For MSI-X, a Function is permitted to cache Address and Data values from unmasked MSI-X Table entries. However, anytime software unmasks a currently masked MSI-X Table entry either by Clearing its Mask bit or by Clearing the Function Mask bit, the Function must update any Address or Data values that it cached from that entry. If software changes the Address or Data value of an entry while the entry is unmasked, the result is undefined.

IMPLEMENTATION NOTE
Per Vector Masking with MSI/MSI-X

Devices and drivers that use MSI or MSI-X have the challenge of coordinating exactly when new interrupt messages are generated. If hardware fails to send an interrupt message that software expects, an interrupt event might be “lost”. If hardware sends an interrupt message that software is not expecting, a “spurious” interrupt might result.

Per-Vector Masking (PVM) can be used to assist in this coordination. For example, when a software interrupt service routine begins, it can mask the vector to help avoid “spurious” interrupts. After the interrupt service routine services all the interrupt conditions that it is aware of, it can unmask the vector. If any interrupt conditions remain, hardware is required to generate a new interrupt message, guaranteeing that no interrupt events are lost.

PVM is a standard feature with MSI-X and an optional[88] feature for MSI. For devices that implement MSI, implementing PVM as well is highly recommended.

6.1.4.6 Hardware/Software Synchronization

If a Function sends messages with the same vector multiple times before being acknowledged by software, only one message is guaranteed to be serviced. If all messages must be serviced, a device driver handshake is required. In other words, once a Function sends Vector A, it cannot send Vector A again until it is explicitly enabled to do so by its device driver (provided all messages must be serviced). If some messages can be lost, a device driver handshake is not required. For Functions that support multiple vectors, a Function can send multiple unique vectors and is guaranteed that each unique message will be serviced. For example, a Function can send Vector A followed by Vector B without any device driver handshake (both Vector A and Vector B will be serviced).

88. Exception: Within an SR-IOV Device, any PFs or VFs that implement MSI must implement MSI PVM

IMPLEMENTATION NOTE

Servicing MSI and MSI-X Interrupts

When system software allocates fewer MSI or MSI-X vectors to a Function than it requests, multiple interrupt sources within the Function, each desiring a unique vector, may be required to share a single vector. Without proper handshakes between hardware and software, hardware may send fewer messages than software expects, or hardware may send what software considers to be extraneous messages.

A rather sophisticated but resource-intensive approach is to associate a dedicated event queue with each allocated vector, with producer and consumer pointers for managing each event queue. Such event queues typically reside in host memory. The Function acts as the producer and software acts as the consumer. Multiple interrupt sources within a Function may be assigned to each event queue as necessary. Each time an interrupt source needs to signal an interrupt, the Function places an entry on the appropriate event queue (assuming there's room), updates a copy of the producer pointer (typically in host memory), and sends an interrupt message with the associated vector when necessary to notify software that the event queue needs servicing. The interrupt service routine for a given event queue processes all entries it finds on its event queue, as indicated by the producer pointer. Each event queue entry identifies the interrupt source and possibly additional information about the nature of the event. The use of event queues and producer/consumer pointers can be used to guarantee that interrupt events won't get dropped when multiple interrupt sources are forced to share a vector. There's no need for additional handshaking between sending multiple messages associated with the same event queue, to guarantee that every message gets serviced. In fact, various standard techniques for "interrupt coalescing" can be used to avoid sending a separate message for every event that occurs, particularly during heavy bursts of events.

In more modest implementations, the hardware design of a Function's MSI or MSI-X logic sends a message any time a transition to assertion would have occurred on the virtual INTx wire if MSI or MSI-X had not been enabled. For example, consider a scenario in which two interrupt events (possibly from distinct interrupt sources within a Function) occur in rapid succession. The first event causes a message to be sent. Before the interrupt service routine has had an opportunity to service the first event, the second event occurs. In this case, only one message is sent, because the first event is still active at the time the second event occurs (a virtual INTx wire signal would have had only one transition to assertion).

One handshake approach for implementations like the above is to use standard Per-Vector Masking, and allow multiple interrupt sources to be associated with each vector. A given vector's interrupt service routine Sets the vector's Mask bit before it services any associated interrupting events and Clears the Mask bit after it has serviced all the events it knows about. (This could be any number of events.) Any occurrence of a new event while the Mask bit is Set results in the Pending bit being Set. If one or more associated events are still pending at the time the vector's Mask bit is Cleared, the Function immediately sends another message.

A handshake approach for MSI Functions that do not implement Per-Vector Masking is for a vector's interrupt service routine to re-inspect all of the associated interrupt events after Clearing what is presumed to be the last pending interrupt event. If another event is found to be active, it is serviced in the same interrupt service routine invocation, and the complete re-inspection is repeated until no pending events are found. This ensures that if an additional interrupting event occurs before a previous interrupt event is Cleared, whereby the Function does not send an additional interrupt message, that the new event is serviced as part of the current interrupt service routine invocation.

This alternative has the potential side effect of one vector's interrupt service routine processing an interrupting event that has already generated a new interrupt message. The interrupt service routine invocation resulting from the new message may find no pending interrupt events. Such occurrences are sometimes referred to as spurious interrupts, and software using this approach must be prepared to tolerate them.

An MSI or MSI-X message, by virtue of being a Posted Request, is prohibited by transaction ordering rules from passing Posted Requests sent earlier by the Function. The system must guarantee that an interrupt service routine invoked as a result of a given message will observe any updates performed by Posted Requests arriving prior to that message. Thus, the interrupt service routine of a device driver is not required to read from a device register in order to ensure data consistency with previous Posted Requests. However, if multiple MSI-X Table entries share the same vector, the interrupt service routine may need to read from some device specific register to determine which interrupt sources need servicing.

6.1.4.7 Message Transaction Reception and Ordering Requirements

As with all Memory Write transactions, the device that includes the target of the interrupt message (the interrupt receiver) is required to complete all interrupt message transactions as a Completer without requiring other transactions to complete first as a Requester. In general, this means that the message receiver must complete the interrupt message transaction independent of when the CPU services the interrupt. For example, each time the interrupt receiver receives an interrupt message, it could Set a bit in an internal register indicating that this message had been received and then complete the transaction on the bus. The appropriate interrupt service routine would later be dispatched because this bit was Set. The message receiver would not be allowed to delay the completion of the interrupt message on the bus pending acknowledgement from the processor that the interrupt was being serviced. Such dependencies can lead to deadlock when multiple devices send interrupt messages simultaneously.

Although interrupt messages remain strictly ordered throughout the PCI Express Hierarchy, the order of receipt of the interrupt messages does not guarantee any order in which the interrupts will be serviced. Since the message receiver must complete all interrupt message transactions without regard to when the interrupt was actually serviced, the message receiver will generally not maintain any information about the order in which the interrupts were received. This is true both of interrupt messages received from different devices and multiple messages received from the same device. If a device requires one interrupt message to be serviced before another, the device must not send the second interrupt message until the first one has been serviced.

6.1.5 PME Support

PCI Express supports power management events from native PCI Express devices as well as PME-capable PCI devices.

PME signaling is accomplished using an in-band Transaction Layer PME Message (PM_PME) as described in Chapter 5 .

6.1.6 Native PME Software Model

PCI Express-aware software can enable a mode where the Root Complex signals PME via an interrupt. When configured for native PME support, a Root Port receives the PME Message and sets the PME Status bit in its Root Status register. If software has set the PME Interrupt Enable bit in the Root Control register to 1b, the Root Port then generates an interrupt.

If the Root Port is enabled for level-triggered interrupt signaling using the INTx messages, the virtual INTx wire must be asserted whenever and as long as all of the following conditions are satisfied:

- The Interrupt Disable bit in the Command register is set to 0b.
- The PME Interrupt Enable bit in the Root Control register is set to 1b.
- The PME Status bit in the Root Status register is set.

Note that all other interrupt sources within the same Function will assert the same virtual INTx wire when requesting service.

If the Root Port is enabled for edge-triggered interrupt signaling using MSI or MSI-X, an interrupt message must be sent every time the logical AND of the following conditions transitions from FALSE to TRUE:

- The associated vector is unmasked (not applicable if MSI does not support PVM).
- The PME Interrupt Enable bit in the Root Control register is set to 1b.
- The PME Status bit in the Root Status register is set.

Note that PME and Hot-Plug Event interrupts (when both are implemented) always share the same MSI or MSI-X vector, as indicated by the Interrupt Message Number field in the PCI Express Capabilities register.

The software handler for this interrupt can determine which device sent the PME Message by reading the PME Requester ID field in the Root Status register in a Root Port. It dismisses the interrupt by writing a 1b to the PME Status bit in the Root Status register. Refer to Section 7.5.3.14 for more details.

Root Complex Event Collectors provide support for the above described functionality for Root Complex Integrated Endpoints (RCiEPs).

6.1.7 Legacy PME Software Model

Legacy software, however, will not understand this mechanism for signaling PME. In the presence of legacy system software, the system power management logic in the Root Complex receives the PME Message and informs system software through an implementation specific mechanism. The Root Complex may utilize the Requester ID in the PM_PME to inform system software which device caused the power management event.

