for IOTLB. • 1: Hardware supports page-selective, domain-selective, and global invalidates for IOTLB. Hardware implementations reporting this field as Set are recommended to support a Maximum Address Mask Value (MAMV) value of at least 9 (or 18 if supporting 1GB pages with second level translation). This field is applicable only for IOTLB invalidations for second-level translation. Irrespective of value reported in this field, implementations supporting PASID must support page/address selective IOTLB invalidation for first-level translation.
38	RsvdZ	0h	R: Reserved	Reserved.

Bits	Access	Default	Field	Description
37:34	RO	X	SLLPS: Second Level Large Page Support	This field indicates the large page sizes supported by hardware. A value of 1 in any of these bits indicates the corresponding large-page size is supported. The large-page sizes corresponding to various bit positions within this field are: • 0: 21-bit offset to page frame (2MB) • 1: 30-bit offset to page frame (1GB) • 2: Reserved • 3: Reserved Hardware implementations supporting a specific large-page size must support all smaller large-page sizes. i.e., only valid values for this field are 0000b, 0001b, 0011b.
33:24	RO	X	FRO: Fault-recording Register offset	This field specifies the offset of the first fault recording register relative to the register base address of this remapping hardware unit. If the register base address is X, and the value reported in this field is Y, the address for the first fault recording register is calculated as X+(16*Y).
23	R0	X	R: Reserved	Reserved.
22	RO	X	ZLR: Zero Length Read	• 0: Indicates the remapping hardware unit blocks (and treats as fault) zero length DMA read requests to write-only pages. • 1: Indicates the remapping hardware unit supports zero length DMA read requests to write-only pages. DMA remapping hardware implementations are recommended to report ZLR field as Set.
21:16	RO	X	MGAW: Maximum Guest Address Width	This field indicates the maximum guest physical address width supported by second-level translation in remapping hardware. The Maximum Guest Address Width (MGAW) is computed as (N+1), where N is the valued reported in this field. For example, a hardware implementation supporting 48-bit MGAW reports a value of 47 (101111b) in this field. If the value in this field is X, untranslated DMA requests with addresses above $2^{(X+1)}$-1 that are subjected to second-level translation are blocked by hardware. Device-TLB translation requests to addresses above $2^{(X+1)}$-1 that are subjected to second-level translation from allowed devices return a null Translation-Completion Data with R=W=0. Guest addressability for a given DMA request is limited to the minimum of the value reported through this field and the adjusted guest address width of the corresponding page-table structure. (Adjusted guest address widths supported by hardware are reported through the SAGAW field). Implementations must support MGAW at least equal to the physical addressability (host address width) of the platform.

Bits	Access	Default	Field	Description
15:13	RsvdZ	0h	R: Reserved	Reserved.
12:8	RO	X	SAGAW: Supported Adjusted Guest Address Widths	This 5-bit field indicates the supported adjusted guest address widths (which in turn represents the levels of page-table walks for the 4KB base page size) supported by the hardware implementation. A value of 1 in any of these bits indicates the corresponding adjusted guest address width is supported. The adjusted guest address widths corresponding to various bit positions within this field are: • 0: Reserved • 1: 39-bit AGAW (3-level page-table) • 2: 48-bit AGAW (4-level page-table) • 3: 57-bit AGAW (5-level page-table) • 4: Reserved Software must ensure that the adjusted guest address width used to set up the page tables is one of the supported guest address widths reported in this field.
7	RO	X	CM: Caching Mode	• 0: Not-present and erroneous entries are not cached in any of the remapping caches. Invalidations are not required for modifications to individual not present or invalid entries. However, any modifications that result in decreasing the effective permissions or partial permission increases require invalidations for them to be effective. • 1: Not-present and erroneous mappings may be cached in the remapping caches. Any software updates to the remapping structures (including updates to "not-present" or erroneous entries) require explicit invalidation. Hardware implementations of this architecture must support a value of 0 in this field. Refer to Section 6.1 for more details on Caching Mode.
6	RO	X	PHMR: Protected High-Memory Region	• 0: Indicates protected high-memory region is not supported. • 1: Indicates protected high-memory region is supported.
5	RO	X	PLMR: Protected Low-Memory Region	• 0: Indicates protected low-memory region is not supported. • 1: Indicates protected low-memory region is supported.
4	RO	X	RWBF: Required Write-Buffer Flushing	• 0: Indicates no write-buffer flushing is needed to ensure changes to memory-resident structures are visible to hardware. • 1: Indicates software must explicitly flush the write buffers to ensure updates made to memory-resident remapping structures are visible to hardware. Refer to Section 6.8 for more details on write buffer flushing requirements.

Bits	Access	Default	Field	Description
3	RO	X	AFL: Advanced Fault Logging	• 0: Indicates advanced fault logging is not supported. Only primary fault logging is supported. • 1: Indicates advanced fault logging is supported.
2:0	RO	X	ND: Number of domains supported[1]	• 000b: Hardware supports 4-bit domain-ids with support for up to 16 domains. • 001b: Hardware supports 6-bit domain-ids with support for up to 64 domains. • 010b: Hardware supports 8-bit domain-ids with support for up to 256 domains. • 011b: Hardware supports 10-bit domain-ids with support for up to 1024 domains. • 100b: Hardware supports 12-bit domain-ids with support for up to 4K domains. • 101b: Hardware supports 14-bit domain-ids with support for up to 16K domains. • 110b: Hardware supports 16-bit domain-ids with support for up to 64K domains. • 111b: Reserved.

1. Each remapping unit in the platform should support as many number of domains as the maximum number of independently DMA-remappable devices expected to be attached behind it.

10.4.3 Extended Capability Register

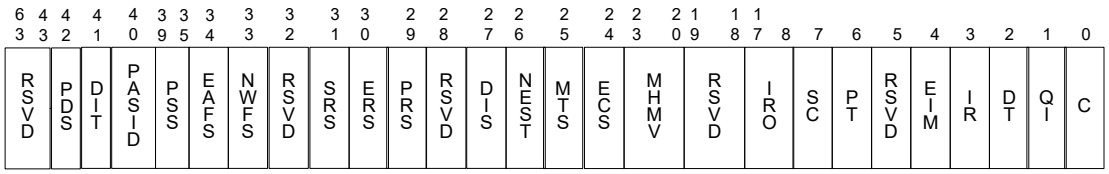


Figure 10-46. Extended Capability Register

Abbreviation	ECAP_REG
General Description	Register to report remapping hardware extended capabilities
Register Offset	010h

Bits	Access	Default	Field	Description
63:43	RsvdZ	0h	R: Reserved	Reserved.
42	RO	X	PDS: Page-request Drain Support	• 0: Hardware does not support Page-request Drain (PD) flag in *inv_wait_dsc*. • 1: Hardware supports Page-request Drain (PD) flag in *inv_wait_dsc*. This field is valid only when Device-TLB support field is reported as Set.
41	RO	X	DIT: Device-TLB Invalidation Throttle	• 0: Hardware does not support Device-TLB Invalidation Throttling. • 1: Hardware supports Device-TLB Invalidation Throttling. This field is valid only when Page Request Support (PRS) field is reported as Set.
40	RO	X	PASID: Process Address Space ID Support	• 0: Hardware does not support requests tagged with Process Address Space IDs. • 1: Hardware supports requests tagged with Process Address Space IDs.
39:35	RO	X	PSS: PASID Size Supported	This field reports the PASID size supported by the remapping hardware for requests-with-PASID. A value of N in this field indicates hardware supports PASID field of N+1 bits (For example, value of 7 in this field, indicates 8-bit PASIDs are supported). Requests-with-PASID with PASID value beyond the limit specified by this field are treated as error by the remapping hardware. This field is valid only when PASID field is reported as Set.

Bits	Access	Default	Field	Description
34	RO	X	EAFS: Extended Accessed Flag Support	• 0: Hardware does not support the extended-accessed (EA) bit in first-level paging-structure entries. • 1: Hardware supports the extended-accessed (EA) bit in first-level paging-structure entries. This field is valid only when PASID field is reported as Set.
33	RO	X	NWFS: No Write Flag Support	• 0: Hardware ignores the 'No Write' (NW) flag in Device-TLB translation-requests, and behaves as if NW is always 0. • 1: Hardware supports the 'No Write' (NW) flag in Device-TLB translation-requests. This field is valid only when Device-TLB support (DT) field is reported as Set.
32	RsvdZ	0h	R: Reserved	Reserved.
31	RO	X	SRS: Supervisor Request Support	• 0: Hardware does not support requests-with-PASID seeking supervisor privilege. • 1: Hardware supports requests-with-PASID seeking supervisor privilege. This field is valid only when PASID field is reported as Set.
30	RO	X	ERS: Execute Request Support	• 0: Hardware does not support requests-with-PASID seeking execute permission. • 1: Hardware supports requests-with-PASID seeking execute permission. This field is valid only when PASID field is reported as Set.
29	RO	X	PRS: Page Request Support	• 0: Hardware does not support page requests. • 1: Hardware supports page requests. This field is valid only when Device-TLB (DT) field is reported as Set.
28	Rsvd	X	R: Reserved	Reserved (defunct).
27	RO	X	DIS: Deferred Invalidate Support	• 0: Hardware does not support deferred invalidations of IOTLB & Device-TLB. • 1: Hardware supports deferred invalidations of IOTLB & Device-TLB. This field is valid only when PASID field is reported as Set.
26	RO	X	NEST: Nested Translation Support	• 0: Hardware does not support nested translations. • 1: Hardware supports nested translations. This field is valid only when PASID field is reported as Set.

Bits	Access	Default	Field	Description
25	RO	X	MTS: Memory Type Support	• 0: Hardware does not support Memory Type in first-level translation and Extended Memory type in second-level translation. • 1: Hardware supports Memory Type in first-level translation and Extended Memory type in second-level translation. This field is valid only when PASID and ECS fields are reported as Set. Remapping hardware units with, one or more devices that operate in processor coherency domain, under its scope must report this field as Set.
24	RO	X	ECS: Extended Context Support	• 0: Hardware does not support extended-root-entries and extended-context-entries. • 1: Hardware supports extended-root-entries and extended-context-entries. Implementations reporting PASID or PRS fields as Set, must report this field as Set.
23:20	RO	X	MHMV: Maximum Handle Mask Value	The value in this field indicates the maximum supported value for the Interrupt Mask (IM) field in the Interrupt Entry Cache Invalidation Descriptor (*iec_inv_dsc*). This field is valid only when the IR field in Extended Capability register is reported as Set.
19:18	RsvdZ	0h	R: Reserved	Reserved.
17:8	RO	X	IRO: IOTLB Register Offset	This field specifies the offset to the IOTLB registers relative to the register base address of this remapping hardware unit. If the register base address is X, and the value reported in this field is Y, the address for the IOTLB registers is calculated as X+(16*Y).
7	R0	X	SC: Snoop Control	• 0: Hardware does not support 1-setting of the SNP field in the page-table entries. • 1: Hardware supports the 1-setting of the SNP field in the page-table entries. Implementations are recommended to support Snoop Control to support software usages that require Snoop Control for assignment of devices behind a remapping hardware unit.
6	R0	X	PT: Pass Through	• 0: Hardware does not support pass-through translation type in context-entries and extended-context-entries. • 1: Hardware supports pass-through translation type in context and extended-context-entries. Pass-through translation is specified through *Translation-Type* (T) field value of 10b in context-entries, or T field value of 010b in extended-context-entries. Hardware implementation supporting PASID must report a value of 1b in this field.

