INTEL® SCALABLE I/O VIRTUALIZATION

Technical Specification

September 2020

Order Number: 337679-002, Rev 1.1

Table of Contents

Table of Figures

List of Tables

Introduction

Intel® Scalable I/O Virtualization (Intel® Scalable IOV) is a scalable and flexible approach to hardware-assisted I/O virtualization. Intel Scalable IOV builds on existing PCI Express* capabilities, enabling it to be easily supported by compliant PCI Express endpoint device designs and the software ecosystem.

This document specifies the Intel Scalable IOV architecture, including host platform and endpoint device capabilities required to support it, and describes a high-level reference software architecture.

Document Organization

Chapter 1 provides an architectural overview of Intel Scalable IOV and its key components.
Chapter 2 specifies endpoint device blueprint and requirements.
Chapter 3 describes the required host platform Root Complex (RC) support.
Chapter 4 describes the reference software architecture.

Audience

This document is for endpoint device developers implementing scalable hardware support for I/O virtualization and sharing, for driver developers for such devices, and for Operating System and Virtual Machine Monitor developers who are enabling hardware-assisted I/O virtualization.

Reference Documents

Intel® Virtualization Technology for Directed I/O Specification, Rev 3.1
Intel® Architecture Instruction Set Extensions Programming Reference
PCI Express* Base Specification, Revision 4.0, Version 1.0
PCI Express* ECN - Deferrable Memory Write (DMWr) and Device 3 Extended Capability

Revision History

Date	Revision	Description
June 2018	1.0	Technical preview release
September 2020	1.1	Specification update

Terms and Abbreviations

Acronym	Term	Description
SR-IOV	Single Root I/O Virtualization	SR-IOV as specified by the PCI Express Base Specification, Revision 4.0, Version 1.0.
Intel Scalable IOV	Intel Scalable I/O Virtualization	Software composable and scalable I/O virtualization as specified by this document.
PF	Physical Function	PCI Express Physical Function as specified by SR-IOV.
VF	Virtual Function	PCI Express Virtual Function as specified by SR-IOV.
ADI	Assignable Device Interface	Assignable Device Interface is the unit of assignment for a device.
DWQ	Dedicated Work Queue	A work queue that can be assigned to a single address domain at a time.
SWQ	Shared Work Queue	A work queue that can be assigned to multiple address domains simultaneously.
PASID	Process Address Space Identifier	Process Address Space ID and its TLP prefix as specified by the PCI Express Base Specification.
RID	Requester ID	Bus/Device/Function number identity for a PCI Express function (PF or VF).
IMS	Interrupt Message Storage	Device-specific interrupt message storage for ADIs.
MSI-X	Message Signaled Interrupts Extended	MSI-X capability as defined by the PCI Express Base Specification.
FLR	Function Level Reset	Function Level Reset as defined by the PCI Express Base Specification.
VMM	Virtual Machine Monitor	System software that creates and manages virtual machines. Also known as Hypervisor.
VM	Virtual Machine	An isolated execution environment constructed by a VMM which runs a guest OS.
Host OS	Host Operating System	The privileged OS that works with the VMM to virtualize the platform.

Acronym	Term	Description
VDCM	Virtual Device Composition Module	A device-specific component that is responsible for composing a Virtual Device. Typically, this is a software component in the VMM or host driver, but other implementations are possible.
VDEV	Virtual Device	A virtual device composed by VDCM that utilizes one or more ADIs.
Guest Driver	Guest Driver	Device-specific software that runs in a VM and manages virtual device operation.
Host Driver	Host Driver	Device-specific software that runs in the host OS and manages physical device operation.
GVA	Guest Virtual Address	Virtual address space of a process executing within a VM.
GPA	Guest Physical Address	Physical address space of a VM as seen by guest software.
HPA	Host Physical Address	Physical address space of hardware machine.
SVM	Shared Virtual Memory	A memory model that enables I/O devices to operate in shared virtual address space with CPU.
ATS	Address Translation Services	Ability for device to request and cache address translations. Refer to the PCI Express Specification.
DMWr	Deferrable Memory Write	Refer to the PCI Express ECN for Deferrable Memory Write (DMWr) .

1 Overview

This chapter provides background on I/O virtualization and introduces the key concepts and components of Intel Scalable I/O Virtualization.

1.1 Virtualization Background

Virtualization allows system software called a virtual machine monitor (VMM), also known as a hypervisor, to create multiple isolated execution environments called virtual machines (VMs), in which operating systems and applications can run. Virtualization is extensively used in enterprise and cloud datacenters as a mechanism to consolidate multiple workloads onto a single physical machine while keeping them isolated from each other.

Containers are another type of isolated environment that are used to package and deploy applications and run them in the isolated environment. Containers may be constructed as either bare-metal containers that are instantiated as OS process groups or as machine containers that utilize the increased isolation properties of hardware support for virtualization. Containers are lighter weight than VMs and can be deployed in much higher density, potentially increasing the number of isolated environments on a system by an order of magnitude. This document primarily refers to isolated domains as VMs, but the principles also apply to other domain abstractions such as containers.

Modern processors provide features to reduce virtualization overhead that may be utilized by VMMs to allow VMs direct access to hardware resources. Intel® Virtualization Technology (Intel® VT-x) defines the Intel® processor hardware capabilities to reduce overheads for processor and memory virtualization. Intel® Virtualization Technology for Directed I/O (Intel® VT-d) defines the platform hardware features for direct memory access (DMA) and interrupt remapping and isolation that can be utilized to minimize overheads of I/O virtualization.

I/O virtualization refers to the virtualization and sharing of I/O devices across multiple VMs or container instances. There are multiple existing approaches for I/O virtualization that may be broadly classified as either software-based or hardware-assisted.

With software-based I/O virtualization, the VMM exposes a virtual device, such as a Network Interface Controller (NIC), to a VM. A software device model in the VMM emulates the behavior of the virtual device. The device model translates from virtual device commands to physical device commands that are forwarded to a physical device. Such software emulation of devices can provide good compatibility to software running within VMs but incurs significant performance overhead, especially for high performance devices. In addition to the performance limitations, emulating virtual devices in software can be complex for programmable devices such as Graphics Processing Units (GPUs) and Field-Programmable Gate Arrays (FPGAs) because these devices perform a variety of complex functions. Variants of software-based I/O virtualization such as 'device paravirtualization' and 'mediated pass-through' can mitigate some of the performance and complexity disadvantages of device emulation.

To avoid the overheads of software-based I/O virtualization, VMMs may make use of platform support for DMA and interrupt remapping (such as Intel VT-d) to support 'direct device assignment', which allows guest software to directly access an assigned device. Direct device assignment provides the best I/O virtualization performance since the hypervisor is no longer in the path of most guest software accesses to the device. However, this approach requires the device to be exclusively assigned to a VM and does not support sharing of the device across multiple VMs.

