Intel® Virtualization Technology for Directed I/O Architecture Specification

February 2006

Order Number: D51397-001

Version 1.0

CONTENTS

FIGURES

TABLES

CHAPTER 1
INTRODUCTION

This document describes the Intel® Virtualization Technology for Directed I/O (VT-d); 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.

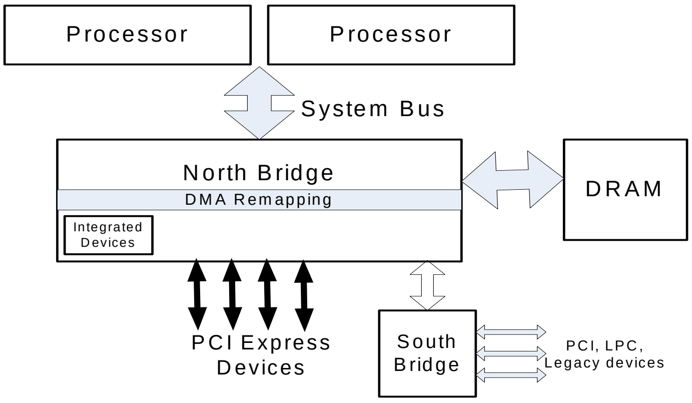

Figure 1-1. General Platform Topology

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 IA-32 Intel® Architecture. For more information, refer to the "*IA-32 Intel® Architecture Software Developer's Manual*".
- Intel® Virtualization Technology for Intel® Itanium® Architecture. For more information, refer to the "*Intel® Virtualization Technology Specification for the Intel® Itanium® Architecture*".

- This specification is based on the PCI Express* 1.1 base specification. Any additional features added to PCI Express that are relevant to I/O virtualization and DMA remapping will be accommodated in future versions of this specification.

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 and virtual machine monitor (VMM) developers utilizing the I/O virtualization hardware capabilities.

1.2 ORGANIZATION

The document is organized as shown in the following table.

Table 1-1. Document Organization

Chapter/ Appendix	Description
1	Introduction
2	I/O Device Assignment
3	DMA Remapping
4	Other Hardware Considerations
5	BIOS Considerations
6	Translation Structure Formats
7	DMA Remapping Registers
8	Extended DMA Remapping Features
A	Fault Reason Encodings

1.3 GLOSSARY

The document uses the terms listed in the following table.

Table 1-2. Glossary

Term	Definition
Chipset / Root-Complex	Used in this specification to refer 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.
Context	A hardware representation of state that identifies a device and the domain to which the device is assigned.
Device ID	A 16-bit device identification number consisting of the PCI Bus number, Device number and Function number. It is also referred to as Source-ID in this document.
DMA Remapping	Translating the address in a DMA request (DVA) to a host physical address (HPA).
Domain	A collection of physical, logical, or virtual resources that are allocated to work together. Domain is used as a generic term for virtual machines, partitions, etc.
DVA	DMA Virtual Address: a virtual address in a DMA request. One example usage of DVA in virtualization is for it to be the same as the Guest physical address (GPA).
GAW	Guest Address Width. GAW refers to the DMA virtual addressability limit.
GPA	Guest Physical Address is the view of physical memory from software running in a partition. GPA is also used in this document as an example usage for DMA virtual addresses (DVA).
Guest	Software running within a virtual machine environment (partition).
HAW	Host Address Width. HAW refers to the DMA physical addressability limit for a platform.
HPA	Host Physical Address.
IOTLB	I/O Translation Lookaside Buffer. IOTLB refers to an address translation cache in a DMA remapping hardware unit that caches effective translations from DVA (GPA) to HPA.
MGAW	Maximum Guest Address Width. MGAW refers to the maximum DMA virtual addressability supported by a DMA remapping hardware implementation.
MSI	Message Signalled Interrupts.
PDE Cache	Page Directory Entry cache. This refers to address translation caches in a DMA remapping hardware unit that caches page directory entries at the various page-directory levels. These are also referred to as non-leaf caches in this document.
VMM	Virtual Machine Monitor - A software layer that controls virtualization.

1.4 REFERENCES

For related information, see the documents listed in the following table.

Table 1-3. References

Document	Description
IA32-SDM	IA-32 Intel® Architecture Software Developer's Manual
PCI-EXP	PCI Express* Base Specification - Revision 1.1
ACPI	ACPI Specification - Rev 3.0

CHAPTER 2
OVERVIEW

This chapter provides an overview of Intel® Virtualization Technology for processor virtualization, and describes additional hardware support for I/O virtualization.

2.1 INTEL® VIRTUALIZATION TECHNOLOGY OVERVIEW

Intel® Virtualization Technology 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 the virtualization software, enables multiple usages such as server consolidation, activity partitioning, workload isolation, embedded IT management, legacy software migration, and disaster recovery.

2.2 VMM AND VIRTUAL MACHINES

Intel® Virtualization Technology 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® Virtualization Technology 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 system vendors to provide simple, robust and reliable virtual machine monitor software. VMM relies on hardware support to set policy and operational details for the handling of events, exceptions, and resources allocated to virtual machines.

Intel® Virtualization Technology provides hardware support for processor virtualization. Intel® Virtualization Technology for IA-32 processors is referred to as VT-x. VT-x constitutes a set of virtual-machine extensions (VMX) that support virtualization of processor hardware for multiple software environments by using virtual machines. An equivalent architecture for processor virtualization of the Itanium architecture is defined and is referred to as VT-i.

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. This enables the same device drivers for the legacy device to be run within the guest, and the 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 introduces limitations with respect to performance and functionality.
- New Software Interfaces: This model is similar to I/O emulation, except that instead of emulating legacy devices, VMM software exposes a synthetic (new) interface/device to guest software. The synthetic device interface is often 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: As part of virtualizing the platform to guests, a VMM may directly assign the physical I/O devices to VMs. With direct assignment of devices, 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 without invoking the VMM. 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.
- Device-assisted I/O 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. I/O virtualization through new software interfaces makes a trade-off between compatibility and performance, and native 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 the above I/O virtualization models is the ability to isolate and contain device accesses only to resources owned by the domain managing the device. Intel® Virtualization Technology for Directed I/O provides VMM software with the following capabilities:

- **Assign I/O devices across VMs**: Flexibly assign I/O devices to VMs and extend the protection and isolation properties of VMs for I/O accesses.
- **DMA remapping**: Direct Memory Accesses (DMA) from devices can be independently address-translated.
- **Reliability**: Record and report DMA errors that may otherwise corrupt memory to system software.

For simplicity, the address translation functionality for I/O device DMA requests is referred to as *DMA remapping* in this document.

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 I/O page tables. When the device attempts to access system memory, the DMA remapping hardware intercepts the access and utilizes the I/O page tables to determine whether the access can be permitted; it also determines the actual location to access. Frequently used I/O page table structures can be cached in hardware; the caches are referred to as I/O translation look-aside buffers (IOTLBs). The DMA remapping can be configured flexibly and independently for each device, or across multiple devices.

2.5.2 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 that domain for 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 its robustness and reliability.
- **Feature Support**: An OS may use domains to better manage DMA from legacy 32-bit PCI devices to high memory (above 4GB). This is achieved by allocating 32-bit devices to one or more domains and programming the platform's DMA remapping mechanism 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 2-1.

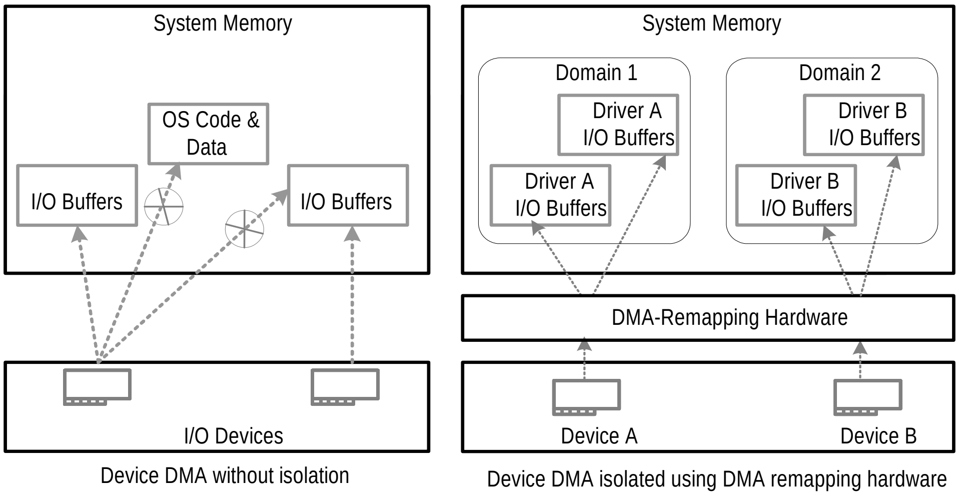

Figure 2-1. Example OS Usage of DMA Remapping

2.5.3 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 without invoking the VMM. The hardware support for DMA remapping enables this direct device assignment without device-specific knowledge in the VMM. See Figure 2-2.

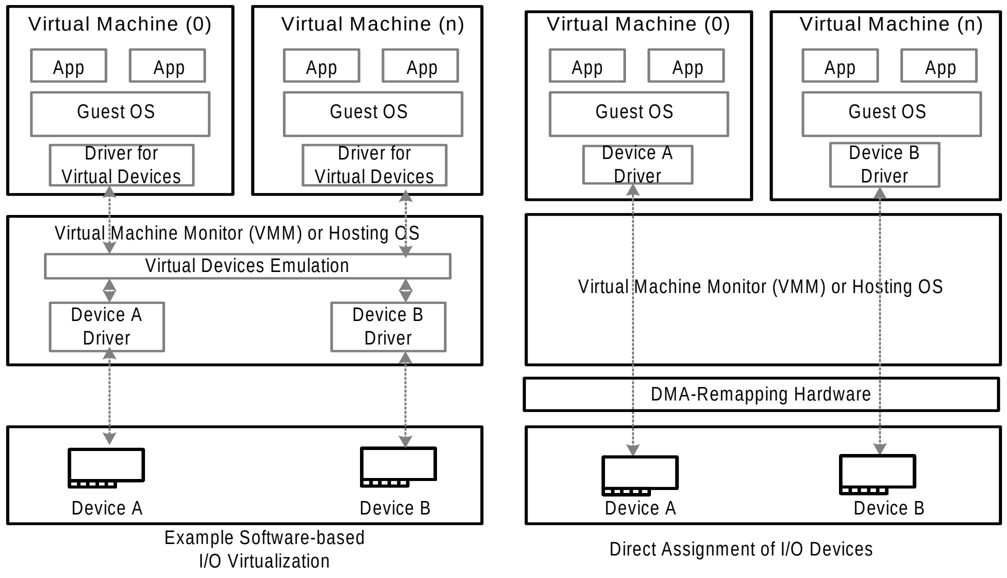
Virtual Machine (0)
Virtual Machine (n)
App
Guest OS
Driver for Virtual Devices
Virtual Machine Monitor (VMM) or Hosting OS
Virtual Devices Emulation
Device A Driver
Device B Driver
Device A
Device B
Example Software-based I/O Virtualization
DMA-Remapping Hardware
Direct Assignment of I/O Devices

Figure 2-2. Example Virtualization Usage of DMA Remapping

In this model, the VMM restricts itself to a controlling function for enabling direct assignment of devices to its partitions. Rather than invoking the VMM for all (or most) I/O requests from a partition, the VMM is invoked only when guest software accesses protected resources (such as I/O 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 DMA 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 (i.e., GPA not identical to HPA), the DMA 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. DMA remapping hardware enables virtualization software to choose the right combination of device assignment and software-based methods for I/O virtualization.

2.5.3.1 DMA Remapping Usages by Guests

A guest OS running in a VM may benefit from the availability of DMA remapping hardware to support the usages described in Section 2.5.2. To support such usages, the VMM may virtualize the DMA remapping hardware to its guests. For example, the VMM may intercept guest accesses to the virtual DMA remapping hardware registers, and manage a shadow copy of the guest DMA remapping structures that is provided to the physical DMA remapping hardware. On updates to the guest I/O page tables, the guest software performs appropriate virtual IOTLB invalidation operations. The virtual IOTLB invalidations may be intercepted by the VMM, the respective shadow page tables updated, and the physical IOTLBs flushed. Due to the non-restartability of faulting DMA transactions (unlike CPU memory management virtualization), a VMM cannot perform lazy updates to its shadow DMA remapping structures. To keep the shadow structures consistent with the guest structures, the VMM may expose virtual IOTLB with eager pre-fetching behavior (including caching of not-present entries) or write-protect the guest DMA remapping structures.

2.5.4 Interaction with Processor Virtualization

Figure 2-3 depicts the interaction between system software, and hardware support for processor level virtualization (VT-x) and Intel® Virtualization Technology for Directed I/O virtualization (VT-d).