Since it is delivered by a Message, PME has edge-triggered semantics in PCI Express, which differs from the level-triggered PME mechanism used for conventional PCI. It is the responsibility of the Root Complex to abstract this difference from system software to maintain compatibility with conventional PCI systems.

6.1.8 Operating System Power Management Notification

In order to maintain compatibility with non-PCI Express-aware system software, system power management logic must be configured by firmware to use the legacy mechanism of signaling PME by default. PCI Express-aware system software must notify the firmware prior to enabling native, interrupt-based PME signaling. In response to this notification, system firmware must, if needed, reconfigure the Root Complex to disable legacy mechanisms of signaling PME. The details of this firmware notification are beyond the scope of this specification, but since it will be executed at system run-time, the response to this notification must not interfere with system software. Therefore, following control handoff to the operating system, firmware must not write to available system memory or any PCI Express resources (e.g., Configuration Space structures) owned by the operating system.

6.1.9 PME Routing Between PCI Express and PCI Hierarchies

PME-capable conventional PCI and PCI-X devices assert the PME# pin to signal a power management event. The PME# signal from PCI or PCI-X devices may either be converted to a PCI Express in-band PME Message by a PCI Express-PCI Bridge or routed directly to the Root Complex.

If the PME# signal from a PCI or PCI-X device is routed directly to the Root Complex, it signals system software using the same mechanism used in present PCI systems. A Root Complex may optionally provide support for signaling PME from PCI or PCI-X devices to system software via an interrupt. In this scenario, it is recommended for the Root Complex to detect the Bus, Device and Function Number of the PCI or PCI-X device that asserted PME#, and use this information to

fill in the PME Requester ID field in the Root Port that originated the hierarchy containing the PCI or PCI-X device. If this is not possible, the Root Complex may optionally write the Requester ID of the Root Port to this field.

Since RCiEPs are not contained in any of the hierarchy domains originated by Root Ports, RCiEPs not associated with a Root Complex Event Collector signal system software of a PME using the same mechanism used in present PCI systems. A Root Complex Event Collector, if implemented, enables the PCI Express Native PME model for associated RCiEPs.

6.2 Error Signaling and Logging

In this document, errors which must be checked and errors which may optionally be checked are identified. Each such error is associated either with the Port or with a specific device (or Function in a Multi-Function Device), and this association is given along with the description of the error. This section will discuss how errors are classified and reported.

6.2.1 Scope

This section explains the error signaling and logging requirements for PCI Express components. This includes errors which occur on the PCI Express interface itself, those errors which occur on behalf of transactions initiated on PCI Express, and errors which occur within a component and are related to the PCI Express interface. This section does not focus on errors which occur within the component that are unrelated to a PCI Express interface. This type of error signaling is better handled through proprietary methods employing device-specific interrupts.

PCI Express 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 defined for more robust error reporting and is implemented with a specific PCI Express Capability structure (refer to Chapter 7 for a definition of this optional capability). This section explicitly calls out all error handling differences between the baseline and the Advanced Error Reporting Capability.

All PCI Express devices support existing, non-PCI Express-aware, software for error handling by mapping PCI Express errors to existing PCI reporting mechanisms, in addition to the PCI Express-specific mechanisms.

6.2.2 Error Classification

PCI Express errors can be classified as two types: Uncorrectable errors and Correctable errors. This classification separates those errors resulting in functional failure from those errors resulting in degraded performance. Uncorrectable errors can further be classified as Fatal or Non-Fatal (see Figure 6-1).

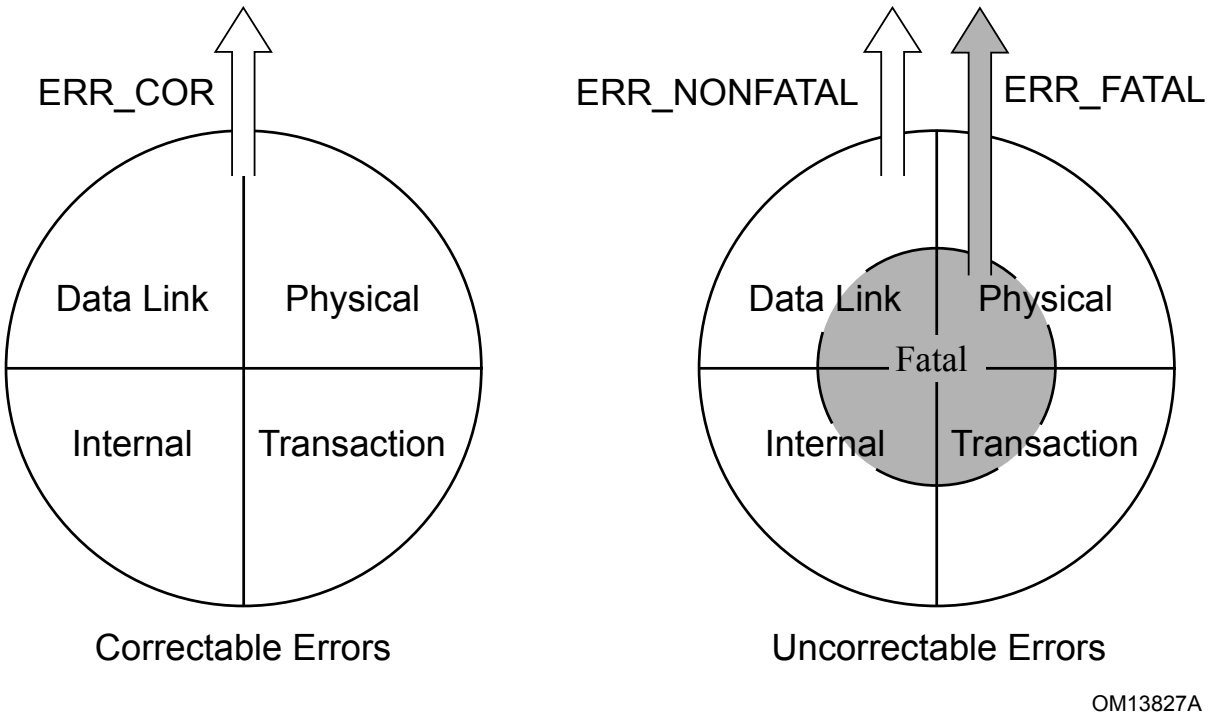

Figure 6-1 Error Classification

Classification of error severity as Fatal, Uncorrectable, and Correctable provides the platform with mechanisms for mapping the error to a suitable handling mechanism. For example, the platform might choose to respond to correctable errors with low priority, performance monitoring software. Such software could count the frequency of correctable errors and provide Link integrity information. On the other hand, a platform designer might choose to map Fatal errors to a system-wide reset. It is the decision of the platform designer to map these PCI Express severity levels onto platform level severities.

6.2.2.1 Correctable Errors

Correctable errors include those error conditions where hardware can recover without any loss of information. Hardware corrects these errors and software intervention is not required. For example, an LCRC error in a TLP that might be corrected by Data Link Level Retry is considered a correctable error. Measuring the frequency of Link-level correctable errors may be helpful for profiling the integrity of a Link.

Correctable errors also include transaction-level cases where one agent detects an error with a TLP, but another agent is responsible for taking any recovery action if needed, such as re-attempting the operation with a separate subsequent transaction. The detecting agent can be configured to report the error as being correctable since the recovery agent may be able to correct it. If recovery action is indeed needed, the recovery agent must report the error as uncorrectable if the recovery agent decides not to attempt recovery.

The triggering of Downstream Port Containment (DPC) is not handled as an error, but it can be signaled as if it were a correctable error, since software that takes advantage of DPC can sometimes recover from the uncorrectable error that triggered DPC. See Section 6.2.10 . An ERR_COR Message that's used for DPC signaling is intended to target system firmware, and may indicate so via the ERR_COR Subclass field.

Similarly, ERR_COR may be used by the System Firmware Intermediary (SFI) capability to signal system firmware, and must indicate so via the ERR_COR Subclass field. See Section 6.7.4 .

6.2.2.2 Uncorrectable Errors

Uncorrectable errors are those error conditions that impact functionality of the interface. There is no mechanism defined in this specification to correct these errors. Reporting an uncorrectable error is analogous to asserting SERR# in PCI/PCI-X. For more robust error handling by the system, this specification further classifies uncorrectable errors as Fatal and Non-fatal.

6.2.2.2.1 Fatal Errors

Fatal errors are uncorrectable error conditions which render the particular Link and related hardware unreliable. For Fatal errors, a reset of the components on the Link may be required to return to reliable operation. Platform handling of Fatal errors, and any efforts to limit the effects of these errors, is platform implementation specific.

6.2.2.2.2 Non-Fatal Errors

Non-fatal errors are uncorrectable errors which cause a particular transaction to be unreliable but the Link is otherwise fully functional. Isolating Non-fatal from Fatal errors provides Requester/Receiver logic in a device or system management software the opportunity to recover from the error without resetting the components on the Link and disturbing other transactions in progress. Devices not associated with the transaction in error are not impacted by the error.