Single Root I/O Virtualization (SR-IOV) is a PCI-SIG* defined specification for hardware-assisted I/O virtualization that defines a standard way for partitioning endpoint devices for direct sharing across multiple VMs or containers. An SR-IOV capable endpoint device supports a Physical Function (PF) and multiple Virtual Functions (VFs). The PF provides resource management for the device and is managed by the host driver running in the host OS. Each VF can be assigned to a VM or container for direct access. SR-IOV is supported by high performance I/O devices such as network and storage controller devices as well as programmable or reconfigurable devices such as GPUs, FPGAs, and other accelerators.

1.2 Intel® Scalable I/O Virtualization

Intel Scalable IOV is an approach to hardware-assisted I/O virtualization that enables highly scalable and high-performance sharing of I/O devices across isolated domains, while containing the cost and complexity for endpoint device hardware to support such scalable sharing.

Figure 1-1 illustrates two example approaches to Intel Scalable IOV, showing how it enables flexible composition of virtual devices for device sharing. Accesses between a VM and a virtual device are defined as either 'direct path' or 'intercepted path'. Direct-path operations on the virtual device are mapped directly to the underlying device hardware for performance, while intercepted-path operations are emulated by the Virtual Device Composition Module (VDCM) for greater flexibility.

The exact mechanism for virtual device composition is implementation specific. For example, Figure 1-1 (a) shows a system that implements VDCM in host OS or VMM software, whereas Figure 1-1 (b) shows a system that implements VDCM in an embedded controller on the platform. VDCM configures the device through

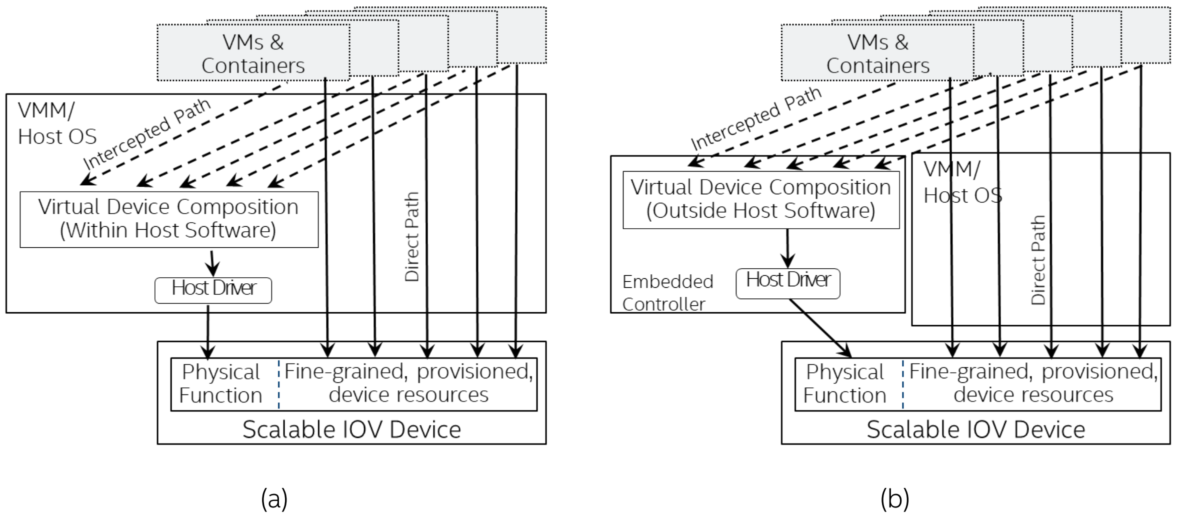

Figure 1-1: Approaches to Intel® Scalable I/O Virtualization

the host driver. VDCM and the host driver may be co-located. For simplicity, this specification primarily uses the example where VDCM is implemented in the host OS or VMM software, but the architecture can be applied to other mechanisms of virtual device composition.

Figure 1-2 illustrates the main benefits of Intel Scalable IOV. Device resources shown as "Q" can be directly mapped to VMs. A VDEV is a virtual device instance that is exposed to a VM. Virtual device composition enables increased sharing scalability and flexibility at lower hardware cost and complexity. It provides system software the flexibility to share device resources with different address domains using different abstractions. For example, application processes may access a device using system calls and VMs may access a device using virtual device interfaces. Virtual device composition can also enable dynamic mapping of VDEVs to device resources, allowing a VMM to over-provision device resources to VMs.

In a data-center with physical machines containing different generations (versions) of the same I/O device, a VMM can use the virtual device composition to present the same VDEV capabilities irrespective of the different generations of physical I/O devices. This ensures that the same guest OS image with a VDEV driver can be deployed or migrated to any of the physical machines.

The Intel Scalable IOV architecture is composed of the following elements:

Endpoint device support	PCI Express endpoint device requirements and capabilities, covered in Chapter 2.
Platform support	Host platform (Root Complex) requirements including enhancements to DMA remapping hardware. These requirements are implemented on Intel® platforms as part of Intel Virtualization Technology for Directed I/O, Rev 3.1 or higher. This is covered in Chapter 3.
Virtual Device Composition Module support	Virtual device composition architecture. This specification describes the software-based virtual device composition architecture in detail, including host system software enabling and device specific software components such as host driver, guest driver, and virtual device composition module (VDCM). This is covered in Chapter 4.

PCI Express endpoint devices may be designed to operate with either Intel Scalable IOV or SR-IOV. Device implementations that already support SR-IOV can maintain it for backwards compatibility while adding the new capabilities to support Intel Scalable IOV. A device capable of both methods should allow software to enable it to operate in one mode or other. Devices may support both methods concurrently or support Intel

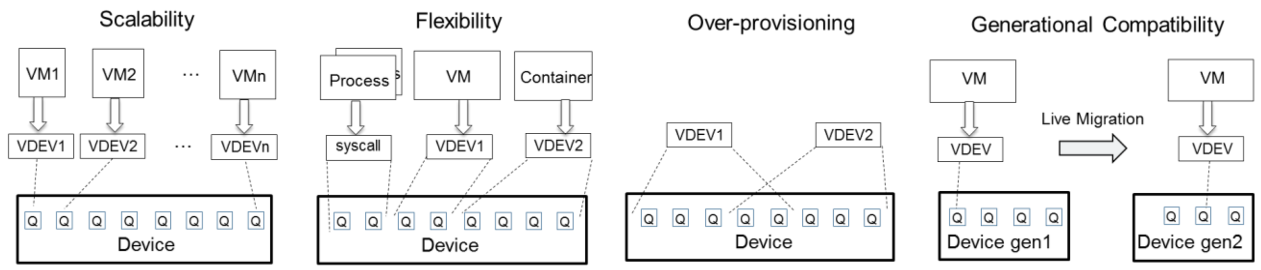

Figure 1-2: Main benefits of Intel® Scalable I/O Virtualization

Scalable-IOV operation in a hierarchical manner over SR-IOV VFs, but these modes of operation are beyond the scope of this document.