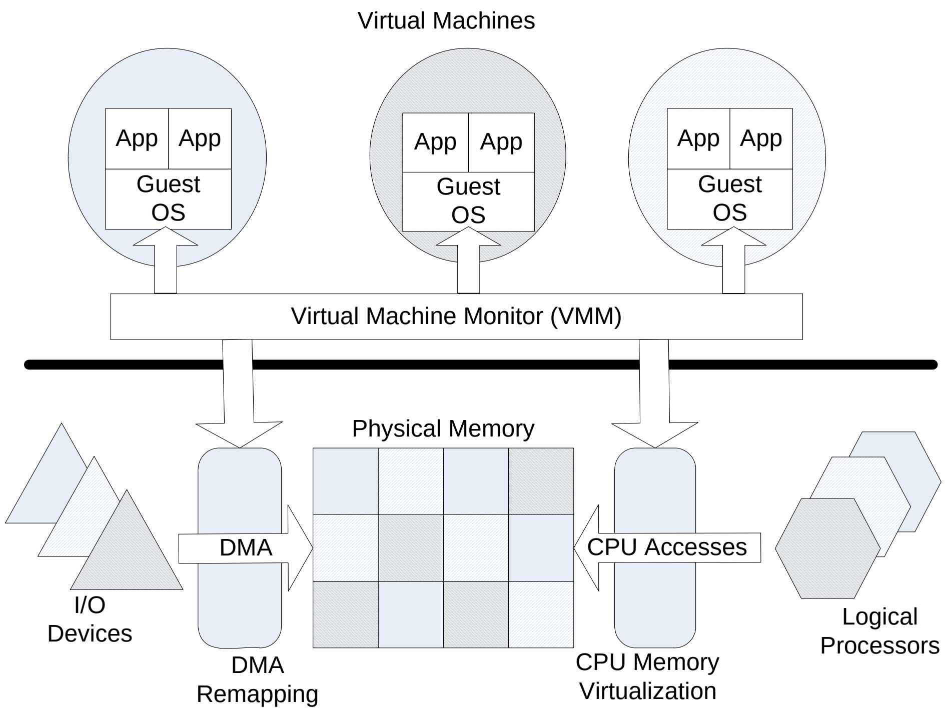

Figure 2-3. Interaction Between I/O and Processor Virtualization

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

CHAPTER 3
DMA REMAPPING

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

3.1 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.

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 all I/O devices are assigned to some domain (possibly a null domain), and by restricting access from each assigned device only to the physical resources allocated to its domain.

The DMA remapping architecture facilitates flexible assignment of I/O devices to an arbitrary number of domains. As described in Section 2.5, each domain has a view of physical address space that may be different than the host physical address space. DMA remapping treats the address specified in a DMA request as a DMA virtual address (DVA). Depending on the software usage model, the DMA virtual address space may be the same as the guest-physical address (GPA) space of the domain to which the I/O device is assigned, or a purely virtual address space defined by software. In either case, 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 an address in a DMA request as a GPA and the translated address as an HPA.

Figure 3-1 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. GPAs in DMA requests initiated by devices 1 & 2 are translated to appropriate HPAs by the DMA remapping hardware.

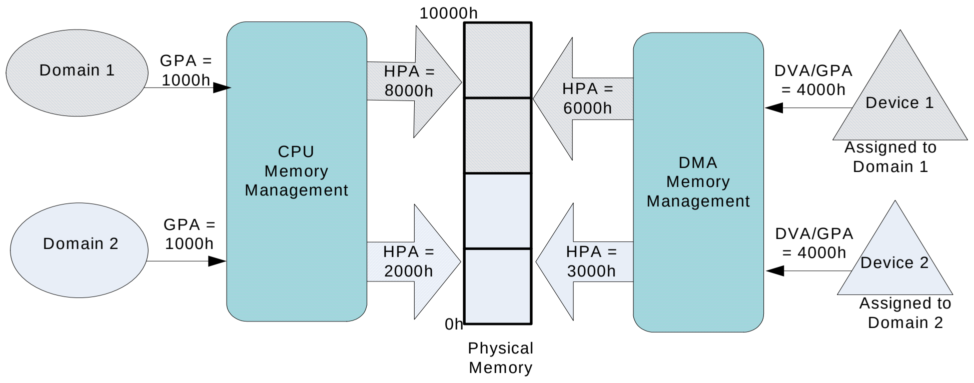

Figure 3-1. DMA Address Translation

The host platform may support one or more DMA remapping hardware units. Each hardware unit supports remapping DMA requests originating within its hardware scope. For example, a desktop chipset implementation may expose a single DMA remapping hardware unit that translates all DMA transactions at the memory controller hub (MCH) component. A server platform with one or more core chipset 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 DMA remapping hardware applies the address translation function to the address in a DMA request to convert it to a host physical address (HPA) before further hardware processing (such as address decoding, snooping of processor caches, and/or forwarding to the memory controllers).

3.2 MAPPING DEVICES TO DOMAINS

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

3.2.1 Source Identifier

The identity of the originator of a DMA request appearing at the address-translation hardware (the “requestor identifier”) is required to identify the device originating the DMA. The attribute identifying the originator of an I/O transaction is referred to as the “source-id” in this document. The DMA 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 mapped to 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 3-2 illustrates the requester-id as defined by the PCI Express Specification.

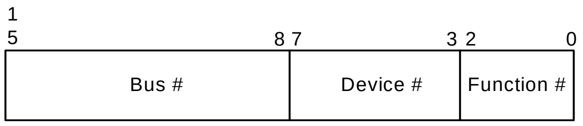

Figure 3-2. Requester Identifier Format

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

3.2.2 Root-Entry

The root-entry functions as the top level structure to map devices on a specific PCI bus to their respective domains. Each root-entry structure contains the following fields:

- **Present flag**: The present field is used by software to indicate to hardware whether the root-entry is present and 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 yet have any downstream devices attached. If the present field of a root-entry used to process a DMA request is clear, the DMA request is blocked, resulting in a translation fault.
- **Context-entry table pointer**: The context-entry table pointer references the context-entry table for devices on the bus identified by the root-entry. Section 3.2.3 describes context entries in further detail.

Figure 3-3 illustrates the root-entry format. The root entries are programmed through the root-entry table. The location of the root-entry table in system memory is programmed through the Root-entry Table Address register (described in Section 7.4.6). The root-entry table is 4KB in size and accommodates 256 root entries to cover the PCI bus number space (0-255). In the case of a PCI device, the bus number (upper 8-bits) encoded in a DMA transaction's source-id field is used to index into the root-entry structure. Figure 3-3 illustrates how these tables are used to map devices to domains.

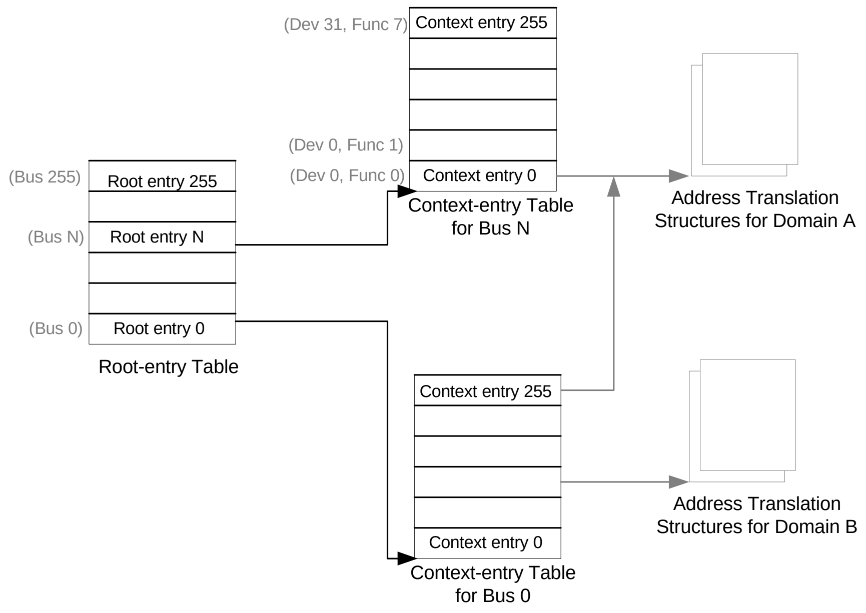

Figure 3-3. Device to Domain Mapping Structures

3.2.3 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 the memory-resident context-entry tables. Each root-entry in the root-entry table contains the pointer to the context-entry table for the corresponding bus number. Each context-entry table contains 256 entries, with each entry representing a unique PCI device function on the bus. For a PCI device, the device and function numbers (lower 8-bits) of a source-id are used to index into the context-entry table.

Each context-entry contains the following attributes:

- **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 always reference the same address translation structure, and vice versa.
- **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 DMA request is clear, the DMA request is blocked, resulting in a translation fault.

- **Translation Type**: The translation-type field indicates the type of the address translation structure that must be used to address-translate a DMA request processed through the context-entry.
- **Address Width**: The address-width field indicates the address width of the domain to which the device corresponding to the context-entry is assigned.
- **Address Space Root**: The address-space-root field provides the host physical address of the address translation structure in memory to be used for address-translating DMA requests processed through the context-entry.
- **Fault Processing Disable Flag**: The fault-processing-disable field enables software to selectively disable recording and reporting of DMA remapping faults detected for DMA requests processed through the context-entry.

Section 6.2 illustrates the exact context-entry format. 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.

3.2.3.1 Context Caching

The DMA remapping architecture enables hardware to cache root-entries and context-entries to minimize the overhead incurred for fetching them from memory. Hardware manages the context-cache and supports context-cache invalidation requests by software.

When modifying device-to-domain mapping structures referenced by multiple DMA remapping hardware units in a platform, software is responsible for explicitly invalidating the caches at each of the hardware units. For more detailed information, see Section 7.4.9.

The architecture defines the following types of context-cache invalidation requests:

1. **Global Invalidation**: All context-cache entries cached at a DMA remapping hardware unit are invalidated through a global invalidate.
2. **Domain-Selective Invalidation**: Context-cache entries corresponding to a specific domain are invalidated through a domain-selective invalidate.
3. **Device-Selective Invalidation**: Context-cache entries corresponding to a specific device within a domain are invalidated through a device-selective invalidate.

Hardware implementations must allow software to specify any of the above three types of invalidation requests, but may perform the actual invalidation at a coarser granularity if the requested invalidation granularity cannot be supported. For example, hardware may perform a domain-selective invalidation on a device-selective invalidation request. Hardware reports to software the granularity at which the actual invalidation was performed. Section 7.4.9 describes the hardware registers for context-cache invalidation.

3.3 ADDRESS TRANSLATION

The translation-type field in a context-entry specifies the translation structure type used to remap DMA requests from an I/O device. The architecture currently defines only a multi-level, page-table-based, address-translation structure type.

3.3.1 Multi-Level Page Table

The multi-level page tables allow software to manage host physical memory at page (4KB) granularity and set up a hierarchical structure with page directories and page tables. Hardware implements the page-walk logic and traverses these structures using the address from the DMA request. The maximum number of page-table levels that need to be traversed is a function of the address width of the domain (specified in the corresponding context entry).

The architecture defines the following features for the multi-level page table structure:

- Super Pages
    - The super-page field in page-table entries enables larger page allocations. When a page-table entry with the super-page 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 page-table entry with the GPA bits not yet used for the page-walk.
    - The architecture currently defines super-pages of size 2^{21}, 2^{39}, 2^{48}, and 2^{57}. Implementations indicate support for specific super-page sizes through the Capability register. Hardware implementations may optionally support these super-page sizes.
- DMA Access Controls
    - DMA access controls make it possible to control DMA accesses to specific regions within a domain’s address space. These controls are defined through the read and the write permission fields.
    - If hardware encounters a page-table entry with the read field clear as part of address-translating a DMA read request, the request is blocked.
    - If hardware encounters a page-table entry with the write field clear as part of address translating a DMA write request, the request is blocked.

Figure 3-4 shows a multi-level (3-level) page-table structure with 4KB page mappings and 4KB page tables. Figure 3-5 shows the same thing with 2MB super pages.

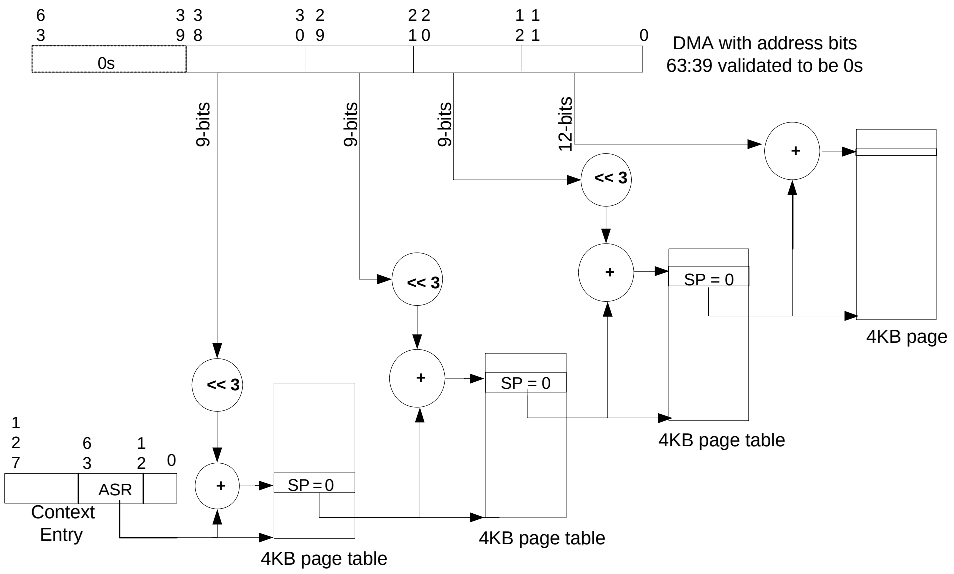

Figure 3-4. Example Multi-level Page Table

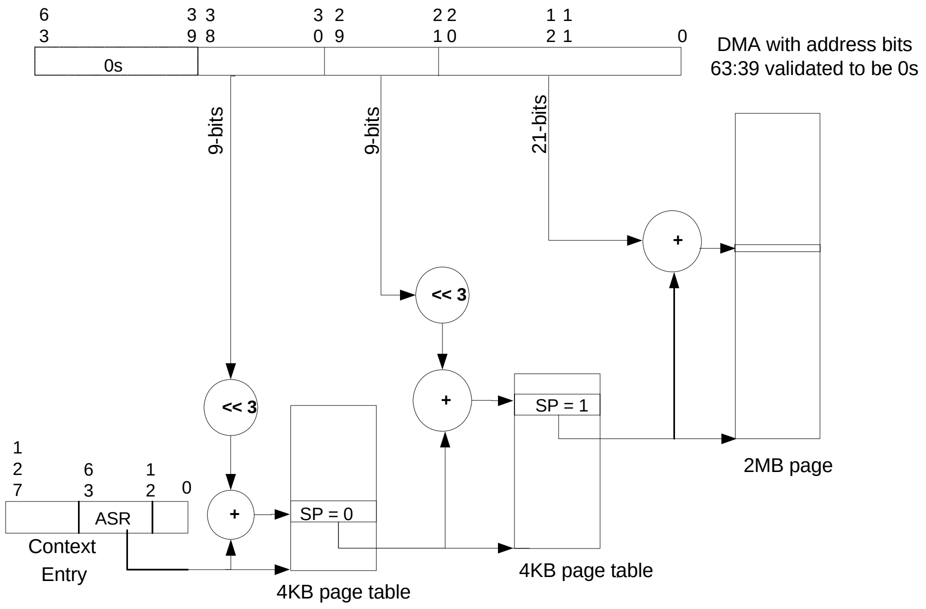

Figure 3-5. Example Multi-level Page Table (with 2MB Super Pages)

3.3.1.1 Adjusted Guest Address Width (AGAW)

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. For example, a domain to which 2GB of memory is allocated has a GAW of 31, and an AGAW of 39. AGAW is calculated as follows:

```
 R = (GAW - 12) MOD 9;
if (R == 0) {
   AGAW = GAW;
} else {
   AGAW = GAW + 9 - R;
}
if (AGAW > 64)
   AGAW = 64;
```

The AGAW indicates the number of levels of page-walk. Hardware implementations report the supported AGAWs through the Capability register. Software must ensure that it uses an AGAW supported by the underlying hardware implementation when setting up page tables for a domain.

