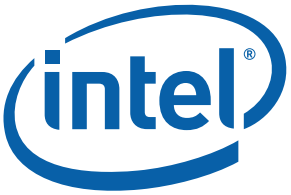


Intel® Scalable I/O Virtualization

Technical Specification

June 2018

Reference Number: 337679-001, Revision: 1.0

Contents

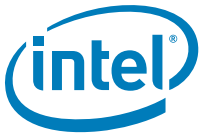
Figures

Tables

Revision History

Document Number	Revision Number	Description	Date
337679-001	1.0	• Initial Release	June 2018

§



CHAPTER 1
INTRODUCTION

The introduction of the Single Root I/O Virtualization (SR-IOV) specification in 2007 by PCI-SIG* was a notable advancement toward hardware-assisted high performance I/O virtualization and sharing for PCI Express* devices. Since then, the compute landscape has evolved beyond deploying virtual machines for server consolidation to hyper-scale data centers which need to seamlessly add resources and dynamic provision containers. The new computing environment demands increased scalability and flexibility for I/O virtualization.

Intel® Scalable I/O Virtualization (Intel® Scalable IOV) lays out a scalable and flexible approach to hardware-assisted I/O virtualization targeting such hyper-scale usages. Intel Scalable IOV builds on an already existing set of 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 a high-level reference software architecture, and delineates the host platform and endpoint device capabilities required to support it.

1.1 DOCUMENT ORGANIZATION

A description of this document's content follows:

Chapter 1 — Gives an introduction to this document, describing organization and audience. Also provides definition of key terms and abbreviations used in document.

Chapter 2 — Provides an architectural overview of Intel Scalable IOV and its key components.

Chapter 3 — Specifies endpoint device blueprint and requirements to support Intel Scalable IOV.

Chapter 4 — Describes the host platform Root Complex (RC) support for Intel Scalable IOV.

Chapter 5 — Describes the reference software architecture to enable Intel Scalable IOV.

1.2 AUDIENCE

This document is aimed at endpoint device developers considering scalable hardware support for I/O virtualization and sharing. The document is also expected to be used by Operating System (OS) and Virtual Machine Monitor (VMM) developers who are enabling hardware-assisted I/O virtualization for emerging hyperscale usages.

1.3 REFERENCE DOCUMENTS

Intel® Virtualization Technology for Directed I/O Specification, Rev 3.0:

https://software.intel.com/en-us/download/intel-virtualization-technology-for-directed-io-architecture-specification

PCI Express* Base Specification, Revision 4.0, Version 1.0

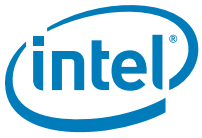
http://pcisig.com/specifications

1.4 DEFINITION OF TERMS

The following includes definition of terms and acronyms used in this document.

Table 1-1.Definitions

Acronym	Term	Definition
ADI	Assignable Device Interface	Assignable Device Interface is the unit of assignment for Intel® Scalable IOV capable device.
FLR	Function Level Reset	Function Level Reset as per PCI Express* Base Specification.
Guest Driver	Guest Driver	Device-specific software that runs inside a VM and manages virtual device operation.
Host Driver	Host Driver	Device-specific software that runs inside a host OS and manages physical device operation.
Host OS	Host Operating System	The privileged OS that works with the VMM to virtualize the platform.
IMS	Interrupt Message Storage	Device-specific interrupt message storage for ADIs on Intel Scalable IOV capable device.
Intel® Scalable IOV	Intel® Scalable I/O Virtualization	Software composable and scalable I/O virtualization as specified by this document.
MSI-X	Message Signaled Interrupts Extended	MSI-X capability as defined by PCI Express* Base Specification.
PF	Physical Function	PCI Express Physical Function as specified by SR-IOV.
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 PF or VF.
SR-IOV	Single Root I/O Virtualization	SR-IOV as specified by PCI Express Base Specification, Revision 4.0, Version 1.0
VDCM	Virtual Device Composition Module	Device-specific software component that runs on host responsible for composing a Virtual Device.
VDEV	Virtual Device	A virtual device composed by VDCM utilizing one or more ADIs.
VF	Virtual Function	PCI Express Virtual Function as specified by SR-IOV.
VMM	Virtual Machine Monitor	System software that creates and manages virtual machines. Also known as Hypervisor.



CHAPTER 2
INTEL® SCALABLE I/O VIRTUALIZATION OVERVIEW

This chapter provides the background on existing approaches to I/O virtualization and introduces the key concepts and components of Intel® Scalable I/O Virtualization and how it is envisioned to address the requirements not met by existing I/O virtualization and sharing approaches.

2.1 VIRTUALIZATION TODAY

Virtualization allows a system software called virtual machine monitor (VMM), also known as a hypervisor, to create multiple isolated execution environments called virtual machines (VMs) in which operating systems (OSs) and applications can run. Virtualization is extensively used in modern enterprise and cloud data centers as a mechanism to consolidate multiple workloads onto a single physical machine while still keeping them isolated from each other. Besides VMs, containers provide 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 and can be deployed in much higher density than VMs, potentially increasing the number of container instances on a system by an order of magnitude.

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) for IA-32 Intel® Architecture (Intel® VT-x) defines the Intel® processor hardware capabilities to reduce overheads for processor and memory virtualization. Intel® Virtualization Technology (Intel® VT) 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 hypervisor exposes a virtual device, such as a Network Interface Controller (NIC), to a VM. A software device model in the hypervisor or host OS emulates the behavior of the virtual device. The device model translates from virtual device commands to physical device commands before forwarding to the 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 device accesses in software can be too complex for programmable devices such as Graphics Processing Units (GPUs) and Field-Programmable Gate Arrays (FPGAs) because these devices perform a variety of functions versus only a fixed set of functions. Variants of software-based I/O virtualization such as 'device paravirtualization' and 'mediated pass-through' allow to mitigate some of the performance and complexity disadvantages with device emulation.

To avoid the software-based I/O virtualization overheads, hypervisors may make use of platform support for DMA and interrupt remapping capability (such as Intel® VT-d) to support 'direct device assignment' allowing guest software to directly access the assigned device (Physical Function). This direct device assignment provides the best I/O virtualization performance since the hypervisor is no longer in the way 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 may support one or more Physical

Functions (PFs), each of which may support multiple Virtual Functions (VFs). The PF functions as the resource management entity for the device and is managed by the PF driver in the host OS. Each VF can be assigned to a VM or container for direct access. SR-IOV is supported by multiple 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 emerging accelerators.

2.2 SCALABLE I/O VIRTUALIZATION

As hyper-scale models proliferate along with increasing number of processing elements on modern processors, a typical standard high-volume server is used to host an order of magnitude more number of bare-metal or machine containers than traditional VMs. Many of these usages such as network function virtualization (NFV) or heterogeneous computing with accelerators require high performance hardware-assisted I/O virtualization. These dynamically provisioned high-density usages (i.e., of the order of 1000 domains) demand more scalable and fine-grained I/O virtualization solutions than required by traditional virtualization usages supported by SR-IOV capable devices.