1.2.1 Separation of Direct-path and Intercepted-path Operations

The hardware-software interface for most I/O devices may be divided into (a) slow-path control and configuration operations that are less frequent and have less impact on overall device performance; and (b) fast-path command and completion operations that are frequent and have higher impact on overall device performance. This distinction of slow-path and fast-path operations is practiced by high performance I/O devices that support direct user-mode access. Intel Scalable IOV extends such device designs to define a dynamically composable approach to I/O virtualization.

Intel Scalable IOV distinguishes intercepted-path and direct-path accesses. Intercepted-path accesses from VMs go through the virtual device composition module, while direct-path accesses are mapped directly to the device. Which operations and accesses are distinguished as intercepted path versus direct path is controlled by the device implementation. Typically, slow-path operations are treated as intercepted-path accesses and fast-path operations are treated as direct-path accesses. For example, intercepted-path accesses typically include initialization, control, configuration, management, QoS, error processing, and reset, whereas direct-path accesses typically include data-path operations involving work submission and work completion processing.

1.2.2 Assignable Device Interfaces

High performance I/O devices support a large number of command/completion interfaces for efficient multiplexing/demultiplexing of I/O. Intel Scalable IOV defines an approach to assign these device interfaces to isolated domains at a fine granularity. The architecture defines the granularity of sharing of a device as an 'Assignable Device Interface' (ADI). Each ADI instance on the device encompasses the set of resources on the device that are allocated by software to support the direct-path operations for a virtual device. See section 2.1 for examples of the types of resources that may constitute ADIs.

1.2.3 Platform Scalability Using PASIDs

All ADIs on a device function use the same PCIe* Requester ID (Bus/Device/Function number) corresponding to the device's PCIe Function. Process Address Space Identifiers (PASID) are used to distinguish upstream memory transactions performed for different ADIs and to convey the address space targeted by the transaction. The 20-bit PASID associated with a transaction is conveyed in a PCI Express PASID TLP Prefix. Refer to the PCI Express specification for details on the PASID TLP Prefix. A platform with support for Intel Scalable IOV enables a unique address translation function for upstream requests for each PASID, as described in Chapter 3.

1.2.4 Virtual Device Composition

A device-specific component called the Virtual Device Composition Module (VDCM) is responsible for managing virtual device instances.

Figure 1-3 illustrates an example software architecture where VDCM is implemented in host software. The figure calls out key components to describe the architecture and is not intended to illustrate all virtualization software or specific implementation choices. Software responsibilities are abstracted between system software (OS/VMM) and device-specific driver software components. The VMM maps direct-path accesses from the guest directly onto the provisioned ADIs for the VDEV. The VMM traps intercepted-path accesses from the guest and forwards them to VDCM for emulation. VDCM emulates the intercepted accesses to the VDEV. If required, it may access the physical device (for example, to read ADI status or configure the ADI's PASID).

Virtualization management software may make use of VDCM interfaces for virtual device resource and state management, enabling capabilities such as suspend, resume, reset, and migration of virtual devices. Depending on the specific VMM implementation, VDCM may be instantiated as a separate user or kernel module or may be packaged as part of the host driver. Chapter 4 further describes the high-level software architecture.

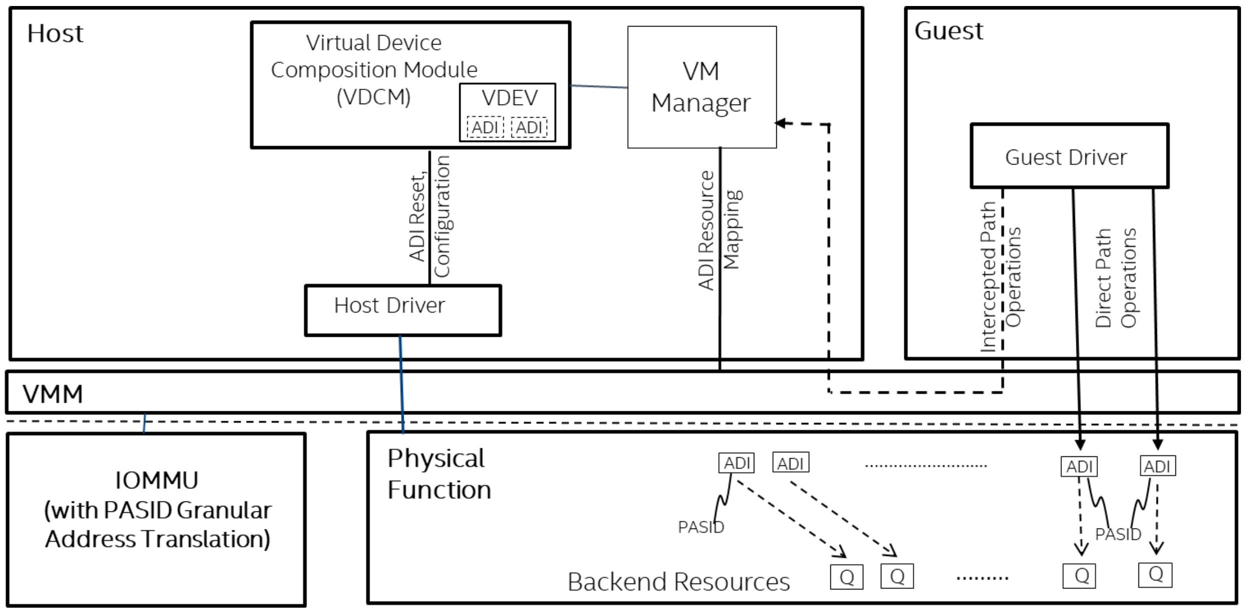

Figure 1-3: Intel® Scalable I/O Virtualization Software Architecture

2 Device Support

This chapter describes the key set of requirements and capabilities for an endpoint device to support Intel Scalable IOV. The requirements apply to both Root-Complex Integrated Endpoint and PCI Express Endpoint devices.

As described in previous chapter, the construct for fine-grained sharing on endpoint devices is Assignable Device Interfaces (ADIs). ADIs form the unit of assignment and isolation for devices and are composed by software to form virtual devices. This chapter describes the requirements for endpoint devices for enumeration, allocation, configuration, management and isolation of ADIs.

2.1 Organizing Device Resources for ADIs

Resources on a device associated with work submission, execution, and completion operations are referred to as device backend resources. These may include command/status registers, on-device queues, references to in-memory queues, local memory on the device, or any other device-specific internal constructs.

An Assignable Device Interface (ADI) refers to the set of device backend resources that are allocated, configured and organized as an isolated unit, forming the unit of device sharing. The type and number of backend resources grouped to compose an ADI is device specific. An ADI may be associated with a device context, rather than with specific device resources. ADIs using shared work queues (SWQ) for work submission may have little to no state or resources associated with them on the device. (See Section 2.3 for more details.)

