for many environments, I/O intensive workloads can suffer significant performance degradation. Each I/O operation - inbound or outbound - must be intercepted and processed by the VI adding significant platform resource overhead.

SR-IOV provides tools to reduce these platform resources overheads. The benefits of SR-IOV are:

- The ability to eliminate VI involvement in main data movement actions - DMA, Memory space access, interrupt processing, etc. Elimination of VI interception and processing of each I/O operation can provide significant application and platform performance improvements.
- Standardized method to control SR-IOV resource configuration and management through Single Root PCI Manager (SR-PCIM).
  - Due to a variety of implementation options - system firmware, VI, operating system, I/O drivers, etc. - SR-PCIM implementation is outside the scope of this specification.
- The ability to reduce the hardware requirements and associated cost with provisioning potentially a significant number of I/O Functions within a device.
- The ability to integrate SR-IOV with other I/O virtualization technologies such as Address Translation Services (ATS), Address Translation and Protection Table (ATPT) technologies, and interrupt remapping technologies to create robust, complete I/O virtualization solutions.

Figure 9-3 illustrates an example SR-IOV capable platform.

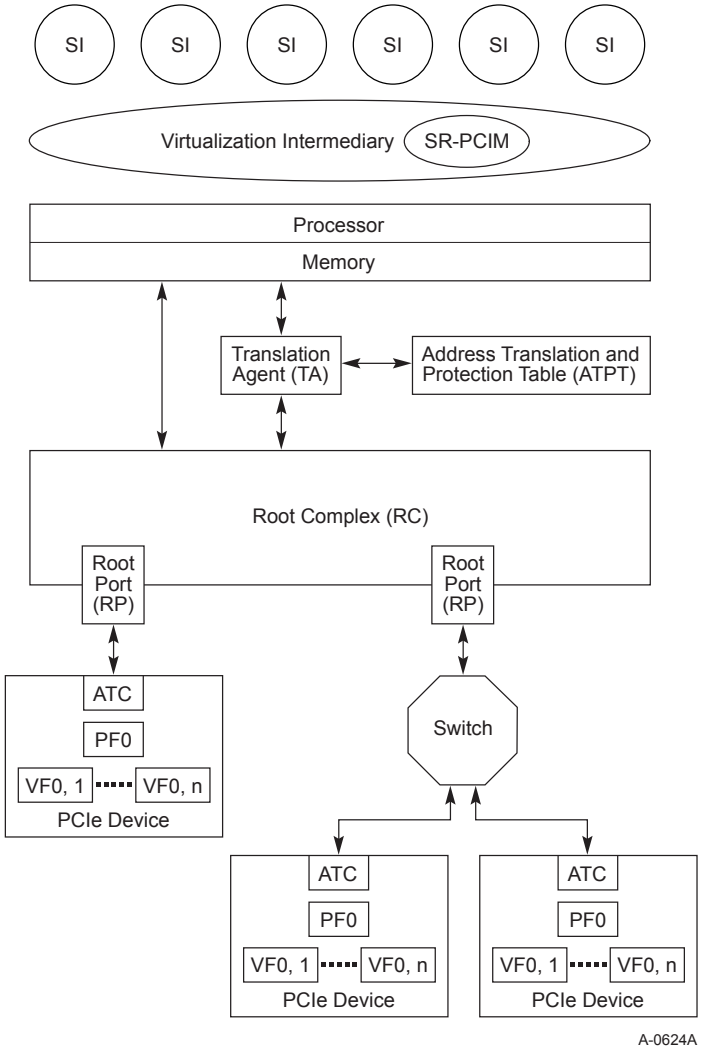

Figure 9-3 Generic Platform Configuration with SR-IOV and IOV Enablers

The SR-IOV generic platform configuration is composed of the following additional functional elements:

- SR-PCIM - Software responsible for the configuration of the SR-IOV Extended Capability, management of Physical Functions and Virtual Functions, and processing of associated error events and overall device controls such as power management and hot-plug services.
- Optional Translation Agent (TA) - A TA is hardware or a combination of hardware and software responsible for translating an address within a PCIe transaction into the associated platform physical address. A TA may contain an Address Translation Cache (ATC) to accelerate translation table access. A TA may also support Address Translation Services (ATS) which enables a PCIe Function to obtain address translations *a priori* to DMA access to the associated memory. See Chapter 10 for more details on ATS benefits and operation.
- Optional Address Translation and Protection Table (ATPT) - An ATPT contains the set of address translations accessed by a TA to process PCIe requests - DMA Read, DMA Write, or interrupt requests. See Address Translation Services (Chapter 10) for additional details.

  - In PCIe, interrupts are treated as memory write operations. Through the combination of a Requester Identifier and the address contained within a PCIe transaction, an interrupt can be routed to any target (e.g., a processor core) transparent to the associated I/O Function.
  - DMA Read and Write requests are translated through a combination of the Routing ID and the address contained within a PCIe transaction.
- Optional Address Translation Cache (ATC) - An ATC can exist in two locations within a platform - within the TA which can be integrated within or sit above an RC - or within a PCIe Device. Within an RC, the ATC enables accelerated translation look ups to occur. Within a Device, the ATC is populated through ATS technology. PCIe transactions that indicate they contain translated addresses may bypass the platform's ATC in order to improve performance without compromising the benefits associated with ATPT technology. See Address Translation Services (Chapter 10) for additional details.
- Optional Access Control Services (ACS) - ACS defines a set of control points within a PCI Express topology to determine whether a TLP should be routed normally, blocked, or redirected. In a system that supports SR-IOV, ACS may be used to prevent device Functions assigned to the VI or different SIs from communicating with one another or a peer device. Redirection may permit a Translation Agent to translate Upstream memory TLP addresses before a peer-to-peer forwarding decision is made. Selective blocking may be provided by the optional ACS P2P Egress Control. ACS is subject to interaction with ATS. See Section 9.3.7.6 for additional details.
- Physical Function (PF) - A PF is a PCIe Function that supports the SR-IOV Extended Capability and is accessible to an SR-PCIM, a VI, or an SI.
- Virtual Function (VF) - A VF is a "light-weight" PCIe Function that is directly accessible by an SI.
  - Minimally, resources associated with the main data movement of the Function are available to the SI. Configuration resources should be restricted to a trusted software component such as a VI or SR-PCIM.
  - A VF can be serially shared by different SI, (i.e., a VF can be assigned to one SI and then reset and assigned to another SI.)
  - A VF can be optionally migrated from one PF to another PF. The migration process itself is outside the scope of this specification but is facilitated through configuration controls defined within this specification.
- All VFs associated with a PF must be the same device type as the PF, (e.g., the same network device type or the same storage device type.)

To compare and contrast a PCIe Device with a PCIe SR-IOV capable device, examine the following set of figures. Figure 9-4 illustrates an example PCIe-compliant Device.

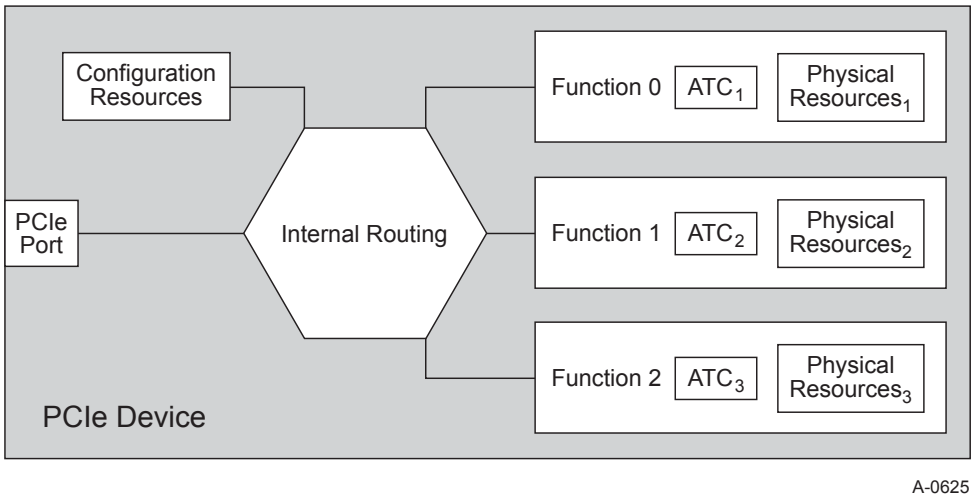

Figure 9-4 Example Multi-Function Device

This figure illustrates an example multi-Function PCIe Device with the following characteristics:

- The PCIe Device shares a common PCIe Link. The Link and PCIe functionality shared by all Functions is managed through Function 0.
  - While this figure illustrates only three Functions, with the use of the Alternative Routing Identifier (ARI) capability, a PCIe Device can support up to 256 Functions.
  - All Functions use a single Bus Number captured through the PCI enumeration process.
- In this example, each PCIe Function supports the ATS capability and therefore has an associated ATC to manage ATS obtained translated addresses.
- Each PCIe Function has a set of unique physical resources including a separate configuration space and BAR.
- Each PCIe Function can be assigned to an SI. To prevent one SI from impacting another, all PCIe configuration operations should be intercepted and processed by the VI.

As this figure illustrates, the hardware resources scale with the number of Functions provisioned. Depending upon the complexity and size of the device, the incremental cost per Function will vary. To reduce the incremental hardware cost, a device can be constructed using SR-IOV to support a single PF and multiple VFs as illustrated in Figure 9-5 .

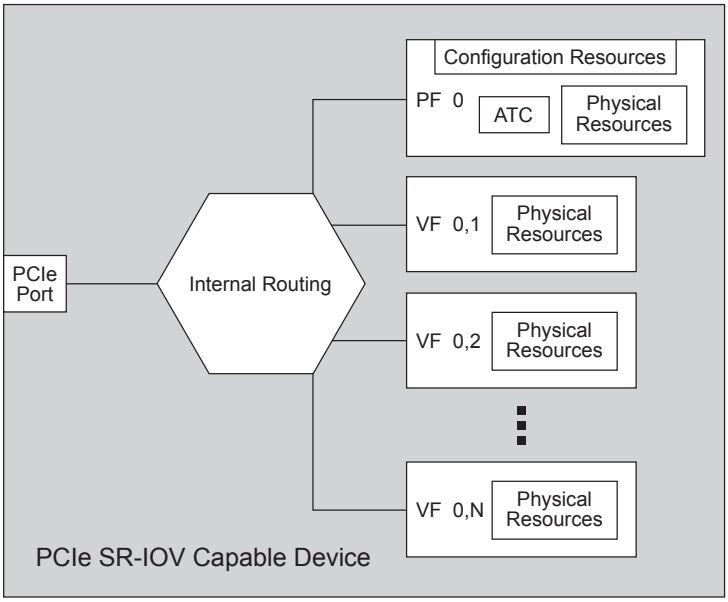

Figure 9-5 Example SR-IOV Single PF Capable Device

The example in Figure 9-5 illustrates a single PF with N VFs. Key observations to note:

- The PF is a PCIe-compliant.
  - Initially and after a conventional reset, a PCIe Function that supports the SR-IOV capabilities defined in this specification shall have the SR-IOV capabilities disabled.
  - To discover the page sizes supported by a PF and its associated VF, the Supported Page Sizes configuration field is read. For additional information on how this field can be used to align PF or VF Memory space apertures on a system page boundary, see Section 9.2.1.1.1 .
- PF nomenclature **PF *M*** designates the PF at Function number M.
- VF nomenclature **VF *M,N*** designates the Nth VF associated with PF M. VFs are numbered starting with 1 so the first VF associated with PF M is VF M,1.
- Each VF shares a number of common configuration space fields with the PF; (i.e., where the fields are applicable to all VF and controlled through a single PF. Sharing reduces the hardware resource requirements to implement each VF.)
  - A VF uses the same configuration mechanisms and header types as a PF.
  - All VFs associated with a given PF share a VF BAR set (see Section 9.3.3.14) and share a VF Memory Space Enable (MSE) bit in the SR-IOV extended capability (see Section 9.3.3.3.4) that controls access to the VF Memory space. That is, if the VF MSE bit is Clear, the memory mapped space allocated for all VFs is disabled.
  - The InitialVFs and TotalVFs fields (see Section 9.3.3.5 and Section 9.3.3.6) are used to discover the maximum number of VFs that can be associated with a PF.
  - If the Device does not support VF migration, TotalVFs and InitialVFs shall contain the same value. If the Device supports VF migration, when TotalVFs is read, the PF must return the number of VFs that can be assigned to the PF. For such a Device, when InitialVFs is read, the PF must return the initial number of VFs assigned to the PF.

- Each Function, PF, and VF is assigned a unique Routing ID. The Routing ID for each PF is constructed as per Section 2.2.4.2 . The Routing ID for each VF is determined using the Routing ID of its associated PF and fields in that PF's SR-IOV Extended Capability.
- All PCIe and SR-IOV configuration access is assumed to be through a trusted software component such as a VI or an SR-PCIM.
- Each VF contains a non-shared set of physical resources required to deliver Function-specific services, (e.g., resources such as work queues, data buffers, etc.) These resources can be directly accessed by an SI without requiring VI or SR-PCIM intervention.
- One or more VF may be assigned to each SI. Assignment policies are outside the scope of this specification.
- While this example illustrates a single ATC within the PF, the presence of any ATC is optional. In addition, this specification does not preclude an implementation from supporting an ATC per VF within the Device.
- Internal routing is implementation specific.
- While many potential usage models exist regarding PF operation, a common usage model is to use the PF to bootstrap the device or platform strictly under the control of a VI. Once the SR-IOV Extended Capability is configured enabling VF to be assigned to individual SI, the PF takes on a more supervisory role. For example, the PF can be used to manage device-specific functionality such as internal resource allocation to each VF, VF arbitration to shared resources such as the PCIe Link or the Function-specific Link (e.g., a network or storage Link), etc. These policy, management, and resource allocation operations are outside the scope of this specification.

Another example usage model is illustrated in Figure 9-6 . In this example, the device supports multiple PFs each with its own set of VFs.

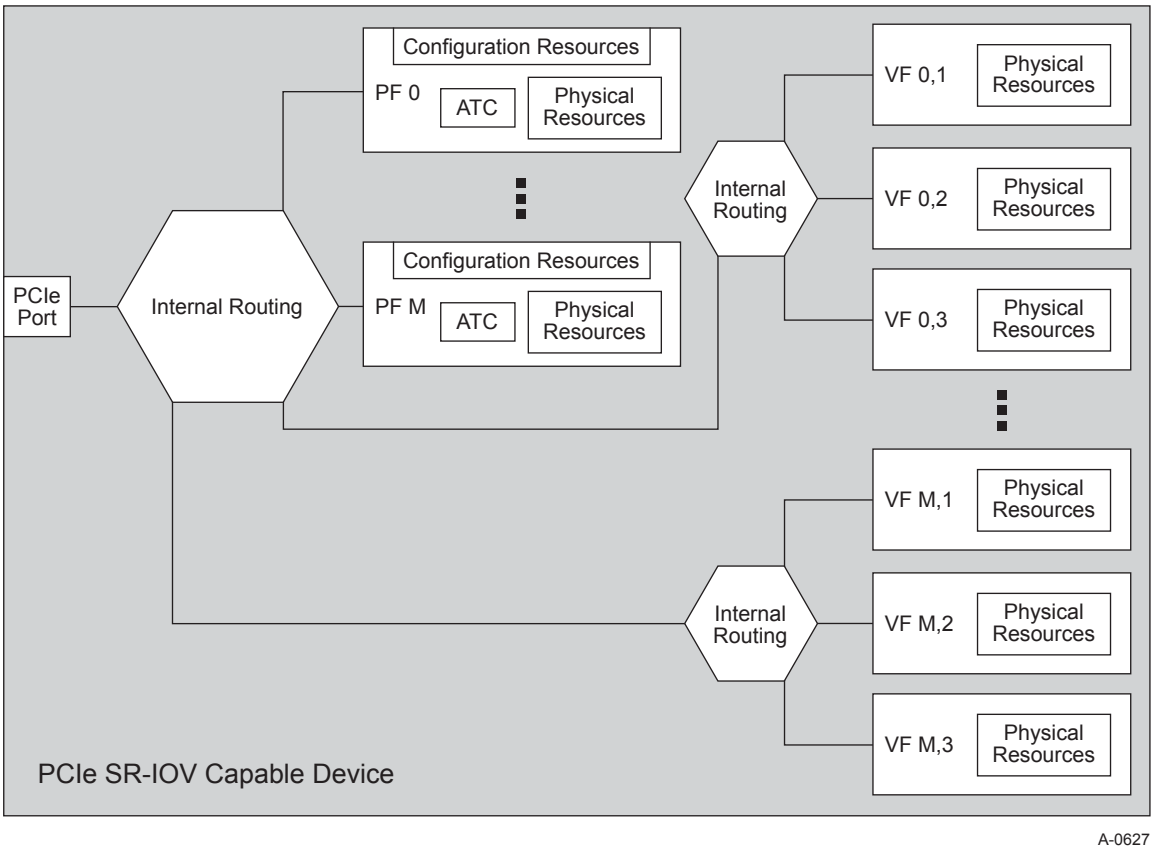

Figure 9-6 Example SR-IOV Multi-PF Capable Device

Key observations to note:

- Each PF can be assigned zero or more VFs. The number of VFs per PF is not required to be identical for all PFs within the device.
- The ARI Extended Capability enables Functions to be assigned to Function Groups and defines how Function Group arbitration can be configured. PFs and VFs can be assigned to Function Groups and take advantage of the associated arbitration capabilities. Within each Function Group, though, arbitration remains implementation specific.
- Internal routing between PFs and VFs is implementation specific.
- For some usage models, all PFs may be the same device type; (e.g., all PFs deliver the same network device or all deliver the same storage device functionality.) For other usage models, each PF may represent a different device type; (e.g., in Figure 9-6 , one PF might represent a network device while another represents an encryption device.)
  - In situations where there is a usage model dependency between device types, such as for each VF that is a network device type, each SI also requires a VF that is an encryption device type. The SR-IOV Extended Capability provides a method to indicate these dependencies. The policies used to construct these dependencies as well as assign dependent sets of VF to a given SI are outside the scope of this specification.

As seen in the prior example, the number of PF and VF can vary based on usage model requirements. To support a wide range of options, an SR-IOV Device can support the following number and mix of PF and VF:

- Using the Alternative Routing Identifier (ARI) capability, a device may support up to 256 PFs. Function Number assignment is implementation specific and may be sparse throughout the 256 Function Number space.
- A PF can only be associated with the Device's captured Bus Number as illustrated in Figure 9-7 .
- SR-IOV Devices may consume more than one Bus Number. A VF can be associated with any Bus Number within the Device's Bus Number range - the captured Bus Number plus any additional Bus Numbers configured by software. See Section 9.2.1.2 for details.
  - Use of multiple Bus Numbers enables a device to support a very large number of VFs - up to the size of the Routing ID space minus the bits used to identify intervening busses.
  - If software does not configure sufficient additional Bus Numbers, then the VFs implemented for the additional Bus Numbers may not be visible.

IMPLEMENTATION NOTE
Function Co-location

The ARI Extended Capability enables a Device to support up to 256 Functions - Functions, PFs, or VFs in any combination - associated with the captured Bus Number. If a usage model does not require more than 256 Functions, implementations are strongly encouraged to co-locate all Functions, PFs, and VFs within the captured Bus Number and not require additional Bus Numbers to access VFs.

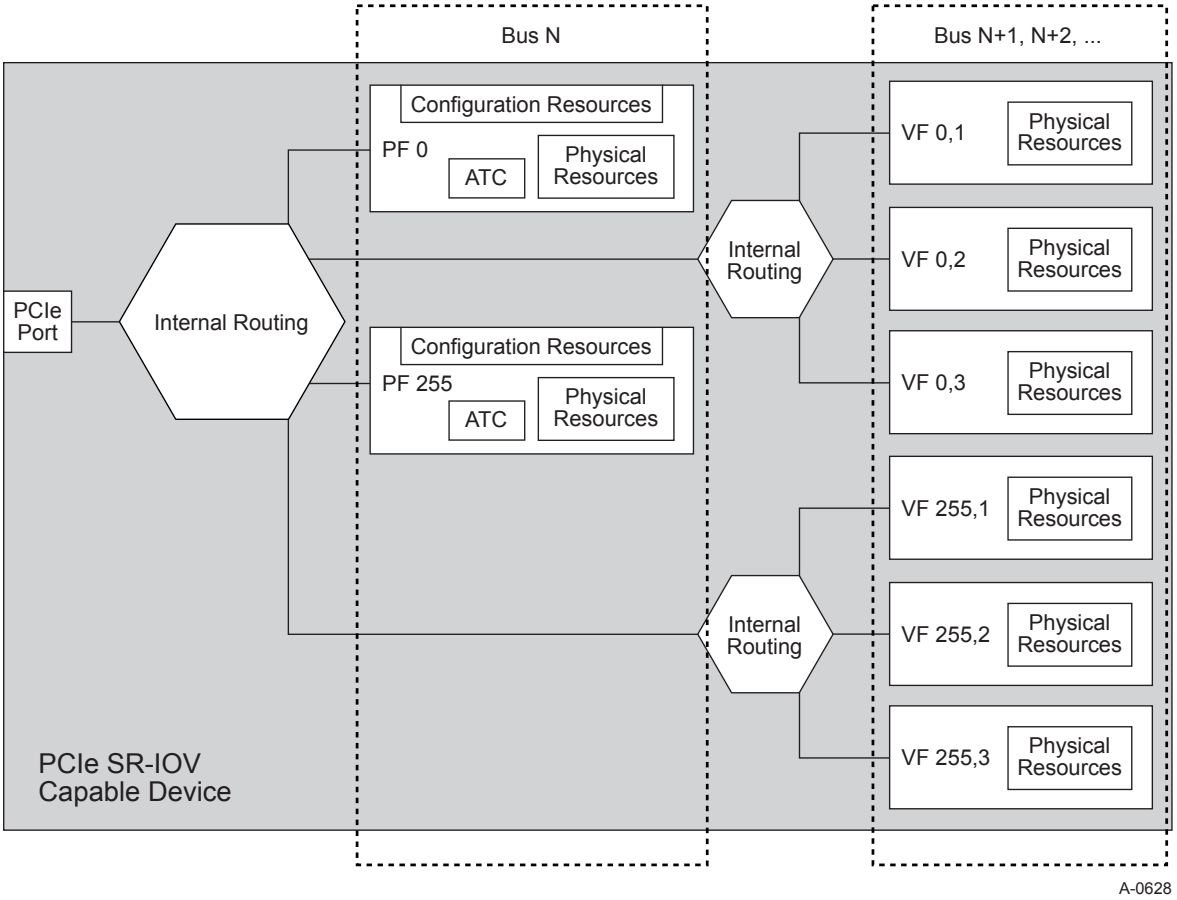

Figure 9-7 Example SR-IOV Device with Multiple Bus Numbers

In this last example, Figure 9-8 , a device implementation may mix any number of Functions, PFs, and VFs.

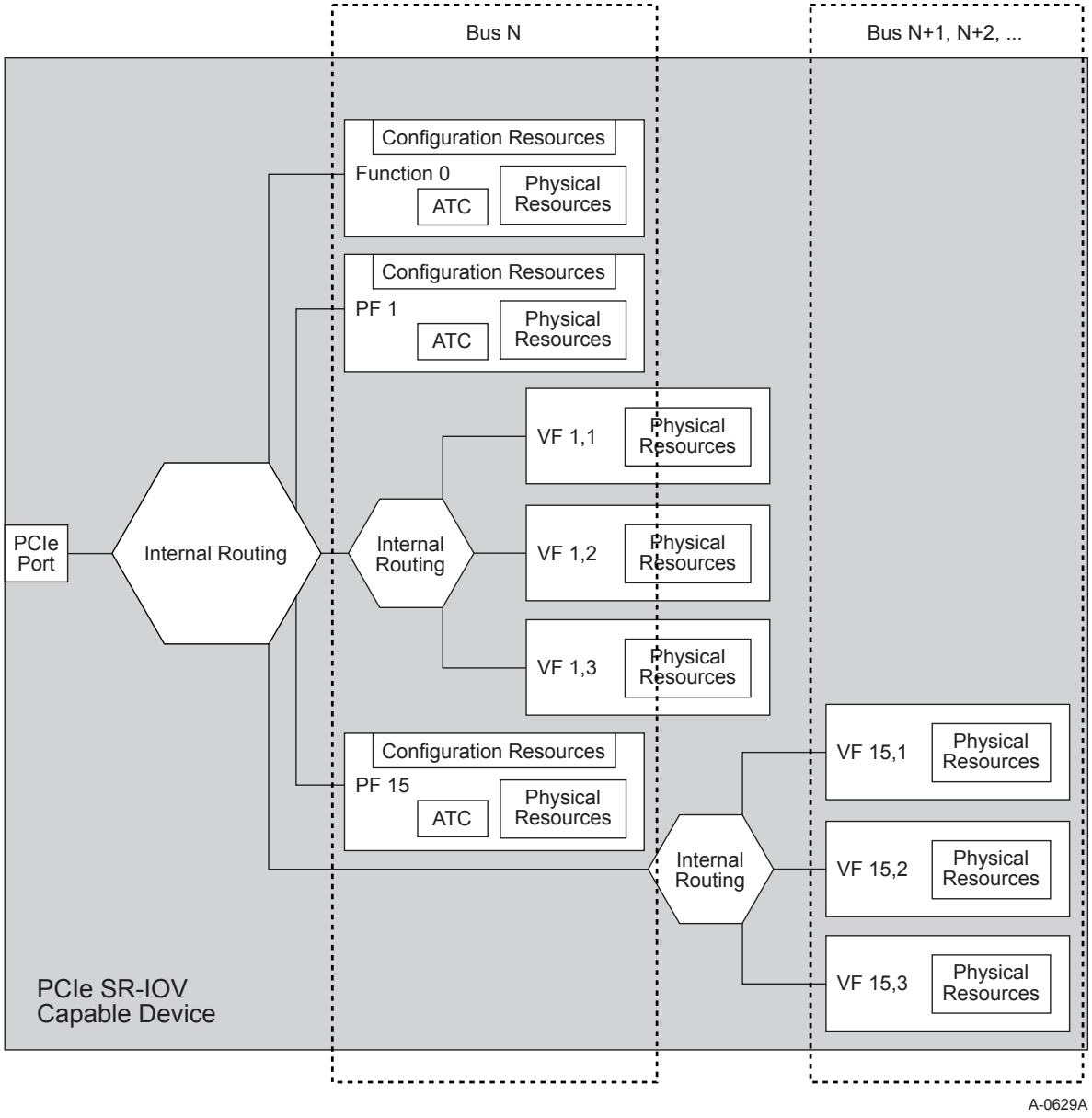

Figure 9-8 Example SR-IOV Device with a Mixture of Function Types

Key observations to note:

- Each Device must contain a Function 0. Function 0 may be a PF (i.e., it may include the SR-IOV extended capability).
- Any mix of Functions can be associated with the captured Bus Number.
  - Non-VFs can only be associated with the captured Bus Number.
- If the ARI Extended Capability is supported, Functions can be assigned to Function Groups. The assignment policy is outside the scope of this specification. If the ARI Extended Capability is not supported, Functions can still use the Function arbitration capabilities as defined in Section 6.3.3.4 .

9.1.1 PCI Technologies Interoperability

Establishing clear interoperability requirements is critical to the success of any technology. To this end, the PCI-SIG I/O virtualization specifications are organized to maximize the interoperability potential for compliant implementations. Conceptually, this is viewed as a set of concentric circles that define how functionality is layered to build an IOV capable component as illustrated in Figure 9-9 .

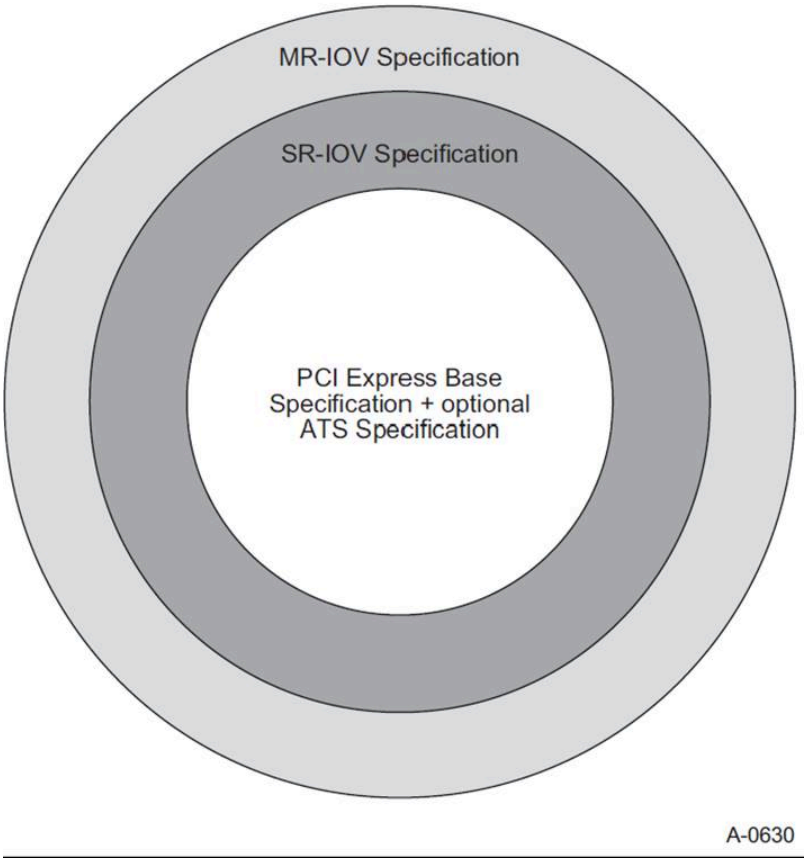

Figure 9-9 I/O Virtualization Interoperability

Key observations to note:

- At their core, I/O virtualization extensions build upon this specification. All IOV implementations must be compliant with the [PCIe-1.1] or later versions. Where applicable, the IOV specifications note relevant deltas between these versions.
  - None of the IOV specifications touch the physical layer.
  - SR-IOV does not touch the data Link or transaction layers specified in this specification.
  - The [MR-IOV] does not touch the transaction layer specified in this specification.
  - All I/O virtualization capabilities are communicated through new PCI Express capabilities implemented in PCI Express extended configuration space.
  - I/O virtualization specifications have no impact on PCI or PCI-X specifications.