Bits	Access	Default	Field	Description
5	R0	X	R: Reserved	Reserved.
4	RO	0	EIM: Extended Interrupt Mode	• 0: On Intel® 64 platforms, hardware supports only 8-bit APIC-IDs (xAPIC Mode). • 1: On Intel® 64 platforms, hardware supports 32-bit APIC-IDs (x2APIC mode). This field is valid only on Intel® 64 platforms reporting Interrupt Remapping support (IR field Set).
3	RO	X	IR: Interrupt Remapping support	• 0: Hardware does not support interrupt remapping. • 1: Hardware supports interrupt remapping. Implementations reporting this field as Set must also support Queued Invalidation (QI)
2	RO	X	DT: Device-TLB support	• 0: Hardware does not support Device-TLBs. • 1: Hardware supports Device-TLBs. Implementations reporting this field as Set must also support Queued Invalidation (QI) Hardware implementations supporting I/O Page Requests (PRS field Set in Extended Capability register) must report a value of 1b in this field.
1	RO	X	QI: Queued Invalidation support	• 0: Hardware does not support queued invalidations. • 1: Hardware supports queued invalidations.
0	RO	X	C: Page-walk Coherency	This field indicates if hardware access to the root, context, extended-context and interrupt-remap tables, and second-level paging structures for requests-without-PASID, are coherent (snooped) or not. • 0:Indicates hardware accesses to remapping structures are non-coherent. • 1:Indicates hardware accesses to remapping structures are coherent. Hardware access to advanced fault log, invalidation queue, invalidation semaphore, page-request queue, PASID-table, PASID-state table, and first-level page-tables are always coherent.

10.4.4 Global Command Register

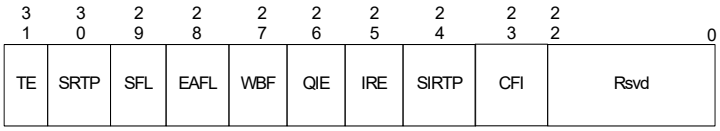


Figure 10-47. Global Command Register

Abbreviation	GCMD_REG
General Description	Register to control remapping hardware. If multiple control fields in this register need to be modified, software must serialize the modifications through multiple writes to this register. For example, to update a bit field in this register at offset X with value of Y, software must follow below steps: 1. Tmp = Read GSTS_REG 2. Status = (Tmp & 96FFFFFFh) // Reset the one-shot bits 3. Command = (Status | (Y << X)) 4. Write Command to GCMD_REG 5. Wait until GSTS_REG[X] indicates command is serviced.
Register Offset	018h

Bits	Access	Default	Field	Description
31	WO	0	TE: Translation Enable	Software writes to this field to request hardware to enable/disable DMA remapping: • 0: Disable DMA remapping • 1: Enable DMA remapping Hardware reports the status of the translation enable operation through the TES field in the Global Status register. There may be active DMA requests in the platform when software updates this field. Hardware must enable or disable remapping logic only at deterministic transaction boundaries, so that any in-flight transaction is either subject to remapping or not at all. Hardware implementations supporting DMA draining must drain any in-flight DMA read/write requests queued within the Root-Complex before completing the translation enable command and reflecting the status of the command through the TES field in the Global Status register. The value returned on a read of this field is undefined.

Bits	Access	Default	Field	Description
30	WO	0	SRTP: Set Root Table Pointer	Software sets this field to set/update the root-table pointer used by hardware. The root-table pointer is specified through the Root Table Address (RTADDR_REG) register. Hardware reports the status of the 'Set Root Table Pointer' operation through the RTPS field in the Global Status register. The 'Set Root Table Pointer' operation must be performed before enabling or re-enabling (after disabling) DMA remapping through the TE field. After a 'Set Root Table Pointer' operation, software must globally invalidate the context-cache and then globally invalidate the IOTLB. This is required to ensure hardware uses only the remapping structures referenced by the new root-table pointer, and not stale cached entries. While DMA remapping is active, software may update the root table pointer through this field. However, to ensure valid in-flight DMA requests are deterministically remapped, software must ensure that the structures referenced by the new root table pointer are programmed to provide the same remapping results as the structures referenced by the previous root-table pointer. Clearing this bit has no effect. The value returned on a read of this field is undefined.
29	WO	0	SFL: Set Fault Log	This field is valid only for implementations supporting advanced fault logging. Software sets this field to request hardware to set/update the fault-log pointer used by hardware. The fault-log pointer is specified through Advanced Fault Log register. Hardware reports the status of the 'Set Fault Log' operation through the FLS field in the Global Status register. The fault log pointer must be set before enabling advanced fault logging (through EAFL field). Once advanced fault logging is enabled, the fault log pointer may be updated through this field while DMA remapping is active. Clearing this bit has no effect. The value returned on read of this field is undefined.

Bits	Access	Default	Field	Description
28	WO	0	EAFL: Enable Advanced Fault Logging	This field is valid only for implementations supporting advanced fault logging. Software writes to this field to request hardware to enable or disable advanced fault logging: • 0: Disable advanced fault logging. In this case, translation faults are reported through the Fault Recording registers. • 1: Enable use of memory-resident fault log. When enabled, translation faults are recorded in the memory-resident log. The fault log pointer must be set in hardware (through the SFL field) before enabling advanced fault logging. Hardware reports the status of the advanced fault logging enable operation through the AFLS field in the Global Status register. The value returned on read of this field is undefined.
27	WO	0	WBF: Write Buffer Flush[1]	This bit is valid only for implementations requiring write buffer flushing. Software sets this field to request that hardware flush the Root-Complex internal write buffers. This is done to ensure any updates to the memory-resident remapping structures are not held in any internal write posting buffers. Refer to Section 6.8 for details on write-buffer flushing requirements. Hardware reports the status of the write buffer flushing operation through the WBFS field in the Global Status register. Clearing this bit has no effect. The value returned on a read of this field is undefined.
26	WO	0	QIE: Queued Invalidation Enable	This field is valid only for implementations supporting queued invalidations. Software writes to this field to enable or disable queued invalidations. • 0: Disable queued invalidations. • 1: Enable use of queued invalidations. Hardware reports the status of queued invalidation enable operation through QIES field in the Global Status register. Refer to Section 6.5.2 for software requirements for enabling/disabling queued invalidations. The value returned on a read of this field is undefined.

Bits	Access	Default	Field	Description
25	WO	0h	IRE: Interrupt Remapping Enable	This field is valid only for implementations supporting interrupt remapping. • 0: Disable interrupt-remapping hardware • 1: Enable interrupt-remapping hardware Hardware reports the status of the interrupt remapping enable operation through the IRES field in the Global Status register. There may be active interrupt requests in the platform when software updates this field. Hardware must enable or disable interrupt-remapping logic only at deterministic transaction boundaries, so that any in-flight interrupts are either subject to remapping or not at all. Hardware implementations must drain any in-flight interrupts requests queued in the Root-Complex before completing the interrupt-remapping enable command and reflecting the status of the command through the IRES field in the Global Status register. The value returned on a read of this field is undefined.
24	WO	0	SIRTP: Set Interrupt Remap Table Pointer	This field is valid only for implementations supporting interrupt-remapping. Software sets this field to set/update the interrupt remapping table pointer used by hardware. The interrupt remapping table pointer is specified through the Interrupt Remapping Table Address (IRTA_REG) register. Hardware reports the status of the 'Set Interrupt Remap Table Pointer' operation through the IRTPS field in the Global Status register. The 'Set Interrupt Remap Table Pointer' operation must be performed before enabling or re-enabling (after disabling) interrupt-remapping hardware through the IRE field. After an 'Set Interrupt Remap Table Pointer' operation, software must globally invalidate the interrupt entry cache. This is required to ensure hardware uses only the interrupt-remapping entries referenced by the new interrupt remap table pointer, and not stale cached entries. While interrupt remapping is active, software may update the interrupt remapping table pointer through this field. However, to ensure valid in-flight interrupt requests are deterministically remapped, software must ensure that the structures referenced by the new interrupt remap table pointer are programmed to provide the same remapping results as the structures referenced by the previous interrupt remap table pointer. Clearing this bit has no effect. The value returned on a read of this field is undefined.

Bits	Access	Default	Field	Description
23	WO	0	CFI: Compatibility Format Interrupt	This field is valid only for Intel® 64 implementations supporting interrupt-remapping. Software writes to this field to enable or disable Compatibility Format interrupts on Intel® 64 platforms. The value in this field is effective only when interrupt-remapping is enabled and Extended Interrupt Mode (x2APIC mode) is not enabled. • 0: Block Compatibility format interrupts. • 1: Process Compatibility format interrupts as pass-through (bypass interrupt remapping). Hardware reports the status of updating this field through the CFIS field in the Global Status register. Refer to Section 5.1.2.1 for details on Compatibility Format interrupt requests. The value returned on a read of this field is undefined.
22:0	RsvdZ	0h	R: Reserved	Reserved.

1. Implementations reporting write-buffer flushing as required in Capability register must perform implicit write buffer flushing as a pre-condition to all context-cache and IOTLB invalidation operations.

10.4.5 Global Status Register

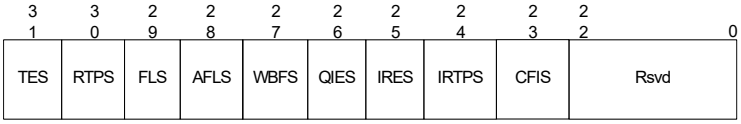


Figure 10-48. Global Status Register

Abbreviation	GSTS_REG
General Description	Register to report general remapping hardware status.
Register Offset	01Ch

Bits	Access	Default	Field	Description
31	RO	0	TES: Translation Enable Status	This field indicates the status of DMA-remapping hardware. • 0: DMA remapping is not enabled • 1: DMA remapping is enabled
30	RO	0	RTPS: Root Table Pointer Status	This field indicates the status of the root-table pointer in hardware. This field is cleared by hardware when software sets the SRTP field in the Global Command register. This field is set by hardware when hardware completes the 'Set Root Table Pointer' operation using the value provided in the Root Table Address register.
29	RO	0	FLS: Fault Log Status	This field: • Is cleared by hardware when software Sets the SFL field in the Global Command register. • Is Set by hardware when hardware completes the 'Set Fault Log Pointer' operation using the value provided in the Advanced Fault Log register.
28	RO	0	AFLS: Advanced Fault Logging Status	This field is valid only for implementations supporting advanced fault logging. It indicates the advanced fault logging status: • 0: Advanced Fault Logging is not enabled • 1: Advanced Fault Logging is enabled
27	RO	0	WBFS: Write Buffer Flush Status	This field is valid only for implementations requiring write buffer flushing. This field indicates the status of the write buffer flush command. It is • Set by hardware when software sets the WBF field in the Global Command register. • Cleared by hardware when hardware completes the write buffer flushing operation.
26	RO	0	QIES: Queued Invalidation Enable Status	This field indicates queued invalidation enable status. • 0: queued invalidation is not enabled • 1: queued invalidation is enabled

Bits	Access	Default	Field	Description
25	RO	0	IRES: Interrupt Remapping Enable Status	This field indicates the status of Interrupt-remapping hardware. • 0: Interrupt-remapping hardware is not enabled • 1:Interrupt-remapping hardware is enabled
24	RO	0	IRTPS: Interrupt Remapping Table Pointer Status	This field indicates the status of the interrupt remapping table pointer in hardware. This field is cleared by hardware when software sets the SIRTP field in the Global Command register. This field is Set by hardware when hardware completes the 'Set Interrupt Remap Table Pointer' operation using the value provided in the Interrupt Remapping Table Address register.
23	RO	0	CFIS: Compatibility Format Interrupt Status	This field indicates the status of Compatibility format interrupts on Intel® 64 implementations supporting interrupt-remapping. The value reported in this field is applicable only when interrupt-remapping is enabled and extended interrupt mode (x2APIC mode) is not enabled. • 0: Compatibility format interrupts are blocked. • 1: Compatibility format interrupts are processed as pass-through (bypassing interrupt remapping).
22:0	RsvdZ	0h	R: Reserved	Reserved.