3.3.1.2 Multi-level Page Table Translation

Address translation of DMA requests processed through a context-entry specifying use of multi-level page table is handled by DMA remapping hardware as follows:

1. If the GPA in the DMA request is to an address above the maximum guest address width supported by the remapping hardware (as reported through the MGAW field in the Capability register), the DMA request is blocked.
2. If the address-width (AW) field programmed in the context-entry is not one of the AGAWs supported by hardware (as reported through the SGAW field in the Capability register), the DMA request is blocked.
3. The address of the DMA request is validated to be within the adjusted address width of the domain to which the device is assigned. DMA requests attempting to access memory locations above address (2^X - 1) are blocked, where X is the AGAW corresponding to the address-width (AW) programmed in the context-entry used to process this DMA request.
4. If the above checks are successful, the address in the DMA request is adjusted (zero-extended) to X bits, where X is the address width (AW) programmed in the context-entry used to process this DMA request.
5. The adjusted address and length specified in the DMA request is used to address translate the DMA request through the multi-level page table referenced by the context entry. Based on the programming of the page-table entries (super-page, read, write attributes), either the adjusted address is successfully translated to a host physical address (HPA), or the DMA request is blocked.
6. For successful address translations, hardware performs the normal processing (address decoding, etc.) of the DMA request as if it was targeting the translated HPA.

3.3.1.3 I/O Translation Lookaside Buffer (IOTLB)

The DMA remapping architecture defines support for caching effective translations[1] for improved address translation performance. The cache of effective translations is referred to as the I/O translation look-aside buffer (IOTLB). Similar to the context-cache, hardware manages the IOTLB, and supports IOTLB invalidation requests by software. Details of the hardware caching and invalidation behavior are described in Section 7.4.9.

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

1. When inserting a leaf page-table entry into the IOTLB, hardware caches the Read (R) and Write (W) attributes as the logical AND of all the respective R and W fields encountered in the page walk reaching up to this leaf entry.

The architecture defines the following types of IOTLB invalidation requests to support the various software usage models:

1. **Global Invalidation**: All address translations cached at a DMA remapping hardware unit are invalidated through a global invalidate.
2. **Domain-Selective Invalidation**: Cached translations belonging to a specific domain are invalidated through a domain-selective invalidate.
3. **Domain Page-Selective Invalidation**: Cached translations corresponding to the specified DMA address(es) of a domain are invalidated through a domain-page-selective invalidate.

Hardware implementations must allow software to specify any of the above types of IOTLB invalidation requests, but may perform the actual invalidation at a coarser granularity if the requested invalidation granularity cannot be supported. Hardware to reports to software the granularity at which the actual invalidation was performed.

3.4 DMA REMAPPING FAULTS

The following table enumerates the various conditions resulting in DMA remapping faults.

Table 3-1. DMA Remapping Fault Conditions

DMA Remapping Fault Conditions	Qualified
The present (P) field in the root-entry used to process the DMA request is clear.	No
The present (P) field in the context-entry used to process the DMA request is clear.	Yes
Hardware detected invalid programming of a context-entry. For example: • The address-width (AW) field was programmed with an SAGAW value not supported by the hardware implementation. • The translation-type (T) field was programmed to indicate a translation type not supported by the hardware implementation. Currently, only the multi-level page-table translation type is defined. • A hardware attempt to access the page-table base through the Address Space Root (ASR) field of the context-entry resulted in error.	Yes
The DMA request attempted to access an address beyond (2^X - 1), where X is[1]: • For multiple level page-table based translation, the minimum of the MGAW reported in the Capability register and the AGAW corresponding to the address-width field in the context-entry used to process the DMA request.	Yes
The Write (W) field in a page-table entry used for address translation of the DMA write request is clear.	Yes
The Read (R) field in a page-table entry used for address translation of the DMA read request is clear.	Yes
A hardware attempt to access the next level page table through the Address (ADDR) field of the page-table entry resulted in error.	Yes

Table 3-1. DMA Remapping Fault Conditions (Contd.)

DMA Remapping Fault Conditions	Qualified
A hardware attempt to access the root-entry table through the root-table address (RTA) field in the Root-entry Table Address register resulted in error.	No
A hardware attempt to access context-entry table through context-entry table pointer (CTP) field resulted in error.	No
Hardware detected reserved field(s) that are not initialized to zero in a root-entry with the present (P) field set.[2]	No
Hardware detected reserved field(s) that are not initialized to zero in a context-entry with the present (P) field set.[2]	Yes
Hardware detected reserved field(s) that are not initialized to zero in a page-table entry with at least one of the read (R) and write (W) fields set.[2]	Yes

NOTES

1. DMA requests to addresses beyond the maximum guest address width (MGAW) supported by hardware may be reported through other means such as through PCI Express Advanced error reporting (AER) at a PCI Express root port.
2. To ensure compatibility with future extensions of this architecture, software must zero reserved fields in all present structures. Hardware behavior is undefined on reserved field violation. Hardware implementations capable of reserved field checking reports reserved field violation as DMA remapping faults.

When a DMA remapping fault is detected:

- The faulting DMA request is always blocked.
- The unqualified fault conditions in Table 3-1 are always logged and reported.
- The qualifying fault conditions in Table 3-1, are logged and reported conditionally. Hardware checks the fault-processing-disable (FPD) field in the context-entry used to process the faulting DMA request to determine if a qualifying fault must be processed. If the FPD field is set, the qualifying fault is not reported. If the FPD field is clear, the qualifying fault is logged and reported.

The following sections provide detailed information about fault logging and reporting.

3.4.1 Fault Logging

DMA remapping faults are processed by logging the fault information and reporting the faults to software through a fault event (interrupt). The 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.

3.4.1.1 Primary Fault Logging

The primary method for logging DMA remapping faults is through Fault Recording registers. The number of fault recording registers supported by a DMA remapping hardware unit is reported through the Capability register. The following information is recorded in the Fault Recording registers:

- **Fault Reason**: The fault reason indicates the specific condition that caused the translation fault. Fault reason encodings are enumerated in Appendix A.
- **Access Type**: This field provides the type (write or read request) of the DMA request that faulted.
- **Source ID**: The source-id indicates the originator of the faulted DMA request.
- **Faulting Address**: The address field provides the page address in the faulted DMA request.

Hardware indicates a pending fault in a fault recording register by setting its Fault (F) field. Additional faults detected from the same requestor may be collapsed by hardware if there is already a pending fault from the same requester. Any fault overflow condition is handled by hardware by not recording the fault and indicating the overflow condition to software through the Primary Fault Overflow (PFO) field in the Fault Status register. Future faults are not recorded or reported until software handles the already reported faults in the fault recording registers, and clears the overflow condition. Section 7.4.18.1 describes the hardware handling of primary fault logging in detail.

3.4.1.2 Advanced Fault Logging

Advanced fault logging is an optional hardware feature. Hardware implementations supporting advanced fault logging report the feature through the Capability register.

Advanced fault logging uses a memory-resident fault log to record 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 DMA remapping hardware. Each fault record contains the same information (fault reason, access type, source-id and faulting address fields) as with primary fault logging. Section 6.4 illustrates the format of the fault record.

When advanced fault logging is enabled, hardware manages a pointer to the next writable entry in the size-aligned fault log region. Whenever a new fault log region is programmed by software, the internal pointer is initialized to the base of the new fault log. The pointer is advanced whenever a fault record is written, for up to the maximum number of entries allowed by the fault log size. If hardware is not able to record a fault because the fault log is full, the fault is not recorded, and the overflow condition is indicated through the Advanced Fault Overflow (AFO) status field in the Advanced Fault Log register. Future faults are not recorded or reported until software clears the overflow condition and re-programs the fault log pointer in hardware. See Section 7.4.18.2 for more detailed information.

3.4.1.3 Fault Priority

The priorities for hardware handling and reporting of fault conditions detected while remapping a DMA request are:

1. Fault conditions encountered on hardware attempt to access the various memory-resident DMA remapping structures.
2. Fault condition due to hardware detecting the target entry being not present.
3. Fault conditions due to erroneous programming of one or more fields in an entry.
4. For hardware implementations validating reserved fields, fault conditions due to one or more non-zero reserved fields.

3.4.2 Fault Reporting

Fault events are reported to software using a message signalled interrupt (MSI), and controlled through the Fault Event Control register. The fault event information (such as interrupt vector, delivery mode, address, etc.) is programmed through the Fault Event Data and Fault Event Address registers.

With both primary and advanced fault logging, hardware can be programmed to generate a fault event when the first fault is detected/recorded. Further fault events are auto-disabled in hardware until software explicitly re-arms fault-event generation. provides further details on hardware fault-event generation behavior with primary and advanced fault logging.

3.4.3 Hardware Handling of Faulting DMA Requests

A DMA request that results in a DMA remapping fault must be blocked by hardware. The exact method of DMA blocking is implementation-specific. For example:

- Faulting DMA write requests may be handled in much the same way as hardware handles write requests to non-existent memory. For example, the DMA request must be 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-directed to a dummy memory location etc.).
- Faulting DMA 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 dummy 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 (preferred). For faulting DMA read requests from PCI Express devices, hardware must indicate either “Completer Abort” (CA) or “Unsupported Request” (UR) in the completion status field of the PCI Express read completion.[1]

1. For PCI Express, a DMA read completion with an error status may cause hardware to generate a PCI Express un-correctable, non-fatal (ERR_NONFATAL) error message.

CHAPTER 4
HARDWARE CONSIDERATIONS

This chapter discusses hardware considerations for DMA remapping that are not covered in other chapters.

4.1 HANDLING INTERRUPT MESSAGES

On Intel platforms, interrupt requests from IOAPICs and MSI-capable devices appear to the root-complex as upstream memory write requests to the address range 0xFEE0_0000h to 0xFEEF_FFFFh. Since this interrupt message address range is architectural and identical between the guest and the host, upstream DMA requests to addresses in the MSI address range are not subject to DMA remapping as described in Chapter 3. Hardware decodes the address in a DMA request to check if it falls in the interrupt address range and bypasses DMA remapping for such transactions. DMA write requests to this range are validated and interpreted as interrupt messages, and DMA read requests to this range are treated as errors.

Software must ensure that the DMA remapping page tables are programmed not to remap regular DMA requests to the above interrupt address range. Hardware behavior is undefined for DMA requests remapped to the interrupt address range through the DMA remapping structures.

Future versions of this specification may define additional behavior for handling interrupt messages.

4.2 ASSIGNING DEVICES BEHIND PCI EXPRESS TO PCI/PCI-X BRIDGES

For PCI Express-to-PCI/PCI-X bridges, the bridge 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.

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

4.3 ASSIGNING 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 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.

4.4 HANDLING DMA REQUESTS CROSSING PAGE BOUNDARY

PCI Express memory requests are specified to disallow Address/Length combinations which cause a memory space access to cross a 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 of DMA requests crossing 4KB boundary from root-complex integrated devices is typically platform-dependent.

Platforms supporting DMA remapping are expected to handle DMA requests that cross 4KB boundary in one of the following ways:

- The platform hardware checks for DMA requests that cross a 4KB boundary and explicitly blocks them. For PCI Express memory requests, this may be implemented by hardware strictly checking for the condition at the PCI Express receivers and handling it as PCI Express error. DMA requests from other devices (such as root-complex integrated devices) violating this rule (and hence blocked by hardware) may be handled in platform-specific ways. In this model, the DMA remapping hardware units never receive DMA requests that cross page boundaries.
- If the platform hardware cannot check for the 4KB crossing condition in DMA requests, the DMA remapping hardware units must detect this condition and re-map the requests as if they were multiple independent DMA requests.

4.5 HANDLING OF ZERO-LENGTH DMA

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 requestor, 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 their destination.

When DMA remapping hardware is enabled, zero-length DMA requests are not processed differently than other DMA requests. Specifically, zero-length DMA requests are address-translated based on the programming of the DMA remapping structures. For successful translations, the transaction is completed in much the same way as it is completed without DMA remapping (for example, the transaction is completed in the coherency domain with all byte enables off). Unsuccessful translations result in DMA remapping faults.

4.6 DMA REMAPPING - SOFTWARE VIEW

The DMA remapping architecture allows hardware implementations supporting a single PCI segment group to expose (to software) the DMA 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 DMA remapping hardware unit that supports one or more integrated devices on the root bus, and additional DMA remapping hardware units for devices behind one or a set of PCI Express root ports. The platform firmware (BIOS) reports each DMA remapping hardware unit in the platform to software. Chapter 5 describes a proposed reporting structure through ACPI constructs.