    - A hierarchy can be composed of a mix of PCI Express and PCI Express-to-PCI/PCI-X Bridges.
        - A PCI Express-to-PCI/PCI-X Bridge and PCI/PCI-X Devices can be serially shared by multiple SIs.
- ATS defines optional functionality applicable to any Function. ATS can be supported in SR-IOV components.
- To implement an SR-IOV Device, SR-IOV requires the Device to be fully compliant with the [PCIe].
    - A hierarchy can be composed of a mix of SR-IOV and non-SR-IOV components. For example, a hierarchy may contain any mix of SR-IOV and non-SR-IOV Endpoint Devices.

9.2 SR-IOV Initialization and Resource Allocation

9.2.1 SR-IOV Resource Discovery

The following sections describe how software determines that a Device is SR-IOV capable and subsequently identifies VF resources through Virtual Function Configuration Space.

9.2.1.1 Configuring SR-IOV Capabilities

This section describes the fields that must be configured before enabling a PF's IOV Capabilities. The VFs are enabled by Setting the PF's VF Enable bit (see Section 9.3.3.3.1) in the SR-IOV extended capability.

The NumVFs field (see Section 9.3.3.7) defines the number of VFs that are enabled when VF Enable is Set in the associated PF.

9.2.1.1.1 Configuring the VF BAR Mechanisms

This section describes how the VF BARs are configured to map memory space. VFs do not support I/O Space and thus VF BARs shall not indicate I/O Space.

The System Page Size field (see Section 9.3.3.13) defines the page size the system will use to map the VF's PCIe memory addresses when the PF's IOV Capabilities are enabled. The System Page Size field is used by the PF to align the Memory space aperture defined by each VF BAR to a system page boundary. The value chosen for the System Page Size must be one of the Supported Page Sizes (see Section 9.3.3.12) in the SR-IOV extended capability.

The behavior of VF BARs is the same as the normal PCI Memory Space BARs (see Section 7.5.1.2.1), except that a VF BAR describes the aperture for each VF, whereas a PCI BAR describes the aperture for a single Function. The attributes for some of the bits in the VF BARs are affected by the VF Resizable BAR Extended Capability (see Section 9.3.7.5) if it is implemented.

- The behavior described in Section 7.5.1.2.1 for determining the memory aperture of a Function's BAR applies to each VF BAR. That is, the size of the memory aperture required for each VF BAR can be determined by writing all "1"s and then reading the VF BAR. The results read back must be interpreted as described in Section 7.5.1.2.1 .
- The behavior for assigning the starting memory space address of each BAR associated with the first VF is also as described in Section 7.5.1.2.1 . That is, the address written into each VF BAR is used by the Device for the starting address of the first VF.
- The difference between VF BARs and BARs described in Section 7.5.1.2.1 is that for each VF BAR, the memory space associated with the second and higher VFs is derived from the starting address of the first VF and the

memory space aperture. For any given VF_v, the starting address of its Memory space aperture for any implemented BAR_b) is calculated according to the following formula:

BAR_b VF_v starting address = VF BAR_b + (v - 1) x (VF BAR_b aperture size)

where VF BAR_b aperture size is the size of VF BAR_b as determined by the usual BAR probing algorithm as described in Section 9.3.3.14 .

VF memory space is not enabled until both VF Enable and VF MSE have been Set (see Section 9.3.3.3.1 and Section 9.3.3.3.4). Note that changing System Page Size (see Section 9.3.3.13) may affect the VF BAR aperture size.

Figure 9-10 shows an example of the PF and VF Memory space apertures.

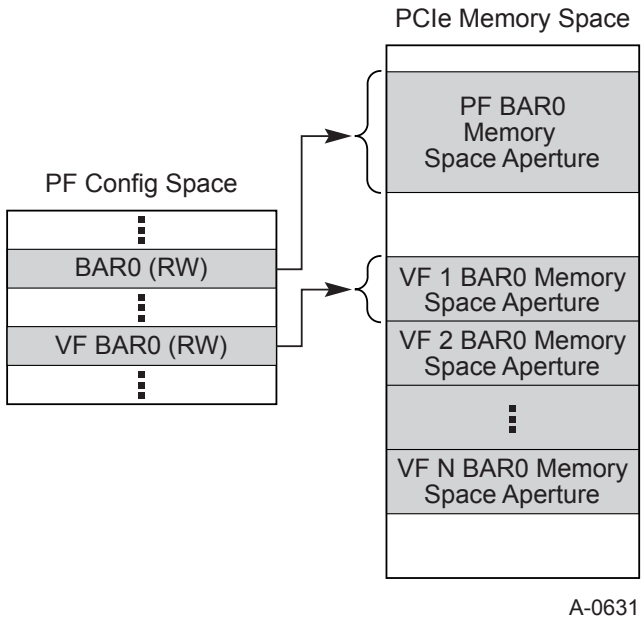

Figure 9-10 BAR Space Example for Single BAR Device

9.2.1.2 VF Discovery

The First VF Offset and VF Stride fields in the SR-IOV extended capability are 16-bit Routing ID offsets. These offsets are used to compute the Routing IDs for the VFs with the following restrictions:

- The value in NumVFs in a PF (Section 9.3.3.7) may affect the values in First VF Offset (Section 9.3.3.9) and VF Stride (Section 9.3.3.10) of that PF.
- The value in ARI Capable Hierarchy (Section 9.3.3.3.5) in the lowest-numbered PF of the Device (for example PF_0) may affect the values in First VF Offset and VF Stride in all PFs of the Device.
- NumVFs of a PF may only be changed when VF Enable (Section 9.3.3.3.1) of that PF is Clear.
- ARI Capable Hierarchy (Section 9.3.3.3.5) may only be changed when VF Enable is Clear in all PFs of a Device.

IMPLEMENTATION NOTE
NumVFs and ARI Capable Hierarchy

After configuring NumVFs and ARI Capable Hierarchy where applicable, software may read First VF Offset and VF Stride to determine how many Bus Numbers would be consumed by the PF's VFs. The additional Bus Numbers, if any, are not actually used until VF Enable is Set.

Table 9-1 describes the algorithm used to determine the Routing ID associated with each VF.

Table 9-1 VF Routing ID Algorithm

VF Number	VF Routing ID
VF 1	(PF Routing ID + First VF Offset) Modulo 2^{16}
VF 2	(PF Routing ID + First VF Offset + VF Stride) Modulo 2^{16}
VF 3	(PF Routing ID + First VF Offset + 2 * VF Stride) Modulo 2^{16}
…	…
VF N	(PF Routing ID + First VF Offset + (N-1) * VF Stride) Modulo 2^{16}
…	…
VF NumVFs (last one)	(PF Routing ID + First VF Offset + (NumVFs-1) * VF Stride) Modulo 2^{16}

All arithmetic used in this Routing ID computation is 16-bit unsigned dropping all carries.

All VFs and PFs must have distinct Routing IDs. The Routing ID of any PF or VF must not overlap with the Routing ID of any other PF or VF given any valid setting of NumVFs across all PFs of a device.

VF Stride and First VF Offset are constants. Except as stated earlier in this section, their values may not be affected by settings in this or other Functions of the Device.

VFs may reside on different Bus Number(s) than the associated PF. This can occur if, for example, First VF Offset has the value 0100h. A VF shall not be located on a Bus Number that is numerically smaller than its associated PF. A VF that is located on the same Bus Number as its associated PF shall not be located on a Device Number that is numerically smaller than the PF[158] .

VFs of an SR-IOV RCiEP Device are associated with the same Root Complex Event Collector (if any) as their PF. Such VFs are not reported in the Root Complex Event Collector Endpoint Association Extended Capability of the Root Complex Event Collector.

As per Section 2.2.6.2 , SR-IOV capable Devices that are associated with an Upstream Port capture the Bus Number from any Type 0 Configuration Write Request. SR-IOV capable Devices do not capture the Bus Number from any Type 1 Configuration Write Requests. SR-IOV capable RCiEPs use an implementation specific mechanism to assign their Bus Numbers.

Switch processing of Bus Numbers is unchanged from base PCIe. In base PCIe, the switch sends all Configuration Requests in the range Secondary Bus Number (inclusive) to Subordinate Bus Number (inclusive) to the Device. Type 1 Configuration Requests addressing the Secondary Bus Number are converted into Type 0 Configuration Requests while

158. SR-IOV Devices immediately below a Downstream Port always have a Device Number of 0 and thus always satisfy this condition.

Type 1 Configuration Requests addressing Bus Numbers between Secondary Bus Number (exclusive) and Subordinate Bus Number (inclusive) are forwarded on to the Device as Type 1 Configuration Requests.

Note: Bus Numbers are a constrained resource. Devices are strongly encouraged to avoid leaving “holes” in their Bus Number usage to avoid wasting Bus Numbers.

All PFs must be located on the Device’s captured Bus Number.

IMPLEMENTATION NOTE
VFs Spanning Multiple Bus Numbers

As an example, consider an SR-IOV Device that supports a single PF. Initially, only PF 0 is visible. Software Sets ARI Capable Hierarchy. From the SR-IOV Extended Capability it determines: InitialVFs is 600, First VF Offset is 1 and VF Stride is 1.

- If software sets NumVFs in the range [0 … 255], then the Device uses a single Bus Number.
- If software sets NumVFs in the range [256 … 511], then the Device uses two Bus Numbers.
- If software sets NumVFs in the range [512 … 600], then the Device uses three Bus Numbers.

PF 0 and VF 0,1 through VF 0,255 are always on the first (captured) Bus Number. VF 0,256 through VF 0,511 are always on the second Bus Number (captured Bus Number plus 1). VF 0,512 through VF 0,600 are always on the third Bus Number (captured Bus Number plus 2).

Software should configure Switch Secondary and Subordinate Bus Number fields to route enough Bus Numbers to the Device. If sufficient Bus Numbers are not available, software should reduce a Device’s Bus Number requirements by not enabling SR-IOV and/or reducing NumVFs for some or all PFs of the Device prior to enabling SR-IOV.

After VF Enable is Set in some PF *n*, the Device must Enable VF *n*,1 through VF *n*,*m* (inclusive) where *m* is the smaller of InitialVFs and NumVFs. A Device receiving a Type 0 Configuration Request targeting an Enabled VF located on the captured Bus Number must process the Request normally. A Device receiving a Type 1 Configuration Request targeting an Enabled VF not located on the captured Bus Number must process the Request normally. A Device receiving a Type 1 Configuration Request targeting the Device's captured Bus Number must follow the rules for handling Unsupported Requests. Additionally, if VF MSE is Set, each Enabled VF must respond to PCIe Memory transactions addressing the memory space associated with that VF.

Functions that are not enabled (i.e., Functions for VFs above *m*) do not exist in the PCI Express fabric. As per Section 2.3.1 , addressing Functions that do not exist will result in Unsupported Request (UR) being returned. This includes Functions on additional Bus Numbers.

IMPLEMENTATION NOTE
Multi-Function Devices with PFs and Switch Functions

SR-IOV devices may consume multiple bus numbers. Additional bus numbers beyond the first one are consecutive and immediately follow the first bus number assigned to the device. If an SR-IOV device also contains PCI-PCI Bridges (with Type 1 Configuration Space Headers), the SR-IOV usage must be accounted for when programming the Secondary Bus Number for those Bridges. Software should determine the last Bus Number used by VFs first and then configure any co-located Bridges to use Bus Numbers above that value.

9.2.1.3 Function Dependency Lists

PCI Devices may have vendor-specific dependencies between Functions. For example, Functions 0 and 1 might provide different mechanisms for controlling the same underlying hardware. In such situations, the Device programming model might require that these dependent Functions be assigned to SIs as a set.

Function Dependency Lists are used to describe these dependencies (or to indicate that there are no Function dependencies). Software should assign PFs and VFs to SIs such that the dependencies are satisfied.

See Section 9.3.3.8 for details.

9.2.1.4 Interrupt Resource Allocation

PFs and VFs support either MSI, MSI-X interrupts, or both if interrupt resources are allocated. VFs shall not implement INTx. Interrupts are described in Section 9.5 .

9.2.2 SR-IOV Reset Mechanisms

This section describes how PCI-base-defined-reset mechanisms affect Devices that support SR-IOV. It also describes the mechanisms used to reset a single VF and a single PF with its associated VFs.

9.2.2.1 SR-IOV Conventional Reset

A Conventional Reset to a Device that supports SR-IOV shall cause all Functions (including both PFs and VFs) to be reset to their original, power-on state as per the rules in Section 6.6.1 . Section 9.3 describes the behavior for the fields defined.

Note: Conventional Reset clears VF Enable in the PF. Thus, VFs no longer exist after a Conventional Reset.

9.2.2.2 FLR That Targets a VF

VFs must support Function Level Reset (FLR).

Note: Software may use FLR to reset a VF. FLR to a VF affects a VF's state but does not affect its existence in PCI Configuration Space or PCI Bus address space. The VFs BARn values (see Section 9.3.3.14) and VF MSE (see Section 9.3.3.3.4) in the PF's SR-IOV extended capability, and the VF Resizable BAR capability values (see Section 9.3.7.5) are unaffected by FLRs issued to the VF.

9.2.2.3 FLR That Targets a PF

PFs must support FLR.

FLR to a PF resets the PF state as well as the SR-IOV extended capability including VF Enable which means that VFs no longer exist.

9.2.3 IOV Re-initialization and Reallocation

If VF Enable is Cleared after having been Set, all of the VFs associated with the PF no longer exist and must no longer issue PCIe transactions or respond to Configuration Space or Memory Space accesses. VFs must not retain any state after VF Enable has been Cleared (including sticky bits).

9.2.4 VF Migration

VF Migration is an optional feature in [MR-IOV]. Devices that do not support the ***MR-IOV Extended Capability*** do not support VF Migration functionality.

In Multi-Root systems, VFs can be migrated between Virtual Hierarchies.

For VF Migration to occur, both VF Migration Capable and VF Migration Enable must be Set. VF Migration Capable indicates to SR-PCIM that VF Migration Hardware is present and that Multi-Root PCI Manager (MR-PCIM) has enabled its use. Hardware support for VF Migration is optional. SR-PCIM support for VF Migration is also optional.

VF Migration Enable indicates to Device hardware and to MR-PCIM that SR-PCIM has also enabled VF Migration.

Supporting VF Migration places the following requirements on SR-PCIM:

- The need to determine if a VF is *Active*, *Dormant*, or *Inactive*. A VF that is Active is available for use by the Single Root (SR). A VF that is Dormant can be configured by SR but will not issue transactions. A VF that is Inactive is not available for use by SR.
- The need to participate in ***Migrate In*** operations. A Migrate In operation is used to offer a VF to SR. A Migrate In operation is initiated by MR-PCIM. SR-PCIM can accept a Migrate In operation and move a VF to the Active.Available state.
- The need to participate in ***Migrate Out*** operations. A Migrate Out operation is used to request the graceful removal of an Active VF from use by SR. SR-PCIM can accept the Migrate Out and move the VF to the Inactive.Unavailable state.
- The need to handle ***Migrate In Retractions***. This is when an offer of a VF to SR is retracted by MR-PCIM and the VF moves back to the Inactive.Unavailable state.

9.2.4.1 Initial VF State

This section describes the initial values in the VF Migration State Array (see Section 9.3.3.15.1).

If InitialVFs (Section 9.3.3.5) is non-zero, VF_1 through $VF_{InitialVFs}$ are in the Active.Available state. If TotalVFs (Section 9.3.3.6) is greater than InitialVFs, $VF_{InitialVFs+1}$ through $VF_{TotalVFs}$ are in the Inactive.Unavailable state. If VF Migration Enable (Section 9.3.3.3.2) is Clear, VFs above InitialVFs are not used.

If InitialVFs is 0, no VFs are in the Active.Available state. If TotalVFs equals InitialVFs all VFs are in the Active.Available state. If TotalVFs is 0, no VFs are associated with this PF and there is no VF Migration State Array.

Figure 9-11 describes this initial VF State.

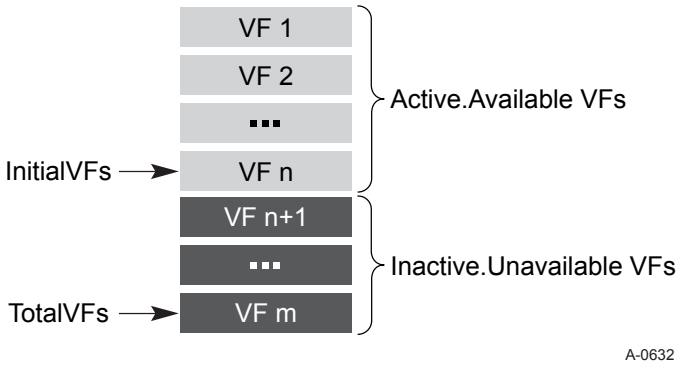

Figure 9-11 Initial VF Migration State Array

9.2.4.2 VF Migration State Transitions

VF migration follows the state diagram shown in Figure 9-12 . The state values shown are contained in the VF State array entry associated with the VF. State transitions indicated by solid lines are initiated by MR-PCIM. State transitions indicated by dotted and dashed lines are initiated by SR-PCIM.

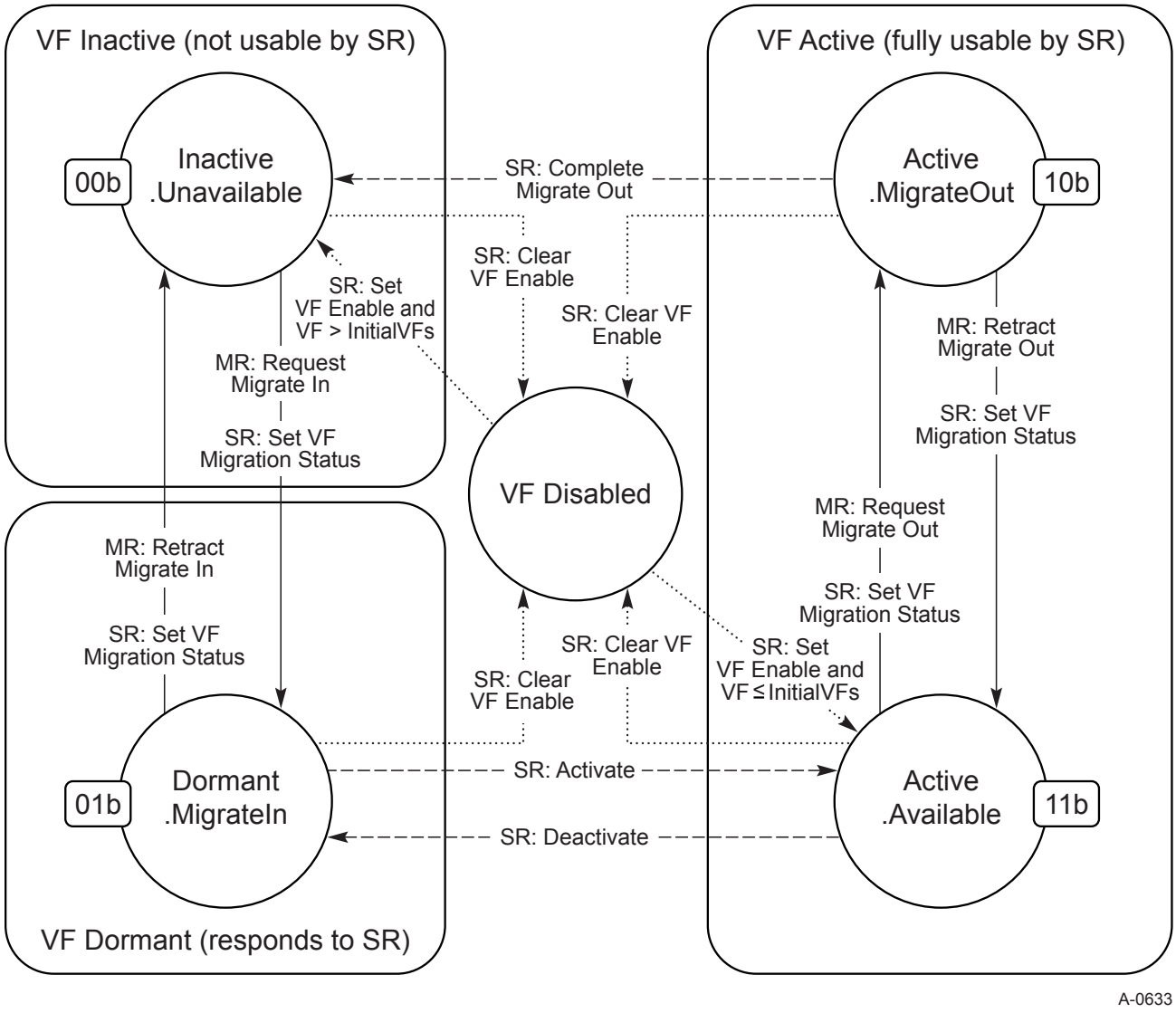

Figure 9-12 VF Migration State Diagram

SR-PCIM initiates a state transition by writing a new value to the VF Migration State Array. Devices ignore write transactions and no state transitions occur when SR-PCIM attempts to initiate any state transition other than in Table 9-2 .

Table 9-2 SR-IOV VF Migration State Table

Current VF Enable	Current VF State	Written VF Enable	Written VF State	Meaning
1	Dormant.MigrateIn	1	Active.Available	SR Activate VF.
1	Active.Available	1	Dormant.MigrateIn	SR Deactivate VF.
1	Active.MigrateOut	1	Inactive.Unavailable	SR Complete Migrate Out.
1	Any	0	Any	SR Disables all VFs.

Current VF Enable	Current VF State	Written VF Enable	Written VF State	Meaning
0	Any	1	Any	SR Enables all VFs. VFs transition to either Active.Available or Inactive.Unavailable based on the VF number and InitialVFs.

IMPLEMENTATION NOTE

Software State Migration Change Detection

SR-PCIM will typically need to verify that a state change took effect by re-reading VF Migration State after writing it.

VFs in states Inactive.Unavailable are not usable by software in any way. Requests targeting an Inactive VF receive Unsupported Request (UR). Within 100 ms of transitioning to the Inactive or Dormant states, the Device must ensure that no new transactions will be issued using the indicated Routing ID.

MR-PCIM initiates state transitions by using a different data structure as specified in [MR-IOV]. The effects of such transitions are visible in the VF Migration State Array and in the VF Migration Status bit. All state transitions initiated by MR-PCIM cause the VF Migration Status bit to be Set.

This migration state machine exists for every VF that supports VF Migration. Migration state machines are not affected by Function Dependency Lists (see Section 9.2.1.3 and Section 9.3.3.8).

VF Migration State does not affect Function state. If VF state needs to be reset as part of a Migrate Out and/or Migrate In operation, SR-PCIM must issue an FLR to accomplish this. VF behavior when VF Migration occurs without an FLR is undefined.

IMPLEMENTATION NOTE

FLR and VF Migration

VF Migration from system A to system B will typically involve one FLR in system A before completing the MigrateOut operation and a second FLR in system B after accepting the MigrateIn operation but before using the VF. System A uses the first FLR to ensure that none of its data leaks out. System B uses the second FLR to ensure that it starts with a clean slate.

9.3 Configuration

9.3.1 SR-IOV Configuration Overview

This section provides SR-IOV-added requirements for implementing PFs and VFs.

PFs are discoverable in configuration space, as with all Functions. PFs contain the SR-IOV Extended Capability described in Section 9.3.3 . PFs are used to discover, configure, and manage the VFs associated with the PF and for other things described in this specification.

9.3.2 Configuration Space

PFs that support SR-IOV shall implement the SR-IOV Extended Capability as defined in the following sections. VFs shall implement configuration space fields and capabilities as defined in the following sections.

9.3.3 *SR-IOV Extended Capability*

The SR-IOV Extended Capability defined here is a PCIe extended capability that must be implemented in each PF that supports SR-IOV. This Capability is used to describe and control a PF's SR-IOV Capabilities.

For Multi-Function Devices, each PF that supports SR-IOV shall provide the Capability structure defined in this section. This Capability structure may be present in any Function with a Type 0 Configuration Space Header. This Capability must not be present in Functions with a Type 1 Configuration Space Header.

Figure 9-13 shows the SR-IOV Extended Capability structure.

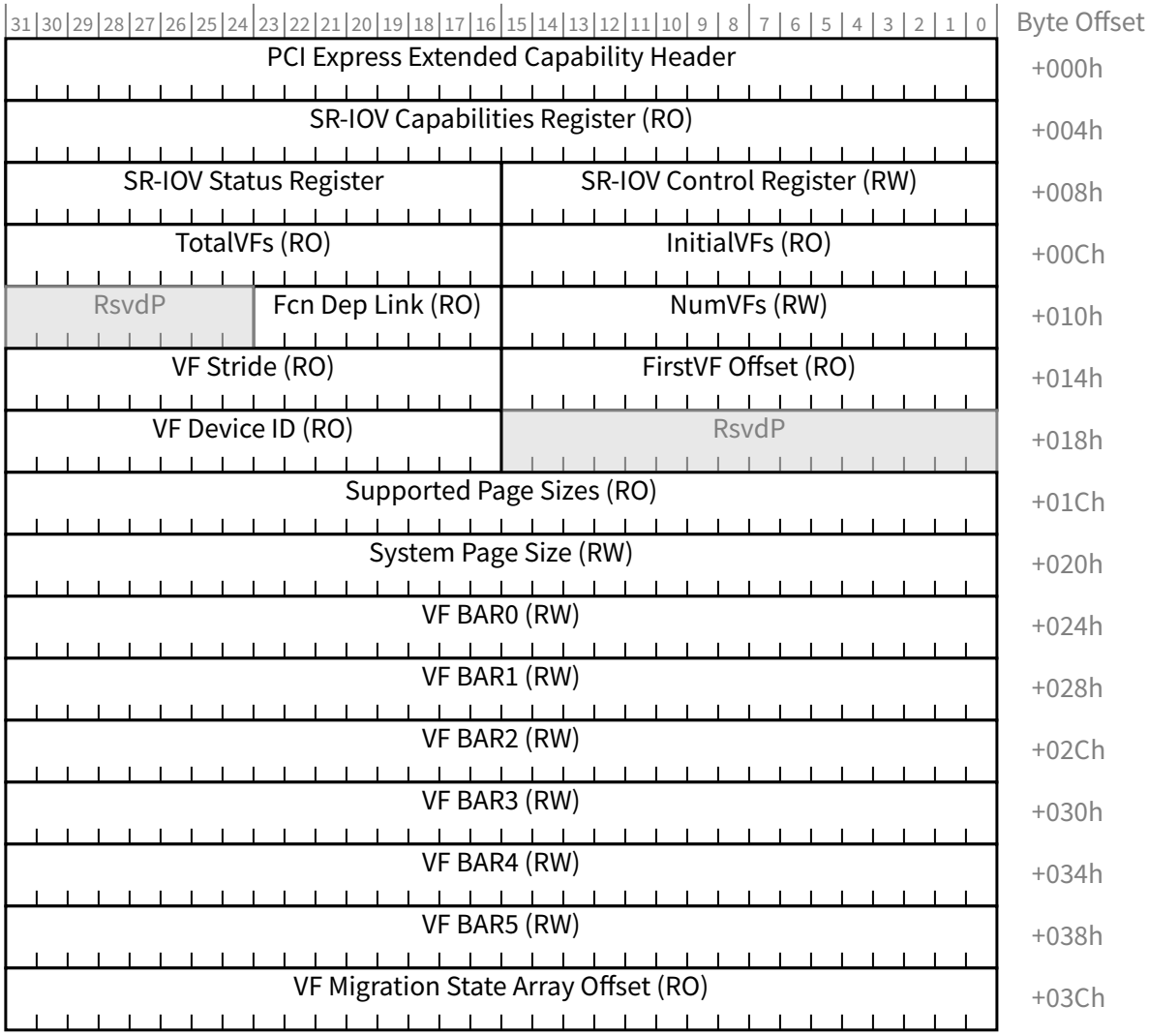

Figure 9-13 SR-IOV Extended Capability

9.3.3.1 SR-IOV Extended Capability Header (Offset 00h)

Table 9-3 defines the SR-IOV Extended Capability header. The Capability ID for the SR-IOV Extended Capability is 0010h.

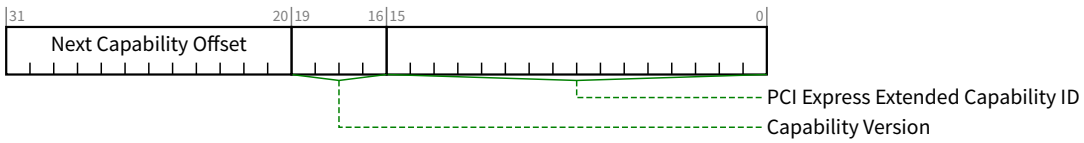

Figure 9-14 SR-IOV Extended Capability Header

Table 9-3 SR-IOV Extended Capability Header

Bit Location	Register Description	Attributes
15:0	***PCI Express Extended Capability ID*** - This field is a PCI-SIG defined ID number that indicates the nature and format of the Extended Capability. The Extended Capability ID for the SR-IOV Extended Capability is 0010h.	RO
19:16	***Capability Version*** - This field is a PCI-SIG defined version number that indicates the version of the Capability structure present. Must be 1h for this version of the specification.	RO
31:20	***Next Capability Offset*** - This field contains the offset to the next PCI Express Capability structure or 000h if no other items exist in the linked list of Capabilities. For Extended Capabilities implemented in Configuration Space, this offset is relative to the beginning of PCI-compatible Configuration Space and thus must always be either 000h (for terminating list of Capabilities) or greater than 0FFh.	RO

9.3.3.2 SR-IOV Capabilities Register (04h)

Table 9-4 defines the layout of the SR-IOV Capabilities field.

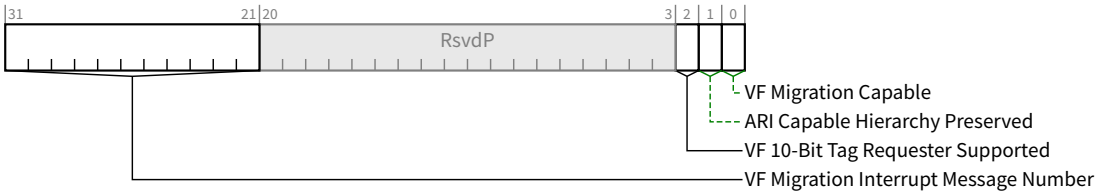

Figure 9-15 SR-IOV Capabilities Register

Table 9-4 SR-IOV Capabilities Register

Bit Location	Register Description	Attributes
0	***VF Migration Capable*** - If Set, the PF is Migration Capable and operating under a Migration Capable MR-PCIM.	RO
1	***ARI Capable Hierarchy Preserved*** *PCI Express Endpoint:* If Set, the ARI Capable Hierarchy bit is preserved across certain power state transitions. *RCiEP*: Not applicable - it is strongly recommended that this bit be hardwired to 0b.	RO
2	***VF 10-Bit Tag Requester Supported*** - If Set, all VFs must support 10-Bit Tag Requester capability. If Clear, VFs must not support 10-Bit Tag Requester capability. If the 10-Bit Tag Requester Supported bit in the PF's Device Capabilities 2 Register is Clear, this bit must be Clear.	HwInit

Bit Location	Register Description	Attributes
31:21	***VF Migration Interrupt Message Number*** - Indicates the MSI/MSI-X vector used for migration interrupts. The value in this field is undefined if VF Migration Capable is Clear.	RO

9.3.3.2.1 VF Migration Capable

VF Migration Capable is Set to indicate that the PF supports VF Migration. If Clear, the PF does not support VF Migration.