6.2.3 Error Signaling

There are three complementary mechanisms which allow the agent detecting an error to alert the system or another device that an error has occurred. The first mechanism is through a Completion Status, the second method is with in-band error Messages, and the third is with Error Forwarding (also known as data poisoning).

Note that it is the responsibility of the agent detecting the error to signal the error appropriately.

Section 6.2.7 describes all the errors and how the hardware is required to respond when the error is detected.

6.2.3.1 Completion Status

The Completion Status field (when status is not Successful Completion) in the Completion header indicates that the associated Request failed (see Section 2.2.8.10). This is one method of error reporting which enables the Requester to associate an error with a specific Request. In other words, since Non-Posted Requests are not considered complete until after the Completion returns, the Completion Status field gives the Requester an opportunity to “fix” the problem at some higher level protocol (outside the scope of this specification). For example, if a Read is issued to prefetchable Memory Space and the Completion returns with an Unsupported Request Completion Status, the Requester would not be in violation of this specification if it chose to reissue the Read Request. Note that from a PCI Express point of view, the reissued Read Request is a distinct Request, and there is no relationship (on PCI Express) between the initial Request and the reissued Request.

6.2.3.2 Error Messages

Error Messages are sent to the Root Complex for reporting the detection of errors according to the severity of the error.

Error messages that originate from PCI Express or Legacy Endpoints are sent to corresponding Root Ports. Errors that originate from a Root Port itself are reported through the same Root Port.

If an optional Root Complex Event Collector is implemented, errors that originate from RCiEPs are sent to the corresponding Root Complex Event Collector. Errors that originate in a Root Complex Event Collector itself are reported through the same Root Complex Event Collector. The Root Complex Event Collector must declare supported RCiEPs as part of its capabilities; each RCiEP must be associated with no more than one Root Complex Event Collector.

When multiple errors of the same severity are detected, the corresponding error Messages with the same Requester ID may be merged for different errors of the same severity. At least one error Message must be sent for detected errors of each severity level. Note, however, that the detection of a given error in some cases will preclude the reporting of certain errors. Refer to Section 6.2.3.2.3 . Also note special rules in Section 6.2.4 regarding non-Function-specific errors in Multi-Function Devices.

Table 6-1 Error Messages

Error Message	Description
ERR_COR	This Message is issued when the Function or Device detects a correctable error on the PCI Express interface. Refer to Section 6.2.2.1 for the definition of a correctable error.
ERR_NONFATAL	This Message is issued when the Function or Device detects a Non-fatal, uncorrectable error on the PCI Express interface. Refer to Section 6.2.2.2.2 for the definition of a Non-fatal, uncorrectable error.
ERR_FATAL	This Message is issued when the Function or Device detects a Fatal, uncorrectable error on the PCI Express interface. Refer to Section 6.2.2.2.1 for the definition of a Fatal, uncorrectable error.

For these Messages, the Root Complex identifies the initiator of the Message by the Requester ID of the Message header. The Root Complex translates these error Messages into platform level events.

IMPLEMENTATION NOTE

Use of ERR_COR, ERR_NONFATAL, and ERR_FATAL

In [PCIe-1.0] and [PCIe-1.0a], a given error was either correctable, non-fatal, or fatal. Assuming signaling was enabled, correctable errors were always signaled with ERR_COR, non-fatal errors were always signaled with ERR_NONFATAL, and fatal errors were always signaled with ERR_FATAL.

In subsequent specifications that support Role-Based Error Reporting, non-fatal errors are sometimes signaled with ERR_NONFATAL, sometimes signaled with ERR_COR, and sometimes not signaled at all, depending upon the role of the agent that detects the error and whether the agent implements AER (see Section 6.2.3.2.4). On some platforms, sending ERR_NONFATAL will preclude another agent from attempting recovery or determining the ultimate disposition of the error. For cases where the detecting agent is not the appropriate agent to determine the ultimate disposition of the error, a detecting agent with AER can signal the non-fatal error with ERR_COR, which serves as an advisory notification to software. For cases where the detecting agent is the appropriate one, the agent signals the non-fatal error with ERR_NONFATAL.

For a given uncorrectable error that's normally non-fatal, if software wishes to avoid continued hierarchy operation upon the detection of that error, software can configure detecting agents that implement AER to escalate the severity of that error to fatal. A detecting agent (if enabled) will always signal a fatal error with ERR_FATAL, regardless of the agent's role.

Software should recognize that a single transaction can be signaled by multiple agents using different types of error Messages. For example, a poisoned TLP might be signaled by intermediate Receivers with ERR_COR, while the ultimate destination Receiver might signal it with ERR_NONFATAL.

6.2.3.2.1 Uncorrectable Error Severity Programming (Advanced Error Reporting)

For device Functions implementing the Advanced Error Reporting Capability, the Uncorrectable Error Severity register allows each uncorrectable error to be programmed to Fatal or Non-Fatal. Uncorrectable errors are not recoverable using defined PCI Express mechanisms. However, some platforms or devices might consider a particular error fatal to a Link or device while another platform considers that error non-fatal. The default value of the Uncorrectable Error Severity register serves as a starting point for this specification but the register can be reprogrammed if the device driver or platform software requires more robust error handling.

Baseline error handling does not support severity programming.

6.2.3.2.2 Masking Individual Errors

Section 6.2.7 lists all the errors governed by this specification and describes when each of the above error Messages are issued. The transmission of these error Messages by class (correctable, non-fatal, fatal) is enabled using the Reporting Enable bits of the Device Control register (see Section 7.5.3.4) or the SERR# Enable bit in the PCI Command register (see Section 7.5.1.1.3).

For devices implementing the Advanced Error Reporting Capability the Uncorrectable Error Mask register and Correctable Error Mask register allows each error condition to be masked independently. If Messages for a particular class of error are not enabled by the combined settings in the Device Control register and the PCI Command register, then no Messages of that class will be sent regardless of the values for the corresponding mask register.

If an individual error is masked when it is detected, its error status bit is still affected, but no error reporting Message is sent to the Root Complex, and the error is not recorded in the Header Log, TLP Prefix Log, or First Error Pointer.

6.2.3.2.3 Error Pollution

Error pollution can occur if error conditions for a given transaction are not isolated to the most significant occurrence. For example, assume the Physical Layer detects a Receiver Error. This error is detected at the Physical Layer and an error is reported to the Root Complex. To avoid having this error propagate and cause subsequent errors at upper layers (for example, a TLP error at the Data Link Layer), making it more difficult to determine the root cause of the error, subsequent errors which occur for the same packet will not be reported by the Data Link or Transaction layers. Similarly, when the Data Link Layer detects an error, subsequent errors which occur for the same packet will not be reported by the Transaction Layer. This behavior applies only to errors that are associated with a particular packet - other errors are reported for each occurrence.

Corrected Internal Errors are errors whose effect has been masked or worked around by a component; refer to Section 6.2.9 for details. Therefore, Corrected Internal Errors do not contribute to error pollution and should be reported when detected.

For errors detected in the Transaction layer and Uncorrectable Internal Errors, it is permitted and recommended that no more than one error be reported for a single received TLP, and that the following precedence (from highest to lowest) be used:

- Uncorrectable Internal Error
- Receiver Overflow
- Malformed TLP
- ECRC Check Failed

- AtomicOp Egress Blocked
- TLP Prefix Blocked
- ACS Violation
- MC Blocked TLP
- Unsupported Request (UR), Completer Abort (CA), or Unexpected Completion
- Poisoned TLP Received or Poisoned TLP Egress Blocked

The Completion Timeout error is not in the above precedence list, since it is not detected by processing a received TLP. Errors listed under the same bullet are mutually exclusive, so their relative order does not matter.

6.2.3.2.4 Advisory Non-Fatal Error Cases

In some cases the detector of a non-fatal error is not the most appropriate agent to determine whether the error is recoverable or not, or if it even needs any recovery action at all. For example, if software attempts to perform a configuration read from a non-existent device or Function, the resulting UR Status in the Completion will signal the error to software, and software does not need for the Completer in addition to signal the error by sending an ERR_NONFATAL Message. In fact, on some platforms, signaling the error with ERR_NONFATAL results in a System Error, which breaks normal software probing.

"Advisory Non-Fatal Error" cases are predominantly determined by the role of the detecting agent (Requester, Completer, or Receiver) and the specific error. In such cases, an agent with AER signals the non-fatal error (if enabled) by sending an ERR_COR Message as an advisory to software, instead of sending ERR_NONFATAL. An agent without AER sends no error Message for these cases, since software receiving ERR_COR would be unable to distinguish Advisory Non-Fatal Error cases from the correctable error cases used to assess Link integrity.