10.4.6 Root Table Address Register

Figure 10-49. Root Table Address Register

Abbreviation	RTADDR_REG
General Description	Register providing the base address of root-table.
Register Offset	020h

Bits	Access	Default	Field	Description
63:12	RW	0h	RTA: Root Table Address	This register points to the base of the page-aligned, 4KB-sized root-table in system memory. Hardware may ignore and not implement bits 63:HAW, where HAW is the host address width. Software specifies the base address of the root table through this register, and programs it in hardware through the SRTP field in the Global Command register. Reads of this register return the value that was last programmed to it.
11	RW	0	RTT: Root Table Type	This field specifies the type of root-table referenced by the Root Table Address (RTA) field. • 0: Root Table • 1: Extended Root Table Root Table entry and Extended-Root Table entry formats are specified in Section 9.1 and Section 9.2 respectively. Software must not modify this field while remapping is active (TES=1 in Global Status register). Hardware implementations reporting extended capabilities as not supported (ECS field Clear in Extended Capability register), treats this field as RsvdZ.
10:0	RsvdZ	0h	R: Reserved	Reserved.

10.4.7 Context Command Register

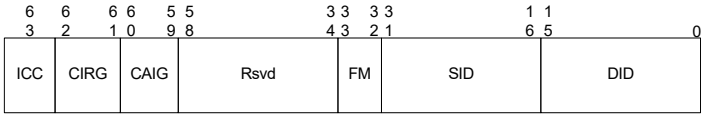


Figure 10-50. Context Command Register

Abbreviation	CCMD_REG
General Description	Register to manage context-cache.The act of writing the uppermost byte of the CCMD_REG with the ICC field Set causes the hardware to perform the context-cache invalidation.
Register Offset	028h

Bits	Access	Default	Field	Description
63	RW	0	ICC: Invalidate Context-Cache	Software requests invalidation of context-cache by setting this field. Software must also set the requested invalidation granularity by programming the CIRG field. Software must read back and check the ICC field is Clear to confirm the invalidation is complete. Software must not update this register when this field is Set. Hardware clears the ICC field to indicate the invalidation request is complete.Hardware also indicates the granularity at which the invalidation operation was performed through the CAIG field. Software must submit a context-cache invalidation request through this field only when there are no invalidation requests pending at this remapping hardware unit. Since information from the context-cache may be used by hardware to tag IOTLB entries, software must perform domain-selective (or global) invalidation of IOTLB after the context-cache invalidation has completed. Hardware implementations reporting a write-buffer flushing requirement (RWBF=1 in the Capability register) must implicitly perform a write buffer flush before invalidating the context-cache. Refer to Section 6.8 for write buffer flushing requirements.

Bits	Access	Default	Field	Description
62:61	RW	0h	CIRG: Context Invalidation Request Granularity	Software provides the requested invalidation granularity through this field when setting the ICC field: • 00: Reserved. • 01: Global Invalidation request. • 10: Domain-selective invalidation request. The target domain-id must be specified in the DID field. • 11: Device-selective invalidation request. The target source-id(s) must be specified through the SID and FM fields, and the domain-id [that was programmed in the context-entry for these device(s)] must be provided in the DID field. Hardware implementations may process an invalidation request by performing invalidation at a coarser granularity than requested. Hardware indicates completion of the invalidation request by clearing the ICC field. At this time, hardware also indicates the granularity at which the actual invalidation was performed through the CAIG field.
60:59	RO	Xh	CAIG: Context Actual Invalidation Granularity	Hardware reports the granularity at which an invalidation request was processed through the CAIG field at the time of reporting invalidation completion (by clearing the ICC field). The following are the encodings for this field: • 00: Reserved. • 01: Global Invalidation performed. This could be in response to a global, domain-selective, or device-selective invalidation request. • 10: Domain-selective invalidation performed using the domain-id specified by software in the DID field. This could be in response to a domain-selective or device-selective invalidation request. • 11: Device-selective invalidation performed using the source-id and domain-id specified by software in the SID and FM fields. This can only be in response to a device-selective invalidation request.
58:34	RsvdZ	0h	R: Reserved	Reserved.
33:32	WO	0h	FM: Function Mask	Software may use the Function Mask to perform device-selective invalidations on behalf of devices supporting PCI-Express Phantom Functions. This field specifies which bits of the function number portion (least significant three bits) of the SID field to mask when performing device-selective invalidations.The following encodings are defined for this field: • 00: No bits in the SID field masked • 01: Mask bit 2 in the SID field • 10: Mask bits 2:1 in the SID field • 11: Mask bits 2:0 in the SID field The context-entries corresponding to the source-ids specified through the SID and FM fields must have the domain-id specified in the DID field. The value returned on a read of this field is undefined.

Bits	Access	Default	Field	Description
31:16	WO	0h	SID: Source-ID	Indicates the source-id of the device whose corresponding context-entry needs to be selectively invalidated.This field along with the FM field must be programmed by software for device-selective invalidation requests. The value returned on a read of this field is undefined.
15:0	RW	0h	DID: Domain-ID	Indicates the id of the domain whose context-entries need to be selectively invalidated. This field must be programmed by software for both domain-selective and device-selective invalidation requests. The Capability register reports the domain-id width supported by hardware. Software must ensure that the value written to this field is within this limit. Hardware ignores (and may not implement) bits 15:N, where N is the supported domain-id width reported in the Capability register.

10.4.8 IOTLB Registers

IOTLB registers consists of two adjacently placed 64-bit registers:

- IOTLB Invalidate Register (IOTLB_REG)
- Invalidate Address Register (IVA_REG)

Offset	Register Name	Size	Description
XXXh	Invalidate Address Register	64	Register to provide the target address for page-selective IOTLB invalidation. The offset of this register is reported through the IRO field in Extended Capability register.
XXXh + 008h	IOTLB Invalidate Register	64	Register for IOTLB invalidation command

These registers are described in the following sub-sections.

10.4.8.1 IOTLB Invalidate Register

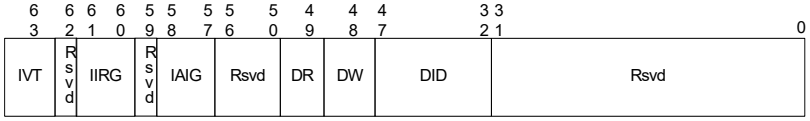


Figure 10-51. IOTLB Invalidate Register

Abbreviation	IOTLB_REG
General Description	Register to invalidate IOTLB. The act of writing the upper byte of the IOTLB_REG with the IVT field Set causes the hardware to perform the IOTLB invalidation.
Register Offset	XXXh + 0008h (where XXXh is the location of the IVA_REG)

Bits	Access	Default	Field	Description
63	RW	0	IVT: Invalidate IOTLB	Software requests IOTLB invalidation by setting this field. Software must also set the requested invalidation granularity by programming the IIRG field. Hardware clears the IVT field to indicate the invalidation request is complete. Hardware also indicates the granularity at which the invalidation operation was performed through the IAIG field. Software must not submit another invalidation request through this register while the IVT field is Set, nor update the associated Invalidate Address register. Software must not submit IOTLB invalidation requests when there is a context-cache invalidation request pending at this remapping hardware unit. Hardware implementations reporting a write-buffer flushing requirement (RWBF=1 in Capability register) must implicitly perform a write buffer flushing before invalidating the IOTLB. Refer to Section 6.8 for write buffer flushing requirements.
62	RsvdZ	0	R: Reserved	Reserved.

Bits	Access	Default	Field	Description
61:60	RW	0h	IIRG: IOTLB Invalidation Request Granularity	When requesting hardware to invalidate the IOTLB (by setting the IVT field), software writes the requested invalidation granularity through this field. The following are the encodings for the field. • 00: Reserved. • 01: Global invalidation request. • 10: Domain-selective invalidation request. The target domain-id must be specified in the DID field. • 11: Page-selective-within-domain invalidation request. The target address, mask, and invalidation hint must be specified in the Invalidate Address register, and the domain-id must be provided in the DID field. Hardware implementations may process an invalidation request by performing invalidation at a coarser granularity than requested. Hardware indicates completion of the invalidation request by clearing the IVT field. At that time, the granularity at which actual invalidation was performed is reported through the IAIG field.
59	RsvdZ	0	R: Reserved	Reserved.
58:57	RO	Xh	IAIG: IOTLB Actual Invalidation Granularity	Hardware reports the granularity at which an invalidation request was processed through this field when reporting invalidation completion (by clearing the IVT field). The following are the encodings for this field. • 00: Reserved. This indicates hardware detected an incorrect invalidation request and ignored the request. Examples of incorrect invalidation requests include detecting an unsupported address mask value in Invalidate Address register for page-selective invalidation requests. • 01: Global Invalidation performed. This could be in response to a global, domain-selective, or page-selective invalidation request. • 10: Domain-selective invalidation performed using the domain-id specified by software in the DID field. This could be in response to a domain-selective or a page-selective invalidation request. • 11: Page-selective-within-domain invalidation performed using the address, mask and hint specified by software in the Invalidate Address register and domain-id specified in DID field. This can be in response to a page-selective-within-domain invalidation request.
56:50	RsvdZ	0h	R: Reserved	Reserved.
49	RW	0h	DR: Drain Reads	This field is ignored by hardware if the DRD field is reported as Clear in the Capability register. When the DRD field is reported as Set in the Capability register, the following encodings are supported for this field: • 0: Hardware may complete the IOTLB invalidation without draining DMA read requests. • 1: Hardware must drain DMA read requests. Refer Section 6.5.5 for description of DMA draining.

Bits	Access	Default	Field	Description
48	RW	0h	DW: Drain Writes	This field is ignored by hardware if the DWD field is reported as Clear in the Capability register. When the DWD field is reported as Set in the Capability register, the following encodings are supported for this field: • 0: Hardware may complete the IOTLB invalidation without draining DMA write requests. • 1: Hardware must drain relevant translated DMA write requests. Refer Section 6.5.5 for description of DMA draining.
47:32	RW	0h	DID: Domain-ID	Indicates the ID of the domain whose IOTLB entries need to be selectively invalidated. This field must be programmed by software for domain-selective and page-selective invalidation requests. The Capability register reports the domain-id width supported by hardware. Software must ensure that the value written to this field is within this limit. Hardware may ignore and not implement bits 47:(32+N), where N is the supported domain-id width reported in the Capability register.
31:0	RsvdP	Xh	R: Reserved	Reserved.

10.4.8.2 Invalidate Address Register

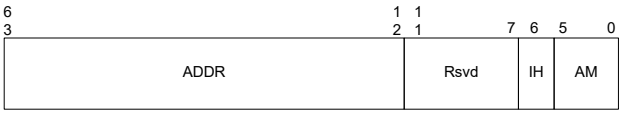


Figure 10-52. Invalidate Address Register

Abbreviation	IVA_REG
General Description	Register to provide the DMA address whose corresponding IOTLB entry needs to be invalidated through the corresponding IOTLB Invalidate register. This register is a write-only register. A value returned on a read of this register is undefined.
Register Offset	XXXh (XXXh is QWORD aligned and reported through the IRO field in the Extended Capability register)

Bits	Access	Default	Field	Description
63:12	WO	0h	ADDR: Address	Software provides the second-level-input-address that needs to be page-selectively invalidated. To make a page-selective-within-domain invalidation request to hardware, software must first write the appropriate fields in this register, and then issue the page-selective-within-domain invalidate command through the IOTLB_REG. Hardware ignores bits 63:N, where N is the maximum guest address width (MGAW) supported. A value returned on a read of this field is undefined.
11:7	RsvdZ	0	R: Reserved	Reserved.
6	WO	0	IH: Invalidation Hint	The field provides hints to hardware about preserving or flushing the non-leaf (context-entry) entries that may be cached in hardware: • 0: Software may have modified both leaf and non-leaf second-level paging-structure entries corresponding to mappings specified in the ADDR and AM fields. On a page-selective-within-domain invalidation request, hardware must invalidate the cached entries associated with the mappings specified by DID, ADDR and AM fields, in both IOTLB and paging-structure caches. Refer to Section 6.5.1.2 for exact invalidation requirements when IH=0. • 1: Software has not modified any second-level non-leaf paging entries associated with the mappings specified by the ADDR and AM fields. On a page-selective-within-domain invalidation request, hardware may preserve the cached second-level mappings in paging-structure-caches. Refer to Section 6.5.1.2 for exact invalidation requirements when IH=1. A value returned on a read of this field is undefined.