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

4.7 HANDLING DMA TO RESERVED SYSTEM MEMORY

PC platforms commonly use reserved memory for platform-specific usages. These memory regions are typically marked as reserved by BIOS in the system memory map provided to software. Memory exposed to system software for operating system use through the system memory map is commonly referred to as “OS-visible memory”.Usages of system reserved memory are platform-specific. Some common examples of private memory usages include: integrated UMA graphics controllers using reserved system memory to host the graphics translation table (GTT) and blitting, management controllers using reserved memory to store/run management firmware, and BIOS/SMM code utilizing reserved memory for USB legacy keyboard emulation.

Platform implementations supporting private 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 means to distinguish accesses to system reserved memory. These means 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 DMA to OS-visible system memory and DMA 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 DMA access to these ranges for proper operation. For legacy reasons, system software is expected to provide unity mapping for these address ranges, and to provide both read and write access privileges to at least the identified devices. For these devices, the system software is also responsible for ensuring that the DMA virtual addresses generated by the system software do not overlap with the reserved system memory address ranges. Refer to Section 5.4 for details on the reporting of reserved memory regions.

4.8 PEER TO PEER CONSIDERATIONS

When DMA remapping hardware is enabled, peer-to-peer requests through the root complex must be handled as follows:

- The address in the DMA request must be treated as a DMA virtual address and address-translated to an 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 of PCI Express peer-to-peer requests changes the contents (address field) of the PCI Express transaction layer packet (TLP), for 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.

Existing PCI Express specifications do not provide the capability to restrict peer-to-peer accesses between functions of a multi-function endpoint, or across PCI Express switch-ports. Due to this limitation, to prevent address aliasing, system software must avoid using all or relevant peer address ranges as valid DMA virtual addresses on such configurations.

4.9 HANDLING OF ISOCHRONOUS DMA

PC platforms support varying classes of isochronous DMA traffic to support different real-time usages and devices. Examples of Isochronous traffic supported by current PC platforms include:

- **Legacy Isochrony**: Legacy isochronous traffic refers to the support typically provided to applications that use legacy audio and USB controllers.
- **PCI Express Isochrony**: PCI Express specification defines support for isochronous traffic to support real-time and QoS usages. An example of this class of isochrony is applications that use high-definition audio controllers.
- **Integrated Graphics Isochrony**: This applies to traffic generated by integrated graphics subsystems which typically have multiple real-time DMA streams for display, overlay, cursor, and legacy VGA usages.

There may be other classes of differentiated DMA streams in the platform to support future usages. Different classes of isochrony are typically implemented through traffic classification, virtual channels (VC), and priority mechanisms. DMA remapping of isochronous traffic requires special handling since the remapping process potentially adds additional latency to the isochronous paths.

Hardware recommendations and software requirements for efficient remapping of isochronous traffic are described below. The classes of isochrony that needs to be subject to these requirements are platform-specific.

- Root-complex hardware may utilize dedicated resources to support remapping of DMA accesses from isochronous devices. This may be in the form of dedicated DMA remapping hardware units for remapping isochronous devices, or through reservation of hardware resources (such as entries in various DMA remapping caching structures) for remapping isochronous DMA.
- Since snooping platform caches typically introduces varying amounts of latency, DMA remapping units that support isochronous DMA may not snoop platform caches when accessing the various remapping structures in memory. This hardware behavior for the mappings used by DMA remapping units is reported to software through the remapping unit's capability registers. This implies that software managing these DMA remap units must explicitly ensure any software updates to remapping structures are visible in memory for hardware to utilize the updated structures.
- DMA remapping units supporting isochronous traffic may pre-fetch and cache address translations. Under normal operation, to maintain isochronous guarantees, software must specify page-selective granular invalidations to ensure the invalidation operations do not impact isochronous DMA requests that may be active at the time of invalidation. The Capability register of each DMA remapping unit indicates to software if the remapping unit manages the DMA with such QoS requirements.
- In order to limit dependencies on isochronous flows, the root-complex hardware may require the DMA remapping structures to be un-shared across DMA remapping units handling isochronous devices (thereby ensuring that the addresses hosting DMA remapping structures used for remapping isochronous and non-isochronous traffic do not overlap). The Capability register of each DMA remapping unit indicates to software whether hardware requires such spatial separation of remapping structures. If specified, to achieve isochronous guarantees, software must avoid sharing the remapping structures with other remapping units.

CHAPTER 5
BIOS CONSIDERATIONS

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

5.1 DMA REMAPPING REPORTING STRUCTURE

Table 5-1. DMA Remapping Reporting (DMAR) Table

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 DMA 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.

Table 5-1. DMA Remapping Reporting (DMAR) Table (Contd.)

Field	Byte Length	Byte Offset	Description
Reserved	11	37	Reserved.
DMA 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) structures. These structures are described below.

5.2 DMA REMAPPING STRUCTURE TYPES

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

Table 5-2. DMA Remapping Structure Types

Value	Description
0	DMA Remapping Hardware Unit Definition (DRHD) Structure
1	Reserved Memory Region Reporting (RMRR) Structure
>1	Reserved for future use. Software may skip structures it does not understand by skipping the appropriate number of bytes indicated by the Length field.

5.3 DMA REMAPPING HARDWARE UNIT DEFINITION STRUCTURE

A DMA remapping hardware unit definition structure uniquely represents a DMA remapping hardware unit present in the platform. There must exactly be one instance of this structure for each DMA remapping hardware unit supported by the platform. There must be at least one DMA remapping hardware unit definition structure in a platform support DMA remapping.

Table 5-3. DMA Remapping Hardware Unit Definition (DRHD) Structure

Field	Byte Length	Byte Offset	Description
Type	2	0	0 - DMA Remapping Hardware Unit Definition structure
Length	2	2	Varies (16 + size of Device Scope Structure)
Flags	1	4	• Bit 0: INCLUDE_ALL: If set, this DMA remapping hardware unit handles DMA from all sources except those that are covered under the scope of other DMA remapping hardware units. Such a remapping unit definition structure entry has no Device Scope field and must appear after all other DMA remapping hardware unit definition structure entries. If clear, this DMA remapping hardware unit handles DMA only from the sources identified through the Device Scope field. • Bits 1-7: Reserved.
Reserved	3	5	Reserved.
Register Base Address	8	8	Base address of DMA remapping hardware register set for this unit.
Device Scope []	-	16	If the INCLUDE_ALL flag is set, the Device Scope structure is omitted. If the INCLUDE_ALL flag is clear, the Device Scope structure contains one or more Device Scope Entries that identify devices under the scope of this DMA remapping hardware unit. The Device Scope structure is described below.

5.3.1 Device Scope Structure

The Device Scope Structure is expected to describe one or more PCI or PCI Express devices. Therefore, it is made up of one or more Device Scope Entries. Each entry points to a specific device. In the following definition, the generic term PCI is used to describe conventional PCI, PCI-X, and PCI-Express. 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 Device Scope Entry itself is a variable length structure that uniquely identifies a PCI sub-hierarchy or a specific endpoint in the system. 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. The Device Scope Entry Type defines whether the entry refers to an endpoint or a sub-hierarchy. The data formats of both types of entries are identical, and the Device Scope Entry Type field is used only to interpret the data.

Table 5-4. Device Scope Entry Structure

Field	Byte Length	Byte Offset	Description
Device Scope Entry Type	1	0	The following enumerations are defined for this field. • 0x01: PCI Endpoint Device - This Device Scope Entry identified by the PCI Path field is a PCI endpoint device. • 0x02: PCI Sub-hierarchy - This Device Scope Entry identified by the PCI Path field is a PCI-PCI bridge. In this case, the specified bridge device and all its downstream devices are included in the scope. Other Values are reserved for future use.
Length	1	1	Length of this Entry in Bytes. (4 + X), where X is the size in bytes of the “PCI Path” field.
Segment Number	1	2	The PCI Segment Number of the device(s) identified by this Device Scope Entry
Starting Bus Number	1	3	A Device Scope Entry describes a device under a specific PCI Host Bridge.This field describes the bus number of the first PCI Bus produced by the PCI Host Bridge.

Table 5-4. Device Scope Entry Structure (Contd.)

Field	Byte Length	Byte Offset	Description
PCI Path	2 * N	4	Describes the path from the Host Bridge to a specific PCI device in hierarchal fashion. 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 Starting 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 Endpoint or PCI-PCI Bridge is obtained by recursively walking down these N {Device, Function} pairs.

The following pseudocode describes how to identify the endpoint or PCI-PCI device specified through a Device Scope Entry:

```
n = (DeviceScopeEntry[1] - 4) / 2);   // number of entries in the ‘PCI Path’ field
seg = DeviceScopeEntry[2];             // segment number at offset 2
bus = DeviceScopeEntry[3];             // starting bus number at offset 3
dev = DeviceScopeEntry[4];             // starting device number at offset 4
func = DeviceScopeEntry[5];          // starting function number at offset 5
i = 0;
while (--n) {
   bus = read_secondary_bus_reg(seg, bus, dev, func); // secondary bus# from config reg.
   dev = DeviceScopeEntry[6+i];       // read next device number
   func = DeviceScopeEntry[7+i];     // read next function number
   i = i + 2;
}
target_device = {segment, bus, dev, func}; // Type field in DeviceScopeEntry identifies
                                           // if target device is endpoint or PCI-PCI bridge
```

5.3.2 Device Scope Example

This section provides an example platform configuration with multiple DMA remapping hardware units. The configurations described are hypothetical examples only intended to illustrate the Device Scope structures.

Figure 5-1 illustrates a platform configuration with a single PCI segment and host bridge (with a starting bus number of 0), and supporting four DMA remapping hardware units as follows:

1. DMA 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. DMA 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. DMA remapping hardware unit #3 has under its scope a root-complex integrated endpoint device located at (dev:func) of (29:0).
4. DMA remapping hardware unit #4 has under its scope all other DMA sources not explicitly under the scope of the other DMA 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.

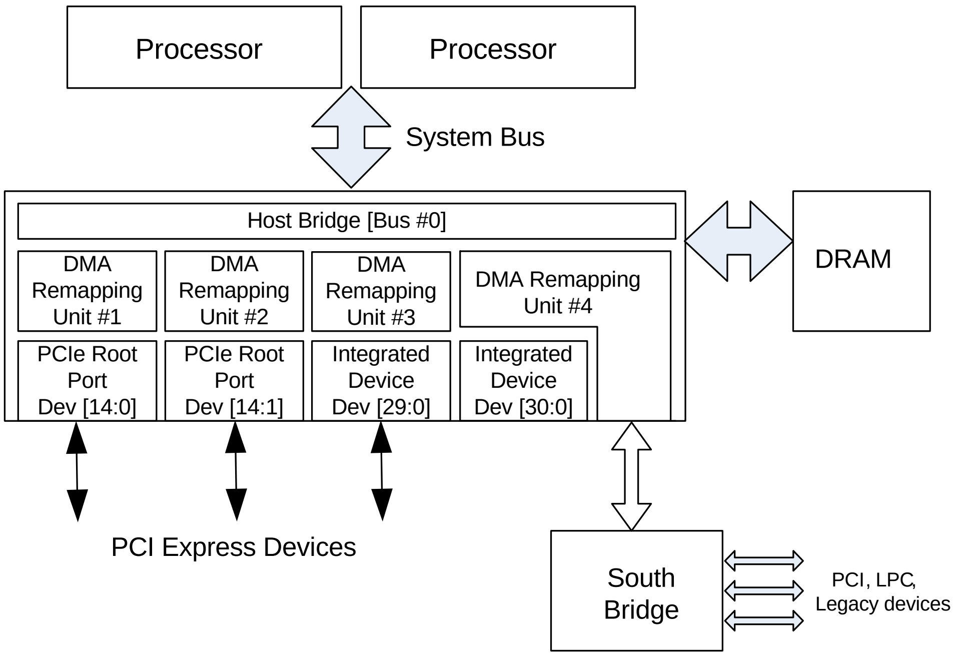

Figure 5-1. Hypothetical Platform Configuration

This platform requires 4 DRHD structures. The Device Scope fields in each DRHD structures are described as below.

- Device Scope for DMA Remapping hardware unit #1:
    - The Device Scope for DMA remapping hardware unit #1 contains only one Device Scope Entry identified as [2, 6, 0, 0, 14, 0].
    - System Software use the Entry Type field value of 2 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 DMA remapping hardware unit.
- Device Scope for DMA Remapping hardware unit #2:
    - The Device Scope for DMA remapping hardware unit #1 contains only one Device Scope Entry identified as [2, 6, 0, 0, 14, 1].
    - System Software use the Entry Type field value of 2 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 DMA remapping hardware unit.
- Device Scope for DMA Remapping hardware unit #3:
    - The Device Scope for DMA remapping hardware unit #3 contains only one Device Scope Entry identified as [1, 6, 0, 0, 29, 0].
    - System software uses the Type field value of 1 to conclude that the scope of DMA remapping hardware unit #2 includes only the endpoint device at PCI Segment 0, Bus 0, Device 29 and Function 0.
- Device Scope for DMA Remapping hardware unit #4:
    - The DHRD structure for DMA remapping hardware unit #4 indicates the INCLUDE_ALL flag and hence does not contain any Device Scope Entries. This hardware unit must be the last in the list of hardware unit definition structures reported.

5.4 RESERVED MEMORY REGION REPORTING STRUCTURE

Reserved memory regions indicate system memory regions reserved by the BIOS that may be a target of DMA requests from one or more devices while the system software is active. These reserved memory regions are typically allocated by the BIOS at boot time and reported to the OS as either reserved or unavailable address ranges in the system memory map. BIOS implementations may report each such reserved memory region through the RMRR structures and the devices that requires access to the specified reserved memory region. For proper functioning of the platform, when DMA from these devices is enabled, system software must ensure that the DMA remapping page tables for the respective devices are set up to provide unity mapping for the specified reserved memory regions with both read and write privileges. The platform designer must avoid or limit such regions since they require system software to create gaps in the DMA virtual address range available to system software and its drivers.