Following are the specific cases of Advisory Non-Fatal Errors. Note that multiple errors from the same or different error classes (correctable, non-fatal, fatal) may be present with a single TLP. For example, an unexpected Completion might also be poisoned. Refer to Section 6.2.3.2.3 for requirements and recommendations on reporting multiple errors. For the previous example, it is recommended that Unexpected Completion be reported, and that Poisoned TLP Received not be reported.

If software wishes for an agent with AER to handle what would normally be an Advisory Non-Fatal Error case as being more serious, software can escalate the severity of the uncorrectable error to fatal, in which case the agent (if enabled) will signal the error with ERR_FATAL.

This section covers Advisory Non-Fatal Error handling for errors managed by the PCI Express Extended Capability and AER. Section 6.2.10.3 covers the RP PIO error handling mechanism for Root Ports that support RP Extensions for DPC. RP PIO advisory errors are similar in concept to AER Advisory Non-Fatal Errors, but apply to different error cases and are managed by different controls.

6.2.3.2.4.1 Completer Sending a Completion with UR/CA Status

A Completer generally sends a Completion with an Unsupported Request or Completer Abort (UR/CA) Status to signal an uncorrectable error for a Non-Posted Request.[89] If the severity of the UR/CA error[90] is non-fatal, the Completer must

89. If the Completer is returning data in a Completion, and the data is bad or suspect, the Completer is permitted to signal the error using the Error Forwarding (Data Poisoning) mechanism instead of handling it as a UR or CA.
90. Certain other errors (e.g., ACS Violation) with a Non-Posted Request also result in the Completer sending a Completion with UR or CA Status. If the severity of the error (e.g., ACS Violation) is non-fatal, the Completer must also handle this case as an Advisory Non-Fatal Error. However, see Section 2.7.2.2 regarding certain Requests with Poisoned data that must be handled as uncorrectable errors.

handle this case as an Advisory Non-Fatal Error.[91] A Completer with AER signals the non-fatal error (if enabled) by sending an ERR_COR Message. A Completer without AER sends no error Message for this case.

Even though there was an uncorrectable error for this specific transaction, the Completer must handle this case as an Advisory Non-Fatal Error, since the Requester upon receiving the Completion with UR/CA Status is responsible for reporting the error (if necessary) using a Requester-specific mechanism (see Section 6.2.3.2.5).

6.2.3.2.4.2 Intermediate Receiver

When a Receiver that's not serving as the ultimate PCI Express destination for a TLP detects[92] a non-fatal error with the TLP, this "intermediate" Receiver must handle this case as an Advisory Non-Fatal Error.[93] A Receiver with AER signals the error (if enabled) by sending an ERR_COR Message. A Receiver without AER sends no error Message for this case. An exception to the intermediate Receiver case for Root Complexes (RCs) is noted below.

An example where the intermediate Receiver case occurs is a Switch that detects poison or bad ECRC in a TLP that it is routing. Even though this was an uncorrectable (but non-fatal) error at this point in the TLP's route, the intermediate Receiver handles it as an Advisory Non-Fatal Error, so that the ultimate Receiver of the TLP (i.e., the Completer for a Request TLP, or the Requester for a Completion TLP) is not precluded from handling the error more appropriately according to its error settings. For example, a given Completer that detects poison in a Memory Write Request[94] might have the error masked (and thus go unsignaled), whereas a different Completer in the same hierarchy might signal that error with ERR_NONFATAL.

A Poisoned TLP Egress Blocked error is never handled as an intermediate Receiver case since it is not detected as a part of processing a received TLP.

If an RC detects a non-fatal error with a TLP it normally would forward peer-to-peer between Root Ports, but the RC does not support propagating the error related information (e.g., a TLP Digest, EP bit, or equivalent) with the forwarded transaction, the RC must signal the error (if enabled) with ERR_NONFATAL and also must not forward the transaction. An example is an RC needing to forward a poisoned TLP peer-to-peer between Root Ports, but the RC's internal fabric does not support poison indication.

6.2.3.2.4.3 Ultimate PCI Express Receiver of a Poisoned TLP

When a poisoned TLP is received by its ultimate PCI Express destination, if the severity is non-fatal and the Receiver deals with the poisoned data in a manner that permits continued operation, the Receiver must handle this case[95] as an Advisory Non-Fatal Error.[96] A Receiver with AER signals the error (if enabled) by sending an ERR_COR Message. A Receiver without AER sends no error Message for this case. Refer to Section 2.7.2.2 for special rules that apply for poisoned Memory Write Requests.

An example is a Root Complex that receives a poisoned Memory Write TLP that targets host memory. If the Root Complex propagates the poisoned data along with its indication to host memory, it signals the error (if enabled) with an ERR_COR. If the Root Complex does not propagate the poison to host memory, it signals the error (if enabled) with ERR_NONFATAL.

Another example is a Requester that receives a poisoned Memory Read Completion TLP. If the Requester propagates the poisoned data internally or handles the error like it would for a Completion with UR/CA Status, it signals the error (if enabled) with an ERR_COR. If the Requester does not handle the poison in a manner that permits continued operation, it signals the error (if enabled) with ERR_NONFATAL.

91. If the severity is fatal, the error is not an Advisory Non-Fatal Error, and must be signaled (if enabled) with ERR_FATAL.
92. If the Receiver does not implement ECRC Checking or ECRC Checking is not enabled, the Receiver will not detect an ECRC Error.
93. If the severity is fatal, the error is not an Advisory Non-Fatal Error, and must be signaled (if enabled) with ERR_FATAL.
94. See Section 2.7.2.2 for special rules that apply for poisoned Memory Write Requests.
95. However, see Section 2.7.2.2 regarding certain Requests with Poisoned data that must be handled as uncorrectable errors.
96. If the severity is fatal, the error is not an Advisory Non-Fatal Error, and must be signaled (if enabled) with ERR_FATAL.

6.2.3.2.4.4 Requester with Completion Timeout

This section applies to Requesters other than Root Ports performing programmed I/O (PIO). See Section 6.2.10.3 for related RP PIO functionality in Root Ports that support RP Extensions for DPC.

When the Requester of a Non-Posted Request times out while waiting for the associated Completion, the Requester is permitted to attempt to recover from the error by issuing a separate subsequent Request. The Requester is permitted to attempt recovery zero, one, or multiple (finite) times, but must signal the error (if enabled) with an uncorrectable error Message if no further recovery attempt will be made.

If the severity of the Completion Timeout is non-fatal, and the Requester elects to attempt recovery by issuing a new request, the Requester must first handle the current error case as an Advisory Non-Fatal Error.[97] A Requester with AER signals the error (if enabled) by sending an ERR_COR Message. A Requester without AER sends no error Message for this case.

Note that automatic recovery by the Requester from a Completion Timeout is generally possible only if the Non-Posted Request has no side-effects, but may also depend upon other considerations outside the scope of this specification.

6.2.3.2.4.5 Receiver of an Unexpected Completion

When a Receiver receives an unexpected Completion and the severity of the Unexpected Completion error is non-fatal, the Receiver must handle this case as an Advisory Non-Fatal Error.[98] A Receiver with AER signals the error (if enabled) by sending an ERR_COR Message. A Receiver without AER sends no error Message for this case.

If the unexpected Completion was a result of misrouting, the Completion Timeout mechanism at the associated Requester will trigger eventually, and the Requester may elect to attempt recovery. Interference with Requester recovery can be avoided by having the Receiver of the unexpected Completion handle the error as an Advisory Non-Fatal Error.

6.2.3.2.5 Requester Receiving a Completion with UR/CA Status

When a Requester receives back a Completion with a UR/CA Status, generally the Completer has handled the error as an Advisory Non-Fatal Error, assuming the error severity was non-fatal at the Completer (see Section 6.2.3.2.4.1). The Requester must determine if any error recovery action is necessary, what type of recovery action to take, and whether or not to report the error.

If the Requester needs to report the error, the Requester must do so solely through a Requester-specific mechanism. For example, many devices have an associated device driver that can report errors to software. As another important example, the Root Complex on some platforms returns all 1's to software if a Configuration Read Completion has a UR/CA Status.

Section 6.2.10.3 covers RP PIO controls for Root Ports that support RP Extensions for DPC. Outside of the RP PIO mechanisms, Requesters are not permitted to report the error using PCI Express logging and error Message signaling.

6.2.3.3 Error Forwarding (Data Poisoning)

Error Forwarding, also known as data poisoning, is indicated by setting the EP bit in a TLP. Refer to Section 2.7.2 . This is another method of error reporting in PCI Express that enables the Receiver of a TLP to associate an error with a specific

97. If the severity is fatal, the error is not an Advisory Non-Fatal Error, and must be signaled (if enabled) with ERR_FATAL. The Requester is strongly discouraged from attempting recovery since sending ERR_FATAL will often result in the entire hierarchy going down.
98. If the severity is fatal, the error is not an Advisory Non-Fatal Error, and must be signaled (if enabled) with ERR_FATAL.

Request or Completion. Unlike the Completion Status mechanism, Error Forwarding can be used with either Requests or Completions that contain data. In addition, “intermediate” Receivers along the TLP’s route, not just the Receiver at the ultimate destination, are required to detect and report (if enabled) receiving the poisoned TLP. This can help software determine if a particular Switch along the path poisoned the TLP.