Intel Scalable IOV is a new 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. Depending on the usage model, the isolated domains may be traditional VMs, machine containers, bare-metal containers, or application processes. This document primarily refers to isolated domains as VMs, but the general principles apply broadly to other domain abstractions such as containers.

Figure 2-1 illustrates the high level difference between SR-IOV and Intel Scalable IOV approaches. Unlike the coarse-grained device partitioning approach adopted by SR-IOV to create multiple VFs on a PF, Intel Scalable IOV enables software to flexibly compose virtual devices utilizing the hardware-assists for device sharing at finer granularity. Frequent (i.e., performance critical) operations on the composed virtual device are mapped directly to the underlying device hardware, while infrequent operations are emulated through device-specific composition software in the host. This is different from the existing architecture for SR-IOV devices, where only the device-agnostic PCI Express* architectural resources (such as configuration space registers and MSI-X capability registers) of the virtual device are virtualized in software, and the rest of the virtual device resources (including all other MMIO) are mapped directly to the underlying VF hardware resources.

Figure 2-1. Approaches to Hardware-assisted I/O Virtualization

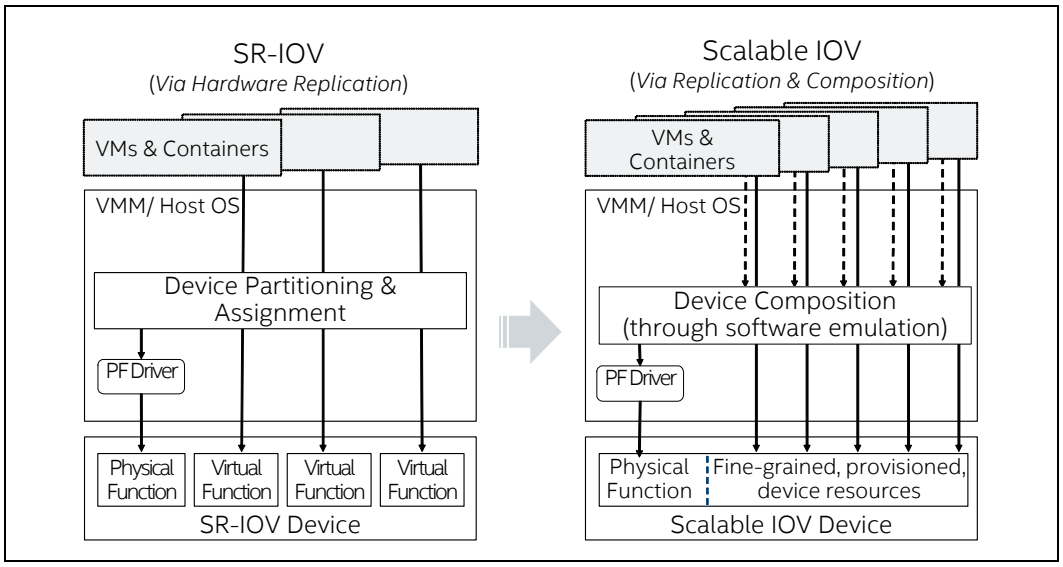
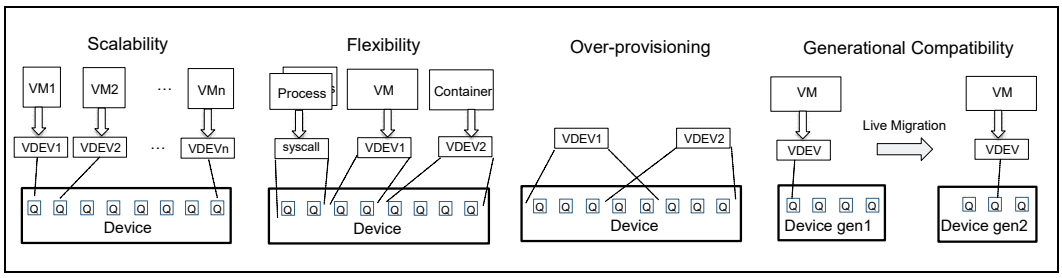


Figure 2-2 illustrates some of the main benefits of Intel Scalable IOV. Fine-grained provisioning of device resources (shown as Q) to VMs along with software emulation of infrequent device accesses enables devices to increase sharing scalability at lower hardware cost and complexity. It provides system software the flexibility to share device resources with different address domains using different abstractions (e.g., application processes to access through system calls and VMs/Containers to access through virtual device interfaces). Through software-controlled dynamic mapping of VDEVs to device resources, it also enables VMMs to over-provision device resources to VMs.

This approach also enables VMMs to easily maintain generational compatibility in a data center. For example, in a data center with physical machines containing different generations (versions) of the same I/O device, a VMM can use the software emulation to virtualize VDEV's MMIO based capability registers to present the same VDEV capabilities irrespective of the different generations of physical I/O device. This is to ensure that the same guest OS image with a VDEV driver can be deployed or migrated to any of the physical machines.

Figure 2-2. Main Benefits of Intel® Scalable I/O Virtualization



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

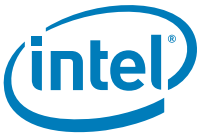
- Endpoint device support: PCI Express endpoint device requirements and capabilities to support Intel Scalable IOV. This is covered in Chapter 3.
- Platform support: Host platform (Root Complex) requirements including enhancements to DMA remapping hardware to support Intel Scalable IOV. This is covered in Chapter 4. These requirements are implemented on Intel platforms as part of Intel Virtualization Technology for Directed I/O, Rev 3.0 or higher.
- Software support: Reference software architecture envisioned for enabling Intel Scalable IOV, including host system software (OS and/or VMM) enabling infrastructure and endpoint device specific software components such as host driver, guest driver, and a virtual device composition module (VDCM). This is covered in Chapter 5.

PCI Express endpoint devices may support requirements to operate with Intel Scalable IOV independent of its support for SR-IOV. This enables device implementations that already support SR-IOV to maintain it for backwards compatibility while adding the capabilities to support Intel Scalable IOV.

This document expects an endpoint physical function capable of both SR-IOV and Intel Scalable-IOV to be enabled to operate in one mode or other, but not concurrently. While it may be possible for endpoint device implementations and supporting software architecture to support SR-IOV and Intel Scalable IOV operation concurrently (or support Intel Scalable-IOV operation in a hierarchical manner over SR-IOV VFs), these modes of operation are beyond the scope of this document.

2.2.1 Separation of Fast-path and Slow-path Operations

PCI Express SR-IOV architecture follows a near complete functional replication of the Physical Function (PF) hardware for its Virtual Functions (VFs). This is realized by most SR-IOV device implementations by replicating most of the PF's hardware/software interface for each of its VFs, including resources such as memory mapped resources, MSI-X storage and capabilities such as Function Level Reset (FLR).



Such functional replication approach can add to device complexity and impose limitations to scale to large number of VFs.