Figure 2-1 illustrates a logical view of ADIs with varying number of device backend resources, and virtualization software composing virtual device instances with one or more ADIs. ADI 1 and ADI 2 are composed of single backend resource 1 and 2 respectively, whereas ADI 3 is composed of multiple backend resources 3, 4, and 5. Virtual device 1 instance (VDEV1) is composed of two ADIs (ADI 1 and ADI 2) whereas VDEV 2 and VDEV k instances are composed of single ADIs (ADI 3 and ADI m respectively). Due to different ADI composition of ADI 3 and ADI m, VDEV 2 gets 3 backend resources whereas VDEV k gets one backend resource.

IMPLEMENTATION NOTE

Example ADIs for various types of devices

- Network controller: Transmit/receive queues associated with a virtual switch interface.
- Storage controller: Command and completion queues associated with a storage namespace.
- GPU: Dynamically created graphics or compute context.
- FPGA: Accelerator Functional Unit.
- Multi-context FPGA: Dynamically created execution context.
- RDMA device: Queue pair.

2.2 Identifying ADI Upstream Requests

Upstream memory requests from all ADIs are tagged with the Requester ID of the device function hosting the ADIs. Requests from different ADIs of the device function are distinguished using a Process Address Space Identifier (PASID). The PCI Express specification defines the Process Address Space Identifier (PASID) in the PASID TLP Prefix of memory transactions. In conjunction with the Requester ID, the PASID identifies the address space associated with the request.

Endpoint devices must support the PASID capability as defined by the PCI Express specification and comply with all associated requirements. Before enabling ADIs, the PASID capability of the device must be enabled. Before an ADI is activated, it must be configured with a PASID value. All upstream memory requests and ATS Translation Requests generated by any ADI must be tagged with the assigned PASID value using the PASID TLP Prefix. ATS Translated Requests by an ADI may be generated without PASID or with the assigned PASID. Refer to the PCI Express specification and related ECNs for details on usage of the PASID TLP Prefix on Translated Requests. Interrupts generated by ADIs are not tagged with the PASID TLP Prefix. Refer to Section 2.5 for identifying ADI interrupts.

Each ADI must have a primary PASID associated with it, which is used for direct-path operations. ADIs may have optional secondary PASIDs whose usage is device dependent. For example, an ADI may be configured to access meta-data, commands, and completions with a secondary PASID that represents a restricted control domain, while data accesses are associated with the primary PASID corresponding to the domain to which the ADI is assigned.

When assigning an ADI to an address domain (e.g., VM, container, or process), the ADI is configured with the unique PASID of the address domain and its memory requests are tagged with the PASID value in the PASID TLP Prefix. If multiple ADIs are assigned to the same address domain, they may be assigned the same PASID. If ADIs belonging to a VDEV assigned to a VM are further mapped to secondary address domains (e.g., application processes) within the VM, each such ADI is assigned a unique PASID corresponding to the secondary address domain. This enables usages such as Shared Virtual Memory within a VM, where a guest application process is assigned an ADI and requests from the ADI are subject to nested translation (GVA to GPA to HPA) by the DMA remapping hardware, which is similar to the nested address translation for CPU accesses by a guest application.

.

2.3 ADIs Using Shared Work Queues

A Shared Work Queue (SWQ) is a special work submission interface that can be used simultaneously by multiple independent software entities such as applications, containers, or VMs. Work submission to SWQ makes use of PCI Express Deferrable Memory Write (DMWr) requests. DMWr provides a mechanism for PCIe Endpoints and hosts to choose to carry out or defer incoming DMWr Requests. DMWr requests destined to an SWQ may be accepted or deferred (i.e., rejected for re-submission) by the Endpoint device. Refer to the PCI Express Deferrable Memory Write (DMWr) and Device 3 Extended Capability ECN for details.

Software submits work to an SWQ on Intel® 64 processors using the ENQCMD (Enqueue Command) or ENQCMDS (Enqueue Command as Supervisor) instructions. ENQCMD/S instructions carry a PASID value in the work descriptor to identify the software entity that is submitting the work descriptor. ENQCMD/S instructions return a Success or Retry (Deferred) indication. Success indicates the work was accepted into the SWQ, while Retry indicates it was not accepted due to SWQ capacity, QoS, or other reasons. On a Retry status, the work submitter may back-off and retry later. Refer to the Intel® Architecture Instruction Set Extensions Programming Reference for more details.

Because work submissions to an SWQ contain a PASID value in the work descriptor, the PASID may not need to be preconfigured in the device for ADIs that use an SWQ.

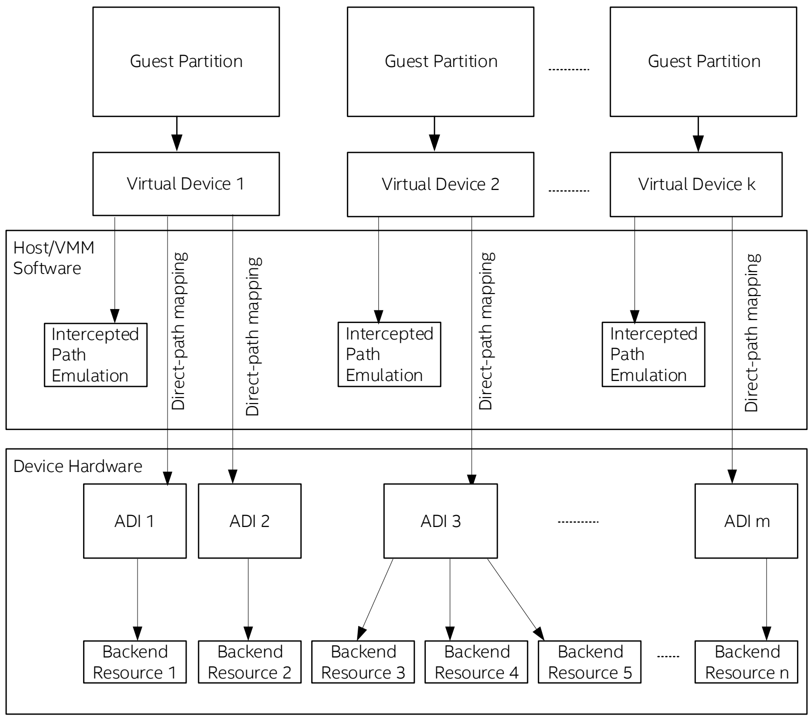

Figure 2-1: Composability of Virtual Devices from ADIs

2.4 ADI Memory Mapped Registers

Each ADI's memory mapped I/O (MMIO) registers are contained within one or more of address ranges mapped by the PCI Express Base Address Registers (BARs) of the device. Each ADI's MMIO registers must be isolated in one or more system page size aligned regions. These may be contiguous or scattered regions within the device's MMIO space. The number and location of system page size regions associated with specific ADIs is device-specific. The system page sizes supported by the device is reported via the Intel Scalable IOV DVSEC Capability described in Section 2.8. For Intel® 64 architecture platforms, the system page size is 4KB.