VF Migration support is an optional feature of [MR-IOV]. If the PF does not implement hardware for VF Migration, this bit shall be implemented as RO value 0b. If the PF implements hardware for VF Migration, this bit is controlled by mechanisms defined in [MR-IOV] and indicates the presence of support for VF Migration.

Devices that implement SR-IOV only shall implement this field as RO value zero.

PFs that support VF Migration must implement MSI or MSI-X interrupts (or both).

9.3.3.2.2 ARI Capable Hierarchy Preserved

ARI Capable Hierarchy Preserved is Set to indicate that the PF preserves the ARI Capable Hierarchy bit across certain power state transitions (see Section 9.3.3.3.5). Components must either Set this bit or Set the No_Soft_Reset bit (see Section 9.6.2). It is recommended that components set this bit even if they also set No_Soft_Reset.

ARI Capable Hierarchy Preserved is only present in the lowest-numbered PF of a Device (for example PF0). ARI Capable Hierarchy Preserved is Read Only Zero in other PFs of a Device.

ARI Capable Hierarchy Preserved does not apply to RCiEPs, and its value is undefined (see Section 9.3.3.3).

9.3.3.2.3 VF 10-Bit Tag Requester Supported

If a PF supports 10-Bit Tag Requester capability, its associated VFs are permitted to support 10-Bit Tag Requester capability as well, but this is optional. Especially for usage models where the bulk of the traffic is spread across several VFs concurrently, it may not be necessary for individual VFs to use 10-Bit Tags so they can support >256 outstanding Non-Posted Requests each.

For a given PF, it is required that either all or none of its associated VFs support 10-Bit Tag Requester capability. This avoids unnecessary implementation and management complexity. See Section 9.3.5.9 .

VFs that support 10-Bit Tag Requester capability have additional requirements beyond other Function types in order to simplify error handling and reduce the possibility of 10-Bit Tag related errors with one VF impacting other traffic.

- If the VF 10-Bit Tag Requester Enable bit in the SR-IOV Control register is Set, then each VF must use 10-Bit Tags for all Non-Posted Requests that it generates.
- For each outstanding 10-Bit Tag Request, if the VF receives a Completion that matches the outstanding Request other than Tag[9:8] being 00b, the VF must prevent that Request from (eventually) generating a Completion Timeout error, and instead handle the error via a device-specific mechanism that avoids data corruption.

It is strongly recommended that software not configure Unexpected Completion errors to be handled as Uncorrectable Errors. This avoids them triggering System Errors or hardware error containment mechanisms like Downstream Port Containment (DPC).

IMPLEMENTATION NOTE
No VF 10-Bit Tag Completer Supported Bit

There is no VF 10-Bit Tag Completer Supported bit. If a PF supports 10-Bit Tag Completer capability, then all of its associated VFs are required support 10-Bit Tag Completer capability as stated in Section 9.3.5.9 . This helps avoid the complexity of PCIe hierarchies where some Completers support 10-Bit Tag capability and some do not.

9.3.3.2.4 VF Migration Interrupt Message Number

VF Migration Interrupt Message Number must contain the MSI or MSI-X vector number used for the interrupt message generated in association with certain VF migration events.

For MSI, VF Migration Interrupt Message Number must indicate the MSI message number [0 .. 31] used to reference certain SR events described in this section. The Function is required to update this field so that it is correct if the number of MSI Messages assigned to the Function changes when software writes to the Multiple Message Enable field in the Message Control Register for MSI.

For MSI-X, VF Migration Interrupt Message Number must indicate the MSI-X Table entry [0 .. 2047] used to generate the interrupt message.

If both MSI and MSI-X are implemented, they are permitted to use different vectors, though software must enable only one mechanism at a time. If both MSI and MSI-X interrupts are enabled, this field is undefined.

If VF Migration Capable is Clear, this field is undefined.

IMPLEMENTATION NOTE
Migration and MSI

If a PF implements MSI and software sets Multiple Message Enable to a value less than Multiple Message Capable, some sharing of MSI vectors may be occurring. This can create complicated software structures since a single vector might need to be directed to both SR-PCIM and the PF Device Driver.

9.3.3.3 SR-IOV Control Register (Offset 08h)

Table 9-5 defines the layout of the SR-IOV Control fields.

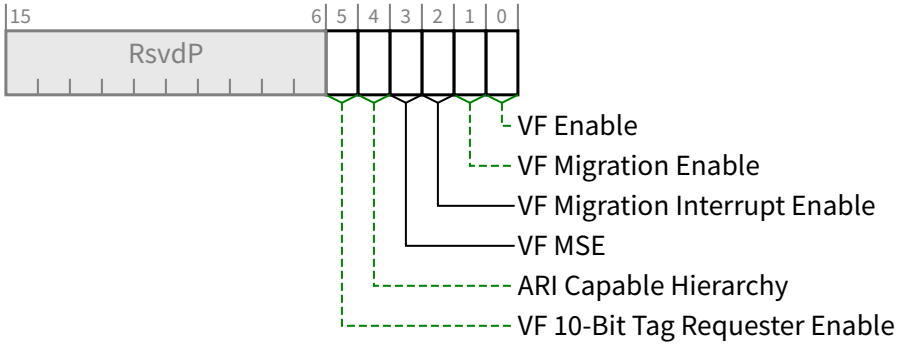

Figure 9-16 SR-IOV Control Register

Table 9-5 SR-IOV Control Register

Bit Location	Register Description	Attributes
0	***VF Enable*** - Enables/Disables VFs. Default value is 0b.	RW
1	***VF Migration Enable*** - Enables/Disables VF Migration Support. Default value is 0b. See Section 9.3.3.3.2 .	RW or RO (see description)
2	***VF Migration Interrupt Enable*** - Enables/Disables VF Migration State Change Interrupt. Default value is 0b.	RW
3	***VF MSE*** - Memory Space Enable for Virtual Functions. Default value is 0b.	RW
4	***ARI Capable Hierarchy*** - *PCI Express Endpoint:* This bit must be RW in the lowest-numbered PF of the Device and hardwired to 0b in all other PFs. If the value of this bit is 1b, the Device is permitted to locate VFs in Function Numbers 8 to 255 of the captured Bus Number. Otherwise, the Device must locate VFs as if it were a non-ARI Device. This bit is not affected by FLR of any PF or VF. Default value is 0b. *RCiEP:* Not applicable - this bit must be hardwired to 0b. Within the Root Complex, VFs are always permitted to be assigned to any Function Number allowed by First VF Offset and VF Stride rules (see Section 9.3.3.9 and Section 9.3.3.10).	RW or RO (see description)
5	***VF 10-Bit Tag Requester Enable*** - If Set, all VFs must use 10-Bit Tags for all Non-Posted Requests they generate. If Clear, VFs must not use 10-Bit Tags for Non-Posted Requests they generate. See Section 9.3.3.2.3 . If software changes the value of this bit while any VFs have outstanding Non-Posted Requests, the result is undefined. If the VF 10-Bit Tag Requester Supported bit in the SR-IOV Capabilities register is Clear, this bit is permitted to be hardwired to 0b. Default value is 0b.	RW or RO

9.3.3.3.1 VF Enable

VF Enable manages the assignment of VFs to the associated PF. If VF Enable is Set, the VFs associated with the PF are accessible in the PCI Express fabric. When Set, VFs respond to and may issue PCI Express transactions following the rules for PCI Express Endpoint Functions.

If VF Enable is Clear, VFs are disabled and not visible in the PCI Express fabric; requests to these VFs shall receive UR and these VFs shall not issue PCI Express transactions.

To allow components to perform internal initialization, after changing the VF Enable bit from Cleared to Set, the system is not permitted to issue Requests to the VFs which are enabled by that VF Enable bit until one of the following is true:

- At least 100 ms has passed
- An FRS Message has been received from the PF with a Reason Code of VF Enabled
- At least VF Enable Time has passed. VF Enable Time is either (1) the Reset Time value in the Readiness Time Reporting capability associated with the VF, or (2) a value determined by system software / firmware[159] .

The Root Complex and/or system software must allow at least 1.0 s after Setting the VF Enable bit, before it may determine that a VF which fails to return a Successful Completion Status for a valid Configuration Request is broken. After Setting the VF Enable bit, the VFs enabled by that VF Enable bit are permitted to return a CRS status to configuration requests up to the 1.0 s limit, if they are not ready to provide a Successful Completion Status for a valid Configuration Request. After a PF transmits an FRS Message with a Reason Code of VF Enabled, no VF associated with that PF is permitted to return CRS without an intervening VF disable or other valid reset condition. After returning a Successful Completion to any Request, no VF is permitted to return CRS without an intervening VF disable or other valid reset condition.

Since VFs don't have an MSE bit (MSE in VFs is controlled by the VF MSE bit in the SR-IOV Extended Capability in the PF), it's possible for software to issue a Memory Request before the VF is ready to handle it. Therefore, Memory Requests must not be issued to a VF until at least one of the following conditions has been met:

- The VF has responded successfully (without returning CRS) to a Configuration Request.
- After issuing an FLR to the VF, at least one of the following is true:
  - At least 1.0 s has passed since the FLR was issued.
  - The VF supports FRS and, after the FLR was issued, an FRS Message has been received from the VF with a Reason Code of FLR Completed.
  - At least FLR Time has passed since the FLR was issued. FLR Time is either (1) the FLR Time value in the Readiness Time Reporting capability associated with the VF or (2) a value determined by system software / firmware[153].
- After Setting VF Enable in a PF, at least one of the following is true:
  - At least 1.0 s has passed since VF Enable was Set.
  - The PF supports FRS and, after VF Enable was Set, an FRS Message has been received from the PF with a Reason Code of VF Enabled.
  - At least VF Enable Time has passed since VF Enable was Set. VF Enable Time is either (1) the Reset Time value in the Readiness Time Reporting capability associated with the VF or (2) a value determined by system software / firmware[160] .

159. For example, ACPI tables.
160. For example, ACPI tables.

The VF is permitted to silently drop Memory Requests after an FLR has been issued to the VF or VF Enable has been Set in the associated PF's SR-IOV Extended Capability until the VF responds successfully (without returning CRS) to any Request.

Clearing VF Enable effectively destroys the VFs. Setting VF Enable effectively creates VFs. Setting VF Enable after it has previously been Cleared shall result in a new set of VFs. If the PF is in the D0 power state, the new VFs are in the $D0_{uninitialized}$ state. If the PF is in a lower power state behavior is undefined (see Sections 9.6.1 and 9.6.2).

When Clearing VF Enable, a PF that supports FRS shall send an FRS Message with FRS Reason VF Disabled to indicate when this operation is complete. The PF is not permitted to send this Message if there are outstanding Non-Posted Requests issued by the PF or any of the VFs associated with the PF. The FRS Message may only be sent after these Requests have completed (or timed out).

After VF Enable is Cleared no field in the SR-IOV Extended Capability or the VF Migration State Array (see Section 9.3.3.15.1) may be accessed until either:

- At least 1.0 s has elapsed after VF Enable was Cleared.
- The PF supports FRS and after VF Enable was Cleared, an FRS Message has been received from the PF with a Reason Code of VF Disabled.

Section 9.3.3.7 NumVFs, Section 9.3.3.5 InitialVFs, Section 9.3.3.6 TotalVFs, Section 9.3.3.9 First VF Offset, Section 9.3.3.13 System Page Size, and Section 9.3.3.14 VF BARx describe additional semantics associated with this field.

9.3.3.3.2 VF Migration Enable

VF Migration Enable must be Set to allow VF Migration on this PF. See [MR-IOV] for details.

If VF Migration Capable is Set, this bit is RW and the default value is 0b. Otherwise, this bit is RO Zero.

This field is RO when VF Enable is Set.

9.3.3.3.3 VF Migration Interrupt Enable

An MSI or MSI-X interrupt message must be sent every time the logical AND of the following conditions transitions from FALSE to TRUE:

- The VF's Bus Master Enable bit is Set.
- The associated vector is unmasked.
- The VF Migration Interrupt Enable bit is Set.
- The VF Migration Status bit is Set.

The MSI or MSI-X vector used is indicated by the VF Migration Interrupt Message Number field.

If VF Migration Capable is Clear, this field is undefined.

Note: VF Migration events are described in Section 9.2.4 .

9.3.3.3.4 VF MSE (Memory Space Enable)

VF MSE controls memory space enable for all Active VFs associated with this PF, as with the Memory Space Enable bit in a Function's PCI Command register. The default value for this bit is 0b.

When VF Enable is Set, VF memory space will respond only when VF MSE is Set. VFs shall follow the same error reporting rules as defined in the [PCIe] if an attempt is made to access a Virtual Function's memory space when VF Enable is Set and VF MSE is Clear.

IMPLEMENTATION NOTE
VF MSE and VF Enable

VF memory space will respond with Unsupported Request when VF Enable is Clear. Thus, VF MSE is "don't care" when VF Enable is Clear; however, software may choose to Set VF MSE after programming the VF BARn registers, prior to Setting VF Enable.

9.3.3.3.5 ARI Capable Hierarchy

For Devices associated with an Upstream Port, ARI Capable Hierarchy is a hint to the Device that ARI has been enabled in the Root Port or Switch Downstream Port immediately above the Device. Software should set this bit to match the ARI Forwarding Enable bit in the Root Port or Switch Downstream Port immediately above the Device.

ARI Capable Hierarchy is only present in the lowest-numbered PF of a Device (for example PF_0) and affects all PFs of the Device. ARI Capable Hierarchy is Read Only Zero in other PFs of a Device.

A Device may use the setting of ARI Capable Hierarchy to determine the values for First VF Offset (see Section 9.3.3.9) and VF Stride (see Section 9.3.3.10). The effect of changing ARI Capable Hierarchy is undefined if VF Enable is Set in any PF. This bit must be set to its default value upon Conventional Reset. This bit is not affected by FLR of any PF or VF. If either ARI Capable Hierarchy Preserved is Set (see Section 9.3.3.2.2) or No_Soft_Reset is Set (see Section 9.6.2), a power state transition of this PF from $D3_{Hot}$ to D0 does not affect the value of this bit (see Section 9.6.2).

ARI Capable Hierarchy does not apply to RCiEPs.

IMPLEMENTATION NOTE
ARI Capable Hierarchy

For a Device associated with an Upstream Port, that Device has no way of knowing whether ARI has been enabled. If ARI is enabled, the Device can conserve Bus Numbers by assigning VFs to Function Numbers greater than 7 on the captured Bus Number. ARI is defined in Section 6.13 .

Since RCiEPs are not associated with an Upstream Port, ARI does not apply, and VFs may be assigned to any Function Number within the Root Complex permitted by First VF Offset and VF Stride (see Section 9.3.3.8 and Section 9.3.3.9).

9.3.3.4 SR-IOV Status Register (Offset 0Ah)

Table 9-6 : defines the layout of the SR-IOV Status field.

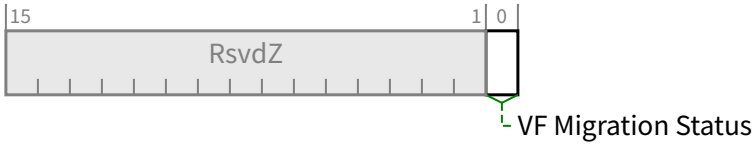

Figure 9-17 SR-IOV Status

Table 9-6 SR-IOV Status

Bit Location	Register Description	Attributes
0	***VF Migration Status*** - Indicates a VF Migration In or Migration Out Request has been issued by MR-PCIM. To determine the cause of the event, software may scan the VF State Array. Default value is 0b.	RW1C

9.3.3.4.1 VF Migration Status

VF Migration Status is Set when VF Enable, VF Migration Capable, and VF Migration Enable are Set and certain state changes occur in a VF Migration State Array entry (see Section 9.2.4 and Section 9.3.3.15.1 for details).

This setting of VF Migration Status is not affected by the value of VF Migration Interrupt Enable or by any controls for MSI or MSI-X.

9.3.3.5 InitialVFs (Offset 0Ch)

InitialVFs indicates to SR-PCIM the number of VFs that are initially associated with the PF.

The minimum value of InitialVFs is 0.

For Devices operating in Single-Root mode, this field is HwInit and must contain the same value as TotalVFs.

For Devices operating in Multi-Root mode, the value of this field may be changed by MR-PCIM when VF Enable is Clear.

Note: The mechanism used by MR-PCIM to affect this field is described in [MR-IOV].

If VF Migration Enable is Set and VF Enable is Cleared and then Set, the value of InitialVFs may change. This is necessary since some VFs may have been migrated to other PFs and may no longer be available to this PF.

9.3.3.6 TotalVFs (Offset 0Eh)

TotalVFs indicates the maximum number of VFs that could be associated with the PF.

The minimum value of TotalVFs is 0.

For Devices operating in Single-Root mode, this field is HwInit and must contain the same value as InitialVFs.

For Devices operating in Multi-Root mode, the value of this field may be changed by MR-PCIM.

Note: The mechanism used by MR-PCIM to affect this field is described in [MR-IOV].

9.3.3.7 NumVFs (Offset 10h)

NumVFs controls the number of VFs that are visible. SR-PCIM sets NumVFs as part of the process of creating VFs. This number of VFs shall be visible in the PCI Express fabric after both NumVFs is set to a valid value and VF Enable is Set. A visible VF has a Function number reserved for it, but might not exist. If the VF exists, it shall respond to PCI Express transactions targeting them, and shall follow all rules defined by this specification. A VF exists if either:

- VF Migration Capable is Clear and the VF number is less than or equal to TotalVFs.
- VF Migration Capable is Set and the associated VF is in either the Active.Available or Dormant.MigrateIn state (see Section 9.2.4)

The results are undefined if NumVFs is set to a value greater than TotalVFs.

NumVFs may only be written while VF Enable is Clear. If NumVFs is written when VF Enable is Set, the results are undefined.

The initial value of NumVFs is undefined.

9.3.3.8 Function Dependency Link (Offset 12h)

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. VF dependencies are the same as the dependencies of their associated 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.

If PF_p and PF_q are in the same Function Dependency List, then any SI that is assigned $VF_{p,n}$ shall also be assigned to $VF_{q,n}$.

IMPLEMENTATION NOTE

Function Dependency Link Example

Consider the following scenario:

SR-IOV Field	PF 0	PF 1	PF 2
Function Dependency Link	1	0	2
NumVFs	4	4	6
First VF Offset	4	4	4
VF Stride	3	3	3

Function Number	Description	Independent
0	PF 0	No
1	PF 1	No
2	PF 2	Yes
3	Function not present	
4	VF 0,1 (aka PF 0 VF 1)	No
5	VF 1,1 (aka PF 1 VF 1)	No
6	VF 2,1 (aka PF 2 VF 1)	Yes
7	VF 0,2	No
8	VF 1,2	No
9	VF 2,2	Yes
10	VF 0,3	No
11	VF 1,3	No
12	VF 2,3	Yes
13	VF 0,4	No
14	VF 1,4	No
15	VF 2,4	Yes
16 to 17	Functions not present	
18	VF 2,5	Yes
19 to 20	Functions not present	
21	VF 2,6	Yes

Function Number	Description	Independent
22 to 255	Functions not present	

In this example, Functions 4 and 5 must be assigned to the same SI. Similarly, Functions 7 and 8, 10 and 11, and 13 and 14 must be assigned together. If PFs are assigned to SIs, Functions 0 and 1 must be assigned together as well. Functions 2, 6, 9, 12, 15, 18, and 21 are independent and may be assigned to SIs in any fashion.

All PFs in a Function Dependency List shall have the same values for the InitialVFs, TotalVFs, and VF Migration Capable fields.

SR-PCIM shall ensure that all PFs in a Function Dependency List have the same values for the NumVFs, VF Enable, and VF Migration Enable fields before any VF in that Function Dependency List is assigned to an SI.

VF Migration and VF Mapping operations occur independently for every VF. SR-PCIM shall not assign a VF to an SI until it can assign all dependent VFs. For example, using the scenario above, if both VF 0,2 (Function 7) and VF 1,2 (Function 8) are in the Inactive.Unavailable or Dormant.MigrateIn states, SR-PCIM shall not assign either VF to an SI until both VFs reach the Active.Available state.

Similarly, SR-PCIM shall not remove a VF from an SI until it can remove all dependent VFs. For example, using the scenario above, both VF 0,2 and VF 1,2 shall be removed from an SI only when they both reach the Active.MigrateOut state. SR-PCIM shall not transition the Functions to Inactive.Unavailable until the SI has stopped using all dependent Functions.

9.3.3.9 First VF Offset (Offset 14h)

First VF Offset is a constant and defines the Routing ID offset of the first VF that is associated with the PF that contains this Capability structure. The first VF's 16-bit Routing ID is calculated by adding the contents of this field to the Routing ID of the PF containing this field ignoring any carry, using unsigned, 16-bit arithmetic.

A VF shall not be located on a Bus Number that is numerically smaller than its associated PF.

This field may change value when the lowest-numbered PF's ARI Capable Hierarchy value changes or when this PF's NumVFs value changes.

Note: First VF Offset is unused if NumVFs is 0. If NumVFs is greater than 0, First VF Offset must not be zero.

9.3.3.10 VF Stride (Offset 16h)

VF Stride defines the Routing ID offset from one VF to the next one for all VFs associated with the PF that contains this Capability structure. The next VF's 16-bit Routing ID is calculated by adding the contents of this field to the Routing ID of the current VF, ignoring any carry, using unsigned 16-bit arithmetic.

This field may change value when the lowest-numbered PF's ARI Capable Hierarchy value changes or when this PF's NumVFs value changes.

Note: VF Stride is unused if NumVFs is 0 or 1. If NumVFs is greater than 1, VF Stride must not be zero.

9.3.3.11 VF Device ID (Offset 1Ah)

This field contains the Device ID that should be presented for every VF to the SI.

VF Device ID may be different from the PF Device ID. A VF Device ID must be managed by the vendor. The vendor must ensure that the chosen VF Device ID does not result in the use of an incompatible device driver.

9.3.3.12 Supported Page Sizes (Offset 1Ch)

This field indicates the page sizes supported by the PF. This PF supports a page size of 2^{n+12} if bit *n* is Set. For example, if bit 0 is Set, the PF supports 4-KB page sizes. PFs are required to support 4-KB, 8-KB, 64-KB, 256-KB, 1-MB, and 4-MB page sizes. All other page sizes are optional.

The page size describes the minimum alignment requirements for VF BAR resources as described in Section 9.3.3.13 .

IMPLEMENTATION NOTE

Non-pre-fetch Address Space

Non-pre-fetch address space is limited to addresses below 4 GB. Pre-fetch address space in 32-bit systems is also limited. Vendors are strongly encouraged to utilize the System Page Size feature to conserve address space while also supporting systems with larger pages.

9.3.3.13 System Page Size (Offset 20h)

This field defines the page size the system will use to map the VFs' memory addresses. Software must set the value of the System Page Size to one of the page sizes set in the Supported Page Sizes field (see Section 9.3.3.12). As with Supported Page Sizes, if bit *n* is Set in System Page Size, the VFs associated with this PF are required to support a page size of 2^{n+12}. For example, if bit 1 is Set, the system is using an 8-KB page size. The results are undefined if System Page Size is zero. The results are undefined if more than one bit is set in System Page Size. The results are undefined if a bit is Set in System Page Size that is not Set in Supported Page Sizes.

When System Page Size is set, the VF associated with this PF is required to align all BAR resources on a System Page Size boundary. Each VF BAR*n* or VF BAR*n* pair (see Section 9.3.3.14) shall be aligned on a System Page Size boundary. Each VF BAR*n* or VF BAR*n* pair defining a non-zero address space shall be sized to consume an integer multiple of System Page Size bytes. All data structures requiring page size alignment within a VF shall be aligned on a System Page Size boundary.

VF Enable must be zero when System Page Size is written. The results are undefined if System Page Size is written when VF Enable is Set.

Default value is 0000 0001h (i.e., 4 KB).

9.3.3.14 VF BAR0 (Offset 24h), VF BAR1 (Offset 28h), VF BAR2 (Offset 2Ch), VF BAR3 (Offset 30h), VF BAR4 (Offset 34h), VF BAR5 (Offset 38h)

These fields must define the VF's Base Address Registers (BARs). These fields behave as normal PCI BARs, as described in Section 7.5.1 . They can be sized by writing all 1s and reading back the contents of the BARs as described in Section 7.5.1.2.1 , complying with the low order bits that define the BAR type fields.

These fields may have their attributes affected by the VF Resizable BAR Extended Capability (see Section 9.3.7.5) if it is implemented.

The amount of address space decoded by each BAR shall be an integral multiple of System Page Size.

Each VF BAR*n*, when "sized" by writing 1s and reading back the contents, describes the amount of address space consumed and alignment required by a single Virtual Function, per BAR. When written with an actual address value, and VF Enable and VF MSE are Set, the BAR maps NumVFs BARs. In other words, the base address is the address of the first VF BAR*n* associated with this PF and all subsequent VF BAR*n* address ranges follow as described below.

VF BARs shall only support 32-bit and 64-bit memory space. PCI I/O Space is not supported in VFs. Bit 0 of any implemented VF BARx must be RO 0b except for a VF BARx used to map the upper 32 bits of a 64-bit memory VF BAR pair.

The alignment requirement and size read is for a single VF, but when VF Enable is Set and VF MSE is Set, the BAR contains the base address for all (NumVFs) VF BAR*n*.

The algorithm to determine the amount of address space mapped by a VF BAR*n* differs from the standard BAR algorithm as follows:

1. Resize the BAR via the VF Resizable BAR Extended Capability (see Section 9.3.7.5) if it is implemented.
2. After reading the low order bits to determine if the BAR is a 32-bit BAR or 64-bit BAR pair, determine the size and alignment requirements by writing all 1s to VF BAR*n* (or VF BAR*n* and VF BAR*n+1* for a 64-bit BAR pair) and reading back the contents of the BAR or BAR pair. Convert the bit mask returned by the read(s) to a size and alignment value as described in Section 7.5.1.2.1 . This value is the size and alignment for a single VF.
3. Multiply the value from step 2 by the value set in NumVFs to determine the total amount of space the BAR or BAR pair will map after VF Enable and VF MSE are Set.

For each VF BAR*n* field, *n* corresponds to one of the VFs BAR spaces. Table 9-7 shows the relationship between *n* and a Function's BAR.

Table 9-7 BAR Offsets

n	BAR Offset in a Type 0 Header
0	10h
1	14h
2	18h
3	1Ch
4	20h
5	24h

The contents of all VF BAR*n* registers are indeterminate after System Page Size is changed.

9.3.3.15 VF Migration State Array Offset (Offset 3Ch)

If VF Migration Capable (Section 9.3.3.2.1) is Set and TotalVFs (Section 9.3.3.6) is not zero, this register shall contain a PF BAR relative pointer to the VF Migration State Array. This register is RO Zero if VF Migration Capable is Clear. The VF Migration State Array is defined in Section 9.3.3.15.1 . The layout of the VF Migration State Array Offset is defined in Table 9-8 .

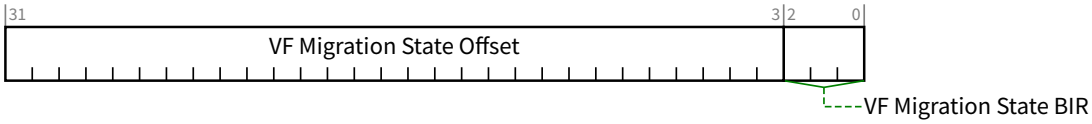

Figure 9-18 VF Migration State Array Offset

Table 9-8 VF Migration State Array Offset

Bit Location	Register Description	Attributes
2:0	***VF Migration State BIR*** - Indicates which one of a Function's Base Address registers, located beginning at 10h in Configuration Space, is used to map the Function's VF Migration State Array into Memory Space. **BIR Value BAR** **0** BAR0 10h **1** BAR1 14h **2** BAR2 18h **3** BAR3 1Ch **4** BAR4 20h **5** BAR5 24h **6** Reserved **7** Reserved For a 64-bit BAR, the VF Migration State BIR indicates the lower DW. This field is undefined if TotalVFs is 0.	RO
31:3	***VF Migration State Offset*** - Used as an offset from the address contained by one of the Function's Base Address registers to point to the base of the VF Migration State Array. The lower three VF Migration State BIR bits are masked off (set to zero) by software to form a 32-bit QW-aligned offset. This field is undefined if TotalVFs is 0.	RO

If a BAR that maps address space for the VF Migration State Array also maps other usable address space not associated with VF Migration, locations used in the other address space must not share any naturally aligned 8-KB address range with one where the VF Migration State Array resides.

9.3.3.15.1 *VF Migration State Array*

The VF Migration State Array is located using the VF Migration State Array Offset register of the SR-IOV Extended Capability block.

The VF Migration State Array has a VF Migration State Entry for each VF. The total size of the VF Migration State array is NumVFs bytes. The VF Migration State Array shall not exist if TotalVFs is 0. Table 9-9 defines the layout of a VF Migration State Array entry.

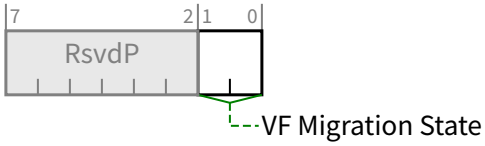

Figure 9-19 VF Migration State Entry

Table 9-9 ***VF Migration State Entry***

Bit Location	Register Description	Attributes
1:0	***VF Migration State*** - State of the associated VF. This field is undefined when VF Enable is Clear. The initial values of the VF Migration State Array are described in Section 9.2.4.1 .	RW

VF Migration State contains the values shown in Table 9-10 .