DMA to these reserved regions may either occur as a result of some operation performed by the system software driver (for example, DMA from UMA graphics controllers to graphics-reserved memory), or may be initiated by non-system software such as the BIOS (for example, in case of DMA performed by a USB controller under BIOS SMM control for legacy keyboard emulation).

Reserved memory ranges that are either not DMA targets, or memory ranges that may be the target of BIOS-initiated DMA only during pre-boot (for example, from a boot disk drive) are not included in the reserved memory region reporting. 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 require DMA accesses.

Table 5-5. Reserved Memory Region Reporting (RMRR) Structure

Field	Byte Length	Byte Offset	Description
Type	2	0	1 - Reserved Memory Region Reporting Structure
Length	2	2	Varies (16 + size of Device Scope structure)
Flag	1	4	• Bit 0: ALLOW_ALL: If set, the Reserved Memory Region applies to devices under the scope of all DMA remapping hardware units. If clear, the Reserved Memory Region is accessed only by devices identified in the Device Scope structure. • Bits 1-7: Reserved
Reserved	3	5	Reserved.
Reserved Memory RegionBase Address	8	8	4KB-aligned base address of the reserved memory region.
Reserved Memory Region Limit Address	8	16	4KB-aligned limit address of the reserved memory region.
Device Scope[]	-	24	• If the ALLOW_ALL flag is set, the Device Scope structure is omitted. • If the ALLOW_ALL flag is clear, 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 the corresponding DMA remapping hardware unit.

CHAPTER 6
TRANSLATION STRUCTURE FORMATS

This chapter describes the DMA remapping memory-resident structures.

6.1 ROOT-ENTRY

The following figure and table describe root-entry.

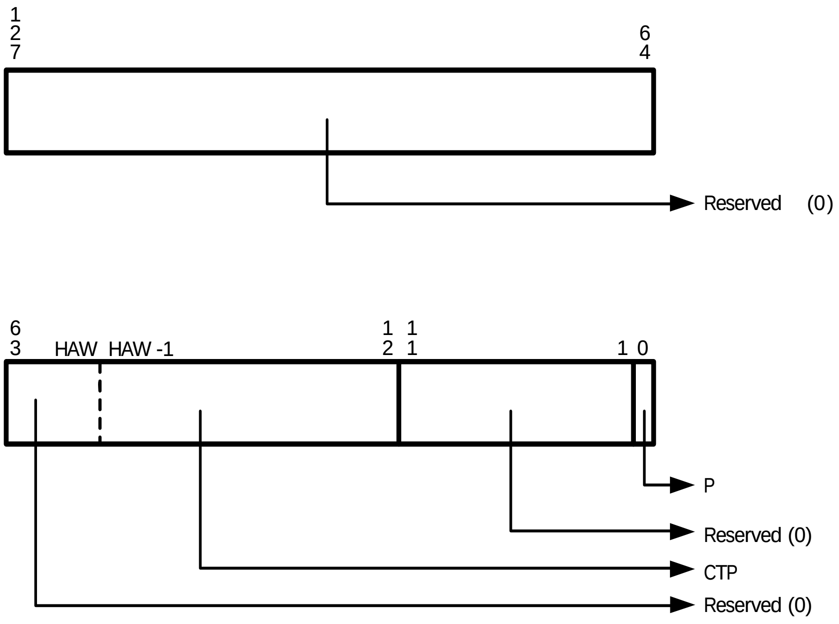

Figure 6-1. Root-Entry Format

Table 6-1. Root-Entry Contents

Bits	Field	Description
127:64	R: Reserved	Reserved. Must be 0. This field is evaluated by hardware only when the present (P) field is set.
63:12	CTP: Context-entry Table Pointer	Pointer to context-entry table for this bus. The context-entry table is 4KB in size and size-aligned. This field is evaluated by hardware only when the present (P) field is set. When evaluated, 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. This field is evaluated by hardware only when the present (P) field is set.
0	P: Present	This field indicates whether the root-entry is present. • 0: Indicates the root-entry is not present. Hardware blocks DMA requests processed through root entries with the present field cleared. • 1: Indicates the root-entry is present. Hardware processes DMA requests per the context-entries referenced by the CTP field.

6.2 CONTEXT-ENTRY

The following figure and table describe the context-entry.

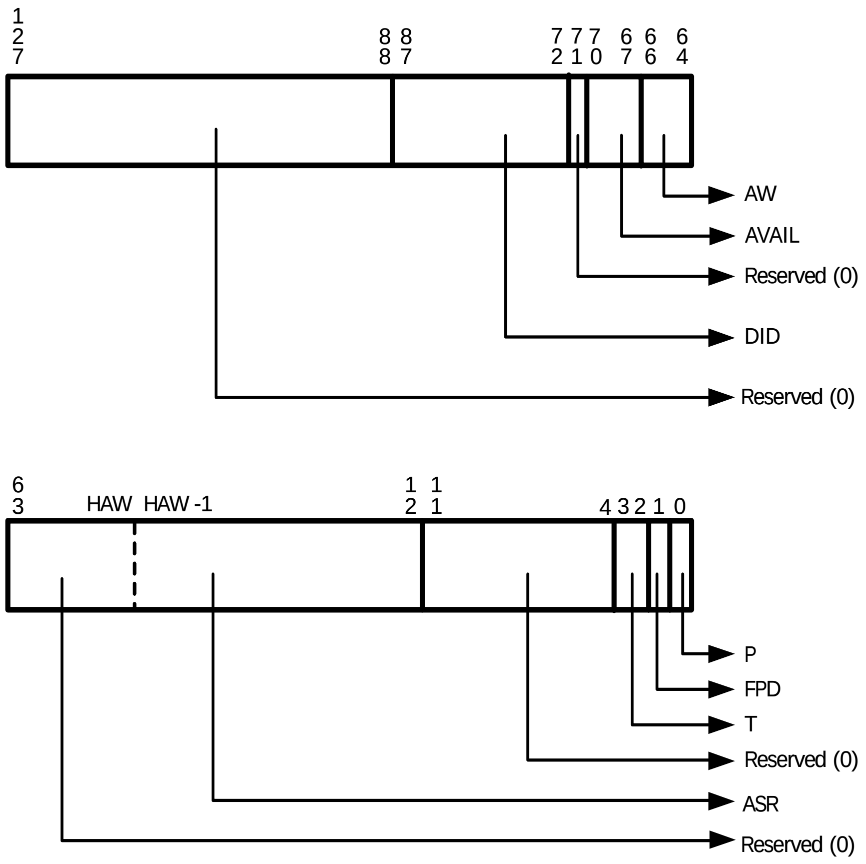

Figure 6-2. Context-Entry Format

Table 6-2. Context-Entry Contents

Bits	Field	Description
127:88	R: Reserved	Reserved. Must be 0. This field is evaluated by hardware only when the present (P) field is set.
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 an 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 the same address translation structure (through the ASR field). Similarly, context-entries referencing the same address translation structure must be programmed with the same domain id. This field is evaluated by hardware only when the present (P) field is set.
71	R: Reserved	Reserved. Must be 0. This field is evaluated by hardware only when the present (P) field is set.
70:67	AVAIL: Available	This field is available to software. Hardware always ignores the programming of this field.
66:64	AW: Address Width	When the translation-type (T) field indicates multi-level page tables, this field indicates the adjusted guest-address-width (AGAW) to be used by hardware for the page-table walk. The following encodings are defined for this field: • 000b: 30-bit AGAW (2-level page table) • 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: 64-bit AGAW (6-level page table) • 101b-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). DMA requests processed through this context-entry and accessing DMA virtual addresses above 2^X-1 (where X is the AGAW value indicated by this field) are blocked[1]. This field is evaluated by hardware only when the present (P) field is set.

Table 6-2. Context-Entry Contents (Contd.)

Bits	Field	Description
63:12	ASR: Address Space Root	Host physical address of the address space root as described below. When the translation-type (T) field indicates multi-level page tables, this field points to the base of page-table root. This field is evaluated by hardware only when the present (P) field is set. When evaluated, hardware treats bits 63:HAW as reserved, where HAW is the host address width of the platform.
11:4	R: Reserved	Reserved. Must be 0. This field is evaluated by hardware only when the present (P) field is set.
3:2	T: Translation Type	This field is evaluated by hardware only when the present (P) field is set. • 00: Indicates ASR field points to a multi-level page table. • 01 - 11: Reserved.
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: Block DMA requests processed through this context-entry. • 1: Process DMA requests through this context-entry based on the programming of other fields.

NOTES

1. DMA requests to addresses beyond the Maximum Guest Address Width (MGAW) supported by hardware may be blocked and reported through other means such as PCI Express Advanced Error Reporting (AER). Such errors (referred to as platform errors) may not be reported as DMA remapping faults and are outside the scope of this specification.

6.3 PAGE-TABLE ENTRY

The following figure and table describe the page-table entry.

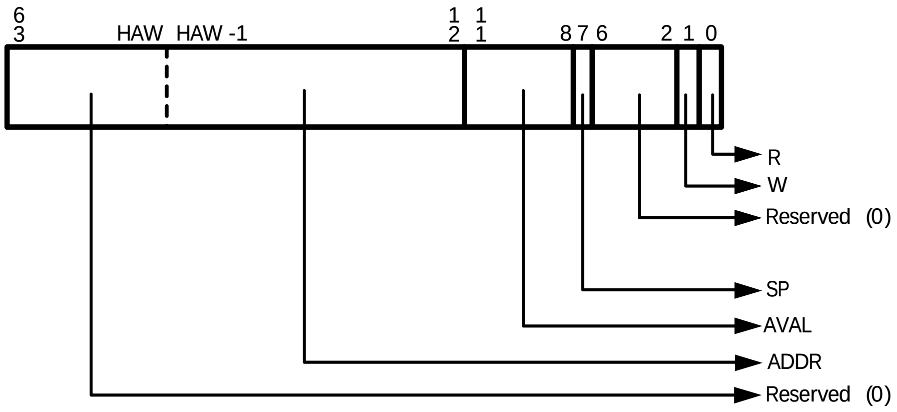

Figure 6-3. Page-Table-Entry Format

Table 6-3. Page-Table-Entry Contents

Bits	Field	Description
63:12	ADDR: Address	Host physical address of the page frame if this is a leaf node. Otherwise a pointer to the next level page table. This field is evaluated by hardware only when at least one of the Read (R) and Write (W) fields is set. When evaluated, hardware treats bits 63:HAW as reserved (0), where HAW is the host address width of the platform.
11:8	AVAIL: Available	This field is available to software. Hardware always ignores the programming of this field.
7	SP: Super Page	This field tells hardware whether to stop the page-walk before reaching a leaf node mapping to a 4KB page: • 0: Continue with the page walk and use the next level table. • 1: Stop the table walk and form the host physical address using the unused bits in the input address for the page walk (N-1):0 along with bits (HAW-1):N of the page base address provided in the address (ADDR) field. Hardware treats the SP field as reserved (0) in: • Leaf page-table entries corresponding to 4KB pages. • Page-directory entries corresponding to super-page sizes not defined in the architecture. • Page-directory entries corresponding to super-page sizes not supported by hardware implementation. (Hardware reports the supported super-page sizes through the Capability register.) This field is evaluated by hardware only when at least one of Read (R) and Write (W) fields is set.
6:2	R: Reserved	Reserved. Must be 0. This field is evaluated by hardware only when at least one of the Read (R) and Write (W) fields is set.
1	W: Writable[1]	Indicates whether the page is writable for DMA: • 0: Indicates the page is not accessible to DMA write requests.DMA write requests processed through this page-table entry are blocked. • 1: Indicates the page is accessible to DMA write requests.
0	R: Readable	Indicates whether the page is readable for DMA: • 0: Indicates the page is not accessible to DMA read requests. DMA read requests processed through this page-table entry are blocked • 1: Indicates the page is accessible to DMA read requests.

NOTES

1. Software may mark a page as not present for all DMA requests by clearing the R and W fields in the corresponding page-table entry.

6.4 FAULT RECORD

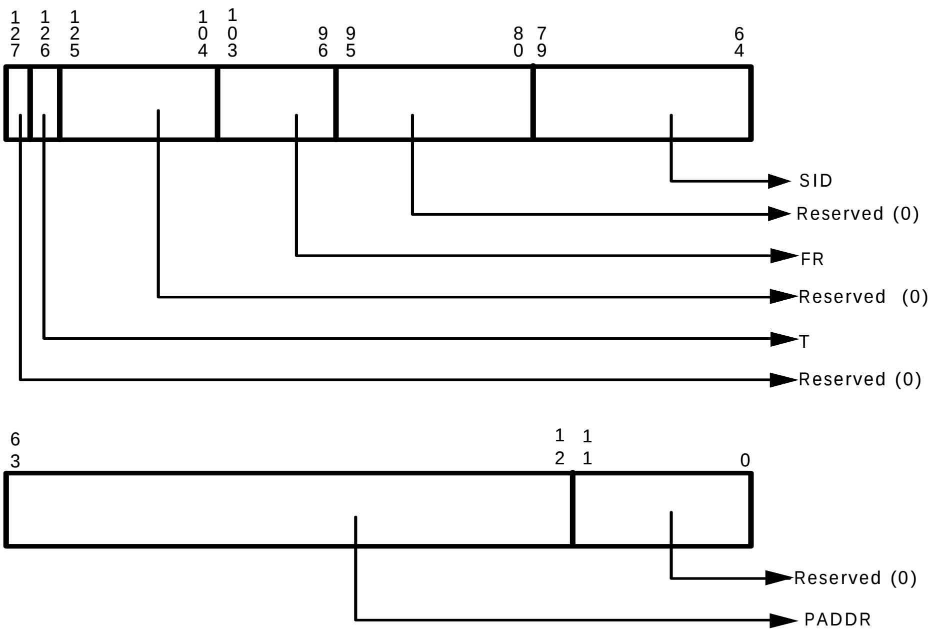

Figure 6-4. Fault-Record Format

Table 6-4. Fault-Record Entry Contents

Bits	Field	Description
127	R: Reserved	Reserved (0).
126	T: Type	Memory access type of faulted DMA request: • 0: DMA Write • 1: DMA Read request
125:104	R: Reserved	Reserved (0).
103:96	FR: Fault Reason	Reason for DMA remapping fault.
95: 80	R: Reserved	Reserved (0).
79:64	SID: Source Identifier	Requester-id in the faulted DMA request.
63:12	PADDR: Page Address	Address (page-granular) in the faulted DMA request.
11:0	R: Reserved	Reserved (0).