Hardware-software interface for most I/O controller implementations may be categorized as (a) slow-path control/configuration operations that are less frequent and has least impact on overall device performance; and (b) fast-path command/completion operations that are frequent and has higher impact on the overall device performance. Such distinction of slow-path versus fast-path operations are already practiced by many high performance I/O devices supporting direct user-mode access. Intel Scalable IOV extends such device designs to define a software composable approach to I/O virtualization and sharing.

Intel Scalable IOV requires endpoint devices to organize its hardware/software interface into fast-path (frequent) and slow-path (infrequent) accesses. Which operations and accesses are distinguished as slow-path versus fast-path is controlled by device implementation. Slow-path accesses typically include initialization, control, configuration, management, error processing, and reset. Fast-path accesses typically include data-path operations involving work submission and work completion processing. With this organization, slow-path accesses to the virtual device from the guest VM are trapped and emulated by device-specific host software while fast-path accesses are directly mapped on to the physical device. Such approach enables simplified device designs (compared to SR-IOV full functional replication), without compromising I/O virtualization scalability or performance. Additionally, the hybrid approach provides increased flexibility for software to compose virtual devices through fine-grained provisioning of device resources.

2.2.2 Assignable Device Interfaces

Modern high performance I/O devices support a large number of command/completion interfaces for efficient multiplexing/de-multiplexing of I/O and in some usages to support user-mode I/O. A few examples of such devices are:

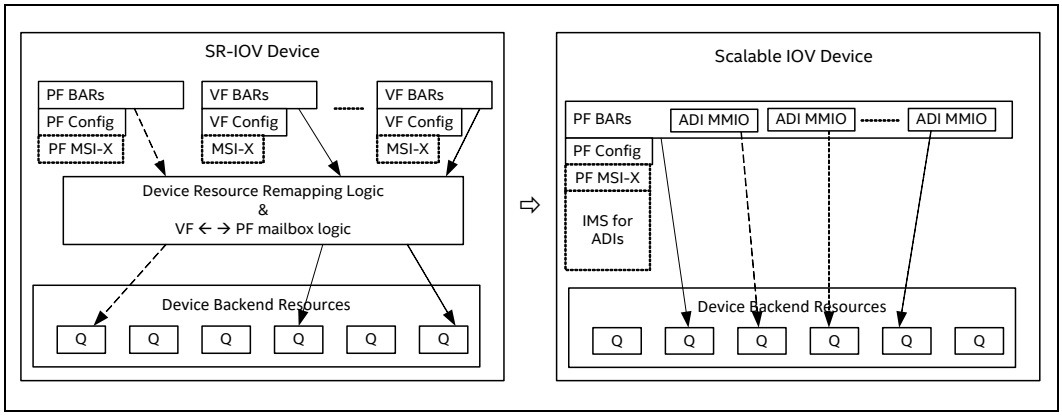
- High-bandwidth network controllers supporting thousands of transmit/ receive (TX/ RX) queues across a large number of Virtual Switch Interfaces (VSIs).
- Storage controllers such as NVM Express devices supporting many command and completion queue pair constructs.
- Accelerator devices such as GPUs supporting a large number of graphics and/or compute contexts.
- Reconfigurable FPGA devices with Accelerator Functional Units (AFUs) supporting a large number of execution contexts.
- RDMA capable devices supporting thousands of Queue Pair interfaces.

Intel Scalable IOV takes advantage of multi-queue/multi-context capable high performance I/O device designs and defines an approach to share these devices at a finer granularity (queues, queue bundles, contexts, etc.) than SR-IOV VF granularity. To achieve this finer-grained sharing, Intel Scalable IOV architecture defines the granularity of sharing of a device as an 'Assignable Device Interface' (ADI) on the device. Each ADI instance on the device encompasses the set of resources on the device that are allocated by software to support the fast-path operations for a virtual device.

Conceptually, ADI is similar to SR-IOV VF, except it is finer-grained and maps to the fast-path operations for a virtual device. Unlike VFs, all ADIs on a Physical Function (PF) share the Requester-ID (Bus/ Device/Function number) of the PF, have no PCI configuration space registers, share the same Base Address Register (BAR) resources of the PF (i.e., no VFBARs), and do not require replicated MSI-X storage. Instead of MSI-X table storage for each ADI, PF implements a device specific Interrupt Message Storage (IMS). IMS is similar to MSI-X table storage in purpose but is not architectural and instead implemented in a device specific manner for maximum flexibility. Additionally, unlike some SR-IOV devices which implement VF←→PF communication channel and 'resource remapping logic' (See Section 3.1) on the device, ADIs may use slow-path emulation to provide such functionality. ADI's memory-mapped register space is laid out such that fast-path registers are in separate system page size regions than the slow-path registers.

The host driver for Intel Scalable IOV capable device defines the collection of device back-end resources that are grouped to form an ADI. Figure 2-3 illustrates the key differences between SR-IOV capable and Intel Scalable IOV capable endpoint devices.

Figure 2-3. Differences between Intel® Scalable IOV and SR-IOV Based Device Sharing



This device-specific and light-weight nature of ADIs, along with the flexibility to emulate portions of the virtual device functionality in device-specific host software, enables device hardware implementations to compose a large number of virtual devices for scalable sharing at lower device cost and complexity compared to equivalent scaling of SR-IOV VFs.

2.2.3 Platform Scalability Using PASIDs

With SR-IOV, each VF in a device is identified by a PCI Express Requester-ID (RID), allowing DMA remapping hardware support in Root Complex (such as Intel® VT-d) to apply unique address translation functions for upstream requests from the VF. RIDs are also used for routing transactions such as read completions for PCI Express device hierarchy, and hence can be a scarce resource on some platform topologies with large I/O fan-out. This can impose scalability limitations on the number of isolated domains a device can support.