Devices must partition their ADI MMIO registers into two categories: (a) MMIO registers accessed for direct-path operations; and (b) MMIO registers accessed for intercepted-path operations. The definition of what operations are designated as intercepted path versus direct path is device-specific. The device must segregate registers in these two categories into distinct system page size regions, to allow the VMM to directly map direct-path operations to one or more constituent ADIs while emulating intercepted-path operations in the VDCM.

Devices should implement prefetchable 64-bit BARs so that address space above 4GB can be used for scaling ADI MMIO resources.

2.5 ADI Interrupts

ADIs capable of generating interrupts must generate only message signaled interrupts, not legacy interrupts. ADIs must not share interrupt resources/messages with the base function or with another ADI. ADIs sharing the same SWQ for work submission must also support separate interrupt messages. Each ADI may support zero or more interrupt messages. For example, an ADI composed of N queues may support N interrupt messages to distinguish work arrivals or completions for each queue.

2.5.1 ADI Interrupt Message Storage (IMS)

Device implementations may support a large number of ADIs, and each ADI may use multiple interrupt messages. To support the large interrupt message storage for all the ADIs, a device-specific construct called Interrupt Message Storage (IMS) is defined. IMS enables devices to store the interrupt messages for ADIs in a device-specific optimized manner without the scalability restrictions of the PCI Express defined MSI-X capability. Support for IMS is indicated by the IMS Support field in the Intel Scalable IOV DVSEC Capability described in Section 2.8.[1]

IMS entries store and generate interrupts using the same interrupt message address and data values as PCI Express MSI-X table entries. Interrupt messages stored in IMS are composed of a DWORD size data payload and a 64-bit address. IMS implementations must allow for dynamic allocation and release of IMS entries as ADIs are dynamically instantiated/revoked to create/destroy virtual devices. IMS must support per-message

[1] IMS may be supported by devices independent of Intel Scalable IOV. Such usages are outside the scope of this document.

masking and pending bit status, similar to the per-vector mask and pending bit array in the PCI Express MSI-X capability.

The size, location, and storage format for IMS is device specific. For example, a device may implement IMS as on-device storage. A device that maintains ADI contexts in memory may implement IMS as part of the context privileged state. In either approach, the device may implement IMS as either one unified storage structure or as de-centralized per-ADI storage structures. If IMS is implemented in host memory, the device may cache IMS entries within the device. If the device implements IMS caching, it must also implement device specific interfaces for the device driver to invalidate the IMS cache entries. Programming of IMS is done by the host driver.

Devices should support IMS for better scalability and dynamic allocation of ADI interrupts. Interrupts generated by ADIs should use the IMS. Interrupts generated by the base function should use the MSI or MSI-X capability. With appropriate device and system software support, ADI interrupts may use MSI-X and base-function interrupts may use IMS.

2.5.2 ADI Interrupt Isolation

IMS is managed by host driver software and is not accessible directly from guest or user-mode drivers. Within the device, IMS storage is not accessible from the ADIs. ADIs can request interrupt generation only through the device's 'Interrupt Message Generation Logic', which allows an ADI to only generate interrupt messages that are associated with that specific ADI. These restrictions ensure that the host driver has complete control over which interrupt messages can be generated by each ADI.

On Intel 64 architecture platforms, message signaled interrupts are issued as DWORD size untranslated memory writes without a PASID TLP Prefix, to address range 0xFEExxxxx. Since all memory requests generated by ADIs include a PASID TLP Prefix, it is not possible for an ADI to generate a DMA write that would be interpreted by the platform as an interrupt message.

2.6 ADI Isolation, Access Control, and QoS

ADIs are isolated from each other and from the base function. Operations or functioning of one ADI must not functionally affect other ADIs or the base function. Every memory request (except ATS translated requests) from an ADI must include the ADI's assigned PASID value in the PASID TLP prefix. The PASID identity for an ADI is modified only by privileged software such as the host driver.

The PCI Express Access Control Service capability is not applicable for isolation between ADIs. Devices must not allow peer-to-peer access between ADIs or between ADIs and the base function (either internal to the device or at I/O fabric egress). Independent of Intel Scalable IOV support, a device may support ACS guidelines for isolation across endpoint functions or devices, per the PCI Express specification.

Although ADIs are functionally isolated, they may have performance effects on each other and on the base function. Devices may define Quality of Service (QoS) controls for ADIs to manage these effects. The definition of QoS for ADIs is device specific and is outside the scope of this specification.

ADI specific errors are errors that can be attributed to a particular ADI, such as malformed commands or address translation errors. Such errors must not impact functioning of other ADIs or the base function.

Handling of ADI specific errors can be implemented in device-specific ways; such errors should be reported directly to the guest that the ADI is assigned to, when possible.

2.7 ADI Reset

Each ADI must be independently resettable without affecting the operation of other ADIs. Reset of an ADI is performed through device-specific interfaces. ADI reset causes the device to abort (discard) all in flight and accepted operations to the ADI by specific domain and reset ADI specific configuration on the device to a known state. For ADIs using shared work queue (SWQ) for work submission, ADI reset must not cause reset of the SWQ since that may affect other ADIs sharing the SWQ.

A VDEV may expose a virtual FLR capability that may be emulated by the VDCM by requesting the device to perform ADI resets for each of the constituent ADIs of the virtual device.

An ADI reset must ensure that the reset is not reported as complete until all of the following conditions are satisfied:

- All DMA write operations by the ADI are drained or aborted
- All DMA read operations by the ADI have completed or aborted
- All interrupts from the ADI have been generated
- If ADI is capable of Address Translation Service (ATS), all ATS requests by the ADI have completed or aborted, and
- If ADI is capable of Page Request Service (PRS), no more page requests will be generated by the ADI. Additionally, either page responses have been received for all page requests generated by the ADI or the ADI will discard page responses for any outstanding page requests by the ADI.

Devices supporting Intel Scalable IOV should support Function Level Reset (FLR) and may support additional device-specific global reset controls. A global reset operation or FLR resets all ADIs and returns the device to a state where no ADIs are configured. A device may also support a device-specific global reset that resets all ADIs but leaves them configured.

Devices may optionally support saving and restoring ADI state, to facilitate operations such as live migration and suspend/resume of virtual devices composed of ADIs. For example, to support ADI suspend, a device may implement an interface to drain (complete) all operations submitted to the ADI.

2.8 Capability Enumeration

An endpoint device function reports support for Intel Scalable IOV via a PCI Express Designated Vendor Specific Extended Capability (DVSEC). This capability may be used by system software and tools to detect endpoint devices supporting Intel Scalable IOV without a dependency on the host driver. The host driver is still responsible for enabling Intel Scalable IOV and related operations through system software specific interfaces. Figure 2-2 illustrates the Intel Scalable IOV DVSEC structure.