CHAPTER 7
DMA REMAPPING REGISTERS

This chapter describes the structure and use of the DMA Remapping Registers.

7.1 REGISTER LOCATION

The register set for each DMA remapping hardware unit in the platform is placed at a 4Kbyte-aligned memory-mapped location. The exact location of the register region is implementation-dependent, and is communicated to system software by BIOS through the DMA remapping hardware reporting structures.

7.2 SOFTWARE ACCESS TO HARDWARE REGISTERS

Software interacts with the DMA 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 the order of lower doubleword first and then the upper double-word.
- When updating registers through multiple accesses (be it in software or due to hardware disassembly), certain registers may have specific requirements on how these 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 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 RsvdZ fields when written.

7.3 REGISTER ATTRIBUTES

The following table defines the attributes used in the DMA remapping Registers. Those registers are discussed in Section 7.4.

Table 7-1. Attribute Definitions for DMA Remapping Registers

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. Read of the field indicates status. A set bit indicating a status may be cleared by writing a 1. Writing a 0 to RW1C fields has no effect.
RW1CS	Sticky Read-only status, Write-1-to-clear status field. Read of the field indicates status. A set bit indicating a status may be cleared by writing a 1. Writing a 0 to RW1C fields 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.

7.4 REGISTER DESCRIPTIONS

The following table summarizes the memory-mapped registers used for DMA remapping.

Table 7-2. DMA Remapping Register

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-Entry Table Address Register	64	Register to set up location of root-entry table.
028h	Context Command Register	64	Register to manage context-entry cache.
030h	Reserved	32	Reserved
034h	Fault Status Register	32	Register to report primary fault logging 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.

Table 7-2. DMA Remapping Register (Contd.)

Offset	Register Name	Size	Description
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-FFFh	Reserved	-	Reserved register space for future extensions.
XXXh	IOTLB Invalidation Units [m][1]	64	Each IOTLB invalidation unit consists of two 64-bit registers: Table 7-10 describes the format of each IOTLB invalidation unit registers.
XXXh	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.

NOTES

1. Hardware implementations may place IOTLB invalidation unit registers and fault recording registers in any reserved addresses above 4KB register region, or place them in subsequent 4KB regions. If one or more subsequent 4KB regions are used, unused addresses in those pages must be treated as reserved by hardware.

7.4.1 Version Register

Table 7-3. 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 DMA remapping driver written for prior architecture versions.
Register Offset	000h

Bits	Access	Default	Field	Description
31:8	RO	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.

7.4.2 Capability Register

Table 7-4. Capability Register

Abbreviation	CAP_REG
General Description	Register to report general DMA remapping hardware capabilities
Register Offset	008h

Bits	Access	Default	Field	Description
63:56	RO	0h	R: Reserved	Reserved.
55	RO	X	DRD: DMA Read Draining	• 0: On IOTLB invalidations, hardware does not support draining of translated DMA read requests queued within the root complex. • 1: On IOTLB invalidations, hardware supports draining of translated DMA read requests queued within the root complex.
54	RO	X	DWD: DMA Write Draining	• 0: On IOTLB invalidations, hardware does not support draining of translated DMA writes queued within the root complex. • 1: On IOTLB invalidations, hardware supports draining of translated DMA writes queued within the root complex.
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 (IVA_REG) register.
47:40	RO	X	NFR: Number of Fault- recording Registers	This field contains the value N-1, where N is the number of fault recording registers supported by hardware. Implementations must support at least one fault recording register (NFR = 0) for each DMA remapping hardware unit in the platform. The maximum number of fault recording registers per DMA remapping hardware unit is 256.
39:38	RO	0h	R: Reserved	Reserved.

Bits	Access	Default	Field	Description
37:34	RO	X	SPS: Super-Page support	This field indicates the super page sizes supported by hardware. A value of 1 in any of these bits indicates the corresponding super-page size is supported. The super-page sizes corresponding to various bit positions within this field are: • 0: 21-bit offset to page frame • 1: 30-bit offset to page frame • 2: 39-bit offset to page frame • 3: 48-bit offset to page frame
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 DMA 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	RO	X	SS: Spatial Separation	• 0: Indicates no spatial separation requirements for this DMA remapping hardware unit. The memory-resident structures used by this DMA remapping unit may be partially or fully shared with other DMA remapping units. • 1: Indicates spatial separation requirements for this DMA remapping hardware unit. The memory-resident structures used by this DMA remapping unit must not be shared with other DMA remapping units. This may be required by specific implementations for remapping isochronous DMA.
22	RO	X	QoS: Quality of Service	• 0: Indicates this DMA remapping hardware unit has no additional quality of service requirements. • 1: Indicates this DMA remapping hardware unit remaps DMA with specific quality of service requirements (such as with isochrony). To guarantee isochronous performance, software must ensure invalidation operations do not impact active DMA streams. This implies that when DMA is active, software performs page-selective invalidations (instead of coarser invalidations).

Bits	Access	Default	Field	Description
21:16	RO	X	MGAW: Maximum Guest Address Width	This field indicates the maximum DMA virtual addressability supported by 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, DMA requests to addresses above $2^{(X+1)}$-1 are always blocked by hardware. 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).
15:13	RO	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) 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: 30-bit AGAW(2-level page table) • 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: 64-bit AGAW (6-level page table) Software must ensure that the adjusted guest address width used to setup the page tables is one of the supported guest address widths reported in this field.

Bits	Access	Default	Field	Description
7	RO	X	CM: Caching Mode	• 0: Hardware does not cache not present and erroneous entries in the context-cache and IOTLB. 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: Hardware may cache not present and erroneous mappings in the context-cache or IOTLB. Any software updates to the DMA remapping structures (including updates to not-present or erroneous entries) require explicit invalidation. Refer to Section 7.4.9 for more details on caching mode. Hardware implementations are recommended to support operation corresponding to CM=0.
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 (through the Global Command register) to ensure updates made to memory-resident DMA remapping structures are visible to hardware.
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.

Bits	Access	Default	Field	Description
2:0	RO	X	ND: Number of domains supported	• 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. • 100b: 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.

7.4.3 Extended Capability Register

Table 7-5. Extended Capability Register

Abbreviation	ECAP_REG
General Description	Register to report DMA remapping hardware extended capabilities
Register Offset	010h

Bits	Access	Default	Field	Description
63:32	RO	0h	R: Reserved	Reserved.
31:24	RO	X	NIU: Number of IOTLB Invalidation Units	This field contains the value N-1, where N is the number of IOTLB invalidation units supported by hardware. Each IOTLB invalidation unit consists of two registers: A 64-bit IOTLB Invalidation Register (IOTLB_REG), followed by a 64-bit Invalidation Address Register (IVA_REG). Implementations must support at least one IOTLB invalidation unit (NIVU = 0) for each DMA remapping hardware unit in the platform. The maximum number of IOTLB invalidation register units per DMA remapping hardware unit is 256.
23:18	RO	0h	R: Reserved	Reserved.
17:8	RO	X	IVO: Invalidation Unit Offset	This field specifies the offset to the first IOTLB invalidation unit relative to the register base address of this DMA remapping hardware unit. If the register base address is X, and the value reported in this field is Y, the address for the first IOTLB invalidation unit is calculated as X+(16*Y). If N is the value reported in NIU field, the address for the last IOTLB invalidation unit is calculated as X+(16*Y)+(16*N).
7:4	RO	0h	R: Reserved	Reserved.

Bits	Access	Default	Field	Description
3	RO	X	FMT: Fault log memory type	• 0: Indicates hardware accesses to fault log memory region are non-coherent. • 1: Indicates hardware accesses to fault log memory region are coherent. This field is valid only for implementations supporting advanced fault logging.
2	RO	X	PMT: Page-table memory type	• 0: Indicates hardware accesses to page tables are non-coherent. • 1: Indicates hardware accesses to page-tables are coherent.
1	RO	X	CMT: Context-entry table memory type	• 0: Indicates hardware accesses to context-entry tables are non-coherent. • 1: Indicates hardware accesses to context-entry tables are coherent.
0	RO	X	RMT: Root-entry table memory type	• 0: Indicates hardware accesses to root-entry table are non-coherent. • 1: Indicates hardware accesses to root-entry table are coherent.

7.4.4 Global Command Register

Table 7-6. Global Command Register

Abbreviation	GCMD_REG
General Description	Register to control DMA remapping hardware. If multiple control fields in this register need to be modified, software must serialize through multiple writes to this register.
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 hardware: • 0: Disable DMA remapping hardware • 1: Enable DMA remapping hardware Hardware reports the status of the translation enable operation through the TES field in the Global Status register. Before enabling (or re-enabling) DMA remapping hardware through this field, software must: • Setup the DMA remapping structures in memory • Flush the write buffers (through WBF field), if write buffer flushing is reported as required. • Set the root-entry table pointer in hardware (through the SRTP field). • Perform global invalidation of the context-cache and global invalidation of IOTLB • If advanced fault logging supported, setup fault log pointer (through SFL field) and enable advanced fault logging (through the EAFL field). 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 translated 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 GSTS_REG. Value returned on 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-entry table pointer used by hardware. The root-entry table pointer is specified through the Root-entry Table Address register. Hardware reports the status of the root table pointer set operation through the RTPS field in the Global Status register. The root table pointer set operation must be performed before enabling or re-enabling (after disabling) DMA remapping hardware. After a root table pointer set operation, software must globally invalidate the context cache followed by global invalidate of IOTLB. This is required to ensure hardware uses only the remapping structures referenced by the new root table pointer, and not any stale cached entries. While DMA remapping hardware 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. Value returned on read of this field is undefined.

Bits	Access	Default	Field	Description
29	WO	0	SFL: Set Fault Log	This field is valid only for implementations supporting advanced fault logging. If advanced fault logging is not supported, writes to this field are ignored. 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 fault log set 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 hardware is active. Clearing this bit has no effect. Value returned on read of this field is undefined.
28	WO	0	EAFL: Enable Advanced Fault Logging	This field is valid only for implementations supporting advanced fault logging. If advanced fault logging is not supported, writes to this field are ignored. 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 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. Value returned on read of this field is undefined.

Bits	Access	Default	Field	Description
27	WO	0	WBF: Write Buffer Flush	This bit is valid only for implementations requiring write buffer flushing. If write buffer flushing is not required, writes to this field are ignored. Software sets this field to request hardware to flush the root-complex internal write buffers. This is done to ensure any updates to the memory-resident DMA remapping structures are not held in any internal write posting buffers. 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. Value returned on read of this field is undefined.
26:0	RsvdP	0h	R: Reserved	Reserved.

7.4.5 Global Status Register

Table 7-7. Global Status Register

Abbreviation	GSTS_REG
General Description	Register to report general DMA 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 hardware is not enabled • 1: DMA remapping hardware 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-Entry 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 bit is valid only for implementations requiring write buffer flushing. This field: • Indicates the status of the write buffer flush operation. • Is set by hardware when software sets the WBF field in the Global Command register. • Is cleared by hardware when hardware completes the write buffer flushing operation.
26:0	RO	0h	R: Reserved	Reserved.

7.4.6 Root-Entry Table Address Register

Table 7-8. Root-Entry Table Address Register

Abbreviation	RTADDR_REG
General Description	Register providing the base address of root-entry table.
Register Offset	020h

Bits	Access	Default	Field	Description
63:12	RW	0h	RTA: Root Table Address	This register points to base of the page-aligned, 4KB-sized root-entry 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-entry 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.
11:0	RsvdP	0h	R: Reserved	Reserved.

7.4.7 Context Command Register

Table 7-9. 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 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 to be 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 not submit another invalidation request through this register while the ICC field is set. Software must submit a context cache invalidation request through this field only when there are no invalidation requests pending at this DMA 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 write-buffer flushing requirement (RWBF=1 in the Capability register) must implicitly perform a write buffer flushing before reporting invalidation complete to software through the ICC field.

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	0h	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 the CAIG 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 invali-dation 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	RsvdP	0h	R: Reserved	Reserved.

Bits	Access	Default	Field	Description
33:32	WO	0h	FM: Function Mask	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 most significant bit of function number in the SID field. • 10: Mask two most significant bits of function number in the SID field. • 11: Mask all three bits of function number in the SID field. The device(s) specified through the FM and SID fields must correspond to the domain-id specified in the DID field. Value returned on read of this field is undefined.
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. Value returned on read of this field is undefined.
15:0	RW	0h	DID: Domain-ID	Indicates the id of the domain whose context-entries needs 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 may ignore and not implement bits 15:N, where N is the supported domain-id width reported in the Capability register.

7.4.8 IOTLB Invalidation Unit Registers

Hardware implementations report the number of IOTLB invalidation units through the extending capability register. Each IOTLB invalidation unit consists of two adjacently placed 64-bit registers: (a) IOTLB Invalidate Register (IOTLB_REG); and (b) Invalidate Address Register (IVA_REG). These registers are described below.

Each IOTLB Invalidation Unit is composed of two 64-bit registers as shown in the following table.