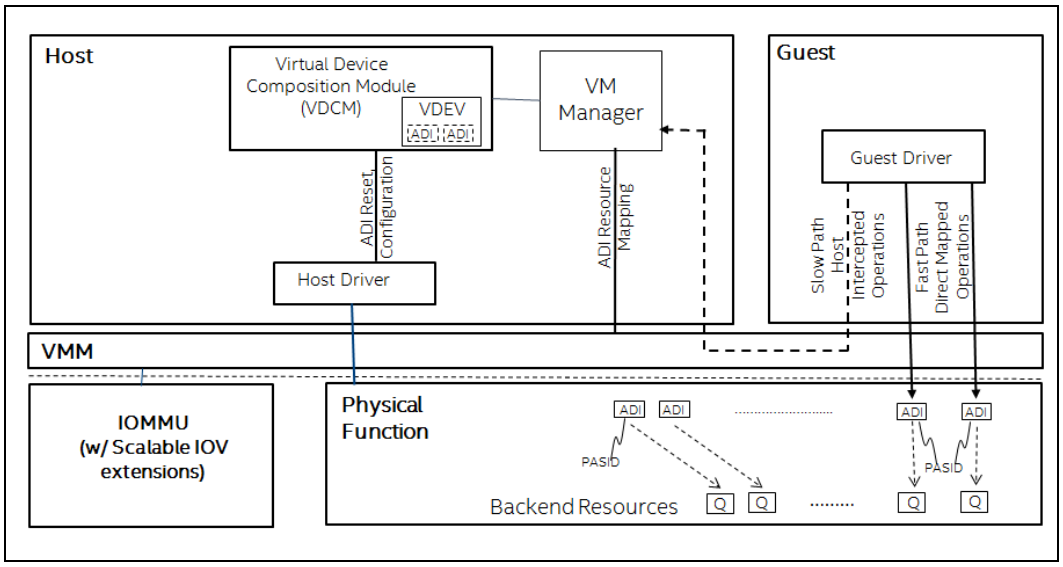
Intel Scalable IOV addresses the platform scalability issue by sharing the RID of the PF with all of its ADIs, and instead assigning ADIs a Process Address Space Identifier (PASID) that is conveyed in upstream transactions using PCI Express PASID TLP Prefix. Refer to the PCI Express specification for details on the PASID TLP Prefix. The platform support for Intel Scalable IOV described in Chapter 4 enables unique address translation functions for upstream requests at PASID granularity. Unlike RID, PASID is not used for transaction routing on the I/O fabric but instead is used only to convey the address space targeted by the memory transaction. Additionally, PASIDs are 20-bit IDs compared to 16-bit RIDs which gives 16x more identifiers. This use of PASIDs by Intel Scalable IOV enables significantly more number of domains to be supported by I/O devices.

2.2.4 Virtual Device Composability

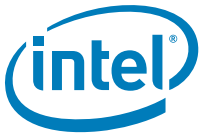
Figure 2-4 illustrates the high-level software architecture for Intel Scalable IOV. The figure calls out key components to describe the architecture and is not intended to illustrate all virtualization software or specific implementation choices. To support broad types of device classes and implementations, the software responsibilities are abstracted between system software (OS/ VMM) and device-specific driver software components.

Intel Scalable IOV introduces a device-specific software component referred to as the Virtual Device Composition Module (VDCM) that is responsible for composing virtual device (VDEV) instances, which it does by emulating VDEV slow-path operations/accesses and mapping the VDEV fast-path accesses to Assignable Device Interface (ADI) instance allocated and configured on the physical device. Unlike SR-IOV VFs, the VDCM allows I/O devices to avoid implementing slow-path operations in hardware and instead focus device hardware to efficiently scale the Assignable Device Interfaces.

Figure 2-4. Intel® Scalable I/O Virtualization Software Architecture



Additionally, virtualization management software may make use of VDCM software interfaces for enhanced 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 5 further describes the high-level software architecture for Intel Scalable IOV.



CHAPTER 3
DEVICE SUPPORT FOR INTEL® SCALABLE I/O VIRTUALIZATION

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 (RCIEP) and PCI Express* Endpoint (PCIEP) devices.

As described in the previous chapter, Intel Scalable IOV defines the construct for fine-grained sharing on endpoint devices as Assignable Device Interfaces (ADIs). ADIs form the unit of assignment and isolation for Intel Scalable IOV capable 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.

3.1 ORGANIZING DEVICE RESOURCES FOR ADIS

Resources on a device associated with fast-path work submission, execution and completion operations are referred to as device back-end 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.

Assignable Device Interface (ADI) refers to the set of device back-end resources that are allocated, configured and organized as an isolated unit, forming the unit of device sharing. The type and number of back-end resources grouped to compose an ADI is device specific. For example, for a network controller device, an ADI may be composed of a set of TX/RX queues and resources associated with a Virtual Switch Interface (VSI). An ADI on a storage controller may be the set of command queues and completion queues associated with a storage namespace. Similarly, an ADI on a GPU may be organized as a set of graphics or compute contexts created on behalf of a virtual-GPU device instance. Depending on the design, ADI on an FPGA device may be an entire Accelerator Function Unit (AFU) or a context on a multi-context capable AFU.

SR-IOV architecture specifies the allocation of PCI Express architectural resources through the VF construct but leaves it to device implementations on how the device back-end resources are allocated and associated with specific VFs. Devices that want to flexibly provision variable number of back-end resources to VFs (e.g., 1 queue-pair to 1 VF and 4 queue-pairs to another VF) need to implement another level of 'resource remapping logic' (See Figure 2-3) within the device to map which back-end resources are accessible through specific VFs and isolated from access through other VFs. Such resource remapping logic in the device increases device complexity as the number of VFs and back-end resources are scaled.

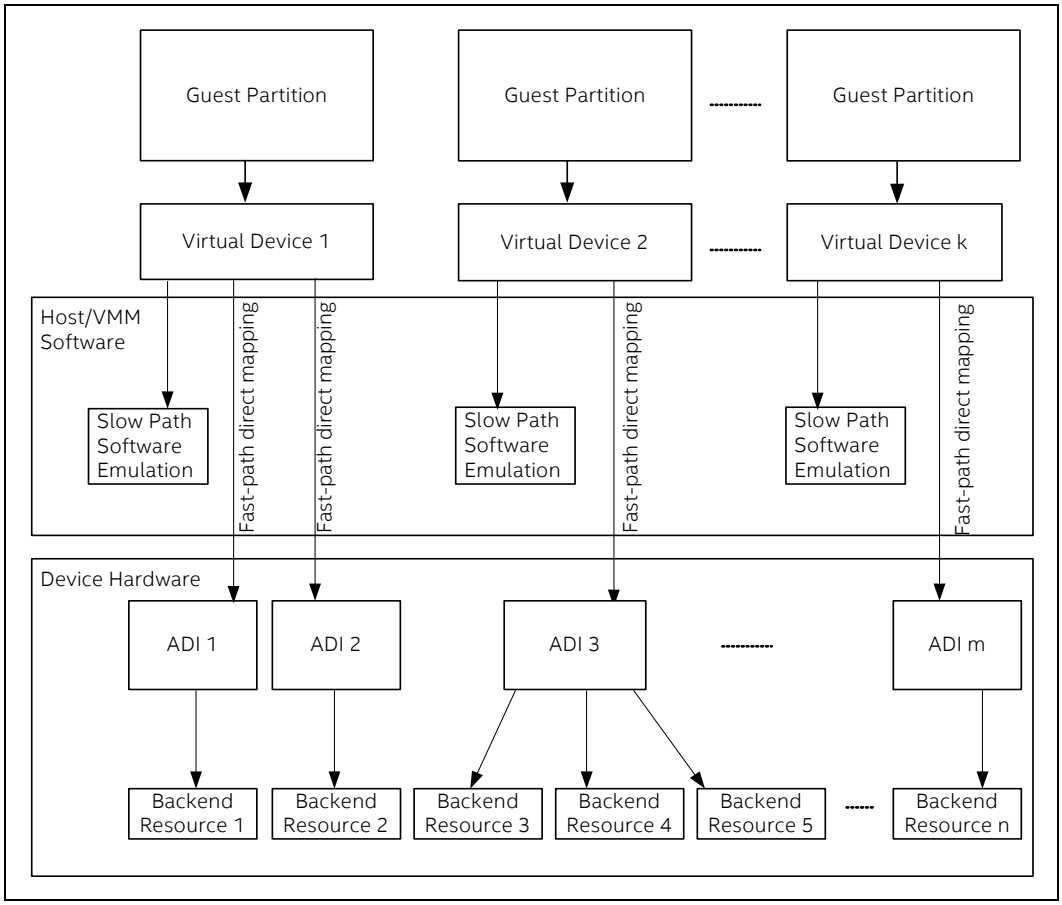
SR-IOV software architecture follows a virtual device instance to be composed of a single VF, whereas the Intel Scalable IOV software architecture allows software to compose a virtual device instance through the use of one or more ADIs. This enables device hardware designs to avoid the need for complex resource remapping logic internal to the device.