6.2.3.4 Optional Error Checking

This specification contains a number of optional error checks. Unless otherwise specified, behavior is undefined if an optional error check is not performed and the error occurs.

When an optional error check involves multiple rules, unless otherwise specified, each rule is independently optional. An implementation may check against all of the rules, none of them or any combination.

Unless otherwise specified, implementation specific criteria are used in determining whether an optional error check is performed.

6.2.4 Error Logging

Section 6.2.7 lists all the errors governed by this specification and for each error, the logging requirements are specified. Device Functions that do not support the Advanced Error Reporting Capability log only the Device Status register bits indicating that an error has been detected. Note that some errors are also reported using the reporting mechanisms in the PCI-compatible (Type 00h and 01h) configuration registers. Section 7.5.1 describes how these register bits are affected by the different types of error conditions described in this section.

For device Functions supporting the Advanced Error Reporting Capability, each of the errors in Table 6-3 , Table 6-4 , and Table 6-5 corresponds to a particular bit in the Uncorrectable Error Status register or Correctable Error Status register. These registers are used by software to determine more precisely which error and what severity occurred. For specific Transaction Layer errors and Uncorrectable Internal Errors, the associated TLP header is recorded.

In a Multi-Function Device, PCI Express errors that are not related to any specific Function within the device, are logged in the corresponding status and logging registers of all Functions in that device.

The following PCI Express errors are not Function-specific:

- All Physical Layer errors
- All Data Link Layer errors
- These Transaction Layer errors:
  - ECRC Check Failed
  - Unsupported Request, when caused by no Function claiming a TLP
  - Receiver Overflow
  - Flow Control Protocol Error
  - Malformed TLP
  - Unexpected Completion, when caused by no Function claiming a Completion
  - Unexpected Completion, when caused by a Completion that cannot be forwarded by a Switch, and the Ingress Port is a Switch Upstream Port associated with a Multi-Function Device
  - Some Transaction Layer errors (e.g., Poisoned TLP Received) may be Function-specific or not, depending upon whether the associated TLP targets a single Function or all Functions in that device.
- Some Internal Errors

- The determination of whether an Internal Error is Function-specific or not is implementation specific.

On the detection of one of these errors, a Multi-Function Device should generate at most one error reporting Message of a given severity, where the Message must report the Requester ID of a Function of the device that is enabled to report that specific type of error. If no Function is enabled to send a reporting Message, the device does not send a reporting Message. If all reporting-enabled Functions have the same severity level set for the error, only one error Message is sent. If all reporting-enabled Functions do not have the same severity level set for the error, one error Message for each severity level is sent. Software is responsible for scanning all Functions in a Multi-Function Device when it detects one of those errors.

6.2.4.1 Root Complex Considerations (Advanced Error Reporting)

6.2.4.1.1 Error Source Identification

In addition to the above logging, a Root Port or Root Complex Event Collector that supports the Advanced Error Reporting Capability is required to implement the Error Source Identification register, which records the Requester ID of the first ERR_NONFATAL/ERR_FATAL (uncorrectable errors) and ERR_COR (correctable errors) Messages received by the Root Port or Root Complex Event Collector. System software written to support Advanced Error Reporting can use the Root Error Status register to determine which fields hold valid information.

If an RCiEP is associated with a Root Complex Event Collector, the RCiEP must report its errors through that Root Complex Event Collector.

For both Root Ports and Root Complex Event Collectors, in order for a received error Message or an internally generated error Message to be recorded in the Root Error Status register and the Error Source Identification register, the error Message must be “transmitted”. Refer to Section 6.2.8.1 for information on how received Messages are forwarded and transmitted. Internally generated error Messages are enabled for transmission with the SERR# Enable bit in the Command register (ERR_NONFATAL and ERR_FATAL) or the Reporting Enable bits in the Device Control register (ERR_COR, ERR_NONFATAL, and ERR_FATAL).

6.2.4.1.2 Interrupt Generation

The Root Error Command register allows further control of Root Complex response to Correctable, Non-Fatal, and Fatal error Messages than the basic Root Complex capability to generate system errors in response to error Messages. Bit fields enable or disable generation of interrupts for the three types of error Messages. System error generation in response to error Messages may be disabled via the PCI Express Capability structure.

If a Root Port or Root Complex Event Collector is enabled for level-triggered interrupt signaling using the INTx messages, the virtual INTx wire must be asserted whenever and as long as all of the following conditions are satisfied:

- The Interrupt Disable bit in the Command register is set to 0b.
- At least one Error Reporting Enable bit in the Root Error Command register and its associated error Messages Received bit in the Root Error Status register are both set to 1b.

Note that all other interrupt sources within the same Function will assert the same virtual INTx wire when requesting service.

If a Root Port or Root Complex Event Collector is enabled for edge-triggered interrupt signaling using MSI or MSI-X, an interrupt message must be sent every time the logical AND of the following conditions transitions from FALSE to TRUE:

- The associated vector is unmasked (not applicable if MSI does not support PVM).

- At least one Error Reporting Enable bit in the Root Error Command register and its associated error Messages Received bit in the Root Error Status register are both set to 1b.

Note that Advanced Error Reporting MSI/MSI-X interrupts always use the vector indicated by the Advanced Error Interrupt Message Number field in the Root Error Status register.

6.2.4.2 Multiple Error Handling (Advanced Error Reporting Capability)

For the Advanced Error Reporting Capability, the Uncorrectable Error Status register and Correctable Error Status register accumulate the collection of errors which correspond to that particular PCI Express interface. The bits remain set until explicitly cleared by software or reset. Since multiple bits might be set in the Uncorrectable Error Status register, the First Error Pointer (when valid) points to the oldest uncorrectable error that is recorded. The First Error Pointer is valid when the corresponding bit of the Uncorrectable Error Status register is set. The First Error Pointer is invalid when the corresponding bit of the Uncorrectable Error Status register is not set, or is an undefined bit.

The Advanced Error Reporting Capability provides the ability to record headers[99] for errors that require header logging. An implementation may support the recording of multiple headers, but at a minimum must support the ability of recording at least one. The ability to record multiple headers is indicated by the state of the Multiple Header Recording Capable bit and enabled by the Multiple Header Recording Enable bit of the Advanced Error Capabilities and Control register. When multiple header recording is supported and enabled, errors are recorded in the order in which they are detected.

If no header recording resources are available when an unmasked uncorrectable error is detected, its error status bit is set, but the error is not recorded. If an uncorrectable error is masked when it is detected, its error status bit is set, but the error is not recorded.

When software is ready to dismiss a recorded error indicated by the First Error Pointer, software writes a 1b to the indicated error status bit to clear it, which causes hardware to free up the associated recording resources. If another instance of that error is still recorded, hardware is permitted but not required to leave that error status bit set. If any error instance is still recorded, hardware must immediately update the Header Log, TLP Prefix Log, TLP Prefix Log Present bit, First Error Pointer, and Uncorrectable Error Status register to reflect the next recorded error. If no other error is recorded, it is recommended that hardware update the First Error Pointer to indicate a status bit that it will never set, e.g., a Reserved status bit. See the Implementation Note below.

If multiple header recording is supported and enabled, and the First Error Pointer is valid, it is recommended that software not write a 1b to any status bit other than the one indicated by the First Error Pointer[100] . If software writes a 1b to such non-indicated bits, hardware is permitted to clear any associated recorded errors, but is not required to do so.

If software observes that the First Error Pointer is invalid, and software wishes to clear any unmasked status bits that were set because of earlier header recording resource overflow, software should be aware of the following race condition. If any new instances of those errors happen to be recorded before software clears those status bits, one or more of the newly recorded errors might be lost.

If multiple header recording is supported and enabled, software must use special care when clearing the Multiple Header Recording Enable bit. Hardware behavior is undefined if software clears that bit while the First Error Pointer is valid. Before clearing the Multiple Header Recording Enable bit, it is recommended that software temporarily mask all uncorrectable errors, and then repetitively dismiss each error indicated by the First Error Pointer.

Since an implementation only has the ability to record a finite number of headers, it is important that software services the First Error Pointer, Header Log, and TLP Prefix Log registers in a timely manner, to limit the risk of missing this information for subsequent errors. A Header Log Overflow occurs when an error that requires header logging is detected

99. If a Function supports TLP Prefixes, then its AER Capability also records any accompanying TLP Prefix along with each recorded header. References to header recording also imply TLP Prefix recording.
100. Status bits for masked errors are an exception. Software can safely clear them if software is certain that they have no recorded headers, as would be the case if they have remained masked since the First Error Pointer was last invalid.

and either the number of recorded headers supported by an implementation has been reached, or the Multiple Header Recording Enable bit is not Set and the First Error Pointer is valid.

Implementations may optionally check for this condition and report a Header Log Overflow error. This is a reported error associated with the detecting Function.

The setting of Multiple Header Recording Capable and the checking for Header Log Overflow are independently optional.