Table 9-10 VF Migration State Descriptions

VF State	VF Exists	Description
00b	No	***Inactive.Unavailable*** - VF does not exist to SR nor is it being migrated in or out.
01b	No	***Dormant.MigrateIn*** - VF is available for use by SR. VF exists but cannot initiate transactions.
10b	Yes	***Active.MigrateOut*** - SR has been requested to relinquish use of the VF.
11b	Yes	***Active.Available*** - Fully functional. Could be assigned to an SI.

The initial values of the VF Migration State Array are described in Section 9.2.4.1 . The VF Migration State Array is returned to a configuration as described in Section 9.2.4.1 within 1.0 s after VF Enable is Cleared.

Software initiates a state transition by writing a new state value to the entry.

Valid state transitions are listed in Table 9-11 and Table 9-12 and shown in Figure 9-12 . Only changes in Table 9-11 can be requested by SR. Changes in Table 9-12 are initiated by MR-PCIM using mechanisms described in the [MR-IOV] and have SR visible effects described here. Any transition issued by SR-PCIM and not listed in Table 9-11 is ignored and does not change the VF State.

Table 9-11 SR-PCIM Initiated VF Migration State Transitions

Current State	New State	Change Initiated By	SR Visible Effects of Change
Dormant.MigrateIn	Active.Available	SR-PCIM	**VF Activate** VF now exists.
Active.Available	Dormant.MigrateIn	SR-PCIM	**VF Deactivate** VF no longer exists.
Active.MigrateOut	Inactive.Unavailable	SR-PCIM	**VF Migrate Out Complete**

Current State	New State	Change Initiated By	SR Visible Effects of Change
			VF no longer exists.

Table 9-12 MR-PCIM Initiated VF Migration State Transitions

Current State	New State	Change Initiated By	SR Visible Effects of Change
Active.Available	Active.MigrateOut	MR-PCIM	**VF Migrate Out Request** VF continues to exist. Sets VF Migration Status.
Inactive.Unavailable	Dormant.MigrateIn	MR-PCIM	**VF Migrate In Request** VF remains non-existent. Sets VF Migration Status.
Dormant.MigrateIn	Inactive.Unavailable	MR-PCIM	**VF Migrate In Retract** VF remains non-existent. Sets VF Migration Status.
Active.MigrateOut	Active.Available	MR-PCIM	**VF Migrate Out Retract** VF continues to exist. Sets VF Migration Status.

9.3.4 PF/VF Configuration Space Header

This section defines the requirements on PF and VF configuration space fields.

The register definitions listed throughout this chapter establish the mapping between existing PCI-SIG specifications and the PF/VF definitions for a PCIe SR-IOV-capable device.

9.3.4.1 PF/VF Type 0 Configuration Space Header

Figure 9-20 details allocation for register fields of PCI Express Type 0 Configuration Space Header.

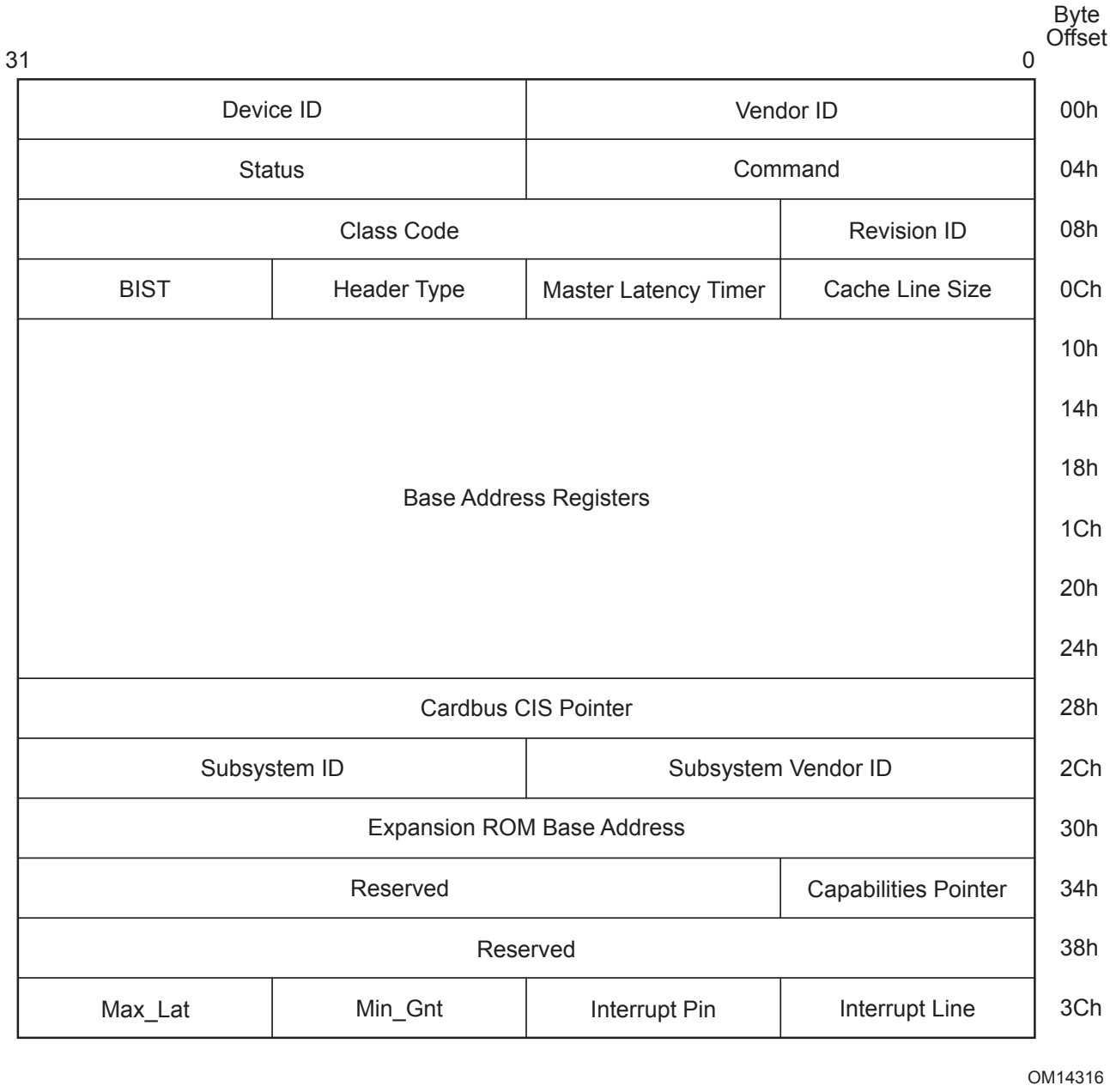

Figure 9-20 PF/VF Type 0 Configuration Space Header

The PCI Express-specific interpretation of registers specific to PCI Express Type 0 Configuration Space Header is defined in this section.

Error Reporting fields are described in more detail in Section 9.4 .

9.3.4.1.1 Vendor ID Register Changes (Offset 00h)

This Read Only register identifies the manufacturer of the Device.

This field in all VFs returns FFFFh when read. VI software should return the Vendor ID value from the associated PF as the Vendor ID value for the VF.

9.3.4.1.2 Device ID Register Changes (Offset 02h)

This Read Only register identifies the particular Device.

This field in all VFs returns FFFFh when read. VI software should return the VF Device ID (see Section 9.3.3.11) value from the associated PF as the Device ID for the VF.

IMPLEMENTATION NOTE

Legacy PCI Probing Software

Returning FFFFh for Device ID and Vendor ID values allows some legacy software to ignore VFs. See Section 7.5.1.1.1 .

9.3.4.1.3 Command Register Changes (Offset 04h)

PF and VF functionality is defined in Section 7.5.1.1.3 except where noted in Table 9-13 . For VF fields marked RsvdP, the PF setting applies to the VF.

Table 9-13 Command Register Changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
0	**I/O Space Enable** - Does not apply to VFs. Must be hardwired to 0b for VFs.	Base	0b
1	**Memory Space Enable** - Does not apply to VFs. Must be hardwired to 0b for VFs. VF Memory Space is controlled by the VF MSE bit in the VF Control register.	Base	0b
2	**Bus Master Enable** - Transactions for a VF that has its Bus Master Enable Set must not be blocked by transactions for VFs that have their Bus Master Enable Cleared.	Base	Base
6	Parity Error Response	Base	RsvdP
8	SERR# Enable	Base	RsvdP
10	**Interrupt Disable** - Does not apply to VFs.	Base	0b

9.3.4.1.4 Status Register Changes (Offset 06h)

PF and VF functionality is defined in Section 7.5.1.1.4 except where noted in Table 9-14 .

Table 9-14 Status Register Changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
3	**Interrupt Status** - Does not apply to VFs.	Base	0b

9.3.4.1.5 Revision ID Register Changes (Offset 08h)

This register specifies a device specific revision identifier.

The value reported in the VF may be different than the value reported in the PF.

9.3.4.1.6 Class Code Register Changes (Offset 09h)

The Class Code register is RO and is used to identify the generic function of the device and, in some cases, a specific register level programming interface. The field in the PF and associated VFs must return the same value when read.

9.3.4.1.7 Cache Line Size Register Changes (Offset 0Ch)

This field is implemented by PCI Express devices as a read-write field for legacy compatibility purposes but has no effect on any PCI Express device behavior. Physical Functions continue to implement this field as RW. For Virtual Functions, this field is RO Zero.

9.3.4.1.8 Latency Timer Register Changes (Offset 0Dh)

This field does not apply to PCI Express. This register must be RO Zero.

9.3.4.1.9 Header Type Register Changes (Offset 0Eh)

This byte identifies the layout of the second part of the predefined header (beginning at byte 10h in Configuration Space) and also whether or not the device contains multiple Functions. Bit 7 in this register is used to identify a Multi-Function Device. For an SR-IOV device, bit 7 in this register is only Set if there are multiple Functions. VFs do not affect the value of bit 7. Bits 6 through 0 identify the layout of the second part of the predefined header. For VFs, this register must be RO Zero.

9.3.4.1.10 BIST Register Changes (Offset 0Fh)

VFs shall not support BIST and must define this field as RO Zero.

If VF Enable is turned on in any PF of a Device, then software must not invoke BIST in any Function associated with that Device.

9.3.4.1.11 Base Address Registers Register Changes (Offset 10h, 14h, … 24h)

For VFs, the values in these registers are RO Zero.

Note: See also Section 9.2.1.1.1 and Section 9.3.3.14 .

9.3.4.1.12 Cardbus CIS Pointer Register Changes (Offset 28h)

For VFs, this register is not used and shall be RO Zero.

9.3.4.1.13 Subsystem Vendor ID Register Changes (Offset 2Ch)

This Read Only field identifies the manufacturer of the subsystem. The field in the PF and associated VFs must return the same value when read.

9.3.4.1.14 Subsystem ID Register Changes (Offset 2Eh)

This Read Only field identifies the particular subsystem. The Device may have a different value in the PF and the VF.

9.3.4.1.15 Expansion ROM Base Address Register Register Changes (Offset 30h)

The Expansion ROM Base Address Register is permitted to be implemented in PFs. The Expansion ROM Base Address Register is RO Zero in VFs. The VI may choose to provide access to the PF Expansion ROM Base Address Register for VFs via emulation.

The PF is not permitted to implement Expansion ROM address decoder sharing (see Section 7.5.1.2.4).

9.3.4.1.16 Capabilities Pointer Register Changes (Offset 34h)

No differences from the description in Section 7.5.1.1.11 .

9.3.4.1.17 Interrupt Line Register Changes (Offset 3Ch)

This field does not apply to VFs. This field must be RO Zero.

9.3.4.1.18 Interrupt Pin Register Changes (Offset 3Dh)

This field does not apply to VFs. This field must be RO Zero.

9.3.4.1.19 Min_Gnt Register/Max_Lat Register Changes (Offset 3Eh/3Fh)

These registers do not apply to PCI Express. They must be RO Zero.

9.3.5 PCI Express Capability Changes

The PCI Express Capability (see Section 7.5.3) is used for identification of a PCI Express device and indicates support for PCI Express features.

Figure 7-21 details allocation of register fields in the PCI Express Capability. PFs and VFs are required to implement this capability subject to the exceptions and additional requirements described below.

9.3.5.1 PCI Express Capabilities Register Changes (Offset 00h)

The PCI Express Capabilities Register is described in Section 7.5.3.2 and the functionality described there applies to both the PF and VF.

9.3.5.2 PCI Express Capabilities Register Changes (Offset 02h)

The PCI Express Capabilities register identifies PCI Express device type and associated capabilities.

The PF and VF functionality is defined in Section 7.5.3.2 .

9.3.5.3 Device Capabilities Register Changes (Offset 04h)

The Device Capabilities Register identifies PCI Express device specific capabilities. Figure 7-24 details allocation of register fields in the Device Capabilities Register; Table 9-15 provides the respective bit definitions.

PF and VF functionality is defined in Section 7.5.3.3 except where noted in Table 9-15 .

Table 9-15 Device Capabilities Register Changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
4:3	Phantom Functions Supported - When the VF Enable is Set, use of Phantom Function numbers by this PF and associated VFs is not permitted and this field must return 00b when read.	Base	00b
25:18	Captured Slot Power Limit Value	Base	undefined
27:26	Captured Slot Power Limit Scale	Base	undefined
28	Function Level Reset Capability - Required for PFs and for VFs.	1b	1b

9.3.5.4 Device Control Register Changes (Offset 08h)

The Device Control Register controls PCI Express device specific parameters. Figure 7-25 details allocation of register fields in the Device Control register; Table 9-16 provides the respective bit definitions.

PF and VF functionality is defined in Section 7.5.3.4 except where noted in Table 9-16 . For VF fields marked RsvdP, the PF setting applies to the VF.

Table 9-16 Device Control Register Changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
0	Correctable Error Reporting Enable See Section 9.4 for details on error reporting.	Base	RsvdP
1	Non-Fatal Error Reporting Enable	Base	RsvdP

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
	See Section 9.4 for details on error reporting.		
2	Fatal Error Reporting Enable See Section 9.4 for details on error reporting.	Base	RsvdP
3	Unsupported Request Reporting Enable See Section 9.4 for details on error reporting.	Base	RsvdP
4	Enable Relaxed Ordering	Base	RsvdP
7:5	Max_Payload_Size	Base	RsvdP
8	Extended Tag Field Enable	Base	RsvdP
9	Phantom Functions Enable	Base	RsvdP
10	Aux Power PM Enable	Base	RsvdP
11	Enable No Snoop	Base	RsvdP
14:12	Max_Read_Request_Size	Base	RsvdP
15	Initiate Function Level Reset - Required for PFs and for VFs. Note: Setting Initiate Function Level Reset in a PF resets VF Enable which means that VFs no longer exist after the FLR is complete.	Base	Base

9.3.5.5 Device Status Register Changes (Offset 0Ah)

The Device Status register provides information about PCI Express device specific parameters. Figure 7-26 details allocation of register fields in the Device Status register; Table 9-17 provides the respective bit definitions.

PF and VF functionality is defined in Section 7.5.3.5 except where noted in Table 9-17 .

Table 9-17 Device Status Register Changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
4	AUX Power Detected	Base	0b
6	Emergency Power Reduction Detected	Base	0b

9.3.5.6 Link Capabilities Register Changes (Offset 0Ch)

The Link Capabilities register identifies PCI Express Link specific capabilities. Figure 7-27 details allocation of register fields in the Link Capabilities register.

PF and VF functionality is defined in Section 7.5.3.6 .

9.3.5.7 Link Control Register Changes (Offset 10h)

The Link Control Register controls PCI Express Link specific parameters. Figure 7-28 details allocation of register fields in the Link Control register.

PF and VF functionality is defined in Section 7.5.3.7 except where noted in Table 9-18 . For VF fields marked RsvdP, the PF setting applies to the VF.

Table 9-18 Link Control Register Changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
1:0	Active State Power Management (ASPM) Control	Base	RsvdP
3	Read Completion Boundary (RCB Must be hardwired to 0b for VFs.	Base	RsvdP
6	Common Clock Configuration	Base	RsvdP
7	Extended Synch	Base	RsvdP
8	Enable Clock Power Management	Base	RsvdP
9	Hardware Autonomous Width Disable	Base	RsvdP

9.3.5.8 Link Status Register Changes (Offset 12h)

The Link Status Register provides information about PCI Express Link specific parameters. Figure 7-29 details allocation of register fields in the Link Status register.

PF functionality is defined in Section 7.5.3.8 . For the VF, all fields in this register are RsvdZ and the PF setting applies to the VF.

9.3.5.9 Device Capabilities 2 Register Changes (Offset 24h)

PF and VF functionality is defined in Section 7.5.3.15 except as noted in Table 9-19 .

Table 9-19 Device Capabilities 2 Register Changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
3:0	**Completion Timeout Ranges Supported** VF value must be identical to PF value.	Base	Base
4	**Completion Timeout Disable Supported** VF value must be identical to PF value.	Base	Base
6	**AtomicOp Routing Supported** Not applicable to Endpoints.	RsvdP	RsvdP
7	**32-bit AtomicOp Completer Supported**	Base	Base

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
	VF value must be identical to PF value.		
8	**64-bit AtomicOp Completer Supported** VF value must be identical to PF value.	Base	Base
9	**128-bit CAS Completer Supported** VF value must be identical to PF value.	Base	Base
16	**10-Bit Tag Completer Supported** VF value must be identical to PF value.	Base	Base
17	**10-Bit Tag Requester Supported** VF value must equal the value of the VF 10-Bit Tag Requester Supported bit in the SR-IOV Capabilities register. See Section 9.3.3.2.3	Base	Base
25:24	**Emergency Power Reduction Supported** VF value must be identical to PF value.	Base	Base
26	**Emergency Power Reduction Initialization Required** VF value must be identical to PF value.	Base	Base

9.3.5.10 Device Control 2 Register Changes (Offset 28h)

PF and VF functionality is defined in Section 7.5.3.16 except as noted in Table 9-20 .

Table 9-20 Device Control 2 Register Changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
3:0	**Completion Timeout Value** The PF value applies to all associated VFs.	Base	RsvdP
4	**Completion Timeout Disable** The PF value applies to all associated VFs.	Base	RsvdP
6	**AtomicOp Requester Enable** The PF value applies to all associated VFs.	Base	RsvdP
8	**IDO Request Enable** The PF value applies to all associated VFs.	Base	RsvdP
9	**IDO Completion Enable** The PF value applies to all associated VFs.	Base	RsvdP
11	**Emergency Power Reduction Request** This bit is only present in one Function associated with an Upstream Port. That Function can never be a VF.	Base	RsvdP

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
12	**10-Bit Tag Requester Enable** The value in the VF 10-Bit Tag Requester Enable bit in the SR-IOV Control register applies to all associated VFs.	Base	RsvdP

9.3.5.11 Device Status 2 Register Changes (Offset 2Ah)

PF and VF functionality is defined in Section 7.5.3.17 .

9.3.5.12 Link Capabilities 2 Register Changes (Offset 2Ch)

PF and VF functionality is defined in Section 7.5.3.18 .

9.3.5.13 Link Control 2 Register Changes (Offset 30h)

PF and VF functionality is defined in Section 7.5.3.19 .

9.3.5.14 Link Status 2 Register Changes (Offset 32h)

PF and VF functionality is defined in Section 7.5.3.20 except where noted in Table 9-21 . The VF fields marked RsvdZ use the value of the associated PF.

Table 9-21 Link Status 2 Register Changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
0	Current De-emphasis Level	Base	RsvdZ

9.3.6 PCI Standard Capabilities

SR-IOV usage of PCI Standard Capabilities is described in Table 9-22 . Items marked n/a are not applicable to PFs or VFs.

Table 9-22 SR-IOV Usage of PCI Standard Capabilities

Capability ID	Description	PF Attributes	VF Attributes
00h	Null Capability	Base	Base
01h	PCI Power Management Interface	Base	Optional. See Section 9.6 .
02h	AGP	n/a	n/a
03h	VPD	Base	Optional. See Section 9.3.6.1 .
04h	Slot Identification	n/a	n/a

Capability ID	Description	PF Attributes	VF Attributes
05h	MSI	Base	See Section 9.5.1.1 .
06h	CompactPCI Hot Swap	n/a	n/a
07h	PCI-X	n/a	n/a
08h	HyperTransport	n/a	n/a
09h	Vendor-specific	Base	Base
0Ah	Debug Port	Base	Base
0Bh	CompactPCI Central Resource Control	n/a	n/a
0Ch	PCI Hot Plug	Base	n/a
0Dh	PCI Bridge Subsystem ID	n/a	n/a
0Eh	AGP 8x	n/a	n/a
0Fh	Secure Device	n/a	n/a
10h	PCI Express	Base	See Section 9.3.5 .
11h	MSI-X	See Section 9.5.1.2 and Section 9.5.1.3 .	See Section 9.5.1.2 and Section 9.5.1.3 .
12h	Serial ATA Data/Index Configuration	Base	n/a
13h	Advanced Features	n/a	n/a
14h	Enhanced Allocation	Base	Must not implement.
15h	Flattening Portal Bridge (FPB)	n/a	n/a

9.3.6.1 VPD Capability

The VPD Capability is optional in PCI. It remains optional in SR-IOV.

VFs and PFs that implement the VPD Capability must ensure that there can be no “data leakage” between VFs and/or PFs via the VPD Capability.

9.3.7 PCI Express Extended Capabilities Changes

SR-IOV usage of PCI Express Extended Capabilities is described in Table 9-23 . Items marked n/a are not applicable to PFs or VFs (e.g., for Capabilities only present in Root Complexes or only present in Function 0).

Table 9-23 SR-IOV Usage of PCI Express Extended Capabilities

Extended Capability ID	Description	PF Attributes	VF Attributes
0000h	Null Capability	Base	Base
0001h	Advanced Error Reporting Extended Capability (AER)	Base	See Section 9.4.2 .
0002h	Virtual Channel Extended Capability (02h)	Base	Must not implement. See Section 9.3.7.1 .
0003h	Device Serial Number Extended Capability	Base	See Section 9.3.7.2
0004h	Power Budgeting Extended Capability	Base	Must not implement. See Section 9.3.7.3
0005h	Root Complex Link Declaration Extended Capability	n/a	n/a
0006h	Root Complex Internal Link Control Extended Capability	n/a	n/a
0007h	Root Complex Event Collector Endpoint Association Extended Capability	n/a	n/a
0008h	Multi-Function Virtual Channel Extended Capability	Base	Must not implement. See Section 9.3.7.1 .
0009h	Virtual Channel Extended Capability (09h)	Base	Must not implement. See Section 9.3.7.1 .
000Ah	RCRB Header Extended Capability	n/a	n/a
000Bh	Vendor-specific Extended Capability	Base	Base
000Ch	Configuration Access Correlation Extended Capability	n/a	n/a
000Dh	ACS Extended Capability	See Section 9.3.7.6 .	See Section 9.3.7.6 .
000Eh	ARI Extended Capability (ARI)	See Section 9.3.7.7 .	See Section 9.3.7.7 .
000Fh	ATS Extended Capability	See Section 9.3.7.8 .	See Section 9.3.7.8 .
0010h	SR-IOV Extended Capability	See Section 9.3.3 .	Must not implement. See Section 9.3.3
0011h	MR-IOV Extended Capability (MR-IOV)	Must not implement. See Section 9.3.7.9 .	Must not implement. See Section 9.3.7.9 .
0012h	Multicast Extended Capability	See Section 9.3.7.10 .	See Section 9.3.7.10 .
0013h	Page Request Extended Capability (PRI)	See Section 9.3.7.11 .	See Section 9.3.7.11
0014h	Reserved for AMD	Base	Base
0015h	Resizable BAR Extended Capability	Base	Must not implement. See Section 9.3.7.4
0016h	Dynamic Power Allocation Extended Capability (DPA)	See Section 9.3.7.12 .	Must not implement. See Section 9.3.7.12 .

Extended Capability ID	Description	PF Attributes	VF Attributes
0017h	TPH Requester Extended Capability (TPH)	See Section 9.3.7.13 .	See Section 9.3.7.13
0018h	LTR Extended Capability	Base	Must not implement. LTR is controlled using Function 0 which is never a VF.
0019h	Secondary PCI Express Extended Capability	Base	Must not implement. 8.0 GT/s is controlled using Function 0 which is never a VF.
001Ah	PMUX Extended Capability	Base	Must not implement. PMUX is controlled using Function 0 which is never a VF.
001Bh	PASID Extended Capability	Base	See Section 9.3.7.14
001Ch	LN Requester Extended Capability (LNR)	Base	Base
001Dh	DPC Extended Capability	n/a	n/a.
001Eh	L1 PM Substates Extended Capability	Base	Must not implement. L1 PM Substates is controlled using Function 0 which is never a VF.
001Fh	Precision Time Management Extended Capability (PTM)	Base	Must not implement. PTM controls the Port and must not be implemented in a VF.
0020h	PCI Express over M-PHY Extended Capability (M-PCIe)	Base	Must not implement. M-PHY is controlled using Function 0 which is never a VF.
0021h	FRS Queueing Extended Capability	n/a	n/a
0022h	Readiness Time Reporting Extended Capability	Base	See Section 9.3.7.15
0023h	Designated vendor-specific Extended Capability	Base	Base
0024h	VF Resizable BAR Extended Capability	See Section 9.3.7.5	Not Implemented. See Section 9.3.7.5
0025h	Data Link Feature Extended Capability	Base	Must not implement.
0026h	Physical Layer 16.0 GT/s Extended Capability	Base	Must not implement.
0027h	Lane Margining at the Receiver Extended Capability	Base	Must not implement.
0028h	Hierarchy ID Extended Capability	Base	Base
0029h	Native PCIe Enclosure Management Extended Capability (NPEM)	Base	Must not implement

9.3.7.1 Virtual Channel/MFVC

VFs use the Virtual Channels of the associated PFs. As such, VFs do not contain any Virtual Channel Capabilities.

VFs shall not contain the MFVC Capability. This Capability is only present in Function 0 which shall never be a VF.

9.3.7.2 Device Serial Number

The Device Serial Number Extended Capability may be present in PFs. If a PF contains the capability, its value applies to all associated VFs. VFs are permitted but not recommended to implement this capability. VFs that implement this capability must return the same Device Serial Number value as that reported by their associated PF.

9.3.7.3 Power Budgeting

The Power Budgeting Extended Capability may be present in PFs, but VFs must not implement it. If a PF contains the capability, it must report values that cover all associated VFs.

9.3.7.4 Resizable BAR

The Resizable BAR Extended Capability may be present in PFs. Since VFs do not implement standard BARs the capability must not be present in a VF. The PF's Resizable BAR settings do not affect any settings in the SR-IOV Extended Capability.

9.3.7.5 VF Resizable BAR Extended Capability

The VF Resizable BAR Extended Capability is permitted to be implemented only in PFs that implement at least one VF BAR, and affects the size and base of a PF's VF BARs. Since VFs do not implement the BARs themselves the capability must not be present in a VF. A PF may implement both a VF Resizable BAR Extended Capability and a Resizable BAR capability, as each capability operates independently.

The VF Resizable BAR Extended Capability is an optional capability that permits PFs to be able to have their VF's BARs resized. The VF Resizable BAR Extended Capability permits hardware to communicate the resource sizes that are acceptable for operation via the VF Resizable BAR Extended Capability and Control registers and system software to communicate the optimal size back to the hardware via the VF BAR Size field of the VF Resizable BAR Control register.

Hardware immediately reflects the size inference in the read-only bits of the appropriate VF BAR. The size inferred is the greater of the values decoded from the System Page Size and VF BAR Size fields. Hardware must Clear any bits that change from read-write to read-only, so that subsequent reads return zero. Software must clear the VF MSE bit in the SR-IOV Control register before writing the VF BAR Size field. After writing the VF BAR Size field, the contents of the corresponding VF BAR are undefined. To ensure that it contains a valid address after resizing the VF BAR, system software must reprogram the VF BAR, and Set the VF MSE bit (unless the resource is not allocated).

The VF Resizable BAR Extended Capability and Control registers are permitted to indicate the ability to operate at 4 GB or greater only if the associated VF BAR is a 64-bit BAR.

It is strongly recommended that a Function not advertise any supported VF BAR size values in this capability that are larger than the space it would effectively utilize if allocated.

IMPLEMENTATION NOTE

Using the Capability During Resource Allocation

System software uses this capability in a similar way to the Resizable BAR capability. System software must first configure the System Page Size register (see Section 9.2.1.1.1). Potential usable memory aperture sizes are reported by the PF, and read, from the VF Resizable BAR Extended Capability and Control registers. It is intended that the software allocate the largest of the reported sizes that it can, since allocating less address space than the largest reported size can result in lower performance. Software then writes the size to the VF Resizable BAR Control register for the appropriate VF BAR for the Function. Following this, the base address is written to the VF BAR.

For interoperability reasons, it is possible that hardware will set the default size of the VF BAR to a low size; a size lower than the largest reported in the VF Resizable BAR Capability Register. Software that does not use this capability to size resources will likely result in sub-optimal resource allocation, where the resources are smaller than desirable, or not allocatable because there is no room for them. It is recommended that system software responsible for allocating resources in a resource constrained environment distribute the limited address space to all memory-mapped hardware, including system RAM, appropriately.

The VF Resizable BAR Extended Capability structure defines a PCI Express Extended Capability which is located in PCI Express Extended Configuration Space, that is, above the first 256 bytes, and is shown below in Figure 9-21 . This structure allows PFs with this capability to be identified and controlled. A Capability register and a Control register are implemented for each VF BAR that is resizable. Since a maximum of 6 VF BARs may be implemented by any PF, the VF Resizable BAR Capability structure can range from 12 bytes long (for a single VF BAR) to 52 bytes long (for all 6 VF BARs).