For example, consider a device that uses queue-pairs (QP) as its back-end resources, and a VM that needs 8 QPs in the virtual device for its workloads. SR-IOV designs will have to map a VF to 8 QPs, with either static partitioning of 8 QPs per VF, or dynamic partitioning of 8 QPs to a VF using resource remapping logic in the device. An equivalent Intel Scalable IOV capable device design may treat each QP as an ADI and use the VDCM software to compose a virtual device using 8 ADIs. The resource remapping logic is implemented in the VDCM software in this case.

Figure 3-1 illustrates a logical view of ADIs with varying number of device back-end resources, and virtualization software composing virtual device instances with one or more ADIs. ADI 1 and ADI 2 are composed of single back-end resource 1 and 2 respectively, whereas ADI 3 is composed of multiple back-end 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 respec-

tively). Due to different ADI composition of ADI 3 and ADI m, VDEV 2 gets 3 back-end resources whereas VDEV k gets one back-end resource.

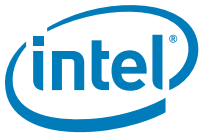
Figure 3-1. Composability of Virtual Devices from ADIs



3.2 IDENTIFYING ADI UPSTREAM REQUESTS

Unlike SR-IOV VFs whose requests are tagged with each VF's unique Requester-ID, requests from all ADIs of a PF are tagged with the Requester-ID of the PF. Instead, requests from ADIs are distinguished through Process Address Space Identifier (PASID) in an end-to-end PASID TLP Prefix. The PCI Express specification defines the Process Address Space Identifier (PASID) in the PASID TLP Prefix of a transaction, in conjunction with the Requester-ID, identifies the address space associated with the request.

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. Platforms with Intel® Virtualization Technology for Directed I/O, Rev 3.0 or higher supports Intel Scalable IOV through PASID-granular address translation capability. Depending on the programming of such 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 and ADIs from one or more devices may be configured to use these PASIDs.



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 (SVM) within a VM, where a guest application process is assigned an ADI, and similar to nested address translation (GVA to GPA to HPA) for CPU accesses by the guest application, requests from its ADI are also subjected to same nested translation by the DMA remapping hardware.

Depending on the usage model, an ADI may also be allowed to use more than one PASID value and in this case the semantics of which PASID value to use with which request is device dependent. For example, an ADI may be configured to access meta-data, commands and completions with one PASID that represents a restricted control domain, while the data accesses are associated with the PASID of the domain to which the ADI is assigned.

Endpoint devices supporting Intel Scalable IOV must support the PASID capability as defined by the PCI Express specification and comply with all associated requirements. Before enabling ADIs on a PF, the PASID capability on the PF must be enabled by software. Before an ADI is activated, it must be configured with PASID value. All upstream memory requests (except ATS translated requests) generated by ADIs must be tagged with the assigned PASID value using the PASID TLP Prefix. ADIs must not be able to generate memory requests (except ATS translated requests) without PASID or generate memory requests with a PASID value in the PASID TLP Prefix that is not its assigned PASID value.

3.3 ADI MEMORY MAPPED REGISTERS

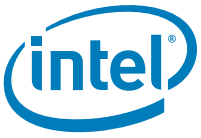
Each ADI's memory mapped I/O (MMIO) registers are hosted within one or more of the PCI Express Base Address Registers (BARs) of the hosting PF. Each ADI's MMIO registers must be contained in one or more system page size and aligned regions, and these may be contiguous or scattered regions within the PF's MMIO space. The exact association between the number and location of system page-size regions within the PF's MMIO to specific ADIs is device-specific. The system page sizes supported by the device is reported via the Intel Scalable IOV capability as described in Section 3.7. For Intel® 64 platforms the system page size is 4KB.

Devices supporting Intel Scalable IOV must partition their ADI MMIO registers into two categories: (a) MMIO registers accessed frequently for fast-path operations; and (b) MMIO registers accessed infrequently for slow-path (control, configuration, management etc.) operations. The definition of what operations are designated as slow-path versus fast-path is device-specific. The PF must locate registers in these two categories in distinct system page size regions. This enables virtualization software to directly map fast-path operations to one or more constituent ADIs while emulating slow-path operations in software.

It is recommended that devices implement 64-bit BARs so that address space above 4GB can be used for scaling ADI MMIO resources. Additionally, since non-prefetchable BARs use MMIO space below 4GB even with 64-bit BARs, devices are recommended to implement prefetchable 64-bit BARs.

3.4 ADI INTERRUPTS

ADIs capable of generating interrupts, must generate only message signaled interrupts (no legacy interrupts). ADIs must not share its interrupt resources/messages with the PF or with another ADI. Each ADI may support one or more interrupt messages. For example, an ADI composed of N queues on a PF may support N interrupt messages to distinguish work arrivals or completions for each queue.



3.4.1 ADI Interrupt Message Storage (IMS)

Intel Scalable IOV enables device implementations to 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 PCI Express defined MSI-X capability.

Even though the IMS storage organization is device-specific, IMS entries store and generate interrupts using the same interrupt message address and data format as with PCI Express MSI-X table entries. Interrupt messages stored in IMS is composed of a DWORD size data payload and a 64-bit address. IMS may also optionally support per-message masking and pending bit status, similar to the per-vector mask and pending bit array in the PCI Express MSI-X capability. The IMS resource is expected to be programmed only by the host driver.

PFs hosting the ADIs should continue to support PCI Express defined MSI or MSI-X capability. Interrupts generated by the PF are expected to use the PF's MSI or MSI-X capability as specified by the PCI Express specification, while interrupts generated by ADIs use the device-specific IMS. Specific host OS/VMM implementations may support the use of IMS for PF's interrupts and/or the use of PF's MSI-X table for ADI interrupts, however, these usages are beyond the scope of this specification.

The size, location, and storage format for IMS is device-specific. For example, some devices may implement IMS as on-device storage, while other stateful devices that manage contexts that are saved to and restored from memory may implement IMS as part of the context privileged state. In either approach, devices may implement IMS as either one unified storage structure or as de-centralized per-ADI storage structures. If IMS is implemented in host memory, ADIs may cache IMS entries on the device. If the device implements IMS caching, it must also implement device specific interfaces for the device-specific driver to invalidate the IMS cache entries.