<table>
<tr><th>Bits</th><th>Access</th><th>Default</th><th>Field</th><th>Description</th></tr>
<tr><td>5:0</td><td>WO</td><td>0</td><td>AM: Address Mask</td><td>The value in this field specifies the number of low order bits of the ADDR field that must be masked for the invalidation operation. This field enables software to request invalidation of contiguous mappings for size-aligned regions. For example:

<table>
<tr><th>Mask Value</th><th>ADDR bits masked</th><th>Pages invalidated</th></tr>
<tr><td>0</td><td>None</td><td>1</td></tr>
<tr><td>1</td><td>12</td><td>2</td></tr>
<tr><td>2</td><td>13:12</td><td>4</td></tr>
<tr><td>3</td><td>14:12</td><td>8</td></tr>
<tr><td>4</td><td>15:12</td><td>16</td></tr>
<tr><td>...</td><td>...</td><td>...</td></tr>
</table>

When invalidating mappings for large-pages, software must specify the appropriate mask value. For example, when invalidating mapping for a 2MB page, software must specify an address mask value of at least 9.

Hardware implementations report the maximum supported address mask value through the Capability register.

A value returned on a read of this field is undefined.</td></tr>
</table>

10.4.9 Fault Status Register

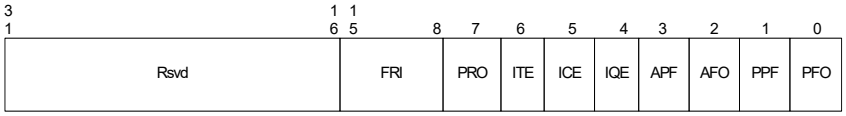


Figure 10-53. Fault Status Register

Abbreviation	FSTS_REG
General Description	Register indicating the various error status.
Register Offset	034h

Bits	Access	Default	Field	Description
31:16	RsvdZ	0h	R: Reserved	Reserved.
15:8	ROS	0	FRI: Fault Record Index	This field is valid only when the PPF field is Set. The FRI field indicates the index (from base) of the fault recording register to which the first pending fault was recorded when the PPF field was Set by hardware. The value read from this field is undefined when the PPF field is Clear.
7	RW1CS	0	PRO: Page Request Overflow	Hardware sets this field to indicate Page Request Queue overflow condition. Fault event is generated based on programming of the Fault Event Control register. Page request message that led to setting this bit and subsequent messages received while this field is already Set are discarded or responded by hardware as described in Section 7.5.1. Software writing a 1 to this field clears it. Hardware implementations not supporting Page Requests (PRS field reported as Clear in Capability register) implement this field as RsvdZ.
6	RW1CS	0h	ITE: Invalidation Time-out Error	Hardware detected a Device-TLB invalidation completion time-out. At this time, a fault event may be generated based on the programming of the Fault Event Control register. Hardware implementations not supporting Device-TLBs implement this bit as RsvdZ.
5	RW1CS	0h	ICE: Invalidation Completion Error	Hardware received an unexpected or invalid Device-TLB invalidation completion. This could be due to either an invalid ITag or invalid source-id in an invalidation completion response. At this time, a fault event may be generated based on the programming of the Fault Event Control register. Hardware implementations not supporting Device-TLBs implement this bit as RsvdZ.

Bits	Access	Default	Field	Description
4	RW1CS	0	IQE: Invalidation Queue Error	Hardware detected an error associated with the invalidation queue. This could be due to either a hardware error while fetching a descriptor from the invalidation queue, or hardware detecting an erroneous or invalid descriptor in the invalidation queue. At this time, a fault event may be generated based on the programming of the Fault Event Control register. Hardware implementations not supporting queued invalidations implement this bit as RsvdZ.
3	RW1CS	0	APF: Advanced Pending Fault	When this field is Clear, hardware sets this field when the first fault record (at index 0) is written to a fault log. At this time, a fault event is generated based on the programming of the Fault Event Control register. Software writing 1 to this field clears it. Hardware implementations not supporting advanced fault logging implement this bit as RsvdZ.
2	RW1CS	0	AFO: Advanced Fault Overflow	Hardware sets this field to indicate advanced fault log overflow condition. At this time, a fault event is generated based on the programming of the Fault Event Control register. Software writing 1 to this field clears it. Hardware implementations not supporting advanced fault logging implement this bit as RsvdZ.
1	ROS	0	PPF: Primary Pending Fault	This field indicates if there are one or more pending faults logged in the fault recording registers. Hardware computes this field as the logical OR of Fault (F) fields across all the fault recording registers of this remapping hardware unit. • 0: No pending faults in any of the fault recording registers • 1: One or more fault recording registers has pending faults. The FRI field is updated by hardware whenever the PPF field is Set by hardware. Also, depending on the programming of Fault Event Control register, a fault event is generated when hardware sets this field.
0	RW1CS	0	PFO: Primary Fault Overflow	Hardware sets this field to indicate overflow of the fault recording registers. Software writing 1 clears this field. When this field is Set, hardware does not record any new faults until software clears this field.

10.4.10 Fault Event Control Register

Figure 10-54. Fault Event Control Register

Abbreviation	FECTL_REG
General Description	Register specifying the fault event interrupt message control bits. Section 7.4 describes hardware handling of fault events.
Register Offset	038h

Bits	Access	Default	Field	Description
31	RW	1	IM: Interrupt Mask	• 0: No masking of interrupts. When a interrupt condition is detected, hardware issues an interrupt message (using the Fault Event Data and Fault Event Address register values). • 1: This is the value on reset. Software may mask interrupt message generation by setting this field.Hardware is prohibited from sending the interrupt message when this field is Set.

Bits	Access	Default	Field	Description

30	RO	0	IP: Interrupt Pending	Hardware sets the IP field whenever it detects an interrupt condition, which is defined as:
				• When primary fault logging is active, an interrupt condition occurs when hardware records a fault through one of the Fault Recording registers and sets the PPF field in the Fault Status register.
				• When advanced fault logging is active, an interrupt condition occurs when hardware records a fault in the first fault record (at index 0) of the current fault log and sets the APF field in the Fault Status register.
				• Hardware detected error associated with the Invalidation Queue, setting the IQE field in the Fault Status register.
				• Hardware detected invalid Device-TLB invalidation completion, setting the ICE field in the Fault Status register.
				• Hardware detected Device-TLB invalidation completion time-out, setting the ITE field in the Fault Status register.
				• Hardware detected Page Request Queue overflow condition, setting the PRO field in the Fault Status register.
				If any of the status fields in the Fault Status register was already Set at the time of setting any of these fields, it is not treated as a new interrupt condition.
				
				The IP field is kept Set by hardware while the interrupt message is held pending. The interrupt message could be held pending due to the interrupt mask (IM field) being Set or other transient hardware conditions.
				
				The IP field is cleared by hardware as soon as the interrupt message pending condition is serviced. This could be due to either:
				• Hardware issuing the interrupt message due to either a change in the transient hardware condition that caused the interrupt message to be held pending, or due to software clearing the IM field.
				• Software servicing all the pending interrupt status fields in the Fault Status register as follows.
				  • When primary fault logging is active, software clearing the Fault (F) field in all the Fault Recording registers with faults, causing the PPF field in the Fault Status register to be evaluated as Clear.
				  • Software clearing other status fields in the Fault Status register by writing back the value read from the respective fields.

29:0	RsvdP	Xh	R: Reserved	Reserved.

10.4.11 Fault Event Data Register

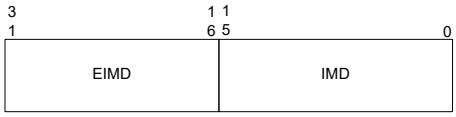


Figure 10-55. Fault Event Data Register

Abbreviation	FEDATA_REG
General Description	Register specifying the interrupt message data.
Register Offset	03Ch

Bits	Access	Default	Field	Description
31:16	RW	0h	EIMD: Extended Interrupt Message Data	This field is valid only for implementations supporting 32-bit interrupt data fields. Hardware implementations supporting only 16-bit interrupt data treat this field as RsvdZ.
15:0	RW	0h	IMD: Interrupt Message data	Data value in the interrupt request. Software requirements for programming this register are described in Section 5.1.6.

10.4.12 Fault Event Address Register

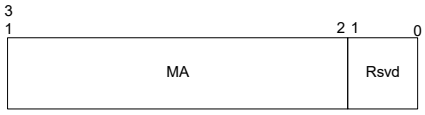


Figure 10-56. Fault Event Address Register

Abbreviation	FEADDR_REG
General Description	Register specifying the interrupt message address.
Register Offset	040h

Bits	Access	Default	Field	Description
31:2	RW	0h	MA: Message address	When fault events are enabled, the contents of this register specify the DWORD-aligned address (bits 31:2) for the interrupt request. Software requirements for programming this register are described in Section 5.1.6.
1:0	RsvdZ	0h	R: Reserved	Reserved.

10.4.13 Fault Event Upper Address Register

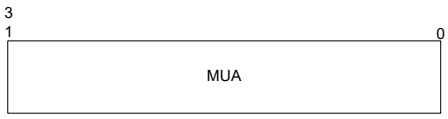


Figure 10-57. Fault Event Upper Address Register

Abbreviation	FEUADDR_REG
General Description	Register specifying the interrupt message upper address.
Register Offset	044h

Bits	Access	Default	Field	Description
31:0	RW	0h	MUA: Message upper address	Hardware implementations supporting Extended Interrupt Mode are required to implement this register. Software requirements for programming this register are described in Section 5.1.6. Hardware implementations not supporting Extended Interrupt Mode may treat this field as RsvdZ.

10.4.14 Fault Recording Registers [n]

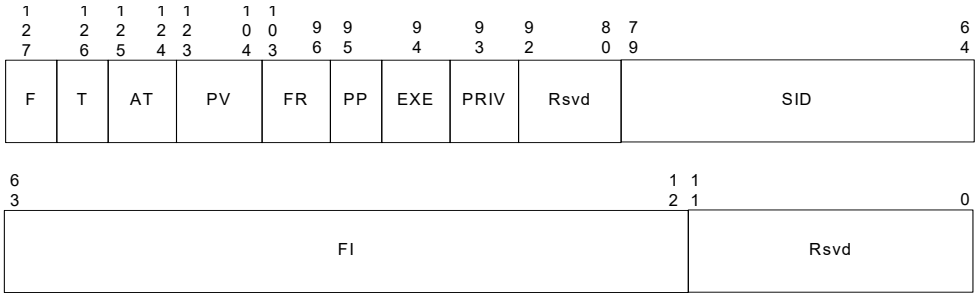


Figure 10-58. Fault Recording Register

Abbreviation	FRCD_REG [n]
General Description	Registers to record fault information when primary fault logging is active.Hardware reports the number and location of fault recording registers through the Capability register. This register is relevant only for primary fault logging. These registers are sticky and can be cleared only through power good reset or by software clearing the RW1C fields by writing a 1.
Register Offset	YYYh (YYYh must be 128-bit aligned)

Bits	Access	Default	Field	Description
127	RW1CS	0	F: Fault[1]	Hardware sets this field to indicate a fault is logged in this Fault Recording register. The F field is Set by hardware after the details of the fault is recorded in other fields. When this field is Set, hardware may collapse additional faults from the same source-id (SID). Software writes the value read from this field to Clear it. Refer to Section 7.3.1 for hardware details of primary fault logging.
126	ROS	X	T: Type	Type of the faulted request: • 0: Write request or Page (PRS) Request • 1: Read request or AtomicOp request This field is relevant only when the F field is Set, and when the fault reason (FR) indicates one of the address translation fault conditions.