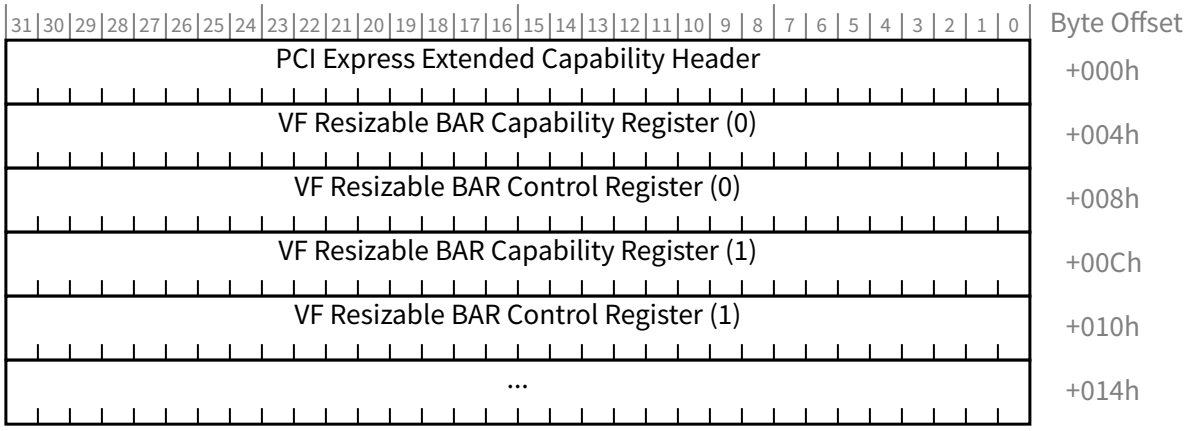

Figure 9-21 VF Resizable BAR Extended Capability

9.3.7.5.1 *VF Resizable BAR Extended Capability Header* (Offset 00h)

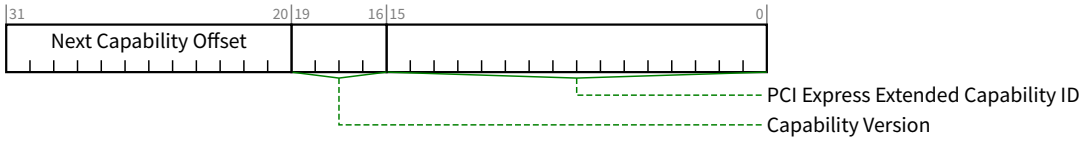

Figure 9-22 VF Resizable BAR Extended Capability Header

Table 9-24 VF Resizable BAR Extended Capability Header

Bit Location	Register Description	Attributes
15:0	***PCI Express Extended Capability ID*** - This field is a PCI-SIG defined ID number that indicates the nature and format of the extended capability. PCI Express Extended Capability ID for the VF Resizable BAR Extended Capability is 0024h.	RO
19:16	***Capability Version*** - This field is a PCI-SIG defined version number that indicates the version of the capability structure present. Must be 1h for this version of the specification.	RO
31:20	***Next Capability Offset*** - This field contains the offset to the next PCI Express Extended Capability structure or 000h if no other items exist in the linked list of capabilities	RO

9.3.7.5.2 *VF Resizable BAR Capability Register* (Offset 04h)

The VF Resizable BAR Capability Register field descriptions are the same as the definitions in the Resizable BAR Capability Register in Table 7-116 . Where those descriptions say ‘BAR’, this register’s description is for ‘VF BAR’. Where those descriptions say ‘Function’, this register’s description is for ‘PF’. Otherwise the field descriptions, the number of bits, their positions, and their attributes are the same. Consequently Figure 7-145 similarly allocates the register fields in this register.

9.3.7.5.3 *VF Resizable BAR Control Register* (Offset 08h)

The VF Resizable BAR Control register bits 31:16 follow the same definitions as the Resizable BAR Control register in Table 7-117 .

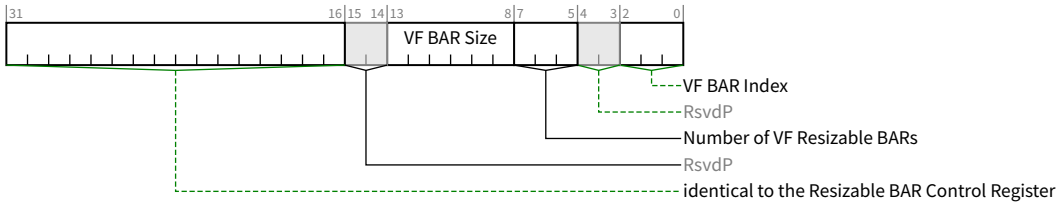

Figure 9-23 VF Resizable BAR Control Register

Table 9-25 VF Resizable BAR Control Register

Bit Location	Register Description	Attributes
2:0	***VF BAR Index*** - This encoded value points to the beginning of this particular VF BAR located in the SR-IOV Extended Capability. **0** VF BAR located at offset 24h **1** VF BAR located at offset 28h **2** VF BAR located at offset 2Ch **3** VF BAR located at offset 30h **4** VF BAR located at offset 34h **5** VF BAR located at offset 38h **others** All other encodings are reserved. For a 64-bit Base Address register, the VF BAR Index indicates the lower DWORD. This value indicates which VF BAR supports a negotiable size.	RO
7:5	***Number of VF Resizable BARs*** - Indicates the total number of resizable VF BARs in the capability structure for the Function. See Figure 9-21 . The value of this field must be in the range of 01h to 06h. The field is valid in VF Resizable BAR Control register (0) (at offset 08h), and is RsvdP for all others.	RO/RsvdP
13:8	***VF BAR Size*** - This is an encoded value. **0** 1 MB (2^{20} bytes) **1** 2 MB (2^{21} bytes) **2** 4 MB (2^{22} bytes) **3** 8 MB (2^{23} bytes) **…** … **43** 8 EB (2^{63} bytes) The default value of this field is equal to the default size of the address space that the VF BAR resource is requesting via the VF BAR's read-only bits. Software must only write values that correspond to those indicated as supported in the VF Resizable BAR Capability and Control registers. Writing an unsupported value will produce undefined results. When this register field is programmed, the value is immediately reflected in the size of the resource, as encoded in the number of read-only bits in the VF BAR.	RW

Bit Location	Register Description	Attributes
31:16	These bits are ***identical to the Resizable BAR Control Register*** bits [31:16] defined in Figure 7-146 . Where those descriptions say 'BAR', this register's description is for 'VF BAR'. Where those descriptions say 'Function', this register's description is for 'PF'.	See Figure 7-146

9.3.7.6 Access Control Services (ACS) Extended Capability Changes

ACS is an optional extended capability. If an SR-IOV Capable Device other than one in a Root Complex implements internal peer-to-peer transactions, ACS is required with additional requirements described below.

PF and VF functionality is defined in Section 7.7.8.2 except where noted in Table 9-26 .

All Functions in SR-IOV Capable Devices (Devices that implement at least one PF) other than RCiEPs that support peer-to-peer transactions within the Device shall implement ACS Egress Control.

Implementation of ACS in RCiEPs is permitted but not required. It is explicitly permitted that, within a single Root Complex, some RCiEPs implement ACS and some do not. It is strongly recommended that Root Complex implementations ensure that all accesses originating from RCiEPs (PFs and VFs) without ACS capability are first subjected to processing by the Translation Agent (TA) in the Root Complex before further decoding and processing. The details of such Root Complex handling are outside the scope of this specification.

Table 9-26 ACS Capability Register Changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
5	***ACS P2P Egress Control*** - Required to be 1b for Functions that are not RCiEPs if peer-to-peer transactions within the Device are supported.	Base	Base

IMPLEMENTATION NOTE

Access Control Services in Systems that Support Direct Assignment of Functions

General-purpose VIs typically have separate address spaces for each SI and for the VI itself. If such a VI also supports direct assignment of a Function to an SI, Untranslated Memory Request transactions issued by directly assigned Functions are under the complete control of software operating within the associated SI and typically reference the address space associated with that SI. In contrast, Memory Request transactions issued by the Host (MMIO requests) and by Functions that are not directly assigned are under the control of the VI and typically reference one or more system address spaces (e.g., the PCIe physical address space, the address space associated with the VI, or the address space associated with some designated SI). General-purpose VIs are not expected to establish a dependency between these various address spaces. Consequently, these address spaces may freely overlap which could lead to unintended routing of TLPs by Switches. For example, Upstream Memory Request TLPs originated by a directly assigned Function and intended for main memory could instead be routed to a Downstream Port if the address in the Request falls within the MMIO address region associated with that Downstream Port. Such unintended routing poses a threat to SI and/or VI stability and integrity.

To guard against this concern, vendors are strongly recommended to implement ACS in platforms that support general purpose VIs with direct assignment of Functions. Such support should include:

- In Switches or Root Complexes located below the TA, the level of ACS support should follow the guidelines established in this document for Downstream Switch Ports that implement an ACS Extended Capability structure.

  Note: Components located above the TA only see Translated Memory Requests, consequently this concern does not apply to those components.

- In SR-IOV devices that are capable of peer-to-peer transactions, ACS support is required.
- In Multi-Function Devices that are capable of peer-to-peer transactions, vendors are strongly recommended to implement ACS with ACS P2P Egress Control.

Additionally, platform vendors should test for the presence of ACS and enable it in Root Complexes and Switches on the path from the TA to a Function prior to directly assigning that Function. If the Function is peer-to-peer capable, ACS should be enabled in the Function as well.

9.3.7.7 Alternative Routing ID Interpretation Extended Capability (ARI) Changes

ARI is not applicable to RCiEPs; all other SR-IOV Capable Devices (Devices that include at least one PF) shall implement the ARI Extended Capability in each Function. PF and VF functionality is defined in Section 7.8.7.2 except where noted in Table 9-27 .

Table 9-27 ARI Capability Register Changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
0	**MFVC Function Groups Capability (M)** - Additional requirements described below.	Base	Base
1	**ACS Function Groups Capability (A)** - Additional requirements described below.	Base	Base

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
15:8	**Next Function Number** - VFs are located using First VF Offset (see Section 9.3.3.10) and VF Stride (see Section 9.3.3.11).	Base	Undefined

Any Device that implements the MFVC Capability with the optional Function Arbitration Table and consumes more than one Bus Number shall implement the MVFC Function Groups Enable (M) capability bit as 1b in Function 0.

Any Device that implements the ACS Capability with the optional Egress Control Vector and consumes more than one Bus Number shall implement the ACS Function Groups Enable (A) capability bit as 1b in Function 0.

9.3.7.8 Address Translation Services Extended Capability Changes (ATS)

ATS support is optional in SR-IOV devices. If a VF implements an ATS capability, its associated PF must implement an ATS capability. The ATS Capabilities in VFs and their associated PFs may be enabled independently.

PF and VF functionality is defined in Section 10.5.1 except where noted in Table 9-28 and Table 9-29 . Attributes shown as ATS are the same as specified in Address Translation Services (Chapter 10).

Table 9-28 ATS Capability Register

Bit Location	PF and VF Register Differences From ATS	PF Attributes	VF Attributes
4:0	**Invalidate Queue Depth** - Hardwired to 0 for VFs. Depth of shared PF input queue.	ATS	RO

Table 9-29 ATS Control Register Changes

Bit Location	PF and VF Register Differences From ATS	PF Attributes	VF Attributes
4:0	**Smallest Translation Unit (STU)** - Hardwired to 0 for VFs. PF value applies to all VFs.	ATS	RO

ATS behavior is specified in Address Translation Services (Chapter 10). ATS requests target the Function being invalidated. ATS responses will have a Routing ID field matching the targeted Function. Each Function is required to implement sufficient queuing to ensure it can hold the maximum number of outstanding Invalidation Requests from a TA (using either input or output queuing).

However, all VFs associated with a PF share a single input queue in the PF. To implement Invalidation flow control, the TA must ensure that the total number of outstanding Invalidate Requests to the shared PF queue (targeted to the PF and its associated VFs) does not exceed the value in the PF Invalidate Queue Depth field.

9.3.7.9 MR-IOV Changes

PFs and VFs shall not contain an MR-IOV Capability.

If present, the MR-IOV Capability is contained in other Functions of the Component. See [MR-IOV] for details.

9.3.7.10 Multicast Changes

Multicast support is optional in SR-IOV devices. If a VF implements a Multicast capability, its associated PF must implement a Multicast capability. PF and VF functionality is defined in Section 7.9.11 except where noted in Table 9-30, Table 9-31, and Table 9-32.

Table 9-30 Multicast Capability Register Changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
5:0	**MC_Max_Group** - PF value applies to all VFs.	Base	RsvdP
13:8	**MC_Window_Size_Requested** - PF value applies to all VFs.	Base	RsvdP
15	**MC_ECRC_Regeneration_Supported** - Not applicable for Endpoints.	Base	RsvdP

Table 9-31 Multicast Control Register Changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
5:0	**MC_Num_Group** - PF value applies to all VFs.	Base	RsvdP

Table 9-32 Multicast Base Address Register Changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
5:0	**MC_Index_Position** - PF value applies to all VFs.	Base	RsvdP
63:12	**MC_Base_Address** - PF value applies to all VFs.	Base	RsvdP

9.3.7.11 Page Request Interface Changes (PRI)

Page Request Interface functionality is defined in Address Translation Services (Chapter 10).

A PF is permitted to support the Page Request Interface. The Page Request Interface of the PF is also used by the associated VFs. The PF is permitted to implement the Page Request Interface capability and the VFs shall not implement it.

Even though the Page Request Interface is shared between PFs and VFs, it sends the requesting Function's ID (PF or VF) in the Requester ID field of the Page Request Message and expects the requesting Function's ID in the Destination Device ID field of the resulting PRG Response Message.

9.3.7.12 Dynamic Power Allocation Changes (DPA)

VFs may not implement the Dynamic Power Allocation Capability.

Power allocation for VFs is managed using the PF's DPA Capability, if implemented.

9.3.7.13 TPH Requester Changes (TPH)

The TPH Requester Extended Capability may be present in VFs, PFs, both or neither. If any VF associated with a given PF contains the capability, all VFs associated with that PF must also support TPH.

All TPH Requester Extended Capability fields in PFs and VFs operate as defined in Section 7.9.13 .

For fields in the TPH Requester Capability Register (offset 04h), a PF and its associated VFs may have different values, but all VFs associated with the same PF must have the same values in all fields.

9.3.7.14 PASID Changes

An Endpoint device is permitted to support PASID. The PASID configuration of the single function (Function or PF) representing the device is also used by all VFs in the device. A PF is permitted to implement the PASID capability, but VFs must not implement it.

Even though the PASID configuration is shared between Functions, PFs and VFs, the device sends the requesting Function's ID (Function, PF or VF) in the Requester ID field of the TLP containing PASID.

9.3.7.15 Readiness Time Reporting Extended Capability Changes

The Readiness Time Reporting Extended Capability may be present in VFs, PFs, both or neither. If any VF associated with a given PF contains the capability, all VFs associated with that same PF must also support Readiness Time Reporting.

The Reset Time field contains the time required following setting of VF Enable (see Section 9.6.1).

The DL_Up Time field is RsvdP.

All VFs associated with the same PF shall report the same time values.

9.4 SR-IOV Error Handling

SR-IOV devices utilize the Error Reporting mechanism defined in Section 6.2 . Errors that are defined as non-function-specific are only logged in the PF.

Section 6.2 defines two error reporting paradigms: the baseline Capability and the Advanced Error Reporting Capability. The baseline error reporting capabilities are required of all PCI Express devices and define the minimum error reporting requirements. The Advanced Error Reporting Capability is optional and is defined for more robust error reporting and is implemented with a specific PCI Express Capability structure.

9.4.1 Baseline Error Reporting

All SR-IOV devices must support the baseline error reporting capabilities, with some modifications to account for the goal of reduced cost and complexity of implementation.

These control bits are only meaningful in the PF. VFs shall use the error reporting control bits in the associated PF when making decisions on generating error Messages.

These following fields are RsvdP in VFs:

- Command register (see Section 9.3.4.1.3)
  - SERR# Enable
  - Parity Error Response
- Device Control register (see Section 9.3.5.4)
  - Correctable Reporting Enable
  - Non-Fatal Reporting Enable
  - Fatal Reporting Enable
  - Unsupported Request (UR) Reporting Enable

Each VF shall implement a mechanism to provide error status independent of any other Function. This is necessary to provide SI isolation for errors that are Function-specific.

The following baseline error reporting status bits must be implemented in each VF:

- Status register (see Section 9.3.4.1.4)
  - Master Data Parity Error
  - Signaled Target Abort
  - Received Target Abort
  - Received Master Abort
  - Signaled System Error
  - Detected Parity Error
- Device Status register (see Section 9.3.5.5)
  - Correctable Error Detected
  - Non-Fatal Error Detected
  - Fatal Error Detected
  - Unsupported Request Detected

Each VF shall use its own Routing ID when signaling errors.

9.4.2 Advanced Error Reporting

The Advanced Error Reporting Capability is optional. If AER is not implemented in the PF, it must not be implemented in the associated VFs. If AER is implemented in the PF, it is optional in the VFs.

Section 6.2.4 classifies errors as Function-specific and non-Function-specific. Each VF that implements the Advanced Error Reporting Capability must maintain its own error reporting status for function-specific errors.

9.4.2.1 VF Header Log

A Device that implements AER in the VFs may share Header Log Registers among VFs associated with a single PF. See Section 9.4.2.10 for details.

Header logging Registers for the PF are independent of its associated VFs and must be implemented with dedicated storage space.

When implementing a reduced set of Header Log Registers, a Function may not have room to log a header associated with an error. In this case, the Function shall update the Uncorrectable Error Status Register and Advanced Error Capabilities and Control register as required by Section 6.2.4 ; however, when the Header Log Register is read, it shall return all 1s to indicate an overflow condition and no header was logged.

9.4.2.2 Advanced Error Reporting Capability Changes

Figure 7-122 describes the AER extended capability structure.

9.4.2.3 Advanced Error Reporting Extended Capability Header Changes (Offset 00h)

This register contains the PCI Express Extended Capability ID, Capability Version, and Next Capability Offset. These fields are unchanged and are described in Section 7.8.4.1 .

9.4.2.4 Uncorrectable Error Status Register Changes (Offset 04h)

The Uncorrectable Error Status register indicates error detection status of individual errors. Errors that are defined as non-Function-specific are logged in the PF. Only Function-specific errors are logged in the VFs.

PF and VF functionality is defined in Section 7.8.4.2 except where noted in Table 9-33 .

Table 9-33 Uncorrectable Error Status Register Changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
4	Data Link Protocol Error Status	Base	0b
5	Surprise Down Error Status	Base	0b
13	Flow Control Protocol Error Status	Base	0b
17	Receiver Overflow Status	Base	0b
18	Malformed TLP Status	Base	0b
19	ECRC Error Status	Base	0b

9.4.2.5 Uncorrectable Error Mask Register Changes (Offset 08h)

PF and VF functionality is defined in Section 7.8.4.3 except where noted in Table 9-34 . For VF fields marked RsvdP, the PF setting applies to the VF. For VF fields marked 0b, the error is not applicable to a VF.

Table 9-34 Uncorrectable Error Mask Register Changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
4	Data Link Protocol Error Mask	Base	0b
5	Surprise Down Error Mask	Base	0b
12	Poisoned TLP Received Mask	Base	RsvdP

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
13	Flow Control Protocol Error Mask	Base	0b
14	Completion Timeout Mask	Base	RsvdP
15	Completer Abort Mask	Base	RsvdP
16	Unexpected Completion Mask	Base	RsvdP
17	Receiver Overflow Mask	Base	0b
18	Malformed TLP Mask	Base	0b
19	ECRC Error Mask	Base	0b
20	Unsupported Request Error Mask	Base	RsvdP
21	ACS Violation Mask	Base	RsvdP

9.4.2.6 Uncorrectable Error Severity Register Changes (Offset 0Ch)

PF and VF functionality is defined in Section 7.8.4.4 except where noted in Table 9-35 . For VF fields marked RsvdP, the PF setting applies to the VF. For VF fields marked 0b, the error is not applicable to a VF.

Table 9-35 Uncorrectable Error Severity Register Changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
4	Data Link Protocol Error Severity	Base	0b
5	Surprise Down Error Severity	Base	0b
12	Poisoned TLP Received Severity	Base	RsvdP
13	Flow Control Protocol Error Severity	Base	0b
14	Completion Timeout Error Severity	Base	RsvdP
15	Completer Abort Error Severity	Base	RsvdP
16	Unexpected Completion Error Severity	Base	RsvdP
17	Receiver Overflow Severity	Base	0b
18	Malformed TLP Severity	Base	0b
19	ECRC Error Severity	Base	0b
20	Unsupported Request Error Severity	Base	RsvdP
21	ACS Violation Severity	Base	RsvdP

9.4.2.7 Correctable Error Status Register Changes (Offset 10h)

The Correctable Error Status register indicates error detection status of individual correctable errors. Errors that are defined as non-Function-specific are logged in the PF. Only Function-specific errors are logged in the VFs.

PF and VF functionality is defined in Section 7.8.4.5 except where noted in Table 9-36 .

Table 9-36 Correctable Error Status Register Changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
0	Receiver Error Status	Base	0b
6	Bad TLP Status	Base	0b
7	Bad DLLP Status	Base	0b
8	REPLAY_NUM Rollover Status	Base	0b
12	Replay Timer Timeout Status	Base	0b
15	Header Log Overflow Status If the VF implements Header Log sharing (see Section 9.4.2.1), this bit is hardwired to 0b.	Base	Base / 0b

9.4.2.8 Correctable Error Mask Register Changes (Offset 14h)

PF and VF functionality is defined in Section 7.8.4.6 except where noted in Table 9-37 . For VF fields marked RsvdP, the PF setting applies to the VF.

Table 9-37 Correctable Error Mask Register Changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
0	Receiver Error Mask	Base	RsvdP
6	Bad TLP Mask	Base	RsvdP
7	Bad DLLP Mask	Base	RsvdP
8	REPLAY_NUM Rollover Mask	Base	RsvdP
12	Replay Timer Timeout Mask	Base	RsvdP
13	Advisory Non-Fatal Error Mask	Base	RsvdP
15	Header Log Overflow Mask - If the VF implements Header Log sharing (see Section 9.4.2.1), this bit is RsvdP.	Base	Base / RsvdP

9.4.2.9 Advanced Error Capabilities and Control Register Changes (Offset 18h)

PF and VF functionality is defined in Section 7.8.4.7 except where noted in Table 9-38 . For VF fields marked RsvdP, the PF setting applies to the VF.

Table 9-38 Advanced Error Capabilities and Control Register Changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
6	ECRC Generation Enable	Base	RsvdP
8	ECRC Check Enable	Base	RsvdP
9	Multiple Header Recording Capable - If the VF implements Header Log sharing (see Section 9.4.2.1), this bit is hardwired to 0b.	Base	Base / 0b
10	Multiple Header Recording Enable - If the VF implements Header Log sharing (see Section 9.4.2.1), this bit is RsvdP.	Base	Base / RsvdP
11	TLP Prefix Log Present - If the VF implements Header Log Sharing (see Section 9.4.2.1), this bit is 0b when the Header Log contains all 1s due to an overflow condition.	Base	Base (see description)

9.4.2.10 Header Log Register Changes (Offset 1Ch)

The Header Log register captures the header for the TLP corresponding to a detected error. See also Section 9.4.2.1 .

A Device that implements AER in the VFs may share Header Log Registers among VFs associated with a single PF. A shared header log must have storage for at least one header.

Header logging Registers for the PF is independent of its associated VFs and must be implemented with dedicated storage space.

When an error is detected in a VF, the error shall be logged as specified in Section 6.2 . If a shared set of Header Log Registers is implemented, a VF may not have room to log a header. In this case, the VF shall update its Uncorrectable Error Status Register and Advanced Error Capabilities and Control register as required in Section 6.2 ; however, when that VF's Header Log Register is read, it shall return all 1s to indicate an overflow condition.

The VF's header log entry shall be locked and remain valid while that VF's First Error Pointer is valid. As defined in Section 6.2 , the First Error Pointer register is valid when the corresponding bit of the Uncorrectable Error Status register is Set. While the header log entry is locked, additional errors shall not overwrite the locked entry for this or any other VF. When a header entry is unlocked, it shall be available to record a new error for any VF sharing the header logs.

PF and VF functionality is defined in Section 7.8.4.8 , except where noted in Table 9-39 .

Table 9-39 Header Log Register changes

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
127:0	Header of TLP associated with error (additional requirements are described above)	Base	Base

9.4.2.11 Root Error Command Register Changes (Offset 2Ch)

This register is not applicable to Devices.

9.4.2.12 Root Error Status Register Changes (Offset 30h)

This register is not applicable to Devices.

9.4.2.13 Error Source Identification Register Changes (Offset 34h)

This register is not applicable to Devices.

9.4.2.14 TLP Prefix Log Register Changes (Offset 38h)

For PFs and VFs, if End-End TLP Prefixes are supported, this register is implemented.

For a VF, if a shared set of Header Log registers is implemented (Section 9.4.2.1), this register's contents are undefined when the Header Log contains all 1s due to an overflow condition.

9.5 SR-IOV Interrupts

SR-IOV capable devices utilize the same Interrupt signaling mechanisms defined in Section 6.1 .

9.5.1 Interrupt Mechanisms

There are three methods of signaling interrupts:

- INTx
- MSI
- MSI-X

PFs may implement INTx. VFs must not implement INTx. PFs and VFs shall implement MSI or MSI-X or both if interrupt resources are requested. Each PF and VF must implement its own unique interrupt capabilities.

9.5.1.1 MSI Interrupts

MSI Capability and PF and VF functionality are defined in Section 7.7 except where noted in Table 9-40 .

Table 9-40 MSI Capability: Message Control

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
8	Per-Vector Masking Capable	1b	1b

9.5.1.2 MSI-X Interrupts

The MSI-X Capability is defined in Section 7.7 and depicted in Figure 9-24 .

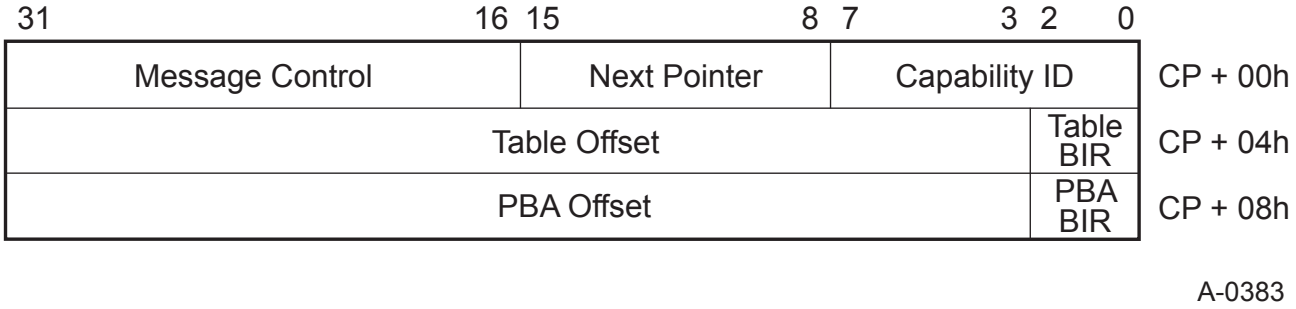

Figure 9-24 MSI-X Capability

PF and VF functionality is identical to that of a Function as defined in Section 7.7.2 .

Note that for VFs the Table Offset and PBA Offset values are relative to the VF's Memory address space.

9.5.1.3 Address Range Isolation

If a BAR that maps address space for the MSI-X Table or MSI-X PBA also maps other usable address space that is not associated with MSI-X structures, locations (e.g., for CSRs) used in the other address space must not share any naturally aligned System Page Size address range with one where either MSI-X structure resides. The MSI-X Table and MSI-X PBA are permitted to co-reside within a naturally aligned System Page Size address range, though they must not overlap with each other.

9.6 SR-IOV Power Management

This section defines the PCI Express SR-IOV power management capabilities and protocols.

The Power Management Capability is required for PFs as described in Chapter 5 .

For VFs, the Power Management Capability is optional.

9.6.1 VF Device Power Management States

If a VF does not implement the Power Management Capability, then the VF behaves as if it had been programmed into the equivalent power state of its associated PF.

If a VF implements the Power Management Capability, the functionality is defined in Section 7.5 except as noted in Section 9.6.4 .

If a VF implements the Power Management Capability, the Device behavior is undefined if the PF is placed in a lower power state than the VF. Software should avoid this situation by placing all VFs in lower power state before lowering their associated PF's power state.

A VF in the D0 state is in the $D0_{active}$ state when the VF has completed its internal initialization and either the VF's Bus Master Enable bit is Set (see Section 9.3.4.1.3) or the VF MSE bit in the SR-IOV Control (see Section 9.3.3.3) Extended Capability is Set. The VF's internal initialization must have completed whenany of the following conditions have occurred:

- The VF has responded successfully (without returning CRS) to a Configuration Request.
- After issuing an FLR to the VF, one of the following is true:
  - At least 1.0 s has passed since the FLR was issued.
  - The VF supports Function Readiness Status and, after the FLR was issued, an FRS Message from the VF with Reason Code FLR Completed has been received.
  - At least FLR time has passed since the FLR was issued. FLR Time is either (1) the FLR Time value in the Readiness Time Reporting capability associated with the VF or (2) a value determined by system software / firmware[161] .
- After Setting VF Enable in a PF, at least one of the following is true:
  - At least 1.0 s has passed since VF Enable was Set.
  - The PF supports Function Readiness Status and, after VF Enable was Set, an FRS Message from the PF with Reason Code VF Enabled has been received.
- After transitioning a VF from $D3_{Hot}$ to D0, at least one of the following is true:
  - At least 10 ms has passed since the request to enter D0 was issued.
  - The VF supports Function Readiness Status and, after the request to enter D0 was issued, an FRS Message from the VF with Reason Code $D3_{Hot}$ to D0 Transition Completed has been received.
  - At least $D3_{Hot}$ to D0 Time has passed since the request to enter D0 was issued. $D3_{Hot}$ to D0 Time is either (1) the $D3_{Hot}$ to D0 Time in the Readiness Time Reporting capability associated with the VF or (2) a value determined by system software / firmware[162] .

9.6.2 PF Device Power Management States

The PF's power management state (D-state) has global impact on its associated VFs. If a VF does not implement the Power Management Capability, then it behaves as if it is in an equivalent power state of its associated PF.

If a VF implements the Power Management Capability, the Device behavior is undefined if the PF is placed in a lower power state than the VF. Software should avoid this situation by placing all VFs in lower power state before lowering their associated PF's power state.

When the PF is placed into the $D3_{Hot}$ state:

- If the No_Soft_Reset bit is Clear then the PF performs an internal reset on the $D3_{Hot}$ to D0 transition and all its configuration state returns to the default values.

  Note: Resetting the PF resets VF Enable which means that VFs no longer exist and any VF specific context is lost after the $D3_{Hot}$ to D0 transition is complete.
- If the No_Soft_Reset bit is Set then the internal reset does not occur. The SR-IOV extended capability retains state, and associated VFs remain enabled.

When the PF is placed into the $D3_{Cold}$ state VFs no longer exist, any VF specific context is lost and PME events can only be initiated by the PF.

161. For example, ACPI tables.
162. For example, ACPI tables.

IMPLEMENTATION NOTE

No_Soft_Reset Strongly Recommended

It is strongly recommended that the No_Soft_Reset bit be Set in all Functions of a Multi-Function Device. This recommendation applies to PFs.