IMPLEMENTATION NOTE
First Error Pointer Register Being Valid

The First Error Pointer (FEP) field is defined to be valid when the corresponding bit of the Uncorrectable Error Status register is set. To avoid ambiguity with certain cases, the following is recommended:

- After an uncorrectable error has been recorded, when the associated bit in the Uncorrectable Error Status register is cleared by software writing a 1b to it, hardware should update the FEP to point to a status bit that it will never set, e.g., a Reserved status bit. (This assumes that the Function does not already have another recorded error to report, as could be the case if it supports multiple header recording.)
- The default value for the FEP should point to a status bit that hardware will never set, e.g., a Reserved status bit.

Here is an example case of ambiguity with Unsupported Request (UR) if the above recommendations are not followed:

- UR and Advisory Non-Fatal Error are unmasked while system firmware does its Configuration Space probing.
- The Function encounters a UR due to normal probing, logs it, and sets the FEP to point to UR.
- System firmware clears the UR Status bit, and hardware leaves the FEP pointing to UR.
- After the operating system has booted, it masks UR.
- Normal probing sets the UR Status bit, but the error is not recorded since UR is masked.

At this point, there's the ambiguity of the FEP pointing to a status bit that is set (thus being valid), when in fact, there is no recorded error that needs to be processed by software.

If hardware relies on this definition of the FEP being valid to determine when it's possible to record a new error, the Function can fail to record new unmasked errors, falsely determining that it has no available recording resources. Hardware implementations that rely on other internal state to determine when it's possible to record a new error might not have this problem; however, hardware implementations should still follow the above recommendations to avoid presenting this ambiguity to software.

6.2.4.3 Advisory Non-Fatal Error Logging

Section 6.2.3.2.4 describes Advisory Non-Fatal Error cases, under which an agent with AER detecting an uncorrectable error of non-fatal severity signals the error (if enabled) using ERR_COR instead of ERR_NONFATAL. For the same cases, an agent without AER sends no error Message. The remaining discussion in this section is in the context of agents that do implement AER.

For Advisory Non-Fatal Error cases, since an uncorrectable error is signaled using the correctable error Message, control/status/mask bits involving both uncorrectable and correctable errors apply. Figure 6-2 shows a flowchart of the sequence. Following are some of the unique aspects for logging Advisory Non-Fatal Errors.

First, the uncorrectable error needs to be of severity non-fatal, as determined by the associated bit in the Uncorrectable Error Severity register. If the severity is fatal, the error does not qualify as an Advisory Non-Fatal Error, and will be signaled (if enabled) with ERR_FATAL.

Next, the specific error case needs to be one of the Advisory Non-Fatal Error cases documented in Section 6.2.3.2.4 . If not, the error does not qualify as an Advisory Non-Fatal Error, and will be signaled (if enabled) with an uncorrectable error Message.

Next, the Advisory Non-Fatal Error Status bit is set in the Correctable Error Status register to indicate the occurrence of the advisory error, and the Advisory Non-Fatal Error Mask bit in the Correctable Error Mask register is checked, and, if set, no further processing is done.

If the Advisory Non-Fatal Error Mask bit is clear, logging proceeds by setting the “corresponding” bit in the Uncorrectable Error Status register, based upon the specific uncorrectable error that’s being reported as an advisory error. If the “corresponding” uncorrectable error bit in the Uncorrectable Error Mask register is clear and the error is one that requires header logging, then the prefix and header are recorded, subject to the availability of resources. See Section 6.2.4.2 .

Finally, an ERR_COR Message is sent if the Correctable Error Reporting Enable bit is set in the Device Control Register.

6.2.4.4 TLP Prefix Logging

For any device Function that supports both TLP Prefixes and Advanced Error Reporting the TLP Prefixes associated with the TLP in error are recorded in the TLP Prefix Log register according to the same rules as the Header Log register (such that both the TLP Prefix Log and Header Log registers always correspond to the error indicated in the First Error Pointer, when the First Error Pointer is valid).

The TLP Prefix Log Present bit (see Section 7.8.4.7) indicates that the TLP Prefix Log register (see Section 7.8.4.12) contains information.

Only End-End TLP Prefixes are logged by AER. Logging of Local TLP Prefixes may occur elsewhere using prefix specific mechanisms.[101]

End-End TLP Prefixes are logged in the TLP Prefix Log register. The underlying TLP Header is logged in the Header Log register subject to two exceptions:

- If the Extended Fmt Field Supported bit is Set (see Section 7.5.3.15), a Function that does not support TLP Prefixes and receives a TLP containing a TLP Prefix will signal Malformed TLP and the Header Log register will contain the first four DWs of the TLP (TLP Prefixes followed by as much of the TLP Header as will fit).
- A Function that receives a TLP containing more End-End TLP Prefixes than are indicated by the Function’s Max End-End TLP Prefixes field must handle the TLP as an error (see Section 2.2.10.2 for specifics) and store the first overflow End-End TLP Prefix in the 1st DW of the Header Log register with the remainder of the Header Log register being undefined.

6.2.5 Sequence of Device Error Signaling and Logging Operations

Figure 6-2 shows the sequence of operations related to signaling and logging of errors detected by a device.

101. For example, errors involving MRI-IOV TLP Prefixes are logged in MR-IOV structures and are not logged in the AER Capability.

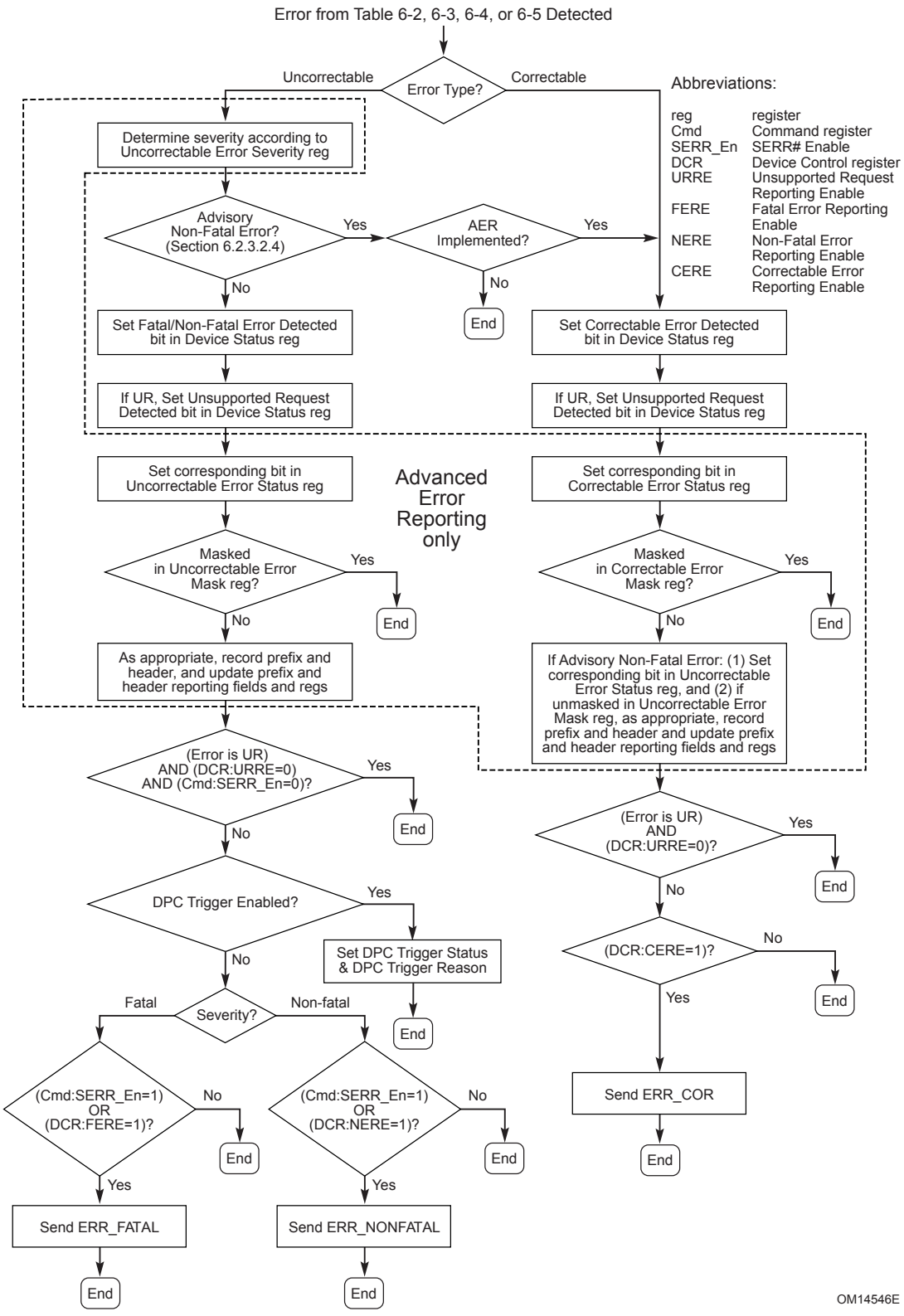

Figure 6-2 Flowchart Showing Sequence of Device Error Signaling and Logging Operations

6.2.6 Error Message Controls

Error Messages have a complex set of associated control and status bits. Figure 6-3 provides a high-level summary in the form of a pseudo logic diagram for how error Messages are generated, logged, forwarded, and ultimately notified to the system. Not all control and status bits are shown. The logic gates shown in this diagram are intended for conveying general concepts, and not for direct implementation.