Bits	Access	Default	Field	Description
125:124	ROS	Xh	AT: Address Type	This field captures the AT field from the faulted request. Hardware implementations not supporting Device-TLBs (DT field Clear in Extended Capability register) treat this field as RsvdZ. When supported, this field is valid only when the F field is Set, and when the fault reason (FR) indicates one of the non-recoverable address translation fault conditions.
123:104	ROS	Xh	PV: PASID Value	PASID value in the faulted request. This field is relevant only when the PP field is Set. Hardware implementations not supporting PASID (PASID field Clear in Extended Capability register) implement this field as RsvdZ.
103:96	ROS	Xh	FR: Fault Reason	Reason for the fault. Appendix A enumerates the various translation fault reason encodings. This field is relevant only when the F field is Set.
95	ROS	X	PP: PASID Present	When Set, indicates the faulted request has a PASID tag. The value of the PASID field is reported in the PASID Value (PV) field. This field is relevant only when the F field is Set, and when the fault reason (FR) indicates one of the non-recoverable address translation fault conditions. Hardware implementations not supporting PASID (PASID field Clear in Extended Capability register) implement this field as RsvdZ.
94	ROS	X	EXE: Execute Permission Requested	When Set, indicates Execute permission was requested by the faulted read request. This field is relevant only when the PP field and T field are both Set. Hardware implementations not supporting PASID (PASID field Clear in Extended Capability register) implement this field as RsvdZ.
93	ROS	X	PRIV: Privilege Mode Requested	When Set, indicates Supervisor privilege was requested by the faulted request. This field is relevant only when the PP field is Set. Hardware implementations not supporting PASID (PASID field Clear in Extended Capability register) implement this field as RsvdZ.
92:80	RsvdZ	0h	R: Reserved	Reserved.
79:64	ROS	Xh	SID: Source Identifier	Requester-id associated with the fault condition. This field is relevant only when the F field is Set.

Bits	Access	Default	Field	Description
63:12	ROS	Xh	FI: Fault Info	When the Fault Reason (FR) field indicates one of the address translation fault conditions, this field contains bits 63:12 of the page address in the faulted request. When PASID Present field is 0 (i.e, faulted request is a request without PASID), hardware treat bits 63:N as reserved (0), where N is the maximum guest address width (MGAW) supported. For requests-with-PASID (PASID Present field = 1), hardware treats bits 63:N as reserved (0), where N corresponds to the largest AGAW value supported by hardware. When the Fault Reason (FR) field indicates interrupt-remapping fault conditions other than Fault Reason 25h, bits 63:48 of this field indicate the interrupt_index computed for the faulted interrupt request, and bits 47:12 are cleared. When the Fault Reason (FR) field indicates interrupt-remapping fault condition of blocked Compatibility mode interrupt (Fault Reason 25h), contents of this field is undefined. This field is relevant only when the F field is Set.
11:0	RsvdZ	0h	R: Reserved	Reserved.

1. Hardware updates to this register may be disassembled as multiple doubleword writes. To ensure consistent data is read from this register, software must first check the Primary Pending Fault (PPF) field in the FSTS_REG is Set before reading the fault reporting register at offset as indicated in the FRI field of FSTS_REG. Alternatively, software may read the highest doubleword in a fault recording register and check if the Fault (F) field is Set before reading the rest of the data fields in that register.

10.4.15 Advanced Fault Log Register

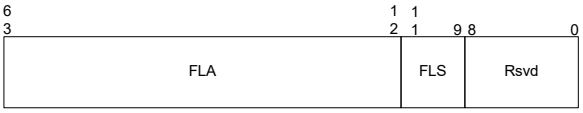


Figure 10-59. Advanced Fault Log Register

Abbreviation	AFLOG_REG
General Description	Register to specify the base address of the memory-resident fault-log region. This register is treated as RsvdZ for implementations not supporting advanced translation-fault logging (AFL field reported as 0 in the Capability register).
Register Offset	058h

Bits	Access	Default	Field	Description
63:12	RW	0h	FLA: Fault Log Address	This field specifies the base of 4KB aligned fault-log region in system memory. Hardware may ignore and not implement bits 63:HAW, where HAW is the host address width. Software specifies the base address and size of the fault log region through this register, and programs it in hardware through the SFL field in the Global Command register. When implemented, reads of this field return the value that was last programmed to it.
11:9	RW	0h	FLS: Fault Log Size	This field specifies the size of the fault log region pointed by the FLA field. The size of the fault log region is 2^X * 4KB, where X is the value programmed in this register. When implemented, reads of this field return the value that was last programmed to it.
8:0	RsvdZ	0h	R: Reserved	Reserved.

10.4.16 Protected Memory Enable Register

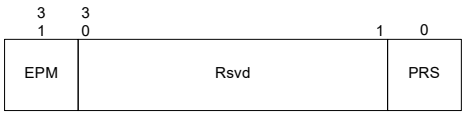


Figure 10-60. Protected Memory Enable Register

Abbreviation	PMEN_REG
General Description	Register to enable the DMA-protected memory regions set up through the PLMBASE, PLMLIMT, PHMBASE, PHMLIMIT registers. This register is always treated as RO for implementations not supporting protected memory regions (PLMR and PHMR fields reported as Clear in the Capability register). Protected memory regions may be used by software to securely initialize remapping structures in memory. To avoid impact to legacy BIOS usage of memory, software is recommended to not overlap protected memory regions with any reserved memory regions of the platform reported through the Reserved Memory Region Reporting (RMRR) structures described in Chapter 8.
Register Offset	064h

Bits	Access	Default	Field	Description

31	RW	0h	EPM: Enable Protected Memory	This field controls DMA accesses to the protected low-memory and protected high-memory regions.
				• 0: Protected memory regions are disabled.
				• 1: Protected memory regions are enabled. DMA requests accessing protected memory regions are handled as follows:
				  • When DMA remapping is not enabled, all DMA requests accessing protected memory regions are blocked.
				  • When DMA remapping is enabled:
				  • DMA requests processed as pass-through (Translation Type value of 10b in Context-Entry) and accessing the protected memory regions are blocked.
				  • DMA requests with translated address (AT=10b) and accessing the protected memory regions are blocked.
				  • DMA requests that are subject to address remapping, and accessing the protected memory regions may or may not be blocked by hardware. For such requests, software must not depend on hardware protection of the protected memory regions, and instead program the remapping structures to block requests to protected memory regions.
				
				Remapping hardware access to the remapping structures are not subject to protected memory region checks.
				
				DMA requests blocked due to protected memory region violation are not recorded or reported as remapping faults.
				
				Hardware reports the status of the protected memory enable/disable operation through the PRS field in this register. Hardware implementations supporting DMA draining must drain any in-flight translated DMA requests queued within the Root-Complex before indicating the protected memory region as enabled through the PRS field.
				
				After writing to this field software must wait for the operation to be completed and reflected in the PRS status field (bit 0) before changing the value of this field again.

30:1	RsvdP	Xh	R: Reserved	Reserved.
0	RO	0h	PRS: Protected Region Status	This field indicates the status of protected memory region(s): • 0: Protected memory region(s) disabled. • 1: Protected memory region(s) enabled.

10.4.17 Protected Low-Memory Base Register

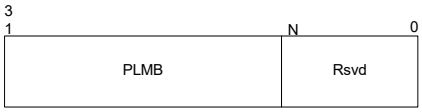


Figure 10-61. Protected Low-Memory Base Register

Abbreviation	PLMBASE_REG
General Description	Register to set up the base address of DMA-protected low-memory region below 4GB. This register must be set up before enabling protected memory through PMEN_REG, and must not be updated when protected memory regions are enabled. This register is always treated as RO for implementations not supporting protected low memory region (PLMR field reported as Clear in the Capability register). The alignment of the protected low memory region base depends on the number of reserved bits (N:0) of this register. Software may determine N by writing all 1s to this register, and finding the most significant bit position with 0 in the value read back from the register. Bits N:0 of this register are decoded by hardware as all 0s. Software must setup the protected low memory region below 4GB. Section 10.4.18 describes the Protected Low-Memory Limit register and hardware decoding of these registers. Software must not modify this register when protected memory regions are enabled (PRS field Set in PMEN_REG)
Register Offset	068h

Bits	Access	Default	Field	Description
31:(N+1)	RW	0h	PLMB: Protected Low-Memory Base	This register specifies the base of protected low-memory region in system memory.
N:0	RsvdZ	0h	R: Reserved	Reserved.

10.4.18 Protected Low-Memory Limit Register

Figure 10-62. Protected Low-Memory Limit Register

Abbreviation	PLMLIMIT_REG
General Description	Register to set up the limit address of DMA-protected low-memory region below 4GB. This register must be set up before enabling protected memory through PMEN_REG, and must not be updated when protected memory regions are enabled. This register is always treated as RO for implementations not supporting protected low memory region (PLMR field reported as Clear in the Capability register). The alignment of the protected low memory region limit depends on the number of reserved bits (N:0) of this register. Software may determine N by writing all 1's to this register, and finding most significant zero bit position with 0 in the value read back from the register. Bits N:0 of the limit register are decoded by hardware as all 1s. The Protected low-memory base and limit registers function as follows: • Programming the protected low-memory base and limit registers the same value in bits 31:(N+1) specifies a protected low-memory region of size $2^{(N+1)}$ bytes. • Programming the protected low-memory limit register with a value less than the protected low-memory base register disables the protected low-memory region. Software must not modify this register when protected memory regions are enabled (PRS field Set in PMEN_REG)
Register Offset	06Ch

Bits	Access	Default	Field	Description
31:(N+1)	RW	0h	PLML: Protected Low-Memory Limit	This register specifies the last host physical address of the DMA-protected low-memory region in system memory.
N:0	RsvdZ	0h	R: Reserved	Reserved.

10.4.19 Protected High-Memory Base Register

Figure 10-63. Protected High-Memory Base Register

Abbreviation	PHMBASE_REG
General Description	Register to set up the base address of DMA-protected high-memory region. This register must be set up before enabling protected memory through PMEN_REG, and must not be updated when protected memory regions are enabled. This register is always treated as RO for implementations not supporting protected high memory region (PHMR field reported as Clear in the Capability register). The alignment of the protected high memory region base depends on the number of reserved bits (N:0) of this register. Software may determine N by writing all 1's to this register, and finding most significant zero bit position below host address width (HAW) in the value read back from the register. Bits N:0 of this register are decoded by hardware as all 0s. Software may setup the protected high memory region either above or below 4GB. Section 10.4.20 describes the Protected High-Memory Limit register and hardware decoding of these registers. Software must not modify this register when protected memory regions are enabled (PRS field Set in PMEN_REG)
Register Offset	070h

Bits	Access	Default	Field	Description
63:(N+1)	RW	0h	PHMB: Protected High-Memory Base	This register specifies the base of protected (high) memory region in system memory. Hardware may ignore and not implement bits 63:HAW, where HAW is the host address width.
N:0	RsvdZ	0h	R: Reserved	Reserved.

10.4.20 Protected High-Memory Limit Register

Figure 10-64. Protected High-Memory Limit Register

Abbreviation	PHMLIMIT_REG
General Description	Register to set up the limit address of DMA-protected high-memory region. This register must be set up before enabling protected memory through PMEN_REG, and must not be updated when protected memory regions are enabled. This register is always treated as RO for implementations not supporting protected high memory region (PHMR field reported as Clear in the Capability register). The alignment of the protected high memory region limit depends on the number of reserved bits (N:0) of this register. Software may determine N by writing all 1's to this register, and finding most significant zero bit position below host address width (HAW) in the value read back from the register. Bits N:0 of the limit register are decoded by hardware as all 1s. The protected high-memory base & limit registers function as follows. • Programming the protected low-memory base and limit registers with the same value in bits HAW:(N+1) specifies a protected low-memory region of size $2^{(N+1)}$ bytes. • Programming the protected high-memory limit register with a value less than the protected high-memory base register disables the protected high-memory region. Software must not modify this register when protected memory regions are enabled (PRS field Set in PMEN_REG)
Register Offset	078h

Bits	Access	Default	Field	Description
63:(N+1)	RW	0h	PHML: Protected High-Memory Limit	This register specifies the last host physical address of the DMA-protected high-memory region in system memory. Hardware may ignore and not implement bits 63:HAW, where HAW is the host address width.
N:0	RsvdZ	0h	R: Reserved	Reserved.