9.6.3 Link Power Management State

VF Power State does not affect Link Power State.

Link Power State is controlled solely by the setting in the PFs regardless of the VF's D-state.

9.6.4 VF Power Management Capability

The following tables list the requirements for PF and VFs Power Management Capability.

PF and VF functionality is defined in Section 7.5 except where noted in Table 9-41 and Table 9-42 .

Table 9-41 SR-IOV Power Management Control/Status (PMCSR)

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
14:13	Data_Scale	Base	00b
12:9	Data_Select	Base	0000b
3	**No_Soft_Reset** -If a VF implements the Power Management Capability, the VF's value of this field must be identical to the associated PF's value.	Base	Base

Table 9-42 SR-IOV Power Management Data Register

Bit Location	PF and VF Register Differences From Base	PF Attributes	VF Attributes
7:0	Data	Base	00000000b

9.6.5 VF EmergencyPower Reduction State

If the Emergency Power Reduction Supported field in a VF is non-zero, that VF enters and exits the Emergency Power Reduction State at the same time as the associated PF. Software can use the Emergency Power Reduction Detected bit in the PF to emulate the corresponding bit in the VF.

ATS Specification

10.

10.1 ATS Architectural Overview

Most contemporary system architectures make provisions for translating addresses from DMA (bus mastering) I/O Functions. In many implementations, it has been common practice to assume that the physical address space seen by the CPU and by an I/O Function is equivalent. While in others, this is not the case. The address programmed into an I/O Function is a "handle" that is processed by the Root Complex (RC). The result of this processing is often a translation to a physical memory address within the central complex. Typically, the processing includes access rights checking to insure that the DMA Function is allowed to access the referenced memory location(s).

The purposes for having DMA address translation vary and include:

- Limiting the destructiveness of a "broken" or miss-programmed DMA I/O Function
- Providing for scatter/gather
- Ability to redirect message-signaled interrupts (e.g., MSI or MSI-X) to different address ranges without requiring coordination with the underlying I/O Function
- Address space conversion (32-bit I/O Function to larger system address space)
- Virtualization support

Irrespective of the motivation, the presence of DMA address translation in the host system has certain performance implications for DMA accesses.

Depending on the implementation, DMA access time can be significantly lengthened due to the time required to resolve the actual physical address. If an implementation requires access to a main-memory-resident translation table, the access time can be significantly longer than the time for an untranslated access. Additionally, if each transaction requires multiple memory accesses (e.g., for a table walk), then the memory transaction rate (i.e., overhead) associated with DMA can be high.

To mitigate these impacts, designs often include address translation caches in the entity that performs the address translation. In a CPU, the address translation cache is most commonly referred to as a translation look-aside buffer (TLB). For an I/O TA, the term address translation cache or ATC is used to differentiate it from the translation cache used by the CPU.

While there are some similarities between TLB and ATC, there are important differences. A TLB serves the needs of a CPU that is nominally running one thread at a time. The ATC, however, is generally processing requests from multiple I/O Functions, each of which can be considered a separate thread. This difference makes sizing an ATC difficult depending upon cost models and expected technology reuse across a wide range of system configurations.

The mechanisms described in this specification allow an I/O Device to participate in the translation process and provide an ATC for its own memory accesses. The benefits of having an ATC within a Device include:

- Ability to alleviate TA resource pressure by distributing address translation caching responsibility (reduced probability of "thrashing" within the TA)
- Enable ATC Devices to have less performance dependency on a system's ATC size
- Potential to ensure optimal access latency by sending pretranslated requests to central complex

This specification will provide the interoperability that allows PCIe Devices to be used in conjunction with a TA, but the TA and its Address Translation and Protection Table (ATPT) are treated as implementation-specific and are outside the scope of this specification. While it may be possible to implement ATS within other PCIe Components, this specification is confined to PCIe Devices and PCIe Root Complex Integrated Endpoints (RCiEPs).

Figure 10-1 illustrates an example platform with a TA and ATPT, along with a set of PCIe Devices and RC Integrated Endpoints with integrated ATC. A TA and an ATPT are implementation-specific and can be distinct or integrated components within a given system design.

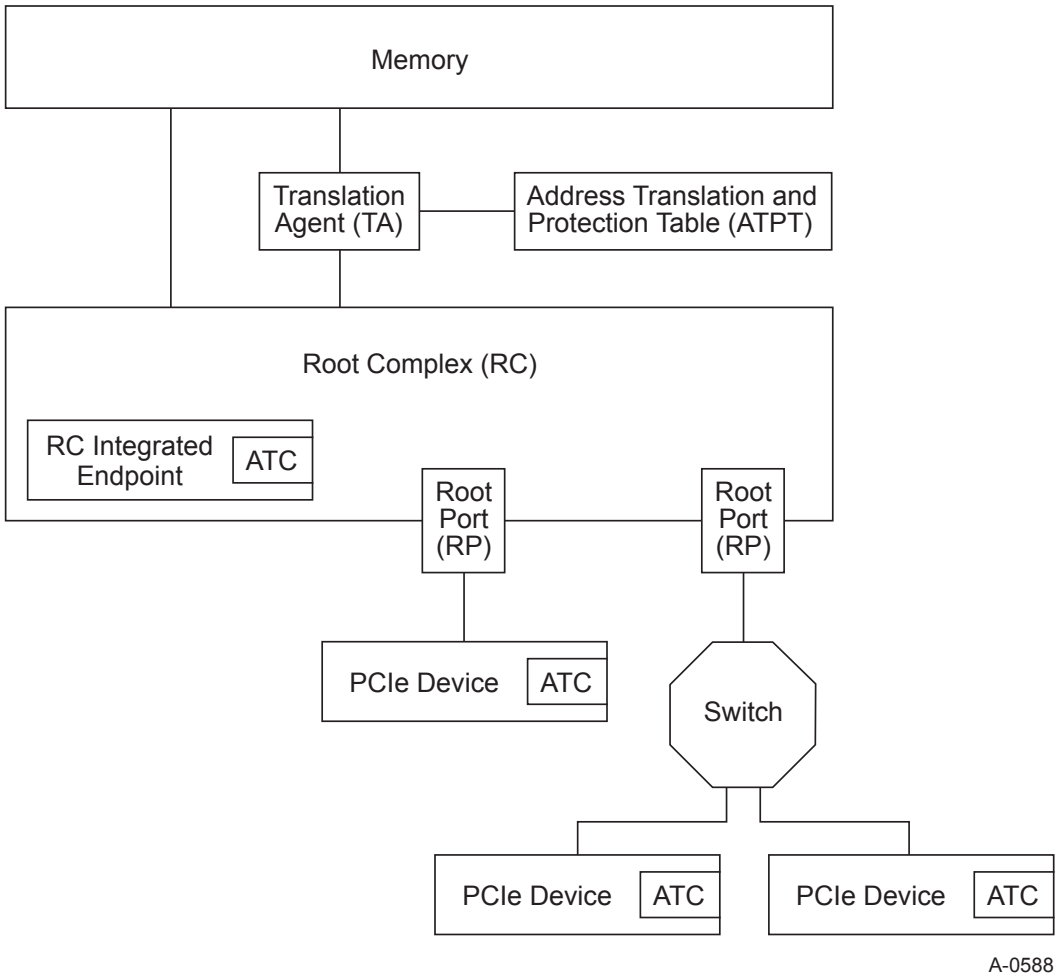

Figure 10-1 Example Illustrating a Platform with TA, ATPT, and ATC Elements

10.1.1 Address Translation Services (ATS) Overview

The ATS chapter provides a new set of TLP and associated semantics. ATS uses a request-completion protocol between a Device[163] and a Root Complex (RC) to provide translation services. In addition, a new AT field is defined within the Memory Read and Memory Write TLP. The new AT field enables an RC to determine whether a given request has been translated or not via the ATS protocol.

163. All references within this chapter to a Device apply equally to a PCIe Device or a Root Complex Integrated Endpoint. ATS does not delineate between these two types in terms of requirements, semantics, configuration, error handling, etc. From a software perspective, an ATS-capable Root Complex Integrated Endpoint must behave the same as an ATS-capable non-integrated Device.

Figure 10-2 illustrates the basic flow of an ATS Translation Request operation.

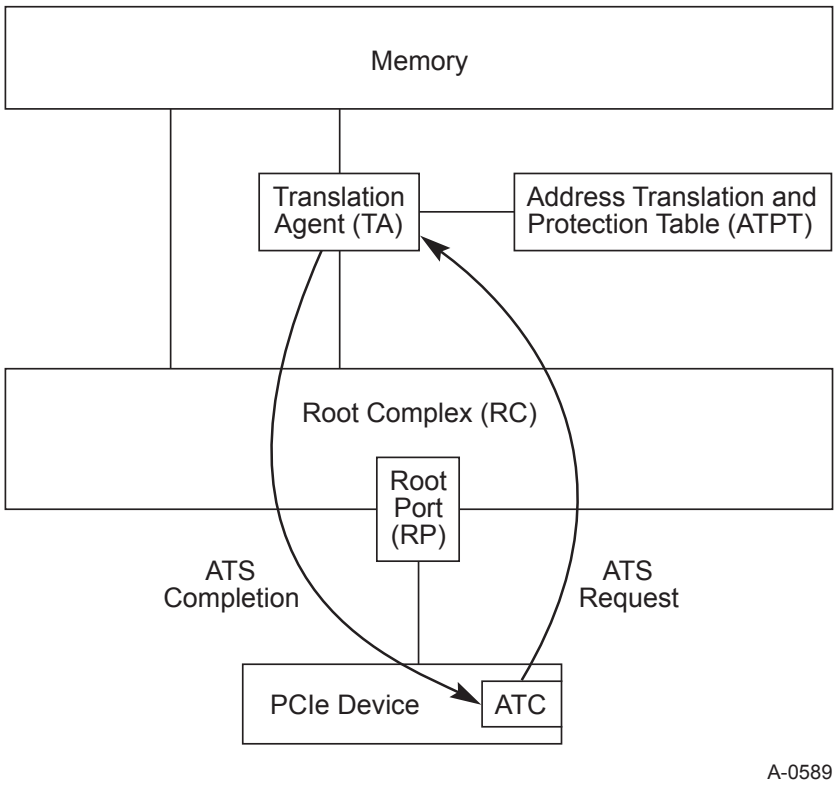

Figure 10-2 Example ATS Translation Request/Completion Exchange

In this example, a Function-specific work request is received by a single-Function PCIe Device. The Function determines through an implementation-specific method that caching a translation within its ATC would be beneficial. There are a number of considerations a Function or software can use in making such a determination; for example:

- Memory address ranges that will be frequently accessed over an extended period of time or whose associated buffer content is subject to a significant update rate
- Memory address ranges, such as work and completion queue structures, data buffers for low-latency communications, graphics frame buffers, host memory that is used to cache Function-specific content, and so forth

Given the variability in designs and access patterns, there is no single criteria that can be applied.

The Function generates an ATS Translation Request which is sent upstream through the PCIe hierarchy to the RC which then forwards it to the TA. An ATS Translation Request uses the same routing and ordering rules as defined in this specification. Further, multiple ATS Translation Requests can be outstanding at any given time; i.e., one may pipeline multiple requests on one or more TC. Each TC represents a unique ordering domain and defines the domain that must be used by the associated ATS Translation Completion.

Upon receipt of an ATS Translation Request, the TA performs the following basic steps:

1. Validates that the Function has been configured to issue ATS Translation Requests.
2. Determines whether the Function may access the memory indicated by the ATS Translation Request and has the associated access rights.

3. Determines whether a translation can be provided to the Function. If yes, the TA issues a translation to the Function.
    a. ATS is required to support a variety of page sizes to accommodate a range of ATPT and processor implementations.
        i. Page sizes are required to be a power of two and naturally aligned.
        ii. The minimum supported page size is 4096 bytes. ATS capable components are required to support this minimum page size.
    b. A Function must be informed of the minimum translation or invalidate size it will be required to support to provide the Function an opportunity to optimize its resource utilization. The smallest minimum translation size must be 4096 bytes.
4. The TA communicates the success or failure of the request to the RC which generates an ATS Translation Completion and transmits via a Response TLP through a RP to the Function.
    a. An RC is required to generate at least one ATS Translation Completion per ATS Translation Request; i.e., there is minimally a 1:1 correspondence independent of the success or failure of the request.
        i. A successful translation can result in one or two ATS Translation Completion TLPs per request. The Translation Completion indicates the range of translation covered.
        ii. An RC may pipeline multiple ATS Translation Completions; i.e., an RC may return multiple ATS Translation Completions and these ATS Translation Completions may be in any order relative to ATS Translation Requests.
        iii. The RC is required to transmit the ATS Translation Completion using the same TC (Traffic Class) as the corresponding ATS Translation Request.
    b. The requested address may not be valid. The RC is required to issue a Translation Completion indicating that the requested address is not accessible.

When the Function receives the ATS Translation Completion and either updates its ATC to reflect the translation or notes that a translation does not exist. The Function proceeds with processing its work request and generates subsequent requests using either a translated address or an untranslated address based on the results of the Completion.

a. Similar to Read Completions, a Function is required to allocate resource space for each completion(s) without causing backpressure on the PCIe Link.
b. A Function is required to discard Translation Completions that might be “stale”. Stale Translation Completions can occur for a variety of reasons.

As one can surmise, ATS Translation Request and Translation Completion processing is conceptually similar and, in many respects, identical to PCIe Read Request and Read Completion processing. This is intentional to reduce design complexity and to simplify integration of ATS into existing and new PCIe-based solutions. Keeping this in mind, ATS requires the following:

- ATS capable components must interoperate with [PCIe-1.1] compliant components.
- ATS is enabled through a new Capability and associated configuration structure. To enable ATS, software must detect this Capability and enable the Function to issue ATS TLP. If a Function is not enabled, the Function is required not to issue ATS Translation Requests and is required to issue all DMA Read and Write Requests with the TLP AT field set to “untranslated”.
- ATS TLPs are routed using either address-based or Requester ID (RID) routing.
- ATS TLPs are required to use the same ordering rules as specified in this specification.
- ATS TLPs are required to flow unmodified through [PCIe-1.1] compliant Switches.
- A Function is permitted to intermix translated and untranslated requests.

- ATS transactions are required not to rely upon the address field of a memory request to communicate additional information beyond its current use as defined by the PCI-SIG.

IMPLEMENTATION NOTE
Address Range Overlap

It is likely that the untranslated and translated address range will overlap, perhaps in their entirety. This is not a requirement of ATS but may be an implementation constraint on the TA so that memory requests will be properly routed.

In contrast to the prior example, Figure 10-3 illustrates an example Multi-Function Device. In this example Device, there are three Functions. Key points to note in Figure 10-3 are:

- Each ATC is associated with a single Function. Each ATS-capable Function must be able to source and sink at least one of each ATS Translation Request or Translation Completion type.
- Each ATC is configured and accessed on a per Function basis. A Multi-Function Device is not required to implement ATS on every Function.
- If the ATC implementation shares resources among a set of Functions, then the logical behavior is required to be consistent with fully independent ATC implementations.

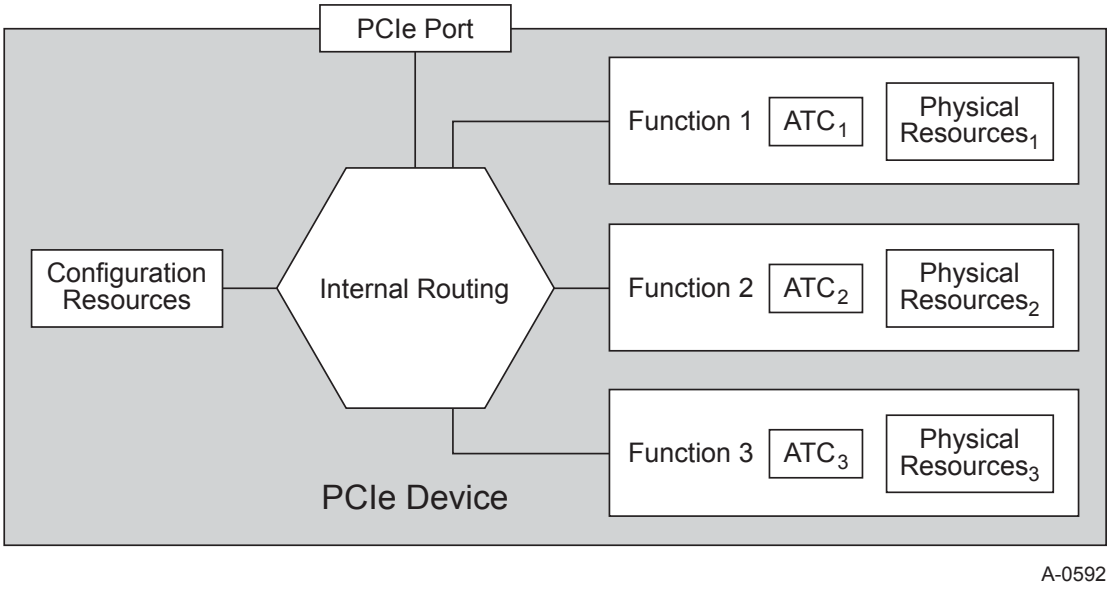

Figure 10-3 Example Multi-Function Device with ATC per Function

Independent of the number of Functions within a Device, the following are required:

- A Function is required not to issue any TLP with the AT field set unless the address within the TLP was obtained through the ATS Translation Request and Translation Completion protocol.

- Each ATC is required to only be populated using the ATS protocol; i.e., each entry within the ATC must be filled via an ATS Translation Completion in response to the Function issuing an ATS Translation Request for a given address.
- Each ATC cannot be modified except through the ATS protocol. That is:
  - Host system software cannot modify the ATC other than through the protocols defined in this specification except to invalidate one or more translations in an ATC. A Device or Function reset would be an example of an operation performed by software to change the contents of the ATC, but a reset is only allowed to invalidate entries not modify their contents.
  - It must not be possible for host system software to use software executing on the Device to modify the ATC.

When a TA determines that a Function should no longer maintain a translation within its ATC, the TA initiates the ATS invalidation protocol. The invalidation protocol consists of a single Invalidation Request and one or more Invalidate Completions.

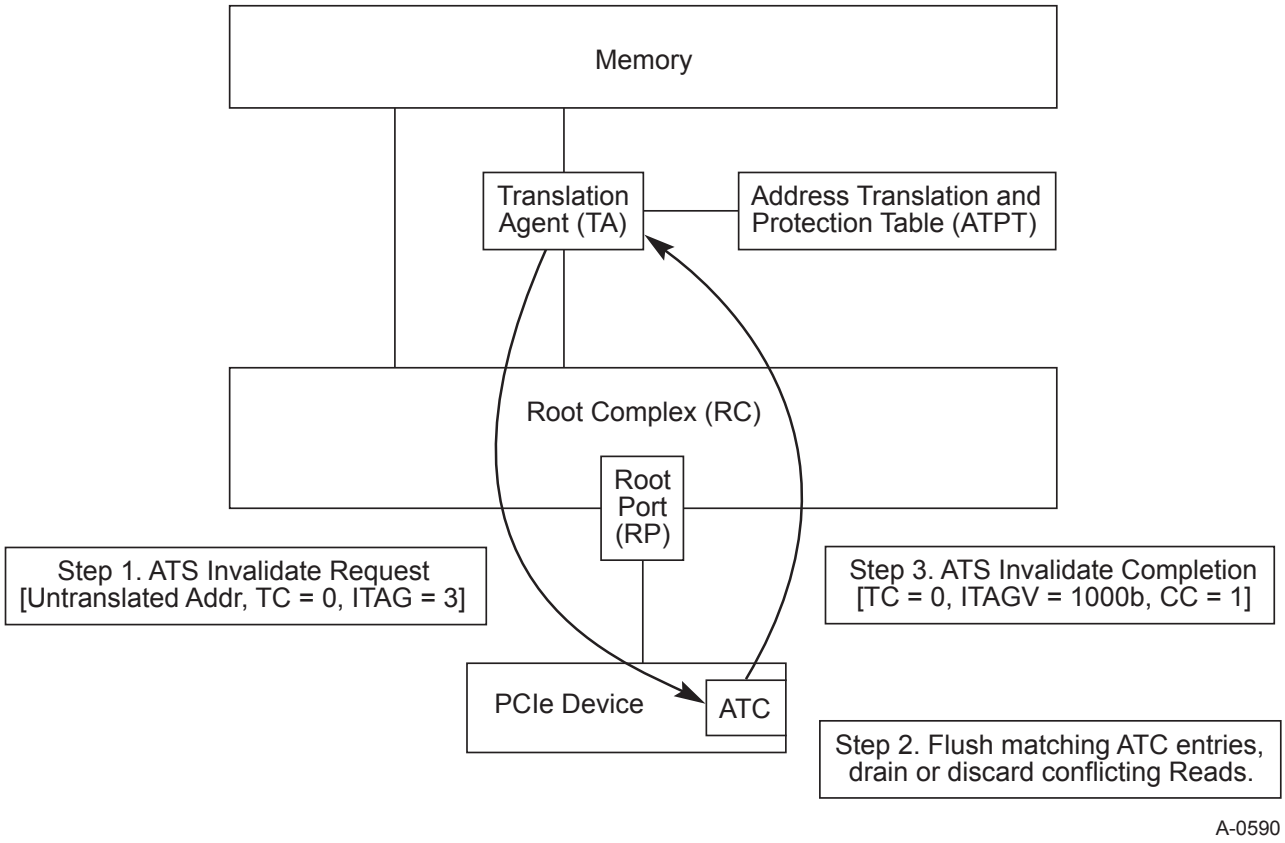

Figure 10-4 Invalidation Protocol with a Single Invalidation Request and Completion

As Figure 10-4 illustrates, there are essentially three steps in the ATS Invalidation protocol:

1. The system software updates an entry in the tables used by the TA. After the table is changed, the TA determines that a translation should be invalidated in an ATC and initiates an Invalidation Request TLP which is transmitted from the RP to the example single-Function Device. The Invalidate Request communicates an untranslated address range, the TC, and an RP unique tag which is used to correlate Invalidate Completions with the Invalidation Request.

2. The Function receives the Invalidate Request and invalidates all matching ATC entries. A Function is not required to immediately flush all pending requests upon receipt of an Invalidate Request. If transactions are in a queue waiting to be sent, it is not necessary for the Function to expunge requests from the queue even if those transactions use an address that is being invalidated.
    a. A Function is required not to indicate the invalidation has completed until all outstanding Read Requests or Translation Requests that reference the associated translated address have been retired or nullified.
    b. A Function is required to ensure that the Invalidate Completion indication to the RC will arrive at the RC after any previously posted writes that use the “stale” address.
3. When a Function has ascertained that all uses of the translated address are complete, it issues one or more ATS Invalidate Completions.
    a. An Invalidate Completion is issued for each TC that may have referenced the range invalidated. These completions act as a flush mechanism to ensure the hierarchy is cleansed of any in-flight transactions which may contain references to the translated address.
        i. The number of Completions required is communicated within each Invalidate Completion. A TA or RC implementation can maintain a counter to ensure that all Invalidate Completions are received before considering the translation to no longer be in use.
        ii. If more than one Invalidation Complete is sent, the Invalidate Completion sent in each TC must be identical in the fields detailed in Section 10.3.2 .
    b. An Invalidate Completion contains the ITAG from Invalidate Request to enable the RC to correlate Invalidate Requests and Completions.

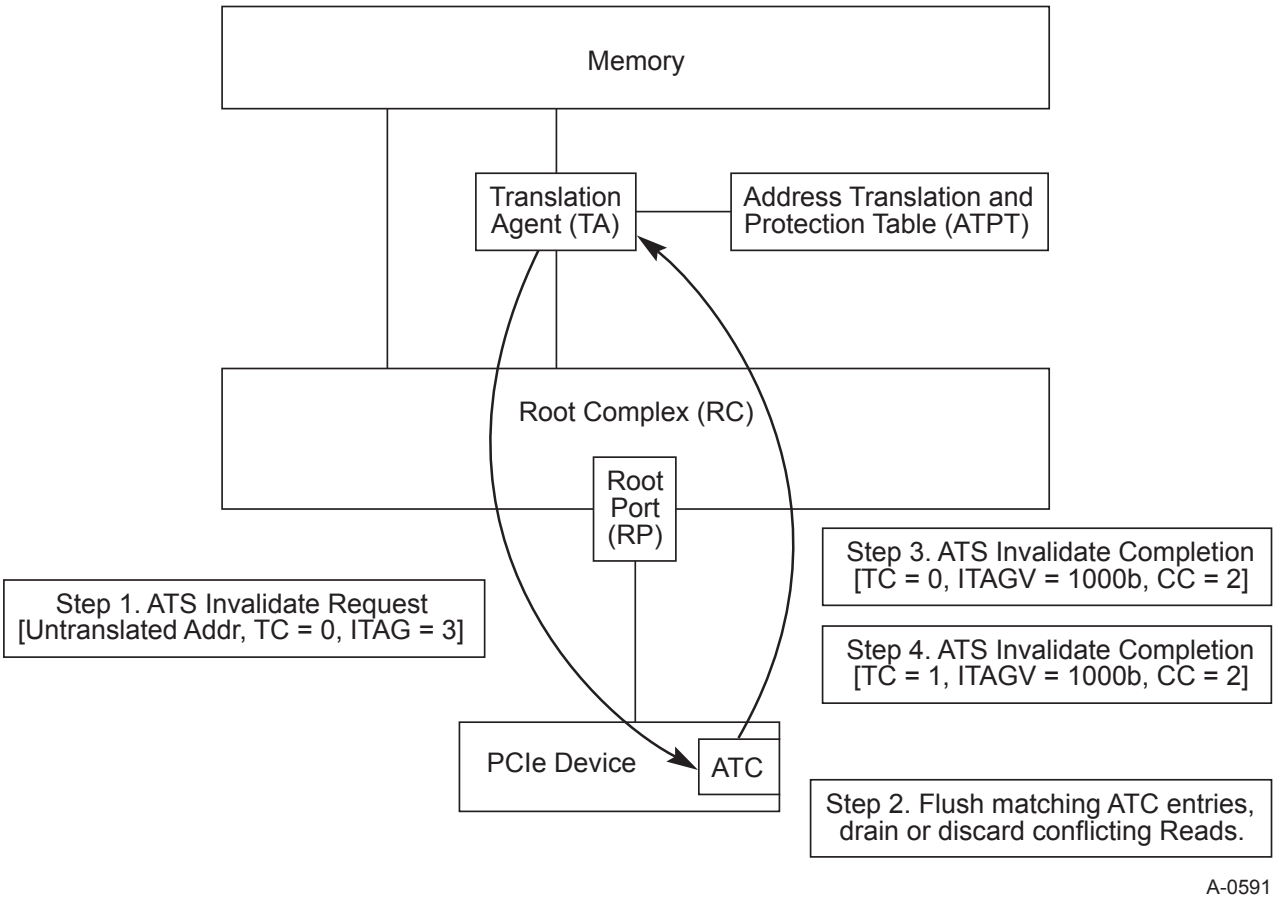

Figure 10-5 Single Invalidate Request with Multiple Invalidate Completions

10.1.2 Page Request Interface Extension

ATS improves the behavior of DMA based data movement. An associated Page Request Interface (PRI) provides additional advantages by allowing DMA operations to be initiated without requiring that all the data to be moved into or out of system memory be pinned.[164] The overhead associated with pinning memory may be modest, but the negative impact on system performance of removing large portions of memory from the pageable pool can be significant.

PRI is functionally independent of the other aspects of ATS. That is, a device that supports ATS need not support PRI, but PRI is dependent on ATS's capabilities.

Intelligent I/O devices can be constructed to make good use of a more dynamic memory interface. Pinning will always have the best performance characteristics from a device's perspective-all the memory it wants to touch is guaranteed to be present. However, guaranteeing the residence of all the memory a device might touch can be problematic and force a sub-optimal level of device awareness on a host. Allowing a device to operate more independently (to page fault when it requires memory resources that are not present) provides a superior level of coupling between device and host.[165]

The mechanisms used to take advantage of a Page Request Interface are very device specific. As an example of a model in which such an interface could improve overall system performance, let us examine a high-speed LAN device. Such a

164. Locked in place so that it cannot be swapped out by the system's dynamic paging mechanism.
165. The alternative is a private interface between a device and its driver that is used to communicate device state so that the driver can ensure the availability of pinned memory resources.

device knows its burst rate and need only have as much physical buffer space available for inbound data as it can receive within some quantum. A vector of unpinned virtual memory pages could be made available to the device, that the device then requests as needed to maintain its burst window. This minimizes the required memory footprint of the device and simplifies the interface with the host, both without negatively impacting performance.

The ability to page, begs the question of page table status flag management. Typical TAs associate flags (e.g., dirty and access indications) with each untranslated address. Without any additional hints about how to manage pages mapped to a Function, such TAs would need to conservatively assume that when they grant a Function permission to read or write a page, that Function will use the permission. Such writable pages would need to be marked as dirty before their translated addresses are made available to a Function.

This conservative dirty-on-write-permission-grant behavior is generally not a significant issue for Functions that do not support paging, where pages are pinned and the cost of saving a clean page to memory will seldom be paid. However, Functions that support the Page Request Interface could pay a significant penalty if all writable pages are treated as dirty, since such Functions operate without pinning their accessible memory footprints and may issue speculative page requests for performance. The cost of saving clean pages (instead of just discarding them) in such systems can diminish the value of otherwise attractive paging techniques. This can cause significant performance issues and risk functional issues in circumstances where the backing store is unable to be written, such as a CD-ROM.

The No Write (NW) flag in Translation Requests indicates that a Function is willing to restrict its usage to only reading the page, independent of the access rights that would otherwise have been granted.

If a device chooses to request only read access by issuing a Translation Request with the NW flag Set and later determines that it needs to write to the page, then the device must issue a new Translation Request.

Upon receiving a Translation Request with the NW flag Clear, TAs are permitted to mark the associated pages dirty. It is strongly recommended that Functions not issue such Requests unless they have been given explicit write permission. An example of write permission is where the host issues a command to a Function to load data from a storage device and write that data into memory.

10.1.3 Process Address Space ID (PASID)

Certain TLPs can optionally be associated with a Process Address Space ID (PASID). This value is conveyed using the PASID TLP Prefix. The PASID TLP Prefix is defined in the Section 6.20 .