Table 7-10. IOTLB Invalidation Unit Registers

Offset	Register Name	Size	Description
XXXh	Invalidate Address Register	64	Register to provide the target address for page-selective IOTLB invalidation.
XXXh + 008h	IOTLB Invalidate Register	64	Register for IOTLB invalidation command

7.4.8.1 IOTLB Invalidate Register

Table 7-11. IOTLB Invalidate Register

Abbreviation	IOTLB_REG
General Description	Register to control page-table entry caching. The act of writing the upper byte of the IOTLB_REG with IVT field set causes the hardware to perform the IOTLB invalidation. There is an IOTLB_REG for each IOTLB Invalidation unit supported by hardware.
Register Offset	XXXh + 0008h (where XXXh is where the corresponding IVA_REG is located)

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 through any of the IOTLB invalidation units when there is a context-cache invalidation request pending at this DMA remapping hardware unit. When more than one IOTLB invalidation units are supported by a DMA remapping hardware unit, software may submit IOTLB invalidation request through any of the currently free units while there are pending requests on other units. Hardware implementations reporting write-buffer flushing requirement (RWBF=1 in Capability register) must implicitly perform a write buffer flushing before reporting invalidation complete to software through the IVT field.

Bits	Access	Default	Field	Description
62:60	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 IIRG field. Following are the encodings for the IIRG field. • 000: Reserved. • 001: Global invalidation request. • 010: Domain-selective invalidation request. The target domain-id must be specified in the DID field. • 011: Domain-page-selective 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. • 101 - 111: Reserved. 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 this time, the granularity at which actual invalidation was performed is reported through the IAIG field.
59:57	RO	0h	IAIG: IOTLB Actual Invalidation Granularity	Hardware reports the granularity at which an invalidation request was processed through this field at the time of reporting invalidation completion (by clearing the IVT field). The following are the encodings for the IAIG field. • 000: Reserved. This indicates hardware detected an incorrect invalidation request and hence ignored the request. • 001: Global Invalidation performed. This could be in response to a global, domain-selective, or domain-page-selective invalidation request. • 010: 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 domain-page-selective invalidation request. • 011: Domain-page-selective 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 domain-page-selective invalidation request. • 100 - 111: Reserved.
56:50	RsvdP	0h	R: Reserved	Reserved.

Bits	Access	Default	Field	Description
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 any translated DMA reads that are queued in the root-complex and yet to be processed. • 1: Hardware must drain all/relevant translated DMA reads that are queued in the root-complex before indicating IOTLB invalidation completion to software. A DMA read request to system memory is defined as drained when root-complex has finished fetching all of its read response data from memory.
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 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 any translated DMA writes that are queued in the root-complex for processing. • 1: Hardware must drain all/relevant translated DMA writes that are queued in the root-complex before indicating IOTLB invalidation completion to software. A DMA write request to system memory is defined as drained when the effects of the write is visible to processor accesses to all addresses targeted by the DMA write.
47:32	RW	0h	DID: Domain-ID	Indicates the ID of the domain whose IOTLB entries needs to be selectively invalidated. This field must be programmed by software for domain-selective, domain-page-selective, and device-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	0h	R: Reserved	Reserved.

7.4.8.2 Invalidate Address Register

Table 7-12. 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. Value returned on reads of this register is undefined. There is an IVA_REG for each IOTLB Invalidation unit supported by hardware.
Register Offset	XXXh (XXXh must be QWORD aligned)

Bits	Access	Default	Field	Description
63:12	WO	0h	ADDR: Address	Software provides the DMA address that needs to be page-selectively invalidated. To request a page-selective invalidation request to hardware, software must first write the appropriate fields in this register, and then issue appropriate page-selective invalidate command through the IOTLB_REG. Hardware ignores bits 63:N, where N is the maximum guest address width (MGAW) supported. Value returned on read of this field is undefined.
11:7	RsvdP	0	R: Reserved	Reserved.
6	WO	0	IH: Invalidation Hint	The field provides hints to hardware to preserve or flush the non-leaf (page-directory) entries that may be cached in hardware: • 0: Software may have modified both leaf and non-leaf page-table entries corresponding to mappings specified in the ADDR and AM fields. On a page-selective invalidation request, hardware must flush both the cached leaf and non-leaf page-table entries corresponding to mappings specified by ADDR and AM fields. • 1: Software has not modified any non-leaf page-table entries corresponding to mappings specified in the ADDR and AM fields. On a page-selective invalidation request, hardware may preserve the cached non-leaf page-table entries corresponding to mappings specified by ADDR and AM fields. Value returned on read of this field is undefined.

Bits	Access	Default	Field	Description
5:0	WO	0	AM: Address Mask	The value in this field specifies the number of low order bits of the ADDR field that must be masked for the invalidation operation. Mask field enables software to request invalidation of contiguous mappings for size-aligned regions. For example:

Mask Value	ADDR bits masked	Pages invalidated
0	None	**1**
1	12	**2**
2	13:12	4
3	14:12	8
4	15:12	16
...	...	...

Hardware implementations report the maximum supported mask value through the Capability register.

Value returned on read of this field is undefined.

7.4.9 Hardware Caching Details

This section describes the architectural behavior associated with the context-cache, IOTLB, and non-leaf (page-directory) caches.

7.4.9.1 Caching Mode Behavior

The Caching Mode (CM) field in Capability register indicates if the underlying implementation may cache not-present and erroneous DMA remapping structure entries. When the CM field is reported as set, any software updates to the DMA remapping structures (including updates to not-present entries) requires explicit invalidation of the caches.

Hardware implementations are highly recommended to not cache not-present and erroneous entries (i.e., report CM field as 0).

Software implementations that may be virtualizing the DMA remapping architecture to other software layers (such as to an operating system running within a guest partition), may report the Caching Mode (CM) of the virtual hardware as set to efficiently virtualize the DMA remapping hardware. For example, software virtualization of DMA remapping architecture typically requires the DMA remapping structures to be shadowed in software. 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 utilize these virtual invalidation operations to keep the shadow tables consistent to any guest structure modifications, without resorting to other less efficient techniques (such as write-protecting the guest structures through CPU page tables).

7.4.9.2 Context Caching

For implementations reporting Caching Mode (CM) as clear in the Capability register, if hardware encounters any of the following fault conditions as part of accessing a root-entry or a context-entry, the resulting entry is not cached in the context-cache.

- Hardware attempt to access the root table through the RTA field in the Root-entry Table Address register resulted in error.
- Present (P) field of the root-entry is clear.
- Hardware detected invalid programming of one or more fields in the present root-entry.
- Hardware attempt to access a context-entry table through the Context Table Pointer (CTP*)* field in the present root-entry resulted in error.
- Present (P) field of the context-entry is clear.
- Hardware detected invalid programming of one or more fields in the present context-entry.
- Hardware checks reserved fields, and detected one or more non-zero reserved fields in the present root-entry or context-entry.

For implementations reporting Caching Mode (CM) as set in the Capability register, above conditions may cause hardware to cache the resulting entry.

Since information from the present context-entries (such as domain-id) may be utilized to tag the IOTLB and/or the non-leaf (page-directory) caches, on root-entry and context-entry modifications that require context-cache invalidation, software must also domain-selectively (or globally) invalidate the IOTLB after the context-cache invalidation is completed.

7.4.9.3 IOTLB

IOTLB caches effective translations for a given DMA address, including the cumulative Read and Write permissions from the page-walk leading to this translation.

For implementations reporting Caching Mode (CM) as clear in the Capability register, IOTLB caches only valid mappings (i.e. results of successful page-walks with effective translations that has at least one of the cumulative Read and Write permissions from the page-walk being set). Specifically, if hardware encounters any of the following conditions, the results are not cached in the IOTLB:

- Conditions listed in Section 7.4.9.2 and Section 7.4.9.3.
- Hardware attempt to access the page table through the ADDR field of the previous page-directory entry in the page walk resulted in error.

- Read (R) and Write (W) fields of the page table entry is clear (not-present entry).
- Hardware detected invalid programming of one or more fields in the present page table entry.
- Hardware checks reserved fields, and detected one or more non-zero reserved fields in the present page table entry.
- The cumulative read and write permissions from the page-walk was both clear (effectively not-present entry).

For implementations reporting Caching Mode (CM) as set in the Capability register, above conditions may cause hardware to cache erroneous or not-present mappings in the IOTLB.

7.4.9.4 Page Directory Entry (PDE) Caching

Support for non-leaf (page-directory entry) caching is hardware implementation dependent, and is transparent to software.

For implementations reporting Caching Mode (CM) as clear in the Capability register, if hardware encounters any of the following fault conditions as part of accessing a page-directory entry, the resulting entry is not cached in the non-leaf caches.

- Hardware attempt to access the page-directory through either the ASR field of context-entry (in case of root page directory), or the ADDR field of the previous page directory entry in the page-walk resulted in error.
- Read (R) and Write (W) fields of the page-directory entry is clear (not present entry).
- Hardware detected invalid programming of one ore more fields in the present page directory entry.
- Hardware checks reserved fields, and detected one or more non-zero reserved fields in the present page directory entry.

For implementations reporting Caching Mode (CM) as set in the Capability register, above conditions may cause hardware to cache the corresponding page-directory entries.

7.4.9.5 PDE Cache Invalidation

Hardware implementations supporting non-leaf (PDE) caching functions as follows:

- Globally invalidate the non-leaf caches on IOTLB global invalidations.
- Domain-selective (or global) invalidation of non-leaf caches on IOTLB domain-selective invalidations.
- For IOTLB page-selective invalidations with Invalidation Hint (IH) field set to 1, hardware is recommended to preserve the non-leaf cache entries.
- For IOTLB page-selective invalidations with IH field clear, hardware must invalidate the appropriate non-leaf cache entries. This may be achieved by invalidating only the cached non-leaf entries corresponding to the mappings specified in the Invalidation Address

registers (IVA_REG), or through a domain-selective (or global) invalidation of the non-leaf caches.

- The results reported by hardware in the IAIG field of the IOTLB Invalidation registers (IOTLB_REG) is always the granularity at which the invalidation was performed on the IOTLB.

7.4.10 Fault Status Register

Table 7-13. Fault Status Register

Abbreviation	FSTS_REG
General Description	Register indicating the primary fault logging status. Section 7.4.18.1 describes hardware behavior for primary fault logging.
Register Offset	034h

Bits	Access	Default	Field	Description
31:16	RO	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.
7:2	RO	0h	R: Reserved	Reserved.
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 DMA 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.

Bits	Access	Default	Field	Description
0	RW1CS	0	PFO: Primary Fault Overflow	Hardware sets this field to indicate overflow of fault recording registers. Software writing 1 clears this field.

7.4.11 Fault Event Control Register

Table 7-14. Fault Event Control Register

Abbreviation	FECTL_REG
General Description	Register specifying the fault event interrupt message control bits. Section 7.4.12 describes hardware handling of fault events.
Register Offset	038h

Bits	Access	Default	Field	Description
31	RW	1	IM: Interrupt Mask	• 0: No masking of interrupt. When a interrupt condition is detected, hardware issues an interrupt message (using the Fault Event Data & 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. Interrupt condition 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 Fault Status register. If the PPF field was already set at the time of recording a fault, it is not treated as a new interrupt condition. • 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 Advanced Fault Log register. If the APF field was already set at the time of detecting/recording a fault, 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 interrupting condition through one of the following ways: • 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 Fault Status register to be evaluated as clear. • When advanced fault logging is active, software clearing the APF field in Advanced Fault Log register.
29:0	RsvdP	0h	R: Reserved	Reserved.

7.4.12 Hardware Handling of Fault Events

The following sub-sections describe the hardware fault event generation behavior when primary or advanced fault logging is active. In both cases, read completions due to software reading the DMA remapping hardware registers must push (commit) any in-flight interrupt messages generated by the respective DMA remapping hardware unit.

7.4.12.1 Fault Event Generation with Primary Fault Logging

When primary fault recording is active, the hardware interrupt generation logic functions as follows:

- If there are pending faults to be processed by software, the fault event is not generated. Hardware checks for pending faults condition through the Primary Pending Faults (PPF) field of the Fault Status register. (The PPF field is computed by hardware as the logical OR of Fault (F) fields across all the Fault Recording registers of the DMA remapping hardware unit.)
- If the above check indicates no pending faults, the Interrupt Pending (IP) field in the Fault Event Control register is set at the time of recording the fault. The Interrupt Mask (IM*)* field of the Fault Event Control register is then checked, and one of the following conditions 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.

If the IP field was set when software clears the IM field, the fault event is generated along with clearing the IP field.

If the IP field was set when a software update to the fault recording registers causes a ‘no pending faults’ condition (i.e., the Fault [F] fields in all the fault recording registers are clear, causing the PPF field to be evaluated as clear), the IP field is cleared.

7.4.12.2 Fault Event Generation with Advanced Fault Logging

When advanced fault recording is active, the hardware interrupt generation logic functions as follows:

- If there are pending faults to be processed by software, the fault event is not generated. Hardware checks for a pending fault condition by checking if the Advanced Pending Faults (APF) field in the Advanced Fault Log register.
- If the above check indicates no pending faults, and the current fault is being recorded to the first fault record (at index 0) of the fault log, the Interrupt Pending (IP) field in the Fault Event Control register is set when the fault is recorded. The Interrupt Mask (IM) field of the Fault Event Control register is then checked, and one of the following conditions applied:
    - If the IM field is clear, the fault event is generated, and the IP field is cleared.
    - If the IM field is set, the interrupt is not generated.

If the IP field is set when software clears the IM field, the fault event is generated, and the IP field is cleared.

If the IP field is set when software clears the APF field in the Advanced Fault Log register (causing a “no pending faults” condition), the IP field is cleared.

7.4.13 Fault Event Data Register

Table 7-15. 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 / RO	0h	EID: Extended Interrupt Message Data	This field is valid only for implementations supporting 32-bit MSI data fields. Hardware implementations supporting only 16-bit MSI data may treat this field as read-only (0).
15:0	RW	0h	ID: Interrupt message data	Data value in the fault-event interrupt message.

7.4.14 Fault Event Address Register

Table 7-16. 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 MSI memory write transaction.
1:0	RO	0h	R: Reserved	Reserved.