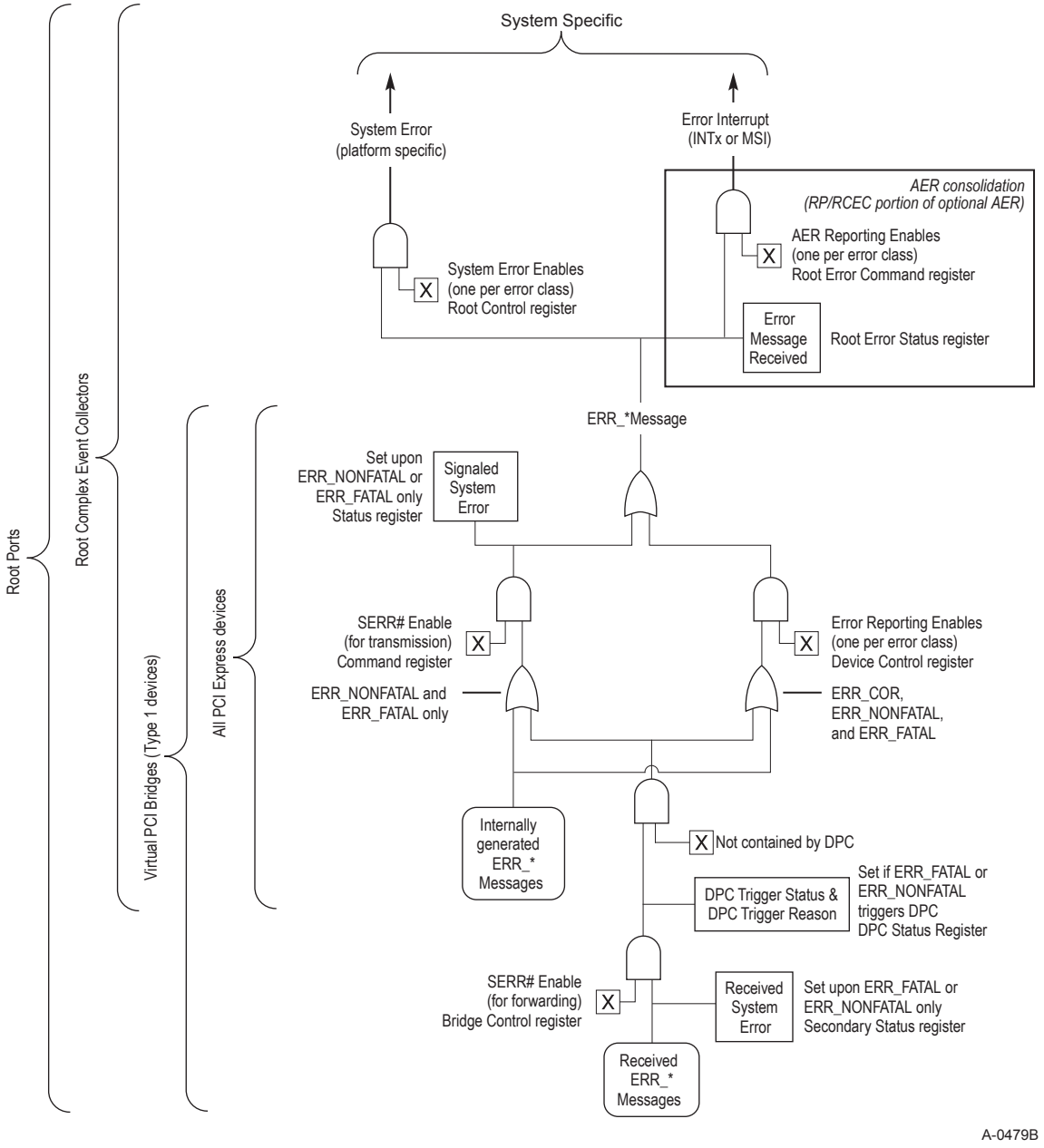

Figure 6-3 Pseudo Logic Diagram for Selected Error Message Control and Status Bits

6.2.7 Error Listing and Rules

Table 6-2 through Table 6-4 list all of the PCI Express errors that are defined by this specification. Each error is listed with a short-hand name, how the error is detected in hardware, the default severity of the error, and the expected action taken by the agent which detects the error. These actions form the rules for PCI Express error reporting and logging.

The Default Severity column specifies the default severity for the error without any software reprogramming. For device Functions supporting the Advanced Error Reporting Capability, the uncorrectable errors are programmable to Fatal or Non-fatal with the Error Severity register. Device Functions without Advanced Error Reporting Capability use the default associations and are not reprogrammable.

The detecting agent action for Downstream Ports that implement Downstream Port Containment (DPC) and have it enabled will be different if the error triggers DPC. DPC behavior is not described in the following tables. See Section 6.2.10 for the description of DPC behavior.

Table 6-2 General PCI Express Error List

Error Name	Error Type (Default Severity)	Detecting Agent Action[102]	References
Corrected Internal Error	Correctable (masked by default)	*Component:* Send ERR_COR to Root Complex.	Section 6.2.9
Uncorrectable Internal Error	Uncorrectable (Fatal and masked by default)	*Component:* Send ERR_FATAL to Root Complex. Optionally, log the prefix/header of the first TLP associated with the error.	Section 6.2.9
Header Log Overflow	Correctable (masked by default)	*Component:* Send ERR_COR to Root Complex.	Section 6.2.4.2

Table 6-3 Physical Layer Error List

Error Name	Error Type (Default Severity)	Detecting Agent Action[103]	References
Receiver Error	Correctable	*Receiver:* Send ERR_COR to Root Complex.	Section 4.2.1.1.3 Section 4.2.1.2 Section 4.2.4.8 Section 4.2.6

Table 6-4 Data Link Layer Error List

Error Name	Error Type (Default Severity)	Detecting Agent Action[104]	References
Bad TLP	Correctable	*Receiver:* Send ERR_COR to Root Complex.	Section 3.6.3.1
Bad DLLP		*Receiver:* Send ERR_COR to Root Complex.	Section 3.6.2.2

102. For these tables, detecting agent action is given as if all enable bits are set to “enable” and, for Advanced Error Handling, mask bits are disabled and severity bits are set to their default values. Actions must be modified according to the actual settings of these bits.
103. For these tables, detecting agent action is given as if all enable bits are set to “enable” and, for Advanced Error Handling, mask bits are disabled and severity bits are set to their default values. Actions must be modified according to the actual settings of these bits.
104. For these tables, detecting agent action is given as if all enable bits are set to “enable” and, for Advanced Error Handling, mask bits are disabled and severity bits are set to their default values. Actions must be modified according to the actual settings of these bits.

Error Name	Error Type (Default Severity)	Detecting Agent Action	References
Replay Timer Timeout		*Transmitter:* Send ERR_COR to Root Complex.	Section 3.6.2.1
REPLAY_NUM Rollover		*Transmitter:* Send ERR_COR to Root Complex.	Section 3.6.2.1
Data Link Protocol Error	Uncorrectable (Fatal)	If checking, send ERR_FATAL to Root Complex.	Section 3.6.2.2
Surprise Down		If checking, send ERR_FATAL to Root Complex.	Section 3.2.1

Table 6-5 Transaction Layer Error List

Error Name	Error Type (Default Severity)	Detecting Agent Action[105]	References
Poisoned TLP Received	Uncorrectable (Non-Fatal)	*Receiver:* Send ERR_NONFATAL to Root Complex or ERR_COR for the Advisory Non-Fatal Error cases described in Section 6.2.3.2.4.1 and Section 6.2.3.2.4.2 . Log the prefix/header of the Poisoned TLP.[106]	Section 2.7.2.2
Poisoned TLP Egress Blocked		*Downstream Port Transmitter:* Send ERR_NONFATAL to Root Complex. Log the prefix/header of the poisoned TLP.	Section 2.7.2.2
ECRC Check Failed		*Receiver (if ECRC checking is supported):* Send ERR_NONFATAL to Root Complex or ERR_COR for the Advisory Non-Fatal Error case described in Section 6.2.3.2.4.1 and Section 6.2.3.2.4.2 . Log the prefix/header of the TLP that encountered the ECRC error.	Section 2.7.1
Unsupported Request (UR)	Uncorrectable (Non-Fatal)	*Request Receiver:* Send ERR_NONFATAL to Root Complex or ERR_COR for the Advisory Non-Fatal Error case described in Section 6.2.3.2.4.1 . Log the prefix/header of the TLP that caused the error.	Table F-1 , Section 2.3.1 , Section 2.3.2 , Section 2.7.2.2 , Section 2.9.1 , Section 5.3.1 , Section 6.2.3.1 , Section 6.2.6 , Section 6.2.8.1 , Section 6.5.7 , Section 7.3.1 , Section 7.3.3 , Section 7.5.1.1.3 , Section 7.5.1.1.4

105. For these tables, detecting agent action is given as if all enable bits are set to “enable” and, for Advanced Error Handling, mask bits are disabled and severity bits are set to their default values. Actions must be modified according to the actual settings of these bits.
106. Advanced Error Handling only.