The PASID TLP Prefix is permitted on:

- Memory Requests (including Untranslated AtomicOp Requests) with Untranslated Addresses
- Address Translation Requests
- Page Request Messages
- ATS Invalidation Requests
- PRG Response Messages

Usage of the PASID TLP Prefix for Untranslated Memory Requests is defined in Section 6.20 . This section describes PASID TLP Prefix for the remaining TLPs.

When a Request does not have a PASID TLP Prefix, the Untranslated Address represents an address space associated with the Requester ID.

When a Request has a PASID TLP Prefix, the Untranslated Address represents an address space associated with both the Requester ID and the PASID value.

When a Response has a PASID TLP Prefix, the PASID value reflects the address space associated the corresponding Request.

Each Function has an independent set of PASID values. The PASID field is 20 bits wide however the effective width is constrained by the lesser of the width supported by the Root Complex (TA) and the width supported by the Function (ATC). Unused upper bits of the PASID value must be 0b.

For Endpoints in systems where a Virtual Intermediary (VI) is present, Untranslated

Addresses with an associated PASID are typically used to represent Guest Virtual Addresses (GVA) and Untranslated Addresses that are not associated with a PASID represent Guest Physical Addresses (GPA). The TA could be designed so that the VI manages the tables used to perform translations from GPA to Translated Addresses while the individual Guest Operating Systems manage tables used to perform translations from GVA to GPA. When translating an address with an associated PASID, the TA performs both translations and returns the resulting Translated Address (i.e., GVA to GPA followed by GPA to Translated Address). The intermediate GPA value is not visible to the ATC.

When an ATC invalidates a cached GPA mapping, it invalidates the GPA mapping and also invalidates all GVA mappings in the ATC. When the GPA invalidate completes, the VI can safely remove pages backing GPA memory range from a Guest Operating System. The VI does not need to know which GVA mappings involved the GPA mapping.

10.2 ATS Translation Services

A TA does translations. An ATC can cache those translations. If an ATC is separated from the TA by PCIe, the memory request from an ATC will need to be able to indicate if the address in the transaction is translated or not. The modifications to the memory transactions are described in this section, as are the transactions that are used to communicate translations between a remote ATC and a central TA.

10.2.1 Memory Requests with Address Type

A Function with an ATC can send memory read/write Requests that contain either translated or untranslated addresses. As shown in Figure 10-6 and Figure 10-7 , the Address Type (AT) field is used to indicate the type of address that is present in the request header.

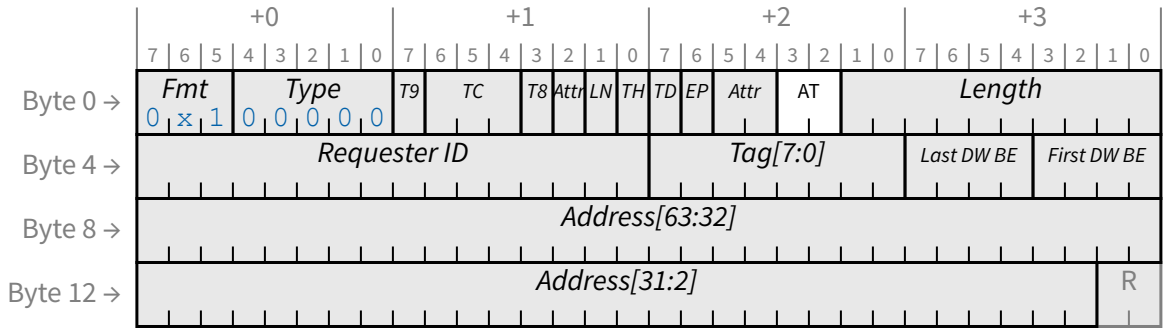

Figure 10-6 Memory Request Header with 64-bit Address

Figure 10-7 Memory Request Header with 32-bit Address

The AT field in the requests is a redefinition of a reserved field in earlier version of this specification. Functions that do not implement an ATC will continue to set the AT field to its defined reserved value (00b). Functions that implement an ATC will set the AT field as listed in Table 10-1 .

Table 10-1 Address Type (AT) Field Encodings

AT[1:0] Coding	Mnemonic	Meaning
00b	Untranslated	A TA may treat the address as either virtual or physical.
01b	Translation Request	The TA will return the translation of the address contained in the address field of the request as a read completion. This value only has meaning for an explicit Translation Request (see Section 10.2.2). The TA will signal an Unsupported Request (UR) if it receives a TLP with the AT field set to 01b in a Memory Request other than Memory Read.
10b	Translated	The address in the transaction has been translated by an ATC. If the Function associated with the SourceID is allowed to present physical addresses to the system memory, then the TA might not translate this address. If the Function is not allowed to present physical addresses, then the TA may treat this as an UR.
11b	Reserved	The TA will signal an Unsupported Request (UR) if it receives a Memory Request TLP with the AT field set to 11b.

The AT field is only defined for Memory Requests. The field remains reserved for other TLPs.

10.2.2 Translation Requests

A Translation Request has a format that is similar to that of a memory read. The AT field is used to differentiate a Translation Request from a normal memory read.

The request header for a Translation Request has the formats illustrated in Figure 10-8 and Figure 10-9 .

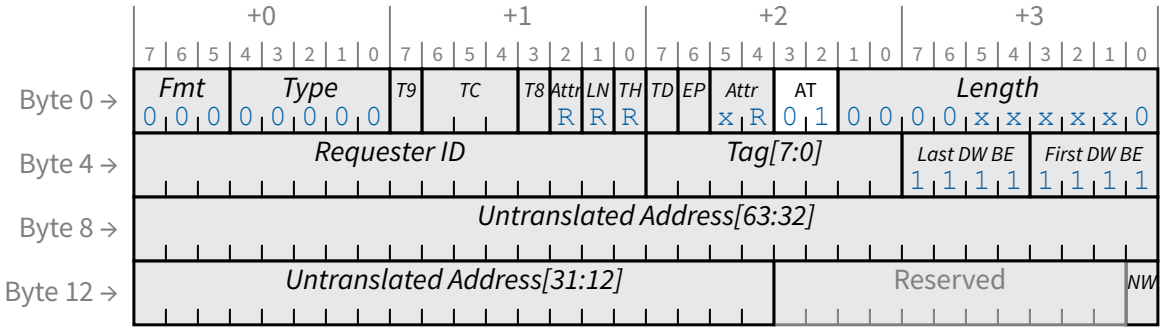

Figure 10-8 64-bit Translation Request Header

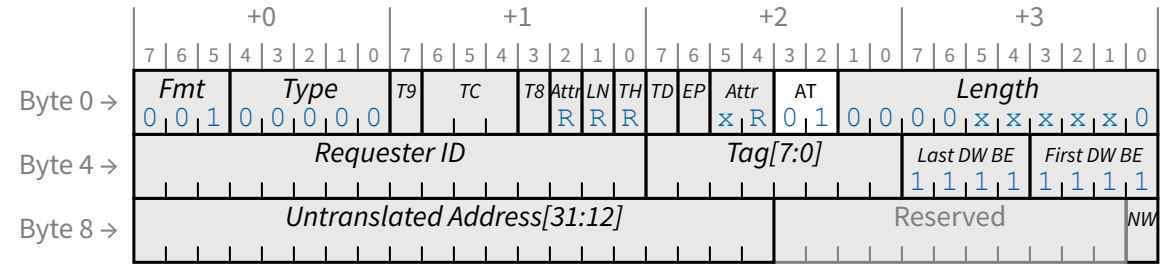

Figure 10-9 32-bit Translation Request Header

Translation Requests have the same completion timeout intervals as Read Requests.

10.2.2.1 Attribute Field

For a Translation Request, the Relaxed Ordering (RO) bit is applicable and permitted to be Set, where it affects the ordering of its associated Translation Completions. The remainder of the Attr field is Reserved. The Requester of a Translation Request must not depend on the TA to guarantee any specific ordering relationship between Translation Completions and any other Requests or Completions. There are no ordering requirements for a Translation Request. A TA may reorder a Translation Request with respect to any other request.

IMPLEMENTATION NOTE

Translation Request Ordering

Because no ordering can be assumed between Translation Requests and other types of Requests, a Translation Request does not make an effective flushing/ordering primitive.

10.2.2.2 Length Field

The Length field is set to indicate how many translations may be returned in response to this request. Each translation is 8 bytes in length and represents one or more STUs (Smallest Translation Unit). The maximum setting for the Length field is the RCB. The Length field in a Translation Request must always indicate an even number of DWORDs. If Length is set to indicate a value greater than allowed, or if the least-significant bit of the Length field is non-zero, then the TA will treat the request as a Malformed Packet.

If the Length field has a value greater than two, then the Function is requesting translations for a range of memory greater than a single STU. The additional translations, if provided, are assumed to be for sequentially-increasing, equal-sized, STU-aligned regions, starting at the requested address.

10.2.2.3 Tag Field

The Tag field has the same meaning as in a Memory Read Request.

10.2.2.4 Untranslated Address Field

A Translation Request includes either a 32-bit or a 64-bit Untranslated Address field. This field indicates the address to be translated. The TA will make decisions about the validity of the request, based on the address in the translation request. The TA is permitted to return fewer translations than requested, but it will not return more.

When multiple translations are requested, the TA will not return a translation if the range of that translation does not overlap the implied range of the Translation Request (this would only apply to translations after the initial value). The implied range of the Translation Request is $[2^{STU+12} * (Length/2)]$ bytes.

The Untranslated Address field in the Translation Request is any address in the range of the first STU. Address bits 11:0 are not present in the Translation Request and are implied to be zero. If a Requester has Page Aligned Request Set (see Section 7.8.8.2), it must ensure that bits 11:2 are zero. If a Requester has Page Aligned Request Clear, it is permitted to supply any value for bits 11:2.[166] The TA must ignore bits 11:2 as well as any low-order bits not required to determine the translation.

For example, if using 64-bit addressing for a Function with the Page Aligned Request bit Set that is programmed with an STU of 1 (i.e., 8192-byte pages), bits 63:13 are significant, bit 12 is ignored by the TA and bits 11:0 are implied to be zero.

10.2.2.5 No Write (NW) Flag

The No Write flag, when Set, indicates that the Function is requesting read-only access for this translation.[167]

The TA may ignore the No Write Flag, however, if the TA responds with a translation marked as read-only then the Function must not issue Memory Write transactions using that translation. In this case, the Function may issue another translation request with the No Write flag Clear, which may result in a new translation completion with or without the W (Write) bit Set.

Upon receiving a Translation Request with the NW flag Clear, TAs are permitted to mark the associated pages dirty. It is strongly recommended that Functions not issue such Requests unless they have been given explicit write permission.

166. Note: The Page Aligned Request bit was added in Revision 1.1 of the ATS Specification.
167. Note: The No Write Flag was added in Revision 1.1 of the ATS Specification.

10.2.2.6 PASID TLP Prefix on Translation Request

If a Translation Request has a PASID TLP Prefix, the Untranslated Address Field is an address within the process address space indicated by the PASID field.

If a Translation Request has a PASID TLP Prefix with either the Privileged Mode Requested or Execute Requested bit Set, these may be used in constructing the Translation Completion Data Entry.

The PASID Extended Capability indicates whether a Function supports and is enabled to send and receive TLPs with the PASID TLP Prefix.

10.2.3 Translation Completion

A Translation Completion (either a Cpl or a CplD) is sent by a TA for each Translation Request. This specification describes the meaning of fields in Translation Completions. Fields not defined in this specification have the same meanings proscribed for Read Completions in this specification. For a Translation Completion, the Relaxed Ordering (RO) bit is applicable and permitted to be Set, if the corresponding Translation Request RO bit was set. The remainder of the Attr field is Reserved.

If the TA was not able to perform the requested translation, a completion with the format shown in Figure 10-10 is used.

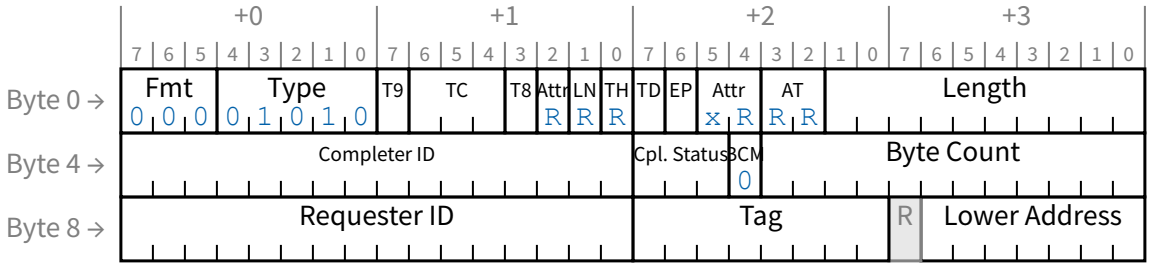

Figure 10-10 Translation Completion with No Data

IMPLEMENTATION NOTE

Byte Count Field for Unsuccessful Translation Completions with No Data

Previous versions of this specification indicated the Byte Count and Lower Address field should be 0000 0000 0000b for Unsuccessful Translation Completions with No Data. It is strongly recommended that implementations do not depend on the Byte Count and Lower Address field being set to any particular value in Unsuccessful Translation Completions with No Data.

The values and meaning for the Completion Status field are listed in Table 10-2 .

Table 10-2 Translation Completion with No Data Status Codes

Value	Status	Meaning
000b	Success	This Completion Status has a nominal meaning of “success”. The TA will not return this value in a Cpl.
001b	Unsupported Request (UR)	Translation Requests from this Function are not supported by the TA. If a Function receives this Completion code, it must disable its ATC and not send requests using translated addresses until the ATC is re-enabled. For transactions the Function may internally have in flight, the Function may either terminate or complete them. The mechanism a Function receiving this code uses to report this condition is outside the scope of this specification. The TA detecting this error is a "Completer Sending a Completion with UR/CA Status" and shall behave as defined in this specification.
010b	CRS	This value is not allowed in any Completion to a request initiated by a PCI Express Function. If received by a Function, it shall be treated as a Malformed TLP.
100b	Completer Abort (CA)	The TA was not able to translate the address because of an error in the TA. This nominally causes an error to be reported to the device driver associated with the ATC. See AER in this specification.
All others	Reserved	A Translation Completion with a Reserved Completion Status value is treated as if the Completion Status was Unsupported Request (001b).

Note: Return values other than *Success* indicate an error.

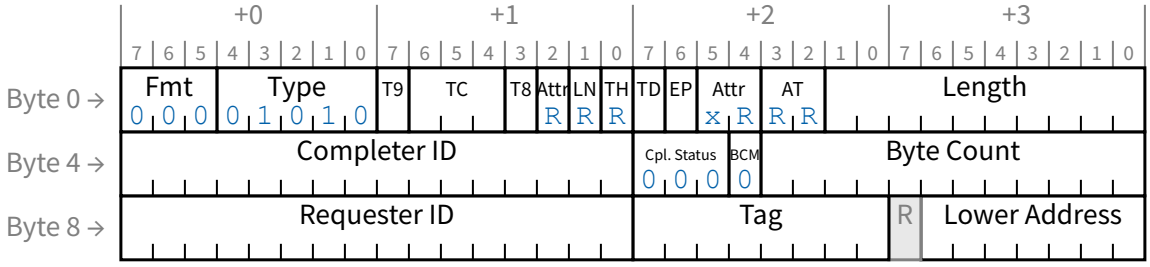

Figure 10-11 Successful Translation Completion

Fields are set in accordance with Section 2.2.9 and Section 2.3 .

Translation Completions must be sent using the same TC as the Translation Request. The Function is not required to verify that the same TC was used.

The Lower Address field will contain a value that will make the packet consistent with RCB semantics. If the result is returned in a single packet, Lower Address is set to RCB minus Byte Count. If the results are returned in multiple packets, the first packet will have a Lower Address field of RCB minus (Total Completion Length * 4) and subsequent packets will have a Lower Address field of 000 0000b. See Section 10.2.4 for additional requirements for multiple packet completions.

If the Completion Status field is 000b, then the translation was successful and a data payload will follow the header. The contents of the data payload are shown in Figure 10-12 .

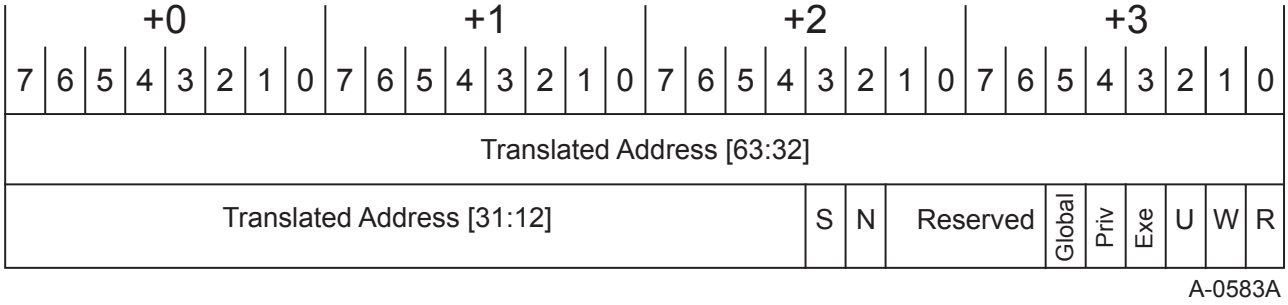

Figure 10-12 Translation Completion Data Entry

Table 10-3 Translation Completion Data Fields

Field	Meaning
S	**Size of translation** - This field is 0b if the translation applies to a 4096-byte range of memory. If this field is 1b, then the translation applies to a range of memory that is larger than 4096 bytes (see Section 10.2.3.1).
N	**Non-snooped accesses** - If this field is 1b, then the read and write requests that use this translation must Clear the No Snoop bit in the Attribute field. If it is 0b, then the Function may use other means to determine if No Snoop should be Set.
Reserved	These bits shall be ignored by the ATC.
Global	**Global Mapping** - If this bit is Set, the ATC is permitted to cache this mapping entry in all PASIDs. If Clear, the ATC is permitted to cache this mapping entry only in the PASID associated with the requesting PASID. This bit may only be Set if the associated Translation Request had a PASID TLP Prefix.
Exe	**Execute Permitted** - If this bit is Set, the requesting Function is permitted to execute code contained in the associated memory range. This bit may be Set only if the associated Translation Request had a PASID TLP Prefix with the Execute Requested bit Set. If this bit is Set, R must also be Set. The Priv bit indicates the Privilege level associated with the Exe bit. If Priv is Set, the Exe bit indicates permissions associated with Privileged Mode entities in the Function. If Priv is Clear, the Exe bit indicates permissions associated with Non- Privileged Mode entities in the Function. This value may be cached if R is Set.
Priv	**Privileged Mode Access** - If this bit is Set, R, W and Exe refer to permissions associated with Privileged Mode entities. If this bit is Clear, R, W and Exe refer to permissions associated with Non-Privileged Mode entities. This bit may only be Set if the associated Translation Request contained a PASID TLP Prefix with the Privileged Mode Requested bit Set. This value must be cached any of the R, W or Exe values are cached.
U	**Untranslated access only** - When this field is Set, the indicated range may only be accessed using untranslated addresses, and the Translated Address field of this Translation Completion Data Entry may not be used in a subsequent Read/Write Request with AT set to Translated. This value may be cached if R or W is Set.

Field	Meaning
R,W	**Read, Write** - These two fields indicate the transaction types that are allowed for requests using the translation. The encodings are: **00b** Neither read nor write transactions are allowed. This translation is considered not to be valid. The contents of the Translated Address, N, U, and Exe fields are undefined. A translation with this value must not be cached in the ATC (see Section 10.2.3.5). **01b** Write Requests that target this range are allowed, but Read Requests are not unless they are zero-length reads. **10b** Read Requests that target this range are allowed (including zero-length reads), but Write Requests are not. **11b** Read and Write Requests that target this range are allowed. The Priv bit indicates the Privilege level associated with R and W. If Priv is Set, R and W indicate permissions associated with Privileged Mode entities in the Function. If Priv is Clear, R and W indicate permissions associated with Non-Privileged Mode entities in the Function.

10.2.3.1 Translated Address Field

If the R and W fields are both Clear, or if U is Set, then the Translated Address field may not be used by the Function for any purpose.

If either the R or W field is Set, and the U field is Clear, then the Translated Address field contains an address that can be used by the Function in a Memory Request with the AT field set to Translated and the Function may cache the Translated Address. When cached, the R and W fields must be stored with the same value as the Translation Completion entry. The address that is cached must be a subset of the address range indicated in the Translation Completion (the subset may include the entire range).

While the Translated Address is cached in the Function's ATC, it shall not be possible for the Function to modify the entry other than to delete it. The entry must be deleted from the ATC when an Invalidation Request is received that has an indicated range that overlaps any portion of the cached address.

A Function is not allowed to make an entry into its ATC unless the entry is in a Translation Completion and the E (Enable) field within the ATS Capability is Set. Entries in an ATC cache that are written before the E field is Set must not be used in Memory Request. They must either be invalidated when the E field is Set or ignored and not used.

10.2.3.2 Translation Range Size (S) Field

If S is Set, then the translation applies to a range that is larger than 4096 bytes. If S = 1b, then bit 12 of the Translated Address is used to indicate whether or not the range is larger than 8192 bytes. If bit 12 is 0b, then the range size is 8192 bytes, but it is larger than 8192 bytes if Set. If S = 1b and bit 12 = 1b, then bit 13 is used to determine if the range is larger than 16384 bytes or not. If bit 13 is 0b, then the range size is 16384 bytes, but it is larger than 16384 bytes if Set.

Low-order address bits are consumed in sequence to indicate the size of the range associated with the translation.

Note: This encoding method is also used to indicate the size of the memory range being invalidated.

Examples for different translation sizes are shown in Table 10-4 .

Table 10-4 Examples of Translation Size Using S Field

Address Bits																				S	Translation Range Size in Bytes
63:32 *	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12		
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	4 K
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	1	8 K
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	1	1	16 K
x	x	x	x	x	x	x	x	x	x	x	x	0	1	1	1	1	1	1	1	1	1	2 M
x	x	x	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1 G
x	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4 G

Note:
* Upper address bits are used to indicate the size for ranges larger than 4 GB.

The size field is set to indicate the range size in multiples of 4096 bytes regardless of the setting of STU. For example, if STU is set to indicate that the minimum translation is 8192 bytes, then S should be Set on all translation returned in a Translation Completion and in all Invalidate Requests. If STU is set to indicate a 16384-byte minimum, then S and bit 12 would both be Set in all translation and invalidate ranges.

If S is Set and bits 63:12 are all 1b, then the behavior is undefined. If S is Set and bit 63 is 0b, and bits 62:12 are all 1b, then the request is to invalidate all translations.

If a Function receives a Translation Completion with a Translation Size field smaller than the Function's programmed STU value, it shall treat the Translation Completion as if it had Completion Status UR.

10.2.3.3 Non-snooped (N) Field

This field is Set to indicate that Read and Write Requests that target memory in the range of this translation must Clear the No Snoop Attribute bit in the Request header. When this field is 0b, the Function is allowed to Set the No Snoop Attribute bit in a Function-specific manner.

Note: When this field is Cleared, the Function is not allowed to Set No Snoop in a Memory Request if the Enable No Snoop field in the Device Control register is Cleared.

The N bit may be cached by the ATC if either R or W is Set.

When U is Set, the meaning of this field is undefined, and the TA may set this field to any value. A translation has a single value for the N field that is not affected by privilege level. An ATC is permitted to cache the N field without regard to the value of the Priv bit.

10.2.3.4 Untranslated Access Only (U) Field

This field is Set when the Function is not allowed to access the implied range of memory using a translated address (the range is implied by the untranslated address in the Translation Request and the offset of the translation in the Translation Completion). The Function may use untranslated addresses to access the range as long as the accesses are allowed by the R and W fields. The Function may cache this translation value if either R or W is Set. If the U field is Set, the Translated Address field in the translation is not necessarily a valid memory address and the Function may not use the value in a Read or Write Request with AT set to Translated.

Note: One of the possible uses of this field is to avoid unnecessary invalidations. If a Function uses translated requests for some portions of memory, but not others, then the U field can be used on the portions for which translated requests are not used. When a translation changes if the U field is Set, then it will not necessarily be required that an Invalidate Request be sent to the Function. An example of this use is a Function with a ring buffer that is used for commands. The ring buffer may be allocated for a long period of time and have very high re-use (locality). For this reason, it is useful for the Function to use translated addresses in its memory request that target the command buffer. The same Function might access data buffers that have poor locality and low reuse. Accesses to the data buffers might best be handled by using untranslated Requests. Setting the U field for the data buffer translations ensures that the Function will not attempt to use a translated value to access the data buffer so, when the data buffer mappings are changed, no Invalidation Request is required. When U is Set and either R or W is Set, the ATC is permitted to cache U, R, W Exe, and Priv, as well as the Translation Range Size (see Section 10.2.3.2). An Invalidation Request is required if these values change.

10.2.3.5 Read (R) and Write (W) Fields

These fields indicate if the returned translation value may be used in a read or write memory request. The ATC may not issue a non-zero read request using the translation value if the R field is Cleared. The ATC may not issue a write request using the translation value if the W field is Cleared. The ATC may not issue any type of request using the translation value if neither the R nor W fields are Set. If both R and W fields are Cleared, the range of the translation is still indicated, but the meaning of the other values in the translation is undefined.

Note: The range of a Translation entry is indicated even if R = W = 0b in order to allow a “hole” in the Translation Completion. For example, if the Translation Request has a Length of six DWs, then up to three translations could be included in the Translation Completion. The first and third translations may have Set R or W but the second could have R = W = 0b. To avoid ambiguity about the size of the indicated gap, the range of the gap is indicated in the Translation Completion even if R = W = 0b.

The R = 0b, W = 0b state is used to indicate that the address field in the translation may not be used to form a translated address value for a subsequent request.

When the host changes the translation in the TA, to make the translation present, the host is not required to send an invalidation indication to the ATC so that it will know of the change in state of the translation. Since the ATC may not be notified of changes of the translation, a translation value of R = W = 0b must not be cached.

If no table entry is found for the requested address, the TA will return a CplD with a single translation value with R = W = 0b.

Note: Implementations should not assume that receiving a translation response with the R or W bits Set *(independent of the value of the U bit)* implies that a subsequent read or write request with the same **untranslated** address will succeed. Although it may be possible for a device and its controlling software to ensure this property, the method for doing so is outside the scope of this specification.

The Priv bit is used to qualify R and W. If Priv is Set, R and W indicate permissions granted to Privileged Mode entities in the Function. If Priv is Clear, R and W indicate permissions granted to Non-Privileged Mode entities in the Function. The R and W values for the two privilege levels are independent. The ATC must not assume any correlation between the Privileged Mode and Non-Privileged Mode permissions associated with a translation.

10.2.3.6 Execute Permitted (Exe)

If Exe is Set, the requesting Function is permitted to execute code in the implied range of memory. If Exe is Clear, the requesting Function is not permitted to execute code in the implied range of memory.

The definition of what it means for a Function to execute code is outside the scope of this specification. Various system components may have different instruction sets. Behavior within the requesting Function when it attempts to execute code that is not permitted by this bit is outside the scope of this specification.

The Exe bit may only be Set the TA supports Execute permissions, the associated Translation Request had a PASID TLP Prefix with an effective value of 1b for the Execute Requested bit[168] and R is Set in the Translation Completion Data Entry. Otherwise, the Exe bit must be Clear.

This value may be cached if R is Set.

The Priv bit is used to qualify the Exe bit. If Priv is Set, the Exe bit indicates permission granted to Privileged Mode entities in the Function. If Priv is Clear, the Exe bit indicates permission granted to Non-Privileged Mode entities in the Function. The Exe bit values for the two privilege levels are independent. The ATC must not assume any correlation between the Privileged Mode and Non-Privileged Mode permissions associated with a translation.

Functions may optionally check that:

- If the Execute Requested bit is Clear in a Translation Request, the Exe bits in the associated Translation Completion Data Entries are also Clear.
- If Exe is Set, R is also Set.

If either optional check fails, the Function shall signal Unexpected Completion (UC). These checks are independently optional.

10.2.3.7 Privileged Mode Access (Priv)

If Priv is Set, R, W, and Exe refer to permissions granted to entities operating in Privileged Mode in the requesting Function. If Priv is Clear, R, W, and Exe refer to permissions granted to entities operating in Non-Privileged Mode in the requesting Function.

The meaning of Privileged Mode and Non-Privileged Mode and what it means for an entity to be operating as in Privileged Mode or in Non-Privileged Mode depends on the protection model of the system and is outside the scope of this specification.

Behavior is outside the scope of this specification when an entity in the requesting Function attempts to access memory that it is not permitted to access.

The Priv bit may only be Set if the TA supports Privileged Mode and the associated Translation Request had a PASID TLP Prefix with an effective value of 1b for the Privileged Mode Requested[169] bit. Otherwise, the Priv bit must be Clear.

The Privileged and Non-Privileged Mode versions of R, W and Exe are independent. An ATC may cache either or both versions of R, W and Exe. An ATC that receives a translation with R=W=0b for one privilege level may not assume anything about what it might receive for the other privilege level.

This value may be cached if R or W is Set. This value must be cached when the corresponding R, W, or Exe values are cached.

Note: Since the Priv bit is Set only when the requesting Function Sets the Privileged Mode Requested bit, Functions that never set that bit should always receive the Priv bit Clear and thus don't need to cache it.

Functions may optionally check that when the Privileged Mode Requested bit is Clear in a Translation Request, the Priv bits in the associated Translation Completion Data Entries are also Clear. If this optional check fails, the Function shall signal Unexpected Completion (UC).

168. The effective value of the bit is 0b unless the bit in the PASID TLP Prefix is 1b and usage of the bit is enabled for the request.
169. The effective value of the bit is 0b unless the bit in the prefix is 1b and usage of the bit is enabled for the request.

IMPLEMENTATION NOTE

Execute Permission and Privilege Mode Enforcement

The requesting Function determines whether a particular Memory Request needs Execute permission or is associated with a Privileged Mode or Non-Privileged Mode entity. The ATC implements the protection checks indicated by the Exe and Priv bits.

10.2.3.8 Global Mapping (Global)

If Global is Set, the requesting Function is permitted to create a Global Mapping entry in the ATC for this translation. If Global is Clear, the requesting Function is not permitted to create a Global Mapping entry in the ATC for this translation. Global Mapping entries apply to all PASIDs of the Function. They permit the ATC to reduce the number of translation requests needed and to reduce the memory needed for caching the results.

A Function is permitted to ignore this bit and always create non-Global Mapping entries in the ATC. This could result in multiple translations being requested for the same Untranslated Address under different PASIDs.

Functions that use this bit must also have the Global Invalidate Supported bit Set (see Section 10.5.1.2).