3.4.2 ADI Interrupt Isolation

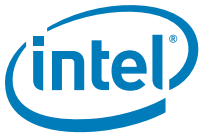
IMS is expected to be managed by host driver software and is not made accessible directly from guest or user-mode drivers. Within the device, IMS storage is not directly accessible from the ADIs, and instead the ADIs can request interrupt generation only through the PF's 'Interrupt Message Generation Logic'. This ensures that ADIs cannot modify IMS contents and an ADI can indirectly generate interrupts only using IMS entries assigned by the host driver to the corresponding 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 while interrupt messages are generated without PASID TLP prefix, it is not possible to generate a DMA write to the interrupt message address (0xFEExxxxx on Intel 64 platforms) through an ADI and cause the platform to interpret it as an interrupt message.

3.5 ADI ISOLATION, ACCESS CONTROL, AND QOS

Operations or functioning of one ADI must not affect functioning of another ADI or functioning of the PF. Every memory request (except ATS translated requests) from an ADI must be with PASID TLP Prefix using the ADI's assigned PASID value in the PASID TLP prefix. The PASID identity for an ADI is expected to be accessed or modified by privileged software such as through the host driver.

Since ADIs on a device are part of the PF, the PCI Express Access Control Service (ACS) capability is not applicable for isolation between ADIs. Instead, devices should disable peer-to-peer access (either internal to the device or at I/O fabric egress), between ADIs and between ADIs and the PF. Independent of Intel Scalable IOV support, PF may support ACS guidelines for isolation across endpoint functions or devices, per PCI Express specification.



Quality of service (QoS) for ADIs can be defined specific to a given device, however specifics for this are outside the scope of this specification. ADI QoS attributes would be managed by the host driver and controlled by the virtualization software through host driver interfaces.

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 PF. Handling of ADI specific errors can be implemented in device-specific ways.

3.6 ADI RESET

Each ADI must be independently resettable without affecting the operation of other ADIs. However, unlike SR-IOV VFs, ADIs do not support Function Level Reset (FLR) capability. Instead, reset of an ADI is performed through software interfaces to the host driver. To support ADI reset, endpoint devices may implement interfaces to abort (discard) in-flight and accepted operations to the ADI by specific domain (or PASID). The virtual device composed out of the ADI may expose virtual FLR capability that may be emulated by the VDCM by requesting the host driver to perform the ADI reset for the constituent ADIs for 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.

PFs are expected to support Function Level Reset (FLR) and may optionally support additional device-specific global reset control. Global reset operation and FLR on a PF resets all of its ADIs and returns the PF to a state where no ADIs are configured.

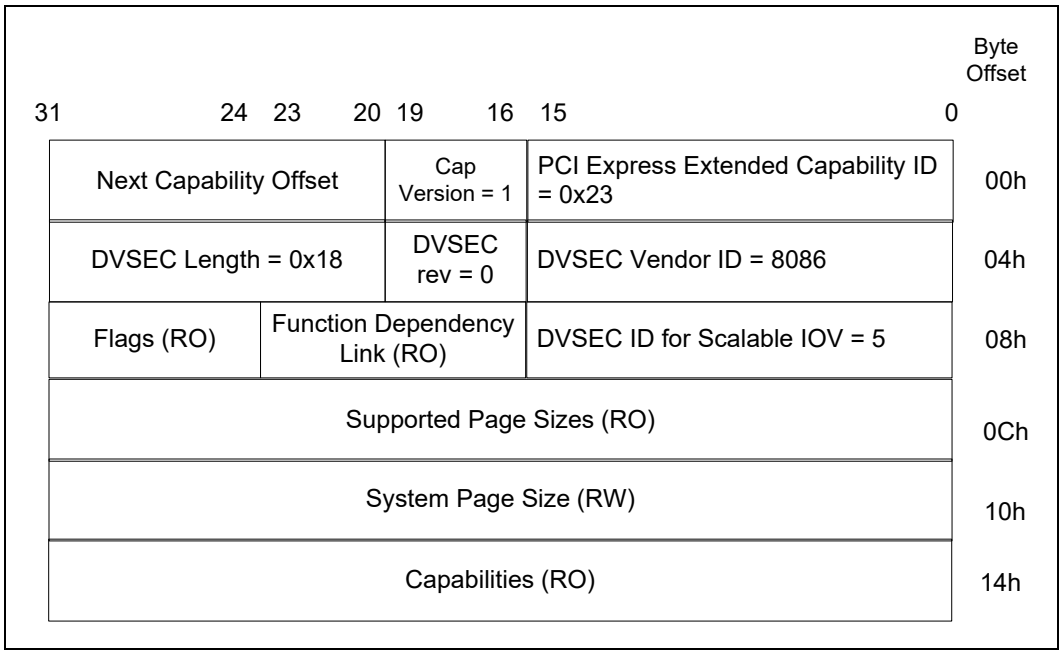
PFs may optionally support saving and restoring ADI state to facilitate operations such as live migration and suspend/resume of virtual devices composed from such ADIs. For example, to support ADI suspend, endpoint devices may implement interfaces to drain (complete) in-flight and accepted operations to the ADI by specific domain (or PASID). ADI suspend, ADI state save, ADI state restore, and ADI resume from restored state are implemented through host driver interfaces.

3.7 INTEL® SCALABLE I/O VIRTUALIZATION CAPABILITY ENUMERATION

A PF reports support for Intel Scalable IOV to system software through its host driver interface. If the host driver software reports support for Intel Scalable IOV, it is expected to support an extended set of interfaces to enumerate, provision, instantiate, and manage ADIs on the PF (See Table 5-1). The system software performs all Intel Scalable IOV specified operations on the device through the host driver.

Additionally, a PCI Express Designated Vendor Specific Extended Capability (DVSEC) is defined for systems software and tools that may need to detect endpoint devices supporting Intel Scalable IOV, without host driver dependency. The host driver is still responsible for enabling Intel Scalable IOV and related operations through system software specific interfaces. Figure 3-2 illustrates Intel Scalable IOV DVSEC structure.

Figure 3-2. 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.