10.4.21 Invalidation Queue Head Register

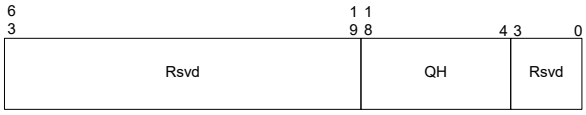


Figure 10-65. Invalidation Queue Head Register

Abbreviation	IQH_REG
General Description	Register indicating the invalidation queue head. This register is treated as RsvdZ by implementations reporting Queued Invalidation (QI) as not supported in the Extended Capability register.
Register Offset	080h

Bits	Access	Default	Field	Description
63:19	RsvdZ	0h	R: Reserved	Reserved.
18:4	RO	0h	QH: Queue Head	Specifies the offset (128-bit aligned) to the invalidation queue for the command that will be fetched next by hardware. Hardware resets this field to 0 whenever the queued invalidation is disabled (QIES field Clear in the Global Status register).
3:0	RsvdZ	0h	R: Reserved	Reserved.

10.4.22 Invalidation Queue Tail Register

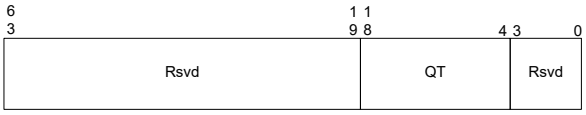


Figure 10-66. Invalidation Queue Tail Register

Abbreviation	IQT_REG
General Description	Register indicating the invalidation tail. This register is treated as RsvdZ by implementations reporting Queued Invalidation (QI) as not supported in the Extended Capability register.
Register Offset	088h

Bits	Access	Default	Field	Description
63:19	RsvdZ	0h	R: Reserved	Reserved.
18:4	RW	0h	QT: Queue Tail	Specifies the offset (128-bit aligned) to the invalidation queue for the command that will be written next by software.
3:0	RsvdZ	0h	R: Reserved	Reserved.

10.4.23 Invalidation Queue Address Register

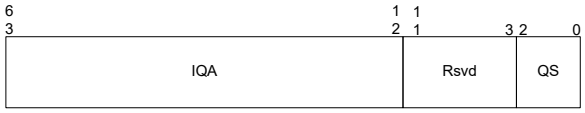


Figure 10-67. Invalidation Queue Address Register

Abbreviation	IQA_REG
General Description	Register to configure the base address and size of the invalidation queue. This register is treated as RsvdZ by implementations reporting Queued Invalidation (QI) as not supported in the Extended Capability register.
Register Offset	090h

Bits	Access	Default	Field	Description
63:12	RW	0h	IQA: Invalidation Queue Base Address	This field points to the base of 4KB aligned invalidation request queue. Hardware may ignore and not implement bits 63:HAW, where HAW is the host address width. Reads of this field return the value that was last programmed to it.
11:3	RsvdZ	0h	R: Reserved	Reserved.
2:0	RW	0h	QS: Queue Size	This field specifies the size of the invalidation request queue. A value of X in this field indicates an invalidation request queue of (2^X) 4KB pages. The number of entries in the invalidation queue is $2^{(X + 8)}$.

10.4.24 Invalidation Completion Status Register

Figure 10-68. Invalidation Completion Status Register

Abbreviation	ICS_REG
General Description	Register to report completion status of invalidation wait descriptor with Interrupt Flag (IF) Set. This register is treated as RsvdZ by implementations reporting Queued Invalidation (QI) as not supported in the Extended Capability register.
Register Offset	09Ch

Bits	Access	Default	Field	Description
31:1	RsvdZ	0h	R: Reserved	Reserved.
0	RW1CS	0	IWC: Invalidation Wait Descriptor Complete	Indicates completion of Invalidation Wait Descriptor with Interrupt Flag (IF) field Set. Hardware implementations not supporting queued invalidations implement this field as RsvdZ.

10.4.25 Invalidation Event Control Register

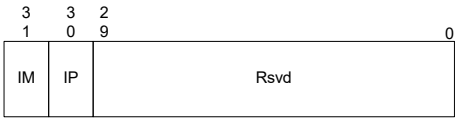


Figure 10-69. Invalidation Event Control Register

Abbreviation	IECTL_REG
General Description	Register specifying the invalidation event interrupt control bits. This register is treated as RsvdZ by implementations reporting Queued Invalidation (QI) as not supported in the Extended Capability register.
Register Offset	0A0h

Bits	Access	Default	Field	Description
31	RW	1	IM: Interrupt Mask	• 0: No masking of interrupt. When a invalidation event condition is detected, hardware issues an interrupt message (using the Invalidation Event Data & Invalidation Event Address register values). • 1: This is the value on reset. Software may mask interrupt message generation by setting this field. Hardware is prohibited from sending the interrupt message when this field is Set.
30	RO	0	IP: Interrupt Pending	Hardware sets the IP field whenever it detects an interrupt condition. Interrupt condition is defined as: • An Invalidation Wait Descriptor with Interrupt Flag (IF) field Set completed, setting the IWC field in the Invalidation Completion Status register. • If the IWC field in the Invalidation Completion Status register was already Set at the time of setting this field, it is not treated as a new interrupt condition. The IP field is kept Set by hardware while the interrupt message is held pending. The interrupt message could be held pending due to interrupt mask (IM field) being Set, or due to other transient hardware conditions. The IP field is cleared by hardware as soon as the interrupt message pending condition is serviced. This could be due to either: • Hardware issuing the interrupt message due to either change in the transient hardware condition that caused interrupt message to be held pending or due to software clearing the IM field. • Software servicing the IWC field in the Invalidation Completion Status register.
29:0	RsvdP	Xh	R: Reserved	Reserved.

10.4.26 Invalidation Event Data Register

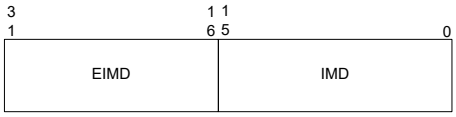


Figure 10-70. Invalidation Event Data Register

Abbreviation	IEDATA_REG
General Description	Register specifying the Invalidation Event interrupt message data. This register is treated as RsvdZ by implementations reporting Queued Invalidation (QI) as not supported in the Extended Capability register.
Register Offset	0A4h

Bits	Access	Default	Field	Description
31:16	RW	0h	EIMD: Extended Interrupt Message Data	This field is valid only for implementations supporting 32-bit interrupt data fields. Hardware implementations supporting only 16-bit interrupt data treat this field as RsvdZ.
15:0	RW	0h	IMD: Interrupt Message data	Data value in the interrupt request. Software requirements for programming this register are described in Section 5.1.6.

10.4.27 Invalidation Event Address Register

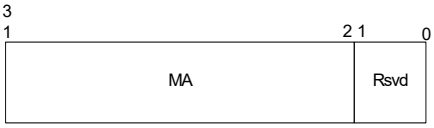


Figure 10-71. Invalidation Event Address Register

Abbreviation	IEADDR_REG
General Description	Register specifying the Invalidation Event Interrupt message address.This register is treated as RsvdZ by implementations reporting Queued Invalidation (QI) as not supported in the Extended Capability register.
Register Offset	0A8h

Bits	Access	Default	Field	Description
31:2	RW	0h	MA: Message address	When fault events are enabled, the contents of this register specify the DWORD-aligned address (bits 31:2) for the interrupt request. Software requirements for programming this register are described in Section 5.1.6.
1:0	RsvdZ	0h	R: Reserved	Reserved.

10.4.28 Invalidation Event Upper Address Register

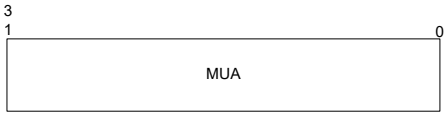


Figure 10-72. Invalidation Event Upper Address Register

Abbreviation	IEUADDR_REG
General Description	Register specifying the Invalidation Event interrupt message upper address.
Register Offset	0ACh

Bits	Access	Default	Field	Description
31:0	RW	0h	MUA: Message upper address	Hardware implementations supporting Queued Invalidations and Extended Interrupt Mode are required to implement this register. Software requirements for programming this register are described in Section 5.1.6. Hardware implementations not supporting Queued Invalidations or Extended Interrupt Mode may treat this field as RsvdZ.

10.4.29 Interrupt Remapping Table Address Register

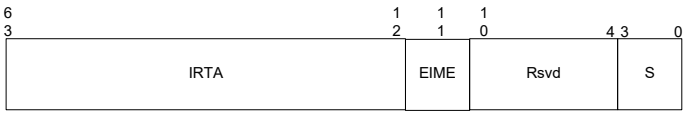


Figure 10-73. Interrupt Remapping Table Address Register

Abbreviation	IRTA_REG
General Description	Register providing the base address of Interrupt remapping table. This register is treated as RsvdZ by implementations reporting Interrupt Remapping (IR) as not supported in the Extended Capability register.
Register Offset	0B8h

Bits	Access	Default	Field	Description
63:12	RW	0h	IRTA: Interrupt Remapping Table Address	This field points to the base of 4KB aligned interrupt remapping table. Hardware may ignore and not implement bits 63:HAW, where HAW is the host address width. Reads of this field returns value that was last programmed to it.
11	RW	0	EIME: Extended Interrupt Mode Enable	This field is used by hardware on Intel® 64 platforms as follows: • 0: xAPIC mode is active. Hardware interprets only 8-bits ([15:8]) of Destination-ID field in the IRTEs. The high 16-bits and low 8-bits of the Destination-ID field are treated as reserved. • 1: x2APIC mode is active. Hardware interprets all 32-bits of Destination-ID field in the IRTEs. This field is implemented as RsvdZ on implementations reporting Extended Interrupt Mode (EIM) field as Clear in Extended Capability register.
10:4	RsvdZ	0h	R: Reserved	Reserved.
3:0	RW	0h	S: Size	This field specifies the size of the interrupt remapping table. The number of entries in the interrupt remapping table is 2^{X+1}, where X is the value programmed in this field.

10.4.30 Page Request Queue Head Register

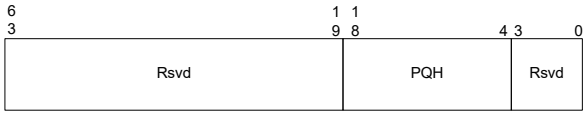


Figure 10-74. Page Request Queue Head Register

Abbreviation	PQH_REG
General Description	Register indicating the page request queue head. This register is treated as RsvdZ by implementations reporting Page Request Support (PRS) as not supported in the Extended Capability register.
Register Offset	0C0h

Bits	Access	Default	Field	Description
63:19	RsvdZ	0h	R: Reserved	Reserved.
18:4	RW	0h	PQH: Page Queue Head	Specifies the offset (16-bytes aligned) to the page request queue for the request that will be processed next by software.
3:0	RsvdZ	0h	R: Reserved	Reserved.

10.4.31 Page Request Queue Tail Register

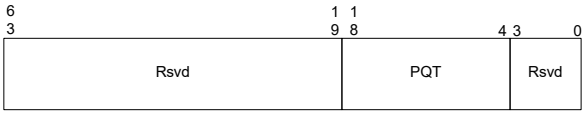


Figure 10-75. Page Request Queue Tail Register

Abbreviation	PQT_REG
General Description	Register indicating the page request queue tail. This register is treated as RsvdZ by implementations reporting Page Request Support (PRS) as not supported in the Extended Capability register.
Register Offset	0C8h

Bits	Access	Default	Field	Description
63:19	RsvdZ	0h	R: Reserved	Reserved.
18:4	RW	0h	PQT: Page Queue Tail	Specifies the offset (16-bytes aligned) to the page request queue for the request that will be written next by hardware.
3:0	RsvdZ	0h	R: Reserved	Reserved.