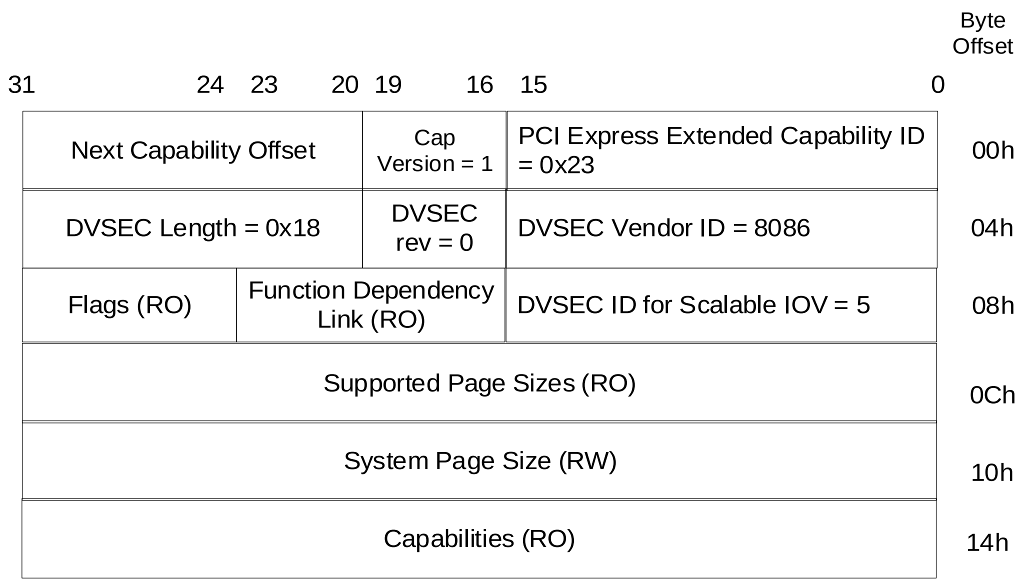

Figure 2-2: DVSEC for Intel® Scalable I/O Virtualization

The fields up to offset 0xa are the standard DVSEC capability header. Refer to the PCI Express DVSEC header for a detailed description of these fields. The remaining fields are described below.

Function Dependency Link (Offset = 0xA, Size = 1 Byte)		
Bit Location	Description	Attributes
7:0	The programming model for a device may have vendor-specific dependencies between sets of Functions. The Function Dependency Link field is used to describe these dependencies. This field describes dependencies between Functions. ADI dependencies are the same as the dependencies of the Functions that they are part of. If a Function is independent from other Functions of a device, this field shall contain its own Function Number. If a Function is dependent on other Functions of a Device, this field shall contain the Function Number of the next Function in the same Function Dependency List. The last Function in a Function Dependency List shall contain the Function Number of the first Function in the Function Dependency List (FDL). Dependencies between Functions are described by the Flags field at offset 0xB.	RO

Flags (Offset = 0xB, Size = 1 Byte)		
Bit Location	**Description**	**Attributes**
0	**H (homogeneous):** When the H flag is reported as set, it indicates that all Functions in the Function Dependency List (FDL) must be enabled for Intel Scalable IOV operation (in a device-specific manner). If some but not all of the Functions in the FDL are enabled for Intel Scalable IOV operation, the behavior is undefined. For example, one Function cannot be in Intel Scalable IOV operation mode and another in SR-IOV operation mode if this flag is reported as set. If the H flag is not Set, then different Functions in the FDL can be in different modes.	RO
7:1	**Reserved**	RO

Supported Page Sizes (Offset = 0xC, Size = 4 Bytes)		
Bit Location	**Description**	**Attributes**
31:0	This field indicates the page sizes supported. A page size of 2^{n+12} is supported if bit *n* is Set. For example, bit 0 indicates support for 4 KB pages. The page size indicates the minimum alignment requirement for ADI MMIO pages so that they can be independently assigned to different address domains. Support for 4 KB pages is required. Devices may support additional page sizes for compatibility with a variety of host platform architectures.	RO

System Page Size (Offset = 0x10, Size = 4 Bytes)		
Bit Location	Description	Attributes
31:0	This field defines the page size the system uses to map the ADIs' MMIO pages. Software must set the value of System Page Size to one of the page sizes set in the Supported Page Sizes field. As with Supported Page Sizes, if bit *n* is set in System Page Size, a page size of 2^{n+12} is used. For example, if bit 1 is set, the device uses an 8 KB page size. The behavior is undefined if System Page Size is zero, more than one bit is set, or a bit is set in System Page Size that is not set in Supported Page Sizes. When System Page Size is written, all ADI MMIO resources are aligned on system page size boundaries. System Page Size must be configured before setting the Memory Space Enable bit in the PCI command register. The behavior is undefined if System Page Size is modified after Memory Space Enable is set. Default value is 0000 0001h indicating a system page size of 4 KB.	RW

Capabilities (Offset = 0x14, Size = 4 Bytes)		
Bit Location	Description	Attributes
0	**IMS Support:** This bit indicates the support for Interrupt Message Storage (IMS) in the device. 0: IMS is not supported by the device. 1: IMS is supported by the device. If virtualization software supports IMS use only for ADIs and not by the base function, then when the base function is directly assigned to a domain, virtualization software may expose a virtual Intel Scalable IOV DVSEC Capability to the domain with the IMS support bit reported as 0.	RO
31:1	**Reserved**	RO

3 Platform Support

The following platform level capabilities are required to support Intel Scalable IOV:

- Support for the PCI Express PASID TLP Prefix in Root Ports and the Root Complex. Refer to the PCI Express Revision 4.0 specification or higher for details on PASID TLP Prefix support.
- PASID-granular address translation in Root Complex.
- Interrupt remapping support in Root Complex.
- PASID translation support in CPUs supporting PCIe Deferrable Memory Write (DMWr).

3.1 Address Space Isolation

The definition of the address space targeted by a PASID value is dependent on the Root Complex DMA remapping hardware capability and the programming of such hardware by software. Depending on the programming of the DMA remapping hardware, the address space targeted by a request with PASID can be a Host Physical Address (HPA), Host Virtual Address (HVA), Host I/O Virtual Address (HIOVA), Guest Physical Address (GPA), Guest Virtual Address (GVA), Guest I/O Virtual Address (GIOVA), etc. All of these address space types can co-exist on a system for different PASID values. ADIs from one or more devices may be configured to use these PASIDs. PASID granular address translation enables upstream requests from each ADI to have a unique address translation. ADIs can be used by VDCM to compose virtual devices that may be assigned to any type of address domain.

For example, Intel platforms support Intel Scalable IOV through the Scalable Mode Translation Support capability of the Intel® Virtualization Technology for Directed I/O, Rev 3.1 or higher. Figure 3-1 illustrates the high-level translation structure organization for Intel scalable mode address translation.