Table 3-1.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 PFs. ADI dependencies are the same as the dependencies of their PFs. If a PF is independent from other PFs of a device, this field shall contain its own Function Number. If a PF is dependent on other PFs of a Device, this field shall contain the Function Number of the next PF in the same Function Dependency List. The last PF in a Function Dependency List shall contain the Function Number of the first PF in the Function Dependency List (FDL). Dependencies between PFs are described by the Flags field at offset 0xB.	RO

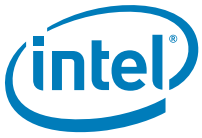


Table 3-2.Flags (Offset = 0xB, Size = 1 Byte)

Bit Location	Description	Attributes
0	H (homogeneous): When the H flag is reported as set, it indicates that all PFs in the Function Dependency List (FDL) must be enabled (in device-specific manner) for Intel® Scalable IOV operation. If some but not all of the PFs in the FDL are enabled for Intel Scalable IOV operation, the behavior is undefined. I.e., one PF cannot be in Intel Scalable IOV operation mode and other in SR-IOV operation mode if this flag is reported as set. If H flag is not set, PFs in the FDL can be in different modes.	RO
7:1	Reserved	RO

Table 3-3.Supported Page Sizes (Offset = 0xC, Size = 4 Bytes)

Bit Location	Description	Attributes
31:0	This field indicates the page sizes supported by the PF. The PF supports a page size of 2n+12 if bit n is set. For example, if bit 0 is set, the PF supports 4 KB pages. The page size describes the minimum alignment requirements for ADI MMIO pages so that they can be independently assigned to different address domains. PFs are required to support 4KB page sizes. PFs may support additional system page sizes for broad compatibility across host platform architectures.	RO

Table 3-4.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 the 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, the ADIs associated with this PF support a page size of 2n+12. 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, the PF aligns all ADI MMIO resources on system page size boundaries. System page size must be configured before setting the Memory Space Enable bit in PCI command register of the PF. The behavior is undefined if system page size is modified after memory space enable bit is set. Default value is 0000 0001h indicating a system page size of 4 KB.	RW

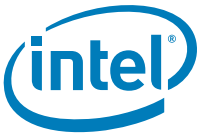


Table 3-5.Capabilities (Offset = 0x14, Size = 4 Bytes)

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 does not support IMS use by the PF itself (IMS use supported only for PFs ADIs), when the PF is directly assigned to a domain, for compatibility, virtualization software may expose a virtual Intel® Scalable IOV capability to the domain with the IMS support bit reported as 0.	RO
31:1	Reserved	RO

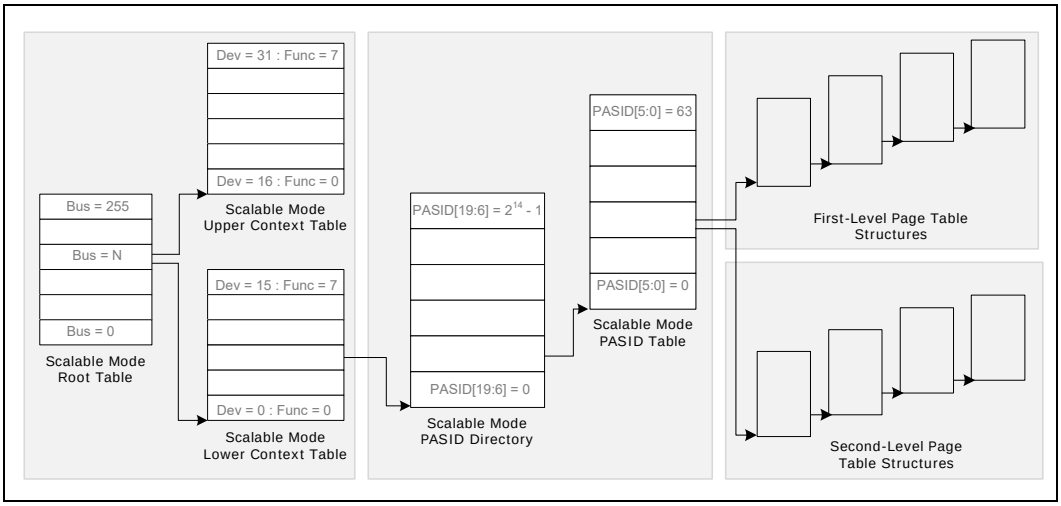
CHAPTER 4
PLATFORM SUPPORT FOR INTEL® SCALABLE IOV

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

- Support for PCI Express* PASID TLP Prefix in supporting Root Ports (RPs), Root Complex (RC), and DMA remapping hardware units. Refer to the PCI Express Rev 4.0 specification for details on PASID TLP Prefix support.
- PASID-granular address translation by DMA remapping hardware as defined by scalable mode address translation in Intel® Virtualization Technology (Intel® VT) for Directed I/O (Intel® VT-d), Rev 3.0 or higher.

Figure 4-1 illustrates the high-level translation structure organization for scalable mode address translation.

Figure 4-1. DMA Remapping Architecture for Intel® Scalable I/O Virtualization



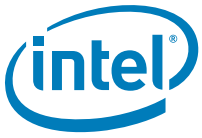
Scalable mode address translation involves a three-stage address translation:

1. First, the Requester-ID (Bus/Device/Function numbers) in upstream requests are used to consult the Root and Context structures that specify translation behavior at Requester-ID (PF or SR-IOV VF) granularity. The context structures refer 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 (target address domain) 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 respective PASID structure entry can be programmed to specify first-level, second-level, pass-through or nested translation functions, 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.

The PASID granular address translation enables upstream requests from each ADI on a PF to have a unique address translation. Any such ADIs on a PF can be used by VDCM to compose virtual devices that may be assigned to any type of address domain (such as guest physical address space of a VM or machine container, I/O virtual address for a bare-metal container, shared CPU virtual address for an application process, or such guest containers or processes operating within a VM).

For interrupt isolation across devices, host system software is expected to enable interrupt remapping and use remappable interrupt message format for all interrupt messages programmed in MSI, MSI-X, or IMS on the device. Refer to Intel Virtualization Technology for Directed I/O specification for details on interrupt remapping.

It is recommended that platforms supporting Intel Scalable IOV also support Posted Interrupts capability. Posted Interrupts enables scalable interrupt virtualization by enabling interrupt messages to operate in guest interrupt vector space without consuming host processor interrupt vectors. It additionally enables direct delivery of virtual interrupts to active virtual processors without hypervisor processing overheads. Refer to Intel Virtualization Technology for Directed I/O architecture specification for details on posted interrupts. Posted interrupt operation is transparent to endpoint devices.



CHAPTER 5
REFERENCE SOFTWARE MODEL FOR INTEL® SCALABLE IOV

This chapter describes software components envisioned for enabling Intel® Scalable IOV. This chapter is not intended to be prescriptive, and instead covers a description of system software (OS or VMM) and device-specific software roles and interactions to compose hardware-assisted virtual devices along with how to manage device operation. The software architecture described in this chapter is focused on I/O virtualization for virtual machines and machine containers. However, the principles can be applied with appropriate software support to other domains such as I/O sharing across bare-metal containers or application processes. Figure 2-4 illustrates the high-level software architecture. The logical components of the reference software architecture are described below.

5.1 HOST DRIVER

Host driver for Intel Scalable IOV capable device is conceptually equivalent to SR-IOV PF driver. The host driver is typically loaded and executed as part of the host OS or hypervisor software. In addition to the role of a normal 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 such as its PASID identity, device-specific Interrupt Message Storage (IMS) for storing ADI's interrupt messages, MMIO register resources for fast-path access to the ADI, and any device-specific resources.

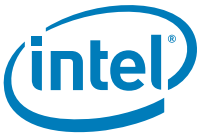
Table 5-1 below illustrates a high-level set of operations that the host driver is envisioned to support for managing ADIs. These operations are invoked through suitable software interfaces defined by specific system software (host OS or hypervisor) implementations.