10.2.4 Completions with Multiple Translations

An ATC is allowed to request that the TA provide translations for a virtually contiguous range of addresses. It does this by setting the Length field in the Translation Request to a value that is two times the number of requested translations as long as the request size (Total Completion Length * 4) is not larger than either Max_Read_Request_Size or RCB.

If multiple translations are requested, the TA may return one or more translations as long as the number of translations does not exceed the number of requested translations. It is not an error for the TA to return fewer translations than requested and no error indication is sent unless there is an error in accessing the data.

If the Translation Completion contains multiple translations, all translations must have the same indicated size. Also, successive translations must apply to the virtual address range that abuts the previous translation in the same completion.

If a translation has both R = 0b and W = 0b, the TA must still set the Size field and the lower bits of the Translated Address field used to encode the completion size to appropriate values.

Each translation in a Translation Completion will have some overlap with the implied memory range of the Translation Request (see Section 10.2.2).

A successful Translation Completion must consist of one or two CplDs.

If a Translation Completion CplD has a Byte Count that is greater than four times the Length field, then additional CplDs are required to complete the transaction.

If a Translation Completion CplD has a Byte Count that is equal to four times the Length field, then the packet completes the request. For such a CplD, if the sum of Byte Count and Lower Address is not a multiple of RCB, then the CplD is the last of a sequence. If no previous CplD for this request has been received, an error has occurred and all translation values should be discarded.

Note: There are multiple reasons that the TA may truncate the results of the completion. For example, the request might ask for a range of addresses, not all of which are defined. This could occur if the first translation is valid but located at the

end of a page of translations. The TA, in looking up the next page of translations, may find that the page is not valid so the addresses are not valid. The range of addresses that are valid would be returned and no error indicated. When truncating a Translation Completion the TA is not allowed to pad the response with invalid entries (R = 0b, W = 0b).

Note: There are multiple reasons that the TA may break a Translation Completion into multiple TLPs. As an example, if the virtual address of the Translation Completion resolves to a table access that crosses an RCB boundary of the memory system, the completion to the TA may be broken into multiple completions by the memory. Rather than require that the TA accumulate the results, it is allowed to send each portion of the Translation Completion to a Function when it is received from the system memory.

10.3 ATS Invalidation

ATS uses the messages shown in this section to maintain consistency between the TA and the ATC. This specification assumes there is a single TA associated with each ATC. The TA (in conjunction with its associated software) must ensure that the address translations cached in the ATC are not stale by issuing Invalidate Requests.

10.3.1 Invalidate Request

When a translation is changed in the TA and that translation might be contained within an ATC in a Function, the TA (in conjunction with its associated software) must send an Invalidate Request to the ATC to maintain proper synchronization between the ATPT and the ATC. An Invalidate Request is used to clear a specific subset of the address range from the ATC. Invalidate Requests are constrained to cover power of 2 multiple of 4096-byte pages.

The format of an Invalidate Request is shown in Figure 10-13 .

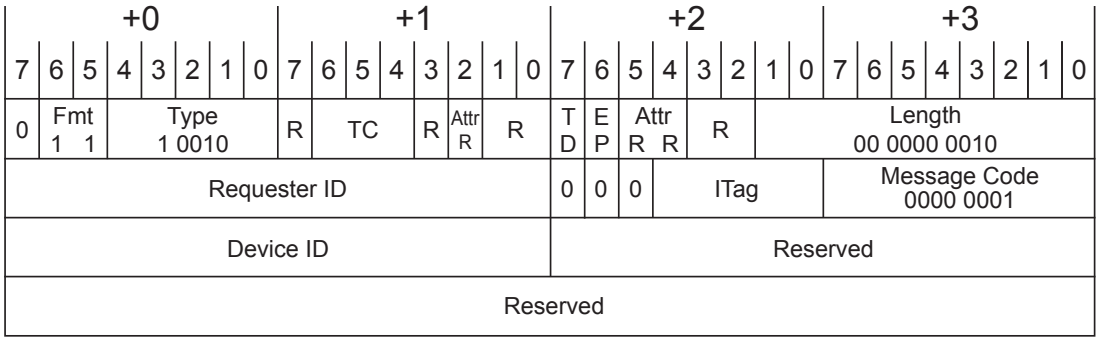

Figure 10-13 Invalidate Request Message

The Invalidate Request is a MsgD transaction with 64 bits of data. Invalidate Request messages may be sent in any TC. The ITag field is constrained to the values 0 to 31 and is used by the TA to uniquely identify requests it issues. A TA must ensure that once an ITag is used, it is not reused until either released by the corresponding Invalidate Completions or by a vendor-specific timeout mechanism (see below).

The TA may have a single pool of ITag values for all invalidates that it issues or it may have a pool for each Device ID or any other combination. A Device with multiple ATCs on different Functions must manage the ITags separately for each Requester ID.

The address range specified in an Invalidate Request may span one or more STU 4096-byte pages. Invalidation ranges are required to be naturally aligned and may not be smaller than STU 4096-byte pages. Upon receiving an Invalidate Request with a range less than STU an ATC may either (1) signal an Unsupported Request or (2) round the range of the request up to a value greater than or equal to the STU.

IMPLEMENTATION NOTE

Invalidate Completion Timeout

Devices should respond to Invalidate Requests within 1 minute (+50% -0%).Having a bounded time permits an ATPT to implement Invalidate Completion Timeouts and reuse the associated ITag values. ATPT designs are implementation-specific. As such, Invalidate Completion Timeouts and their associated error handling are outside the scope of this specification.

The content of the payload is the untranslated address range to be invalidated. The payload format is shown in Remove Word {RD} tag used to help generate ToC/ToF/ToT.

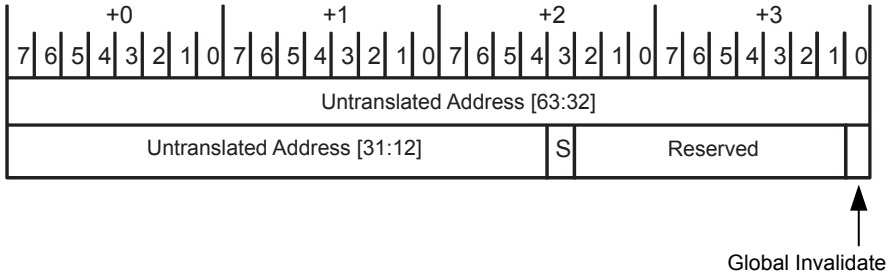

Figure 10-14 Invalidate Request Message Body

The S field is used to indicate if the range being invalidated is greater than 4096 bytes. Its meaning is the same as for the Translation Completion (see Section 10.2.3.1 and Section 10.2.3.2).

The Global Invalidate bit indicates that the Invalidation Request Message affects all PASID values (see Section 10.3.8). This bit is Reserved unless the Invalidation Request has a PASID TLP Prefix. The bit is ignored by the ATC if Global Invalidate Supported bit is Clear (see Section 10.3.8).

10.3.2 Invalidate Completion

When a Function completes an Invalidate operation, it will send one or more Invalidate Completion messages to the TA. These messages must be tagged with information extracted from the Invalidate Request to enable the TA to associate the Invalidate Completions with the Invalidate Request.

The format of the Invalidate Completion message is shown in Figure 10-13 .

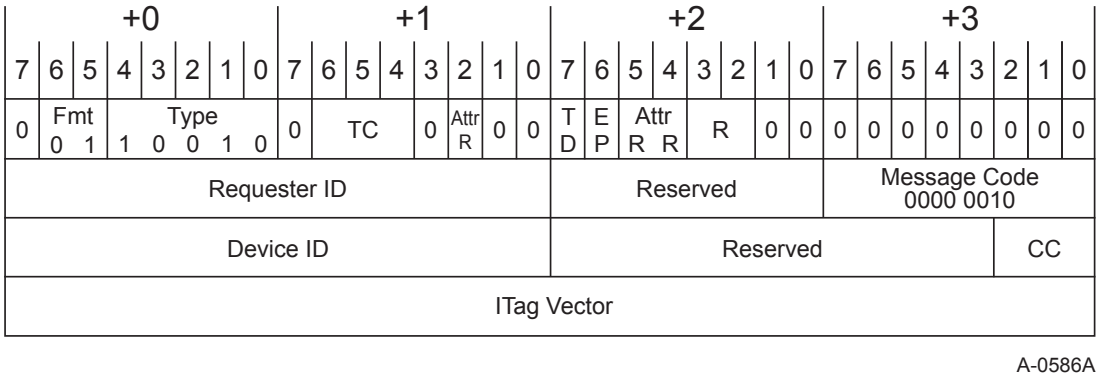

Figure 10-15 Invalidate Completion Message Format

The Invalidate Completion message is a Msg transaction routed by ID. The Requester ID field of the Invalidate Completion message is set to the Requester ID of the Function containing the ATC. The Device ID field of the Invalidate Completion is set to the Requester ID of the TA. The ATC may derive the Requester ID of the TA from the Requester ID field of the corresponding Invalidate Request. Alternatively, since the ATC is only associated with a single TA, the ATC may sample and store the Requester ID from the first Invalidate Request following a Fundamental Reset or FLR. Subsequent Invalidate Completion messages may use this value to set the Device ID field of Invalidate Completion messages.

The Completion Count (CC) field indicates the number of individual Invalidate Completion messages that must be sent for the associated Invalidate Request. Setting the CC field to 0 indicates that eight responses must be sent. The TA is responsible for collecting all the responses associated with a given Tag before considering the corresponding Invalidate Request to be complete.

Invalidate Completion messages may be sent on any TC, independent of the TC the originating Invalidate Request was received. This enables implementations to utilize the Invalidate Completion to push outstanding transactions to the TA to guarantee the required invalidation semantics are met. Implementations that utilize a single Upstream TC are required to send a single Invalidate Completion in the utilized TC.

The ITag Vector field is used to indicate which Invalidate Request has been completed. Each of the 32 possible ITag field values from the Invalidation Request is represented by a single bit in the ITag Vector field. The least significant bit (bit 0; i.e., the right-most bit in the schematic representation of the Invalidate Completion message shown in Figure 10-13) of the ITag Vector field corresponds to the ITag field value of 0. The most significant bit (bit 31) of the ITag Vector field corresponds to the ITag field value of 31. Implementations are allowed to coalesce multiple Invalidate Completions by setting multiple ITag Vector bits in a single message provided the following conditions are met:

- The Invalidate Completions flow in the same TC.
- The Invalidate Completions have the same CC value.
- All fragments of an Invalidate Completion must have identical Request ID, CC, and ITag Vector fields.

A TA that receives an Invalidation Completion for an ITag that has no outstanding Invalidation Request shall report this error using implementation specific mechanisms. One possible such mechanism is to report the Invalidation Completion as an Unexpected Completion (UC).

Functions that do not support ATS will treat an Invalidate Request as UR.

Functions supporting ATS are required to send an Invalidate Completion in response to a Invalidate Request independent of whether the Bus Master Enable bit is Set or not. Note that the above conditions must be satisfied even when Bus Master Enable is Cleared. The method for a device to achieve this is implementation dependent.

IMPLEMENTATION NOTE

Bus Master Enable Change

When Bus Master Enable changes from Set to Clear, no further memory requests should be queued. It is possible that queued write requests are present when BME is Cleared. These requests could block an Invalidate Completion. These requests must be either sent or dropped. This will ensure that all outstanding write transactions that are potentially dependent upon the outstanding invalidation are complete.

10.3.3 Invalidate Completion Semantics

Before an ATC can return an Invalidate Completion for a given Invalidate Request, it must ensure the following conditions are satisfied:

- All new requests initiated by the Function will not utilize stale address translations.
- All outstanding read requests utilizing translated address matching the invalidated range have either completed or been tagged to be discarded (method to discard is implementation specific).
- All outstanding posted writes utilizing a translated address matching the invalidated range have been pushed to the TA. The ATC is required to send a copy of the Invalidate Completion message in each TC in which a posted write has been issued but not known to have been pushed to the TA. The CC field must be set to the same value in each copy of the Invalidate Completion message indicating number of copies sent. The TA is responsible for collecting all sent responses before considering the invalidation to be complete.

IMPLEMENTATION NOTE

Implied TC Flushing

When making the decision as to which TC to send Invalidate Completions, an ATC may infer, in an implementation specific manner, that an issued posted write has been pushed to the TA. For example, a Function that has sent a read transaction to a destination above the TA and received its corresponding response may infer that any preceding posted writes issued in the same TC have been pushed to the TA.

10.3.4 Request Acceptance Rules

In accord with the request acceptance rules enumerated in this section, a Function is not allowed to create a dependency in which the acceptance of a posted transaction is dependent upon the transmission of a posted transaction. Given Invalidate Requests and Invalidate Completions both are posted transactions, Functions must not make the acceptance of an Invalidate Request dependent upon the transmission of an Invalidate Completion. The method for achieving this is implementation specific.

A Function with an ATS capability in its configuration space must be able to accept Invalidate Requests and send Invalidate Completions even if ATS is not enabled.

IMPLEMENTATION NOTE

Invalidate Queue Depth

An ATC is only associated with a single TA. Each TA is limited to a total of 32 outstanding invalidations to any given ATC. This limits the number of outstanding Invalidation Requests active to a single ATC to 32. To avoid a post-to-post dependency, an ATC is required to accept up to 32 Invalidation Requests.

An ATC may choose to implement a maximally sized input queue holding Invalidate Requests. Alternatively, an ATC may choose to implement a maximally sized output queue holding Invalidate Completions. Note that queuing Invalidate Completions requires significantly less state per entry resulting in a potentially more efficient implementation than input queue buffering.

Note that the choice of whether to implement input queuing or output queuing (or a hybrid of both) has no impact on ensuring deadlock free behavior. But implementation choices with regard to queuing may have a significant impact on performance (see Section 10.3.5).

10.3.5 Invalidate Flow Control

Due to the variety of caching architectures and queuing strategies, implementations may vary greatly with respect to invalidation latency and throughput. It is possible that a TA may generate Invalidate Requests at a rate that exceeds the average ATC service rate. When this happens, the credit based flow control mechanisms will throttle the TA issue rate. A side effect of this is congestion spreading to other channels and Links through the credit based flow control mechanism. Depending on the frequency and duration of this congestion, performance may suffer. It is highly recommended that TA and its associated software implement higher level flow control mechanisms.

To assist with the implementation of Invalidate Flow Control, an ATC must publish the number of Invalidate Requests it can buffer before back pressuring the Link. This field applies to all invalidations serviced by the Function, independent of the size of the invalidation. This value is communicated in the Invalidate Queue Depth field in the ATS capability structure (see Section 7.8.8). A value of 0 0000b indicates that invalidate flow control is not necessary to this Function.

IMPLEMENTATION NOTE

Invalidate Flow Control

A Function may indicate that invalidate flow control is not required when one or more of the following is true:

1. The Function can handle invalidations at the maximum arrival rate of Invalidate Requests.
2. The Function will not or very rarely cause Link backpressure (performance loss is negligible).
3. The Function can fully buffer the maximum number of incoming invalidations without back pressuring the Link.

10.3.6 Invalidate Ordering Semantics

Invalidate Requests and Translation Completions may be sent using different TC and are, therefore, unordered with respect to each other (from the Link's perspective). An ATC must ensure that the proper invalidation behavior is maintained when an Invalidate Request bypasses a Translation Completion to an overlapping region.

An ATC must “snoop” its outstanding translation request queue against all arriving Invalidate Requests. When snooping a request for a N*STU sized translation (N is a power of 2), the ATC must snoop the range of addresses starting at the STU aligned region containing the specified address and ending (N-1) STU size pages later.

If an Invalidate Request overlaps the address range in an outstanding Translation Request, the Translation Request must be tagged as invalid and the results of its corresponding Translation Response must be discarded prior to transmission of the Invalidate Completion. If the Translation Response is received before the Invalidate Completion is sent, an implementation is free to issue requests utilizing the translation result provided the Invalidate Completion Semantics (see Section 10.3.3) are satisfied.

IMPLEMENTATION NOTE

Request Range Overlap in Invalidations

In the description above, N is the number of STU sized translations that were requested in the Translation Request. This is equal to (Length field in Translation Request)/2.

As an example:

STU is 00 0010b indicating 16384-byte pages.

An outstanding Translation Request has a Length field of 00 0000 0100b indicating two translations covering a range of 32768 bytes.

The high-order 48 bits of the Translation Request are 0000 0FFF FFFFh.

The low-order 16 bits of the address in the request are 11xx xxxx xxxx xxxxb indicating that the translation request covers a range that overlaps a 32768-byte boundary (in fact, the request crosses a 16-TB boundary).

If two translations are returned, they would cover the two STU sized regions at 0000 0FFF FFFF C000h and 0000 1000 0000 0000h.

An Invalidate Request is received with the high-order 48 bits of 0000 1000 0000h and the low-order 16 bits of 0001 1xxx xxxx xxxxb.

The ATC must detect that a translation associated with a portion of the Translation Request is now invalidated and the Translation Completion associated with the invalidated region must be discarded (for simplification, the ATC is allowed to discard all of the Translation Completion).

It should be noted that, processing of the Invalidate Requests is simplified if Translation Requests do not cross alignment boundaries of the request. The Translation Request from the above example is not aligned to a 32768-byte boundary. If it were broken into two requests, it would be simpler to associate the range of the Invalidate Request with the address in the Translation Request. Breaking the Translation Requests into aligned requests is not a requirement.

10.3.7 Implicit Invalidation Events

The following events will cause the invalidation of all ATC entries:

- Conventional Reset (all forms)
- Function Level Reset
- E field in ATS Capability changes from Clear to Set

The following events will cause the invalidation of all non-Global Mapping ATC entries that were requested using a specific PASID:

- Stopping the use of a PASID as defined inthis specification.

No explicit Invalidate Completion message is sent when these implied invalidate events occur.

> **IMPLEMENTATION NOTE**
> Implicit Invalidation and PASID
>
> Software may not change any of the PASID enable bits when the E field in the ATS Capability is Set. The invalidation that occurs when software Sets the E field also invalidates ATC entries with an associated PASID value.

10.3.8 PASID TLP Prefix and Global Invalidate

The requirements in this section apply to Functions that support the PASID TLP Prefix. For Invalidation Requests that have a PASID TLP Prefix, the ATC shall:

- Optionally signal Unsupported Request (UR) if the associated PASID value is greater than or equal to $2^{\text{Max PASID Width}}$. This error may be signaled anytime an out of range PASID value is present, even when the PASID value is ignored (see below).
- Return an Invalidation Completion if PASID Enable is Clear.
- If the Function supports Global Invalidate (see Section 7.8.8.2):
  - If the Global Invalidate bit in the Request is Set, invalidate Global and non-Global Mapping entries in the ATC within the indicated memory range associated with any PASID value and return an Invalidation Completion. The PASID value in the PASID TLP Prefix is ignored.
  - If the Global Invalidate bit in the Request is Clear, invalidate only non-Global Mapping entries in the ATC within the indicated memory range that were requested using the associated PASID value and return an Invalidation Completion.

Global Mapping entries in the ATC for some or all of the indicated memory range may be retained.

- If the Function does not support Global Invalidate (see Section 7.8.8.2), invalidate entries in the ATC within the indicated memory range that were requested using the associated PASID value and return an Invalidation Completion.
- If no matching entries are present in the ATC, invalidate no ATC entries and return an Invalidation Completion.

For Invalidation Requests that do not have a PASID TLP Prefix, the ATC shall:

- Invalidate ATC entries within the indicate memory range that were requested without a PASID value.
- Invalidate ATC entries at all addresses that were requested with any PASID value.

10.4 Page Request Services

The general model for a page request is as follows:

1. A Function determines that it requires access to a page for which an ATS translation is not available.
2. The Function causes the associated Page Request Interface to send a Page Request Message to its RC. A Page Request Message contains a page address and a Page Request Group (PRG) index. The PRG index is used to identify the transaction and is used to match requests with responses.
3. When the RC determines its response to the request (which will typically be to make the requested page resident), it sends a PRG Response Message back to the requesting Function.
4. The Function can then employ ATS to request a translation for the requested page(s).

A Page Request Message is a PCIe Message Request that is Routed to the Root Complex with a Message Code of 4 (0000 0100b). The mechanism employed at the RC to buffer requests is implementation specific. The only requirement is that an RC not silently discard requests.

All Page Request Messages and PRG Response Messages travel in PCIe Traffic Class 0. A Page Request Message or PRG Response Message with a Traffic Class other than 0 shall be treated as Malformed TLPs by the RC or endpoint that receives the same. Intermediate routing elements (e.g., Switches) shall not detect this error.

The Relaxed Ordering and ID-Based Ordering bits in the Attr field of Page Request Messages and PRG Response messages may be used. The No Snoop bit in the Attr field is reserved.

The page request service allows grouping of page requests into Page Request Groups (PRGs). A PRG can contain one or more page requests. All pages in a PRG are responded to en mass by the host. Individual pages within a PRG are requested with independent Page Request Messages and are recognized as belonging to a common PRG by sharing the same PRG index. The last request of a PRG is marked as such within its Page Request Message. One request credit is consumed per page request (not per PRG).

A PRG Response Message is a PCIe Message Request that is Routed by ID back to the requesting Function. It is used by system software to alert a Function that the page request(s) associated with the corresponding PRG has (have) been satisfied. The page request mechanism does not guarantee any request completion order and all requests are inherently independent of all other concurrently outstanding requests. If a Function requires that a particular request be completed before another request, the initial request will need to complete before the subsequent request is issued. It is valid for a Function to speculatively request a page without ascertaining its residence state and/or to issue multiple concurrently outstanding requests for the same page.

A Page Request Interface is allocated a specific number of page request message credits. An RC (system software) can divide the available credits in any manner deemed appropriate. Any measures the host chooses to employ to ensure that credits are correctly metered by Page Request Interfaces (a Page Request Interface is not using more than its allocation) is an implementation choice. A Page Request Interface is not allowed to oversubscribe the available number of requests (doing so can result in the page request mechanism being disabled if the buffer limit is exceeded at the root). A Page Request Interface's page request allocation is static. It is determined when the Page Request Interface is enabled and can only be changed by disabling and then re-enabling the interface.

10.4.1 Page Request Message

A Function uses a Page Request Message to send page requests to its associated host. A page request indicates a page needed by the Function. The Page Request Interface associated with a Function is given a specific Page Request allocation. A Page Request Interface shall not issue page requests that exceed its page request allocation.

A page request contains the untranslated address of the page that is needed, the access permissions needed for that page, and a PRG index. A PRG Index is a 9-bit scalar that is assigned by the Function to identify the associated page request. Multiple pages may be requested using a single PRG index. When more than a single page is to be associated with a given PRG, the Last flag in the Page Request Record is cleared in all the requests except the last request associated with a given PRG (the flag is set in the last request). Page requests are responded to en mass. No response is possible (except for a Response Failure error) until the last request of a PRG has been received by the root. The number of PRGs that a Function can have outstanding at any given time is less than or equal to the associated Page Request Interface's Outstanding Page Request Allocation. It is valid for a request group to contain multiple requests for the same page and for multiple outstanding PRGs to request the same page.

The first two DWs of a Page Request Message contain a standard PCIe message header. The second two DWs of the message contain page request specific data fields.

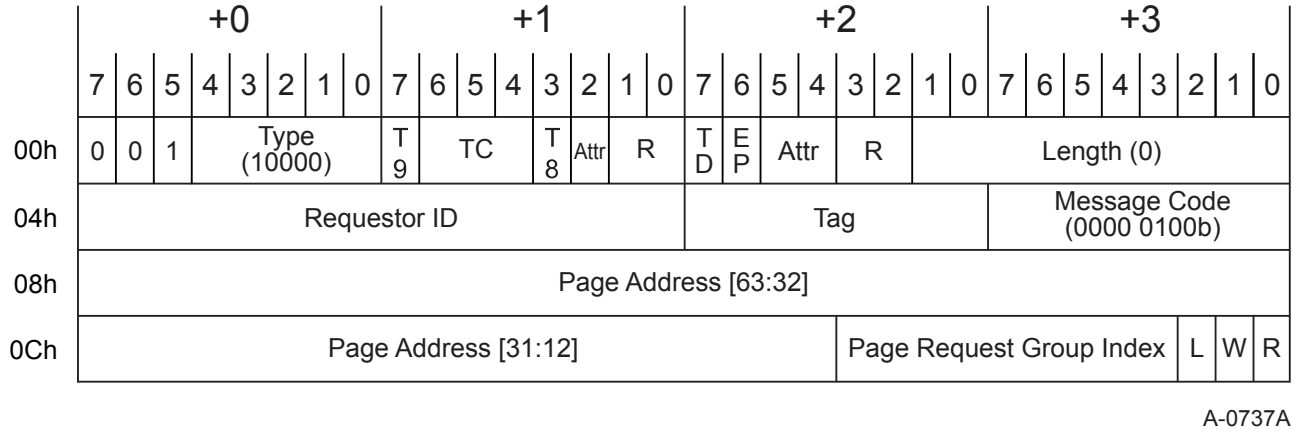

Figure 10-16 Page Request Message

Table 10-5 Page Request Message Data Fields

Field	Meaning
R	**Read Access Requested** - This field, when Set, indicates that the requesting Function seeks read access to the associated page. When Clear, this field indicates that the requesting Function will not read the associated page. The R field must be Set for Page Requests with a PASID TLP Prefix that has the Execute Requested bit Set. If R and W are both Clear and L is Set, this is a Stop Marker (see Section 10.4.1.2.1).
W	**Write Access Requested** - This field, when Set, indicates that the requesting Function seeks write access and/or zero-length read access to the associated page. When Clear, this field indicates that the requesting Function will not write to the associated page. Upon receiving a Page Request Message with the W field Set, the host is permitted to mark the associated page dirty. Thus, Functions must not issue such Requests unless the Function has been given explicit write permission. If R and W are both Clear and L is Set, this is a Stop Marker (see Section 10.4.1.2.1).
L	**Last Request in PRG** - This field, when Set, indicates that the associated page request is the last request of the associated PRG. A PRG can have a single entry, in which case the PRG consists of a single request in which this field is Set. When Clear, this field indicates that additional page requests will be posted using this record's PRG Index.

Field	Meaning
	If R and W are both Clear and L is Set, this is a Stop Marker (see Section 10.4.1.2.1).
Page Request Group Index	**Page Request Group Index** - This field contains a Function supplied identifier for the associated page request. A Function need not employ the entire available range of PRG index values. A host shall never respond with a PRG Index that has not been previously issued by the Function and that is not currently an outstanding request PRG Index (except when issuing a Response Failure, in which case the host need not preserve the associated request's PRG Index value in the error response).
Page Address	**Page Address** - This field contains the untranslated address of the page to be loaded. For pages larger than 4096 bytes, the least significant bits of this field are ignored. For example, the least significant bit of this field is ignored when an 8096-byte page is being requested.

IMPLEMENTATION NOTE
Last Bit and Relaxed Ordering

If multiple page requests are associated with a single PRG index, the last page request of a PRG should have the Relaxed Ordering attribute bit Clear in addition to having the Last flag Set. All other page request messages may have the Relaxed Ordering attribute bit set to any value.

10.4.1.1 PASID TLP Prefix Usage

The PASID Extended Capability indicates whether a Function supports PASID TLP Prefixes and whether it is enabled to send and receive them.

Functions that support the PASID TLP Prefix are permitted to send a PASID TLP Prefix on Page Request Messages. The PASID field contains the process address space of the page being requested and the Execute Requested and Privileged Mode Requested bits indicate the access being requested.

If one Page Request Message in a PRG has a PASID TLP Prefix, all Page Request Messages in that PRG must contain identical PASID TLP Prefixes. Behavior is undefined when the PASID TLP Prefixes are inconsistent.

Functions that support the PASID TLP Prefix and have the PRG Response PASID Required bit Set (see Section 10.5.2.3), expect that PRG Response Messages will contain a PASID TLP Prefix if the associated Page Request Message had a PASID TLP Prefix. For such PRG Response Messages, the Execute Requested and Privileged Mode Requested bits are reserved and the PASID field contains the PASID from the associated Page Request Message.

10.4.1.2 Managing PASID TLP Prefix Usage on PRG Requests

There are rules for stopping and starting the use of a PASID.

This section describes additional rules that apply to Functions that have issued Page Request Messages in a PASID that is being stopped. No additional rules are required to start the usage of the Page Request Interface for a PASID.

When stopping the use of a particular PASID, a Stop Marker Message may be optionally used to avoid waiting for PRG Response Messages before the Function indicates that the stop request for a particular PASID has completed.

To stop without using a Stop Marker Message, the Function shall:

1. Stop queueing new Page Request Messages for this PASID.
2. Finish transmitting any multi-page Page Request Messages for this PASID (i.e. send the Page Request Message with the L bit Set).
3. Wait for PRG Response Messages associated any outstanding Page Request Messages for the PASID.
4. Indicate that the PASID has stopped using a device specific mechanism. This mechanism must indicate that a Stop Marker Message will not be generated.

To stop with the use of a Stop Marker Message the Function shall:

1. Stop queueing new Page Request Messages for this PASID.
2. Finish transmitting any multi-page Page Request Messages for this PASID (i.e. send the Page Request Message with the L bit Set).
3. Internally mark all outstanding Page Request Messages for this PASID as stale. PRG Response Messages associated with these requests will return Page Request Allocation credits and PRG Index values but are otherwise ignored.[170]
4. Indicate that the PASID has stopped using a device specific mechanism. This mechanism must indicate that a Stop Marker Message will be generated.
5. Send a Stop Marker Message to indicate to the host that all subsequent Page Request Messages for this PASID are for a new use of the PASID value.

Note: Steps 4 and 5 may be performed in either order, or in parallel.

10.4.1.2.1 Stop Marker Messages

A Stop Marker Message indicates that a Function has stopped using the Page Request Interface and has transmitted all pending Page Request Messages for a specific PASID. Stop Marker Messages are strongly ordered with respect to Page Request Messages and serve to push Page Request Messages toward the Host. When the Host receives the Stop Marker Message, this indicated that all Page Request Messages associated with the PASID being stopped have been delivered and that any subsequent Page Request Message with the same PASID value are associated with a new incarnation of that PASID value.

Stop Marker Messages do not have a response. They do not have a PRG Index and do not consume Page Request allocation (see Section 10.5.2.5).

The Stop Marker Message bit layout is shown in Figure 10-17 .

170. Page Request Allocation is shared across all PASIDs of the Function (see Section 10.5.2.5). If PRG Response PASID Required is Clear, PRG Index values are shared across all PASIDs of the Function (see Section 10.5.2.3).