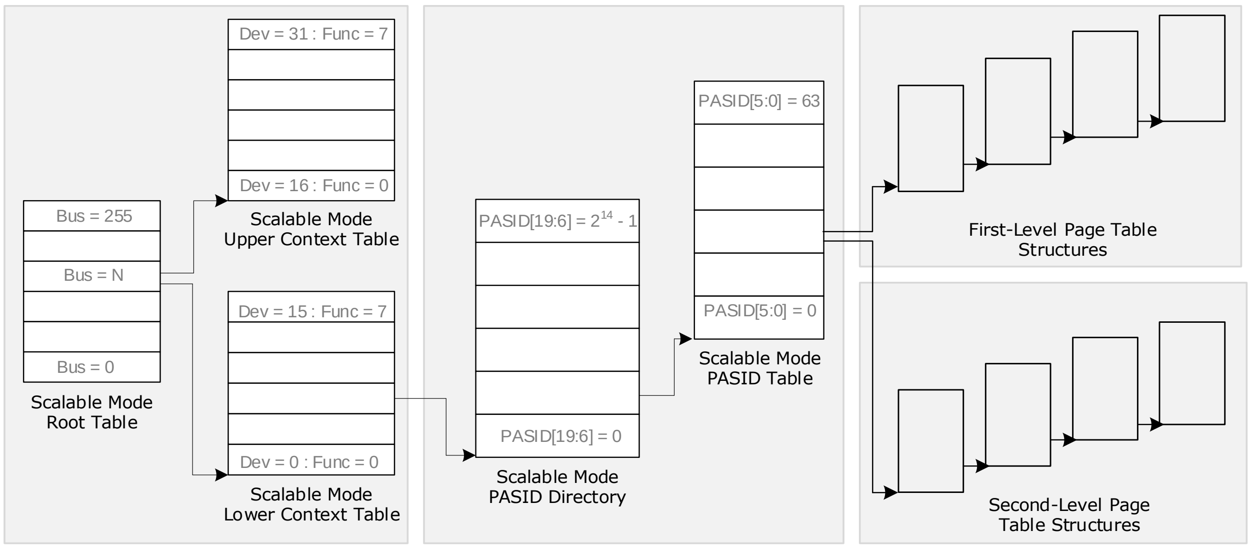

Figure 3-1: Scalable Mode DMA Remapping Architecture for Intel® Scalable I/O Virtualization

Scalable mode address translation involves a three-stage address translation:

1. The Requester ID (Bus/Device/Function number) in the upstream request is used to consult the Root and Context structures that specify translation behavior at Requester ID granularity. The context structure refers to PASID structures.
2. If the request includes a PASID TLP Prefix, the PASID value from the TLP prefix is used to consult the PASID structures that specify translation behavior at PASID granularity. If the request is without a PASID TLP Prefix, the PASID value programmed by software in the Context structure is used instead. For each PASID, the PASID structure entry can be programmed to specify first-level, second-level, pass-through or nested translation, along with references to first-level and second-level page-table structures.
3. Finally, the address in the request is subject to address translation using the first-level, second-level or both page-table structures, depending on the type of translation function.

3.2 Interrupt Isolation

For interrupt isolation across devices, the host platform should support interrupt remapping for all interrupt messages programmed in MSI, MSI-X, or IMS on the device.

The host platform should support direct delivery of virtual interrupts to VMs without hypervisor processing overheads. This also enables virtual interrupts to operate in guest interrupt vector space without consuming host processor interrupt vectors.

Refer to the Intel® VT-d specification for details on interrupt remapping and posted interrupts.

3.3 PASID translation

To support PASID isolation for Shared Work Queues used by VMs, the CPU must provide a way for the PASID to be communicated to the device in the DMWr transaction. On Intel® CPUs, the CPU provides a PASID translation table in the vCPUs' virtual machine control structures. During ENQCMD/ENQCMDS instruction execution in a VM, the PASID translation table is used by the CPU to replace the guest PASID in the work descriptor with a host PASID before the descriptor is sent to the device. Refer to the ENQCMD/ENQCMDS instruction description in the Intel® Architecture Instruction Set Extensions Programming Reference for more details.

4 Reference Software Model

This chapter describes an example software architecture in which Intel Scalable IOV is enabled through VMM software as shown in Figure 1-3. This chapter is not intended to be prescriptive, and instead covers an example description of system software and device-specific software roles and interactions to compose hardware-assisted virtual devices and manage device operation. Specific OS or VMM implementations may choose other methods to enable Intel Scalable IOV.

The software architecture described in this chapter focuses on I/O virtualization for virtual machines and machine containers. However, the principles can be applied to other domains such as I/O sharing across bare-metal containers or application processes. Figure 1-3 illustrates the high-level software architecture. The logical components of the reference software architecture are described below.

4.1 Host Driver

The host driver for an Intel Scalable IOV capable device is conceptually equivalent to an SR-IOV PF driver. The host driver is loaded and executed as part of the host OS or hypervisor software. The host driver reports support for Intel Scalable IOV to system software through the driver interface. In addition to the usual roles of a device driver, the host driver implements software interfaces as defined by the host OS or hypervisor infrastructure to support enumeration, configuration, instantiation, and management of ADIs. The host driver is responsible for configuring each ADI, including aspects such as its PASID identity, Interrupt Message Storage entries, MMIO register resources for direct-path access to the ADI, and any device-specific resources.

Table 4-1 illustrates a high-level set of operations that the host driver supports for managing ADIs. These operations are invoked through suitable software interfaces defined by specific system software implementations.

Description
Intel Scalable IOV capability reporting
Enumeration of types and maximum number of ADIs/VDEVs
Enumeration of resource requirements for each ADI type
Enumeration and setting of generational compatibility for ADIs
Allocation, configuration, reset, drain, abort, release of ADI and its constituent resources
Setting and managing PASID identity of ADIs
Managing device-specific Interrupt Message Storage for ADIs
Enabling guest to host communication channel (if supported)
Configuring device specific QoS properties of ADIs
Suspending/saving state of ADIs, and restoring/resuming state of ADIs

Table 4-1: Host Driver Interfaces for Intel® Scalable I/O Virtualization

4.2 Virtual Device Composition Module

The Virtual Device Composition Module (VDCM) is a device-specific component that is responsible for composing virtual device instances using one or more ADIs allocated by the host driver. The VDCM implements software-based virtualization of intercepted-path operations and arranges for direct-path operations to be submitted directly to the backing ADIs. Each OS or VMM implementation may require the VDCM to be implemented and packaged by device vendors in a specific way. For example, in some OS or hypervisor implementations, the VDCM may be packaged as user-space modules or libraries that are installed as part of the host driver. In other implementations, the VDCM may be a kernel module. If implemented as a library, the VDCM may be statically or dynamically linked with the hypervisor-specific virtual machine resource manager responsible for creating and managing VM resources. If implemented in the host kernel, the VDCM can be part of the host driver.