Table 5-1.Host Driver Interfaces for Intel® Scalable I/O Virtualization

Description
Intel® Scalable IOV capability reporting for the PF.
Enumeration of types and maximum number of ADIs/VDEVs.
Enumeration of resource requirements for each ADI type.
Enumeration and setting of deployment 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 (IMS) for ADIs.
Enabling guest to host communication channel (if supported).
Configuring device-specific QoS properties of ADIs.
Enumerating and managing migration compatibility of ADIs.
Suspending/saving state of ADIs, and restoring/resuming state of ADIs.

5.2 VIRTUAL DEVICE COMPOSITION MODULE

The Virtual Device Composition Module (VDCM) is a device specific-component that is responsible for composing virtual device (VDEV) instances using one or more ADIs allocated by the host driver. VDCM implements software-based virtualization of VDEV slow-path operations and arranges for fast-path



operations to be submitted directly to the backing ADIs. OS or VMM implementations supporting such hardware-assisted virtual device composition may require VDCM to be implemented and packaged by device vendors in different ways. For example, in some OS or hypervisor implementations, VDCM may be packaged as user-space modules or libraries that are installed as part of the device's host driver. In other implementations, VDCM may be a kernel module. If implemented as a library, 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, VDCM can be part of the host driver.

5.3 GUEST DRIVER

Guest driver for Intel Scalable IOV capable device is conceptually equivalent to SR-IOV device VF driver. The guest driver manages the VDEV instances composed by the VDCM. Fast-path accesses by the guest driver are issued directly to the ADIs behind the VDEV, while slow-path accesses are intercepted and virtualized by the VDCM. Similar to implementation choices available for SR-IOV PF and VF drivers, for a target OS, the guest driver can be deployed as a separate driver or as a unified driver that supports both host and guest functionality. For existing SR-IOV devices, if the VDEV can be composed to behave like an existing VF, the Intel Scalable IOV guest driver can even be same as the SR-IOV VF driver for backward compatibility.

5.4 VIRTUAL DEVICE

Virtual Device (VDEV) is the abstraction through which a shared physical device is exposed to guest software. VDEVs are typically exposed to guest OS as virtual PCI Express enumerated devices, with 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 PF but implementations are possible where they are allocated across multiple PFs (for example to support device fault tolerance or load balancing).

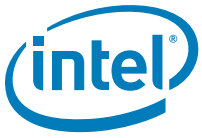
A PF may support multiple types of ADIs, both in terms of number of back-end resources (see Figure 2 1) and in terms of functionality. Similarly, more than one type of VDEV compositions are possible (with respect to the number of backing ADIs, functionality of ADIs, etc.) on a device. The VDCM may publish support for composing multiple 'VDEV types', enabling the virtual machine resource manager to request different types of VDEV instances for assigning to virtual machines. VDCM uses the host OS and VMM defined interfaces to allocate and configure resources needed to compose a VDEV. VDEV instances may be assigned to VMs in the same way as SR-IOV VFs.

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 (vNIC) 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 VDCM to dynamically create accelerator contexts as ADIs.

5.4.1 Virtual Device Memory Mapped Registers Composition

A VDEV's MMIO registers may be composed with any of below methods for any system page size regions of the VDEV MMIO space.

- Direct Mapped to ADI MMIO: As part of composing a VDEV instance, the VDCM defines the system page size ranges in VDEV virtual BARs in guest physical address (GPA) space that need to be mapped to MMIO page ranges of backing ADIs in host physical address (HPA) space. The VDCM may request the hypervisor to set up GPA to HPA mappings in the CPU virtualization page tables, enabling direct access by the guest driver to the ADI. These direct mapped MMIO ranges support fast-path operations to the ADIs.



- VDEV MMIO Intercepted and Emulated by VDCM: Slow-path registers for a 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. These intercepts are provided to the VDCM module composing the VDEV instance, so that it may virtualize such intercepted accesses by itself or through interactions with the host driver. To minimize the software complexity on slow-path access emulation, host OS or virtualization providers may restrict guest drivers to use simple memory move operations of 8 Bytes or less to access VDEV's slow-path MMIO resources.

  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 hosted 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) or without depending on guest OS or hypervisor specific para-virtualized software interfaces.

5.4.2 Virtual Device Interrupts

VDEVs 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.

For typical virtual device compositions, there may be two sources of interrupts delivered as VDEV interrupts to the guest driver. One source is the VDCM software itself that may generate virtual interrupts on behalf of the VDEV to be delivered to the guest. These are purely software generated interrupts by the slow-path operations of the VDEV emulated by VDCM. The other source of interrupts is the ADI instances on the device that are used to support fast-path operations of the VDEV. ADI generated interrupts use interrupt messages stored in the Interrupt Message Storage (IMS).

When the guest OS programs the virtual MSI or MSI-X register, the operation is intercepted and virtualized by the VDCM. For slow-path virtual interrupts, the VDCM requests virtual interrupt injection to the guest through the VMM software interfaces. For fast-path interrupts from ADIs, the VDCM invokes the host driver to allocate and configure required interrupt message address and data in the IMS. This is conceptually similar to how MSI-X interrupts for SR-IOV VFs are virtualized by some virtualization software today, except the interrupt messages are programmed in the IMS by host driver as opposed to in MSI-X table by PCI driver.

5.4.3 Communication Channel between Guest Driver and VDCM

For device-specific usages and reasons, Intel Scalable IOV capable devices may choose to build communication channels between the guest driver and the VDCM. These communication channels can be built in a guest and host system software agnostic manner with either of below methods.

- Software emulated communication channel: Such channel is composed by the VDCM using one or more system page size regions in VDEV MMIO space set up as fully memory-backed to enable

sharing of data between the guest and the host. A host intercepted system page size region in VDEV MMIO space is also set up to signal a guest action to the host. Refer to Section 5.4.1 for VDEV memory-mapped register composition details. Optionally, a virtual interrupt may also be setup 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.

5.4.4 ADIs Supporting Shared Virtual Memory

Shared Virtual Memory (SVM) refers to usages where a device is operating in the CPU virtual address space of the applications sharing the device. SVM usage is enabled with system software programming the DMA remapping hardware to reference the CPU page tables for requests with PASID representing the target applications virtual address space. Devices supporting such SVM capability does not require pages that are accessed by the device to be pinned and instead supports PCI Express Address Translation Services (ATS) and Page Request Service (PRS) capabilities to support recoverable device page-faults. Refer to PCI Express specification for details on ATS and PRS capabilities.

A device supporting Intel Scalable IOV can independently support SVM usages on ADIs allocated to host applications or for ADIs allocated to guest applications through the VDEV instance assigned to the guest VM. Both the host and guest SVM usages are transparent to the ADI operation. The only difference is in the address translation function programming of the Root Complex DMA remapping hardware. The address translation function programmed for PASIDs representing host SVM usage refers to respective CPU virtual address to physical address translation, while the address translation function programmed for PASIDs representing guest SVM usage refers to respective nested address (guest virtual address to guest physical address and further to host physical address) translation.