10.4.32 Page Request Queue Address Register

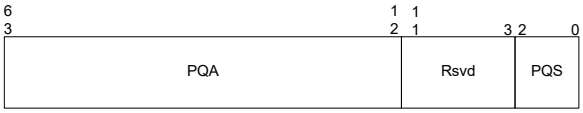


Figure 10-76. Page Request Queue Address Register

Abbreviation	PQA_REG
General Description	Register to configure the base address and size of the page request queue. This register is treated as RsvdZ by implementations reporting Page Request Support (PRS) as not supported in the Extended Capability register.
Register Offset	0D0h

Bits	Access	Default	Field	Description
63:12	RW	0h	PQA: Page Request Queue Base Address	This field points to the base of 4KB aligned page request queue. Hardware may ignore and not implement bits 63:HAW, where HAW is the host address width. Software must configure this register before enabling page requests in any extended-context-entries.
11:3	RsvdZ	0h	R: Reserved	Reserved.
2:0	RW	0h	PQS: Page Queue Size	This field specifies the size of the page request queue. A value of X in this field indicates an invalidation request queue of (2^X) 4KB pages. The number of entries in the page request queue is $2^{(X + 8)}$.

10.4.33 Page Request Status Register

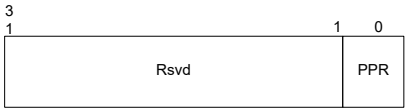


Figure 10-77. Page Request Status Register

Abbreviation	PRS_REG
General Description	Register to report pending page request in page request queue. This register is treated as RsvdZ by implementations reporting Page Request Support (PRS) as not supported in the Extended Capability register.
Register Offset	0DCh

Bits	Access	Default	Field	Description
31:1	RsvdZ	0h	R: Reserved	Reserved.
0	RW1CS	0	PPR: Pending Page Request	Indicates pending page requests to be serviced by software in the page request queue. This field is Set by hardware when a page request entry (*page_req_dsc*) is added to the page request queue.

10.4.34 Page Request Event Control Register

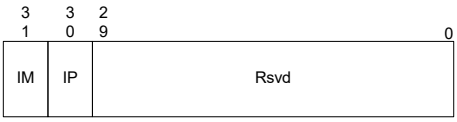


Figure 10-78. Page Request Event Control Register

Abbreviation	PECTL_REG
General Description	Register specifying the page request event interrupt control bits. This register is treated as RsvdZ by implementations reporting Page Request Support (PRS) as not supported in the Extended Capability register.
Register Offset	0E0h

Bits	Access	Default	Field	Description
31	RW	1	IM: Interrupt Mask	• 0: No masking of interrupt. When a page request event condition is detected, hardware issues an interrupt message (using the Page Request Event Data & Page Request Event Address register values). • 1: This is the value on reset. Software may mask interrupt message generation by setting this field. Hardware is prohibited from sending the interrupt message when this field is Set.
30	RO	0	IP: Interrupt Pending	Hardware sets the IP field whenever it detects an interrupt condition. Interrupt condition is defined as: • A page request entry (page_req_dsc) was added to page request queue, resulting in hardware setting the Pending Page Request (PPR) field in Page Request Status register. • If the PPR field in the Page Request Event Status register was already Set at the time of setting this field, it is not treated as a new interrupt condition. The IP field is kept Set by hardware while the interrupt message is held pending. The interrupt message could be held pending due to interrupt mask (IM field) being Set, or due to other transient hardware conditions. The IP field is cleared by hardware as soon as the interrupt message pending condition is serviced. This could be due to either: • Hardware issuing the interrupt message due to either change in the transient hardware condition that caused interrupt message to be held pending or due to software clearing the IM field. • Software servicing the PPR field in the Page Request Event Status register.
29:0	RsvdP	Xh	R: Reserved	Reserved.

10.4.35 Page Request Event Data Register

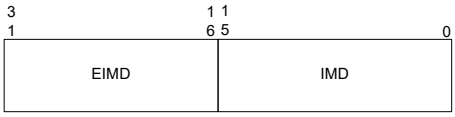


Figure 10-79. Page Request Event Data Register

Abbreviation	PEDATA_REG
General Description	Register specifying the Page Request Event interrupt message data. This register is treated as RsvdZ by implementations reporting Page Request Support (PRS) as not supported in the Extended Capability register.
Register Offset	0E4h

Bits	Access	Default	Field	Description
31:16	RW	0h	EIMD: Extended Interrupt Message Data	This field is valid only for implementations supporting 32-bit interrupt data fields. Hardware implementations supporting only 16-bit interrupt data treat this field as RsvdZ.
15:0	RW	0h	IMD: Interrupt Message data	Data value in the interrupt request. Software requirements for programming this register are described in Section 5.1.6.

10.4.36 Page Request Event Address Register

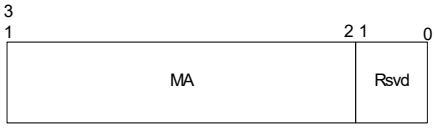


Figure 10-80. Page Request Event Address Register

Abbreviation	PEADDR_REG
General Description	Register specifying the Page Request Event Interrupt message address.This register is treated as RsvdZ by implementations reporting Page Request Support (PRS) as not supported in the Extended Capability register.
Register Offset	0E8h

Bits	Access	Default	Field	Description
31:2	RW	0h	MA: Message address	When fault events are enabled, the contents of this register specify the DWORD-aligned address (bits 31:2) for the interrupt request. Software requirements for programming this register are described in Section 5.1.6.
1:0	RsvdZ	0h	R: Reserved	Reserved.

10.4.37 Page Request Event Upper Address Register

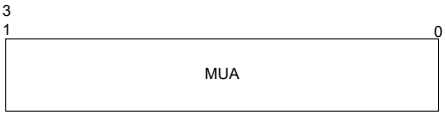


Figure 10-81. Page Request Event Upper Address Register

Abbreviation	PEUADDR_REG
General Description	Register specifying the Page Request Event interrupt message upper address.
Register Offset	0ECh

Bits	Access	Default	Field	Description
31:0	RW	0h	MUA: Message upper address	This field specifies the upper address (bits 63:32) for the page request event interrupt. Software requirements for programming this register are described in Section 5.1.6. Hardware implementations not supporting Extended Interrupt Mode may treat this field as RsvdZ.

10.4.38 MTRR Capability Register

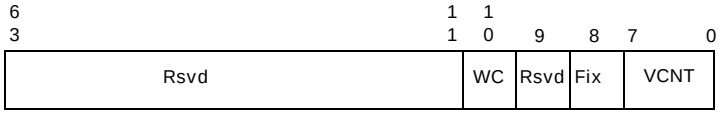


Figure 10-82. MTRR Capability Register

Abbreviation	MTRRCAP_REG
General Description	Register reporting the Memory Type Range Register Capability. This register is treated as RsvdZ by implementations reporting Memory Type Support (MTS) as not supported in the Extended Capability register. When implemented, value reported in this register must match IA32_MTRRCAP Model Specific Register (MSR) value reported by the host IA-32 processor(s).
Register Offset	100h

Bits	Access	Default	Field	Description
63:11	RsvdZ	0h	R: Reserved	Reserved.
10	RO	X	WC: Write Combining	• 0: Write-combining (WC) memory type is not supported. • 1: Write-combining (WC) memory type is supported.
9	RsvdZ	0	R: Reserved	Reserved.
8	RO	X	FIX: Fixed Range MTRRs Supported	• 0: No fixed range MTRRs are supported • 1: Fixed range MTRRs (MTRR_FIX64K_00000 through MTRR_FIX4K_0F8000) are supported
7:0	RO	X	VCNT: Variable MTRR Count	Indicates number of variable range MTRRs supported

10.4.39 MTRR Default Type Register

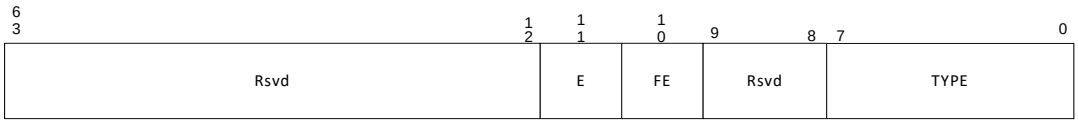


Figure 10-83. MTRR Default Type Register

Abbreviation	MTRRDEF_REG
General Description	Register for enabling/configuring Memory Type Range Registers. This register is treated as RsvdZ by implementations reporting Memory Type Support (MTS) as not supported in the Extended Capability register.
Register Offset	108h

Bits	Access	Default	Field	Description
63:12	RsvdZ	0h	R: Reserved	Reserved.
11	RW	0	E: MTRR Enable	• 0: Disable MTRRs; UC memory type is applied. FE field has no effect. • 1: Enable MTRRs. FE field can disable the fixed-range MTRRs. Type specified in the default memory type field is used for areas of memory not already mapped by either fixed or variable MTRR.
10	RW	0	FE: Fixed Range MTRR Enable	• 0: Disable fixed range MTRRs. • 1: Enable fixed range MTRRs. When fixed range MTRRs are enabled, they take priority over the variable range MTRRs when overlaps in ranges occur. If the fixed-range MTRRs are disabled, the variable range MTRRs can still be used and can map the range ordinarily covered by the fixed range MTRRs.
9:8	RsvdZ	0h	R: Reserved	Reserved.
7:0	RW	0h	TYPE: Default Memory Type	Indicates default memory type used for physical memory address ranges that do not have a memory type specified for them by an MTRR. Legal values for this field are 0,1,4, 5 and 6.

10.4.40 Fixed-Range MTRRs

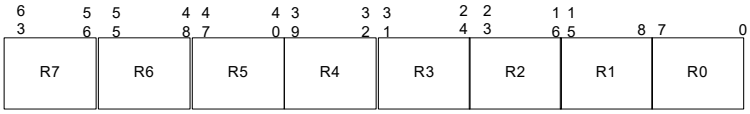


Figure 10-84. Fixed-Range MTRR Format

Abbreviation	MTRR_FIX64_00000_REG MTRR_FIX16K_80000_REG MTRR_FIX16K_A0000_REG MTRR_FIX4K_C0000_REG MTRR_FIX4K_C8000_REG MTRR_FIX4K_D0000_REG MTRR_FIX4K_D8000_REG MTRR_FIX_4K_E0000_REG MTRR_FIX_4K_E8000REG MTRR_FIX4K_F0000_REG MTRR_FIX_4K_F8000_REG
General Description	Fixed-range Memory Type Range Registers. These include 11 registers as illustrated in Table 32. These registers are treated as RsvdZ by implementations reporting Memory Type Support (MTS) as not supported in the Extended Capability register.
Register Offsets	120h, 128h, 130h, 138h, 140h, 148h, 150h, 158h, 160h, 168h, 170h

Bits	Access	Default	Field	Description
63:56	RW	0h	R7	Register Field 7
55:48	RW	0h	R6	Register Field 6
47:40	RW	0h	R5	Register Field 5
39:32	RW	0h	R4	Register Field 4
31:24	RW	0h	R3	Register Field 3
23:16	RW	0h	R2	Register Field 2
15:8	RW	0h	R1	Register Field 1
7:0	RW	0h	R0	Register Field 0