4.3 Guest Driver

The guest driver for an Intel Scalable IOV capable device is conceptually equivalent to an SR-IOV VF driver. The guest driver manages the VDEV instances composed by the VDCM. Direct-path accesses by the guest driver are issued directly to the ADIs behind the VDEV, while intercepted-path accesses are intercepted and virtualized by the VDCM. The guest and host drivers can be implemented as a unified driver that supports both host and guest functionality or as two separate drivers. For existing SR-IOV devices, if the VDEV can be composed to behave like an existing VF, the Intel Scalable IOV guest driver can be same as the SR-IOV VF driver.

4.4 Virtual Device

A virtual device (VDEV) is the abstraction through which a shared physical device is exposed to guest software. A VDEV is typically exposed to a guest OS as a virtual PCI Express device. A VDEV has virtual resources such as virtual Requester ID, virtual configuration space registers, virtual memory BARs, virtual MSI-X table, etc. Each VDEV may be backed by one or more ADIs. The ADIs backing a VDEV typically belong to the same physical function, but implementations are possible where they are allocated across multiple functions (for example to support device fault tolerance or load balancing).

A device may support multiple types of ADIs, both in terms of number of backend resources (see Figure 2-1) and in terms of functionality. Similarly, a VDCM may support more than one type of VDEV composition, with respect to the number of backing ADIs, functionality of ADIs, etc., enabling the virtual machine resource manager to request different types of VDEV instances for assigning to virtual machines. The VDCM uses the host OS and VMM defined interfaces to allocate and configure resources needed to compose a VDEV.

A VDEV may be composed of a static number of ADIs that are pre-allocated at the time of VDEV instantiation or composed dynamically by the VDCM in response to guest driver requests to allocate/free resources. An example of statically allocated ADIs is a virtual NIC with a fixed number of RX/TX queues. An example of dynamically allocated ADIs is a virtual accelerator device, where context allocation requests are virtualized by the VDCM to dynamically create accelerator contexts as ADIs.

4.4.1 Virtual Device Memory Mapped Registers Composition

As part of composing a VDEV instance, the VDCM implements the behavior of the virtual MMIO regions in guest physical address space. A VDEV's MMIO registers may be implemented using any of the following methods. Each system page size region of the VDEV MMIO space may use a different method.

- Direct Mapped to ADI MMIO: Direct-path registers of the VDEV virtual MMIO space are mapped directly to the physical MMIO space of the device. The VDCM requests the hypervisor to set up GPA to HPA mappings for these regions in the CPU virtualization page tables, enabling direct access by the guest driver to the ADI.
- VDEV MMIO Intercepted and Emulated by VDCM: Intercepted-path registers of the VDEV are virtualized by the VDCM by requesting the hypervisor to not map these MMIO regions in the host processor virtualization page-tables, thus forcing host intercepts when the guest driver accesses these registers. The intercepted accesses are provided to the VDCM to virtualize, either by itself or through interactions with the host driver.

  VDEV registers that are read frequently and have no read side-effects, but require VDCM intercept and emulation on write accesses, may be mapped as read-only to backing memory pages provided by VCDM. This supports high performance read accesses to these registers along with virtualizing their write side-effects by intercepting on guest write accesses. 'Write intercept only' registers must be contained in separate system page size regions from the 'read-write intercept' registers on the VDEV MMIO layout.
- VDEV MMIO Mapped to Memory: VDEV registers that have no read or write side effects may be mapped to memory with read and write access. These registers may contain parameters or data for a subsequent operation performed by writing to an intercepted register. Device implementations may also use this approach to define virtual registers for VDEV-specific communication channel between the guest driver and the VDCM. The guest driver writes data to the memory backed virtual registers without host intercepts, followed by a mailbox register access that is intercepted by the VDCM. This optimization reduces host intercept and instruction emulation cost for passing data between guest and host. Such approach may enable guest drivers to implement such channels with VDCM more generally than hardware-based communication doorbells (as often implemented between SR-IOV VFs and PF) and without depending on guest OS or hypervisor specific para-virtualized software interfaces.

4.4.2 Virtual Device Interrupts

A VDEV may expose a virtual MSI or virtual MSI-X capability that is emulated by the VDCM. The guest driver requests VDEV interrupt resources normally through guest OS interfaces, and the guest OS may service this by programming one or more Interrupt Messages through the virtual MSI or virtual MSI-X capability of the VDEV.

Two sources of interrupts may generate VDEV interrupts to the guest driver. One source is the VDCM software itself generating virtual interrupts on behalf of the VDEV. These are purely software generated interrupts arising from the intercepted-path operations of the VDEV being emulated by the VDCM. The other source of interrupts is the ADI instances on the device that are used to support direct-path operations of the VDEV.

When the guest OS programs the virtual MSI or MSI-X register, the operation is intercepted and virtualized by the VDCM. For intercepted-path virtual interrupts, the VDCM requests virtual interrupt injection to the guest through the VMM software interfaces. For direct-path interrupts from ADIs, the VDCM invokes the host driver to allocate and configure required interrupt message address and data in the IMS.

4.4.3 Communication Channel Between Guest Driver and VDCM

For device-specific usages and reasons, devices may choose to build communication channels between the guest driver and the VDCM. These communication channels can be built in a manner that is independent of specific guest and host system software, using one of the following methods.

- Software emulated communication channel: This type of channel is implemented by the VDCM by setting up one or more system page size regions in VDEV MMIO space as fully memory-backed, to be used to share data between the guest and the host. The VDCM also sets up an intercepted-path register in VDEV MMIO space to be used by the guest to signal an action to the host. A virtual interrupt may be used by the VDCM to signal the guest about completion of asynchronous communication channel actions.
- Hardware mailbox-based communication channel: If the communication between the guest driver and the host driver is frequent and the software emulation-based communication channel overhead is significant, the device may implement communication channels based on hardware mailboxes. This is similar to communication channels between SR-IOV VFs and PF in some existing designs.

4.4.4 ADIs Supporting Shared Virtual Memory

Shared Virtual Memory (SVM) is a usage where a device operates in the CPU virtual address space of the application accessing the device. SVM usage is enabled by system software programming the DMA remapping hardware to reference the CPU page tables. Devices supporting SVM do not require pages that are accessed by the device to be pinned, but instead use the PCI Express Address Translation Services and Page Request Services capabilities to implement recoverable device page faults. Refer to the PCI Express specification for details on ATS and PRS capabilities.

Like Intel Scalable IOV, devices supporting SVM use PASIDs to distinguish different application virtual address spaces. A device that supports both SVM and Intel Scalable IOV will support SVM both for ADIs assigned to host applications and for ADIs assigned to guest applications. The distinction between host and guest SVM usages is transparent to the device. The only difference is in the address translation function programming of the DMA remapping hardware for each PASID. The address translation function programmed for a PASID representing host SVM usage refers to the CPU virtual address to physical address translation, while the address translation function programmed for a PASID representing guest SVM usage refers to nested address translation (guest virtual address to guest physical address and then to host physical address).