7.4.15 Fault Event Upper Address Register

Table 7-17. Fault Event Upper Address Register

Abbreviation	FEUADDR_REG
General Description	Register specifying the interrupt message address. For platforms supporting only interrupt messages in the 32-bit address range, this register is treated as read-only (0).
Register Offset	044h

Bits	Access	Default	Field	Description
31:0	RW/RO	0h	MUA: Message upper address	This register needs to be implemented only if hardware supports 64-bit message addresses. If implemented, the contents of this register specify the upper 32-bits of a 64-bit MSI write transaction. If hardware does not support 64-bit messages, the register is treated as read-only (0).

7.4.16 Fault Recording Registers [n]

Table 7-18. Fault Recording Register

Abbreviation	FRCD_REG [n]
General Description	Registers to record DMA remapping 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 via software clearing the RW1C fields by writing a 1.
Register Offset	XXXh (XXXh 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 the PADDR, SID, FR and T fields. When this field is set, hardware may collapse additional faults from the same requestor (SID). Software writes the value read from this field to clear it. Refer to Section 7.4.18.1 for hardware details of primary fault logging.
126	ROS	0	T: Type	Type of the faulted DMA request: • 0: DMA write • 1: DMA read request This field is relevant only when the F field is set.
125:104	RO	0h	R: Reserved	Reserved.
103:96	ROS	0h	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.
95:80	RO	0h	R: Reserved	Reserved.
79:64	ROS	0h	SID: Source Identifier	Requester-id of the faulted DMA request. This field is relevant only when the F field is set.

Bits	Access	Default	Field	Description
63:12	ROS	0h	PADDR: Page Address	This field contains the address (page-granular) in the faulted DMA request. Hardware may treat bits 63:N as reserved (0), where N is the maximum guest address width (MGAW) supported. This field is relevant only when the F field is set.
11:0	RO	0h	R: Reserved	Reserved.

NOTES

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 as 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 as set before reading the rest of the data fields in that register

7.4.17 Advanced Fault Log Register

Table 7-19. Advanced Fault Log Register

Abbreviation	AFLOG_REG
General Description	Register to specify the base address of memory-resident fault-log region. This register is treated as read-only (0) for implementations not supporting advanced translation fault logging (AFL field reported as 0 in the Capability register). This register is sticky and can be cleared only through powergood reset or via software clearing the RW1C fields by writing a 1.
Register Offset	058h

Bits	Access	Default	Field	Description
63:12	RW	0h	FLA: Fault Log Address	This field specifies the base of size-aligned fault-log region in system memory. Hardware may ignore 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.
8:2	RsvdP	0h	R: Reserved	Reserved.
1	RW1CS	0h	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.
0	RW1CS	0h	AFO: Advanced Fault Overflow	Hardware sets this field to indicate an advanced fault log overflow condition. Software writing 1 to this field clears it.

7.4.18 Hardware Handling of Fault Logging

For both primary and advanced fault logging, hardware logs fault information following a circular first-in-first-out (FIFO) model. The following sub-sections describe the hardware fault logging behavior when primary or advanced fault logging is active.

7.4.18.1 Hardware Handling of Primary Fault Logging

Hardware maintains an internal index to reference the Fault Recording register to which the next fault can be recorded. The index is reset to zero when the DMA remapping hardware is enabled/disabled through the Global Command register, and increments whenever a fault is recorded to 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 DMA remapping hardware unit.

Hardware maintains the Primary Pending Fault (PPF) field 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 DMA remapping fault:

- If hardware supports compressing[1] of multiple faults from the same requestor, it compares the SID field of each Fault Recording register with Fault (F) field set to the source-id of the currently faulted DMA 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 the F field in this register 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 no overflow condition, hardware records the current fault information in the Fault Recording register (FR, T, SID, PADDR, fields) 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 PPF field also to be set). The internal index is incremented, and a fault event is generated based on the programming of the Fault Event Control register. Section 7.4.12.1 describes hardware behavior for fault event generation for primary fault logging.

1. Hardware implementations supporting only a limited number of fault recording registers per DMA remapping hardware unit are recommended to collapse multiple pending faults from the same requestor.

Software is expected to process the faults reported through the fault recording registers in a circular FIFO fashion starting from the Fault Recording register referenced by the FRI field, until it finds a fault recording register with no faults (F field clear).

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 these registers, and clear the overflow status by writing a 1 to the 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.

7.4.18.2 Hardware Handling of Advanced Fault Logging

When advanced fault recording is active, hardware maintains an internal index into the memory-resident fault log where the next 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 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 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 DMA requestor 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 DMA remapping hardware registers must push (commit) any in-flight fault record writes to the fault log by the respective DMA remapping hardware unit.

When a DMA remapping fault is detected, hardware advanced fault logging functions as follows:

- If hardware supports compressing multiple back-to-back faults from same requestor, it compares the source-id of the currently faulted DMA request to the internally maintained ‘source-id from previous fault’. If a match is detected, the fault information is not recorded.
- Else if the internal index wrap-around condition is set (implying fault log is full), hardware sets the Advanced Fault Overflow (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 APF field in the Advanced Fault Log register and 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).

- Else, hardware sets the APF field and increments the internal index. A fault event is generated based on the programming of the Fault Event Control register. Section 7.4.12.2 describes hardware behavior for fault event generation for advanced fault logging.

7.4.19 Protected Memory Enable Register

Table 7-20. Protected Memory Enable Register

Abbreviation	PMEN_REG
General Description	Register to enable the DMA protected memory regions setup through the PLMBASE, PLMLIMT, PHMBASE, PHMLIMIT registers. This register is always treated as RO (0) for implementations not supporting protected memory regions (PLMR and PHMR fields reported as 0 in the Capability register). Protected memory regions may be used by software to securely initialize DMA remapping structures in memory.
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: DMA accesses to protected memory regions are handled as follows: • If DMA remapping hardware is not enabled, DMA requests (including those to protected regions) are not blocked. • If DMA remapping hardware is enabled, DMA requests are translated per the programming of the DMA remapping structures. Software may program the DMA remapping structures to allow or block DMA to the protected memory regions. • 1: DMA accesses to protected memory regions are handled as follows: • If DMA remapping hardware is not enabled, DMA requests to protected memory regions are blocked. These DMA requests are not recorded or reported as DMA remapping faults. • If DMA remapping hardware is enabled, hardware may or may not block DMA to the protected memory region(s). Software must not depend on hardware protection of the protected memory regions, and must ensure the DMA remapping structures are properly programmed to not allow DMA to the protected memory regions. 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.
30:1	RsvdP	0h	R: Reserved	Reserved.
0	RO	0h	PRS: Protected Region Status	This field indicates the status of protected memory region: • 0: Protected memory region(s) not enabled. • 1: Protected memory region(s) enabled.

7.4.20 Protected Low-Memory Base Register

Table 7-21. Protected Low-Memory Base Register

Abbreviation	PLMBASE_REG
General Description	Register to setup the base address of DMA protected low-memory region. This register must be setup before enabling protected memory through PMEN_REG, and must not be updated when protected memory regions are enabled. When CMD.LOCK.PMRC command is invoked, this register is locked (treated RO). When CMD.UNLOCK.PMRC command is invoked, this register is unlocked (treated RW). This register is always treated as RO for implementations not supporting protected low memory region (PLMR field reported as 0 in the Capability register). The alignment of the protected low memory region base depends on the number of reserved bits (N) of this register. Software may determine the value of 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 this register is decoded by hardware as all 0s.
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	RsvdP	0h	R: Reserved	Reserved.

7.4.21 Protected Low-Memory Limit Register

Table 7-22. Protected Low-Memory Limit Register

Abbreviation	PLMLIMIT_REG
General Description	Register to setup the limit address of DMA protected low-memory region. This register must be setup before enabling protected memory through PMEN_REG, and must not be updated when protected memory regions are enabled. When CMD.LOCK.PMRC command is invoked, this register is locked (treated RO). When CMD.UNLOCK.PMRC command is invoked, this register is unlocked (treated RW). This register is always treated as RO for implementations not supporting protected low memory region (PLMR field reported as 0 in the Capability register). The alignment of the protected low memory region limit depends on the number of reserved bits (N) of this register. Software may determine the value of 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 is decoded by hardware as all 1s. The Protected low-memory base & limit registers functions as follows: • Programming the protected low-memory base and limit registers with 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.
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	RsvdP	0h	R: Reserved	Reserved.

7.4.22 Protected High-Memory Base Register

Table 7-23. Protected High-Memory Base Register

Abbreviation	PHMBASE_REG
General Description	Register to setup the base address of DMA protected high-memory region. This register must be setup before enabling protected memory through PMEN_REG, and must not be updated when protected memory regions are enabled. When CMD.LOCK.PMRC command is invoked, this register is locked (treated RO). When CMD.UNLOCK.PMRC command is invoked, this register is unlocked (treated RW). This register is always treated as RO for implementations not supporting protected high memory region (PHMR field reported as 0 in the Capability register). The alignment of the protected high memory region base depends on the number of reserved bits (N) of this register. Software may determine the value of 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 is decoded by hardware as all 0s.
Register Offset	070h

Bits	Access	Default	Field	Description
63:(N+1)	RW	0h	PHMB: Protected High-Memory Base	This register specifies the base of size aligned, protected memory region in system memory. Hardware may ignore and not implement bits 63:HAW, where HAW is the host address width.
N:0	RsvdP	0h	R: Reserved	Reserved.

7.4.23 Protected High-Memory Limit Register

Table 7-24. Protected High-Memory Limit Register

Abbreviation	PHMLIMIT_REG
General Description	Register to setup the limit address of DMA protected high-memory region. This register must be setup before enabling protected memory through PMEN_REG, and must not be updated when protected memory regions are enabled. When CMD.LOCK.PMRC command is invoked, this register is locked (treated RO). When CMD.UNLOCK.PMRC command is invoked, this register is unlocked (treated RW). This register is always treated as RO for implementations not supporting protected high memory region (PHMR field reported as 0 in the Capability register). The alignment of the protected high memory region limit depends on the number of reserved bits (N) of this register. Software may determine the value of 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 is decoded by hardware as all 1s. The protected high-memory base & limit registers functions 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.
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	RsvdP	0h	R: Reserved	Reserved.

CHAPTER 8
EXTENDED DMA REMAPPING FEATURES

This chapter summarizes extended I/O virtualization features being considered to support future extensions to the PCI Express specification that are relevant to DMA remapping.

8.1 ON-DEVICE IOTLBS

The DMA remapping architecture described in Section 3 supports address translation of DMA requests received by a platform's core logic chipset components. Section 3.3.1.3 describes the use of IOTLBs in these core logic chipset components to cache frequently used I/O page tables to improve the DMA address translation performance. IOTLBs improve DMA remapping performance by avoiding the multiple memory accesses required to access the I/O page tables for DMA address translation. However, the efficiency of IOTLBs in the core logic is directly proportional to the hit rates of IOTLB lookups, which indirectly depends on the number of DMA virtual address localities and the number of simultaneously active DMA streams in the platform.

One approach to scaling IOTLBs is to allow I/O devices to participate in the DMA remapping process with IOTLBs implemented at these devices. IOTLBs at the devices alleviate pressure for IOTLB resources in the chipset, and provide opportunities for devices to improve performance by pre-fetching address translations before issuing DMA requests.

IOTLB support in I/O devices requires standardized mechanisms for:

- I/O devices to request and receive translations from the chipset
- I/O devices to indicate if a DMA request has translated or un-translated addresses
- Translations cached at the on-device IOTLBs to be invalidated

Extensions to PCI Express to support these mechanisms are being defined in the PCISIG and are expected to be presented in future PCI Express specifications.

8.2 DMA REMAPPING - EXTENDED FEATURES

The extended features in Intel® Virtualization Technology for Directed I/O enable these upcoming extensions to PCI Express by providing support for remote IOTLBs. Specifically, the following extended features are considered:

- Capability for software to control which devices are allowed to participate in DMA remapping
- Ability to respond to address translation requests from enabled devices

- Ability to bypass DMA address translation for DMA requests with translated addresses from enabled devices
- Extended methods for supporting on-device IOTLB invalidations

As the PCI Express extensions are standardized and defined, future versions of this specification will include more details of these extended capabilities.

APPENDIX A FAULT REASON ENCODINGS

The following table describes the meaning of the codes assigned to various faults.

Table A-1. Fault Reason Encodings

Encoding	Fault Reason Description
0x00	Reserved. Used by software when initializing fault records (for advanced fault logging)
0x01	The present (P) field in the root-entry used to process the DMA request is clear.
0x02	The present (P) field in the context-entry used to process the DMA request is clear.
0x03	Hardware detected invalid programming of a context-entry. For example: • The address width (AW) field was programmed with a SAGAW value not supported by the hardware implementation. • The translation-type (T) field was programmed to indicate a translation type not supported by the hardware implementation. • A hardware attempt to access the page table base through the Address Space Root (ASR) field of the context-entry resulted in error.
0x04	The DMA request attempted to access an address beyond (2X - 1), where X is: • For multi-level page-table based translation, 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 used to process the DMA request.
0x05	The Write (W) field in a page-table entry used for address translation of the DMA write request is clear.
0x06	The Read (R) field in a page-table entry used for address translation of the DMA read request is clear.
0x07	A hardware attempt to access the next level page table through the Address (ADDR) field of the page-table entry resulted in error.
0x08	A hardware attempt to access the root-entry table through the root-table address (RTA) field in the Root-entry Table Address register resulted in error.
0x09	A hardware attempt to access context-entry table through context-entry table pointer (CTP) field resulted in error.
0x0A	Hardware detected reserved field(s) that are not initialized to zero in a root-entry with present (P) field set.
0x0B	Hardware detected reserved field(s) that are not initialized to zero in a context-entry with present (P) field set.

Table A-1. Fault Reason Encodings (Contd.)

Encoding	Fault Reason Description
0x0C	Hardware detected reserved field(s) that are not initialized to zero in a page-table entry with at least one of read (R) and write (W) field set.
0x0D - 0xFF	Reserved.