Table 32. Address Mapping for Fixed-Range MTRRs

Address Range (hexadecimal)								MTRR
63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0	
70000-7FFFF	60000-6FFFF	50000-5FFFF	40000-4FFFF	30000-3FFFF	20000-2FFFF	10000-1FFFF	00000-0FFFF	MTRR_FIX64K_00000_REG
9C000-9FFFF	98000-98FFF	94000-97FFF	90000-93FFF	8C000-8FFFF	88000-8BFFF	84000-87FFF	80000-83FFF	MTRR_FIX16K_80000_REG
BC000-BFFFF	B8000-B8FFF	B4000-B7FFF	B0000-B3FFF	AC000-AFFFF	A8000-ABFFF	A4000-A7FFF	A0000-A3FFF	MTRR_FIX16K_A0000_REG
C7000-C7FFF	C6000-C6FFF	C5000-C5FFF	C4000-C4FFF	C3000-C3FFF	C2000-C2FFF	C1000-C1FFF	C0000-C0FFF	MTRR_FIX4K_C0000_REG
CF000-CFFFF	CE000-CEFFF	CD000-CDFFF	CC000-CCFFF	CB000-CBFFF	CA000-CAFFF	C9000-C9FFF	C8000-C8FFF	MTRR_FIX4K_C8000_REG
D7000-D7FFF	D6000-D6FFF	D5000-D5FFF	D4000-D4FFF	D3000-D3FFF	D2000-D2FFF	D1000-D1FFF	D0000-D0FFF	MTRR_FIX4K_D0000_REG
DF000-DFFFF	DE000-DEFFF	DD000-DDFFF	DC000-DCFFF	DB000-DBFFF	DA000-DAFFF	D9000-D9FFF	D8000-D8FFF	MTRR_FIX4K_D8000_REG
E7000-E7FFF	E6000-E6FFF	E5000-E5FFF	E4000-E4FFF	E3000-E3FFF	E2000-E2FFF	E1000-E1FFF	E0000-E0FFF	MTRR_FIX4K_E0000_REG
EF000-EFFFF	EE000-EEFFF	ED000-EDFFF	EC000-ECFFF	EB000-EBFFF	EA000-EAFFF	E9000-E9FFF	E8000-E8FFF	MTRR_FIX4K_E8000_REG
F7000-F7FFF	F6000-F6FFF	F5000-F5FFF	F4000-F4FFF	F3000-F3FFF	F2000-F2FFF	F1000-F1FFF	F0000-F0FFF	MTRR_FIX4K_F0000_REG
FF000-FFFFF	FE000-FEFFF	FD000-FDFFF	FC000-FCFFF	FB000-FBFFF	FA000-FAFFF	F9000-F9FFF	F8000-F8FFF	MTRR_FIX4K_F8000_REG

10.4.41 Variable-Range MTRRs

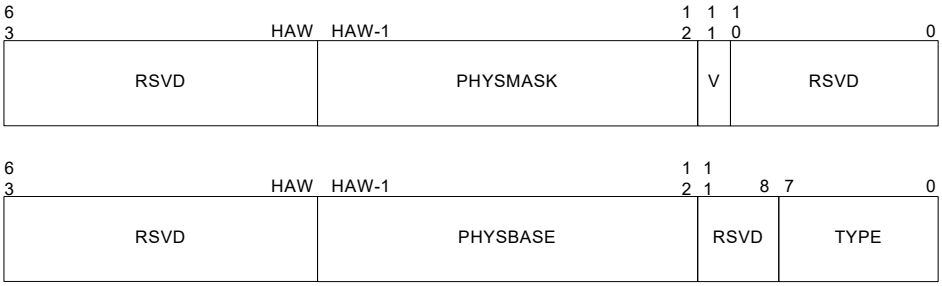


Figure 10-85. Variable-Range MTRR Format

Abbreviation	MTRR_PHYSBASE0_REG, MTRR_PHYSMASK0_REG MTRR_PHYSBASE1_REG, MTRR_PHYSMASK1_REG MTRR_PHYSBASE2_REG, MTRR_PHYSMASK2_REG MTRR_PHYSBASE3_REG, MTRR_PHYSMASK3_REG MTRR_PHYSBASE4_REG, MTRR_PHYSMASK4_REG MTRR_PHYSBASE5_REG, MTRR_PHYSMASK5_REG MTRR_PHYSBASE6_REG, MTRR_PHYSMASK6_REG MTRR_PHYSBASE7_REG, MTRR_PHYSMASK7_REG MTRR_PHYSBASE8_REG, MTRR_PHYSMASK8_REG MTRR_PHYSBASE9_REG, MTRR_PHYSMASK9_REG
General Description	Variable-range Memory Type Range Registers. Each Variable-range MTRR register includes a low 64-bit Base register and a high 64-bit Mask register. VCNT field in MTRRCAP_REG reports number of Variable-range MTRRs supported by hardware. These registers are treated as RsvdZ by implementations reporting Memory Type Support (MTS) as not supported in the Extended Capability register.
Register Offsets	180h, 188h, 190h, 198h, A0h, 1A8h, 1B0h, 1B8h, 1C0h, 1C8h, 1D0h, 1D8h, 1E0h, 1E8h, 1F0h, 1F8h, 200h, 208h, 210h, 218h

Bits	Access	Default	Field	Description
63:HAW	RsvdZ	0h	R: Reserved	Reserved
HAW-1:12	RW	0h	PHYSMASK: Physical Mask	Mask for range
11	RW	0	V: Valid	• 0: Not Valid • 1: Valid
10:0	RsvdZ	0h	R: Reserved	Reserved

Bits	Access	Default	Field	Description
63:HAW	RsvdZ	0h	R: Reserved	Reserved
HAW-1:12	RW	0h	PHYSBASE: Physical Base	Base address of range
11:8	RsvdZ	0h	R: Reserved	Reserved
7:0	RW	0h	TYPE: Type	Memory type for range

Appendix A Non-Recoverable Fault Reason Encodings

The following table describes the summary of fault reason codes assigned to various non-recoverable faults. Refer to Chapter 7 for details on how the fault reasons codes map to different request types.

Encoding	Fault Reason Description
0h	Reserved. Used by software when initializing fault records (for advanced fault logging)
DMA Remapping Fault Conditions	
1h	The Present (P) field in the root-entry (or UP/LP fields in the extended-root-entry) used to process a request is 0.
2h	The Present (P) field in the context-entry (or extended-context-entry) used to process a request is 0.
3h	Invalid programming of a context-entry or extended-context-entry. For example: • The Address width (AW) field is programmed with a value either not supported by hardware or inconsistent with Translation-Type (T) field. • The Translation-Type (T) field is programmed to indicate a translation type not supported by the hardware implementation. • Hardware attempt to access the second-level paging entry referenced through the SLPTPTR field resulted in error. • Hardware attempt to access the PASID-entry referenced through the PASIDTPTR field referenced an address above HAW or resulted in hardware error.
4h	Input-address to second-level translation is above (2^X - 1), where X is the minimum of the maximum guest address width (MGAW) reported through the Capability register and the value in the Address-Width (AW) field of the context-entry (or extended-context-entry) used to process the request.
5h	A non-recoverable address translation fault resulted due to lack of write permission.
6h	A non-recoverable address translation fault resulted due to lack of read permission. For implementations reporting ZLR field as set in the Capability register, this fault condition is not applicable for zero-length read requests to write-only pages.
7h	Hardware attempt to access a second-level paging entry (SL-PML4E, SL-PDPE, SL-PDE, or SL-PTE) referenced through the Address (ADDR) field in a preceding second-level paging entry (SL-PML5E, SL-PML4E, SL-PDPE, SL-PDE) resulted in error.
8h	Hardware attempt to access a root-entry (or extended-root-entry) referenced through the Root-Table Address (RTA) field in the root-table Address Register resulted in error.
9h	Hardware attempt to access a context-entry (or extended-context-entry) referenced through the Context Table Pointer (CTP) field (or UCTP/LCTP fields in an extended-root-entry) resulted in error.
Ah	Non-zero reserved field in a root-entry with Present (P) field Set (or extended-root-entry with UP/LP field Set).
Bh	Non-zero reserved field in a context-entry or extended-context-entry with Present (P) field Set.
Ch	Non-zero reserved field in a second-level paging entry (SL-PML5E, SL-PML4E, SL-PDPE, SL-PDE, or SL-PTE) with at least one of Read (R) and Write (W) fields (and Execute (E) field, if it is enabled and is applicable for the type of request) is Set.

Encoding	Fault Reason Description
Dh	Translation request, translated request, or untranslated-request-with-PASID explicitly blocked due to the programming of the Translation Type (T) field in the present context-entry or extended-context-entry.
Eh - Fh	Reserved.
10h	The PASID Enable (PASIDE) field in extended-context-entry (with P=1) used to process the untranslated request with PASID is 0.
11h	The PASID value in the untranslated request with PASID is larger than the maximum PASID-value supported by the PASID-Table-Size (PTS) field in the extended-context-entry (with P=PASIDE=1) used to process the request.
12h	The Present (P) field in the PASID-entry used to process the untranslated request with PASID is 0.
13h	Non-zero reserved field in a PASID-entry with the Present (P) field Set.
14h	Input-address in the request with PASID is not Canonical (i.e., address bits 63:N do not have the same value as address bit [N-1], where N is 48-bits with 4-level paging and 57-bits with 5-level paging).
15h	Hardware attempt to access the FL-PML4 (FL-PML5 with 5-level paging) entry referenced through the FLPTPTR field in the PASID-entry resulted in error.
16h	Non-zero reserved field in first-level paging entry (FL-PML5E, FL-PML4E, FL-PDPE, FL-PDE, or FL-PTE) with Present (P) field Set.
17h	Hardware attempt to access a first-level paging entry (FL-PML4E, FL-PDPE, FL-PDE, or FL-PTE) referenced through the Address (ADDR) field in a preceding first-level paging entry (FL-PML5E, FL-PML4E, FL-PDPE, or FL-PDE) resulted in error.
18h	A non-recoverable address translation fault resulted for untranslated request with PASID with Execute-Requested (ER) field Set, due to lack of execute permission.
19h	The Execute Requests Enable (ERE) field is 0 in extended-context-entry (with P=1) used to process untranslated request with PASID with Execute-Requested (ER) field Set.
1Ah	The Supervisor Requests Enable (SRE) field is 0 in PASID-entry (with P=1) used to process untranslated requests with PASID with Privileged-mode-Requested (PR) field Set.
1Bh	Root Table Type (RTT) field is 0 in Root-table Address register (RTADDR_REG) used to process the request with PASID.
1Ch	Privilege checks by first-level translation blocked an untranslated request with PASID with user-request privilege
1Dh - 1Fh	Reserved.

Encoding	Fault Reason Description
Interrupt Remapping Fault Conditions	
20h	Decoding of the interrupt request per the Remappable request format detected one or more reserved fields as Set.
21h	The interrupt_index value computed for the interrupt request is greater than the maximum allowed for the interrupt Remapping table-size configured by software, or hardware attempt to access the IRTE corresponding to the interrupt_index value referenced an address above Host Address Width (HAW).
22h	The Present (P) field in the IRTE corresponding to the interrupt_index of the interrupt request is Clear.
23h	Hardware attempt to access the interrupt Remapping table through the Interrupt Remapping Table Address (IRTA) field in the Interrupt Remap Table Address Register resulted in error.
24h	Hardware detected one ore more reserved fields that are not initialized to zero in an IRTE with Present (P) field Set.
25h	On Intel® 64 platforms, hardware blocked an interrupt request in Compatibility format either due to Extended Interrupt Mode Enabled (EIME field Set in Interrupt Remapping Table Address Register) or Compatibility format interrupts disabled (CFIS field Clear in Global Status Register).
26h	Hardware blocked a Remappable interrupt request due to verification failure of the interrupt requester's source-id per the programming of SID, SVT and SQ fields in the corresponding IRTE with Present (P) field Set.
27h	Hardware attempt to access the Posted Interrupt Descriptor (PID) through the Posted Descriptor Address High/Low fields of an IRTE for posted interrupts resulted in error.
28h	Hardware detected one ore more reserved fields that are not initialized to zero in an Posted Interrupt Descriptor (PID).
29-2Fh	Reserved.
30h	Hardware blocked a Page (PRS) Request processed through an extended-context-entry with either Present (P) or Page Request Enable (PRE) fields Clear.
31h - FFh	Reserved.

This Page Is Left Intentionally Blank