Error Name	Error Type (Default Severity)	Detecting Agent Action	References
Completion Timeout	Uncorrectable (Non-Fatal)	*Requester:* Send ERR_NONFATAL to Root Complex or ERR_COR for the Advisory Non-Fatal Error case described in Section 6.2.3.2.4.4 . If the Completion Timeout Prefix/Header Log Capable bit is Set in the Advanced Error Capabilities and Control register, log the prefix/header of the Request TLP that encountered the error.	Section 2.8
Completer Abort		*Completer:* Send ERR_NONFATAL to Root Complex or ERR_COR for the Advisory Non-Fatal Error case described in Section 6.2.3.2.4.1 . Log the prefix/header of the Request that encountered the error.	Section 2.3.1
Unexpected Completion		*Receiver:* Send ERR_COR to Root Complex. This is an Advisory Non-Fatal Error case described in Section 6.2.3.2.4.5 . Log the prefix/header of the Completion that encountered the error.	Section 2.3.2
ACS Violation		*Receiver (if checking):* Send ERR_NONFATAL to Root Complex or ERR_COR for the Advisory Non-Fatal Error case described in Section 6.2.3.2.4.1 . Log the prefix/header of the Request TLP that encountered the error.	
MC Blocked TLP		*Receiver (if checking):* Send ERR_NONFATAL to Root Complex. Log the prefix/header of the Request TLP that encountered the error.	Section 6.14.4

Error Name	Error Type (Default Severity)	Detecting Agent Action	References
AtomicOp Egress Blocked	Uncorrectable (Non-Fatal)	*Egress Port:* Send ERR_COR to Root Complex. This is an Advisory Non-Fatal Error case described in Section 6.2.3.2.4.1 . Log the prefix/header of the AtomicOp Request that encountered the error.	Section 6.15.2
TLP Prefix Blocked		*Egress Port:* Send ERR_NONFATAL to Root Complex or ERR_COR for the Advisory Non-Fatal Error case described in Section 6.2.3.2.4.1 . Log the prefix/header of the TLP that encountered the error.	Section 2.2.10.2
Receiver Overflow	Uncorrectable (Fatal)	*Receiver (if checking):* Send ERR_FATAL to Root Complex.	Section 2.6.1.2
Flow Control Protocol Error		*Receiver (if checking):* Send ERR_FATAL to Root Complex.	Section 2.6.1
Malformed TLP		*Receiver:* Send ERR_FATAL to Root Complex. Log the prefix/header of the TLP that encountered the error.	Section 2.2.2 , Section 2.2.3 , Section 2.2.5 , Section 2.2.7 , Section 2.2.8.1 , Section 2.2.8.2 , Section 2.2.8.3 , Section 2.2.8.4 , Section 2.2.8.5 , Section 2.2.8.10 , Section 2.2.10 , Section 2.2.10.1 , Section 2.2.10.2 , Section 2.3 , Section 2.3.1 , Section 2.3.1.1 , Section 2.3.2 , Section 2.5 , Section 2.5.3 , Section 2.6.1 , Section 2.6.1.2 , Section 6.2.4.4 , Section 6.3.2

For all errors listed above, the appropriate status bit(s) must be set upon detection of the error. For Unsupported Request (UR), additional detection and reporting enable bits apply (see Section 6.2.5).

IMPLEMENTATION NOTE

Device UR Reporting Compatibility with Legacy and 1.0a Software

With 1.0a device Functions that do not implement Role-Based Error Reporting,[107] the Unsupported Request Reporting Enable bit in the Device Control Register, when clear, prevents the Function from sending any error Message to signal a UR error. With Role-Based Error Reporting Functions, if the SERR# Enable bit in the Command Register is set, the Function is implicitly enabled[108] to send ERR_NONFATAL or ERR_FATAL messages to signal UR errors, even if the Unsupported Request Reporting Enable bit is clear. This raises a backward compatibility concern with software (or firmware) written for 1.0a devices.

With software/firmware that sets the SERR# Enable bit but leaves the Unsupported Request Reporting Enable and Correctable Error Reporting Enable bits clear, a Role-Based Error Reporting Function that encounters a UR error will send no error Message if the Request was non-posted, and will signal the error with ERR_NONFATAL if the Request was posted. The behavior with non-posted Requests supports PC-compatible Configuration Space probing, while the behavior with posted Requests restores error reporting compatibility with PCI and PCI-X, avoiding the potential in this area for silent data corruption. Thus, Role-Based Error Reporting devices are backward compatible with envisioned legacy and 1.0a software and firmware.

6.2.7.1 Conventional PCI Mapping

In order to support conventional PCI driver and software compatibility, PCI Express error conditions, where appropriate, must be mapped onto the PCI Status register bits for error reporting.

In other words, when certain PCI Express errors are detected, the appropriate PCI Status register bit is set alerting the error to legacy PCI software. While the PCI Express error results in setting the PCI Status register, clearing the PCI Status register will not result in clearing bits in the Uncorrectable Error Status register and Correctable Error Status register. Similarly, clearing bits in the Uncorrectable Error Status register and Correctable Error Status register will not result in clearing the PCI Status register.

The PCI command register has bits which control PCI error reporting. However, the PCI Command register does not affect the setting of the PCI Express error register bits.

6.2.8 Virtual PCI Bridge Error Handling

Virtual PCI Bridge configuration headers are associated with each PCI Express Port in a Root Complex or a Switch. For these cases, PCI Express error concepts require appropriate mapping to the PCI error reporting structures.

6.2.8.1 Error Message Forwarding and PCI Mapping for Bridge - Rules

In general, a TLP is either passed from one side of the Virtual PCI Bridge to the other, or is handled at the ingress side of the Bridge according to the same rules which apply to the ultimate recipient of a TLP. The following rules cover PCI Express specific error related cases. Refer to Section 6.2.6 for a conceptual summary of Error Message Controls.

107. As indicated by the Role-Based Error Reporting bit in the Device Capabilities register. See Section 7.8.3 .
108. Assuming the Unsupported Request Error Mask bit is not set in the Uncorrectable Error Mask Register if the device implements AER.

- If a Request does not address a space mapped to either the Bridge's internal space, or to the egress side of the Bridge, the Request is terminated at the ingress side as an Unsupported Request
- Poisoned TLPs are forwarded according to the same rules as non-Poisoned TLPs
  - When forwarding a Poisoned Request Downstream:
    - Set the Detected Parity Error bit in the Status register
    - Set the Master Data Parity Error bit in the Secondary Status register if the Parity Error Response Enable bit in the Bridge Control register is set
  - When forwarding a Poisoned Completion Downstream:
    - Set the Detected Parity Error bit in the Status register
    - Set the Master Data Parity Error bit in the Status register if the Parity Error Response bit in the Command register is set
  - When forwarding a Poisoned Request Upstream:
    - Set the Detected Parity Error bit in the Secondary Status register
    - Set the Master Data Parity Error bit in the Status register if the Parity Error Response bit in the Command register is set
  - When forwarding a Poisoned Completion Upstream:
    - Set the Detected Parity Error bit in the Secondary Status register
    - Set the Master Data Parity Error bit in the Secondary Status register if the Parity Error Response Enable bit in the Bridge Control register is set
- ERR_COR, ERR_NONFATAL, and ERR_FATAL are forwarded from the secondary interface to the primary interface, if the SERR# Enable bit in the Bridge Control Register is set. A Bridge forwarding an error Message must not set the corresponding Error Detected bit in the Device Status register. Transmission of forwarded error Messages by the primary interface is controlled by multiple bits, as shown in Figure 6-3 .
- For a Root Port, error Messages forwarded from the secondary interface to the primary interface must be enabled for "transmission" by the primary interface in order to cause a System Error via the Root Control register or (when the Advanced Error Reporting Capability is present) reporting via the Root Error Command register and logging in the Root Error Status register and Error Source Identification register.
- For a Root Complex Event Collector (technically not a Bridge), error Messages "received" from associated RCiEPs must be enabled for "transmission" in order to cause a System Error via the Root Control register or (when the Advanced Error Reporting Capability is present) reporting via the Root Error Command register and logging in the Root Error Status register and Error Source Identification register.

6.2.9 Internal Errors

An Internal Error is an error associated with a PCI Express interface that occurs within a component and which may not be attributable to a packet or event on the PCI Express interface itself or on behalf of transactions initiated on PCI Express. The determination of what is considered an Internal Error is implementation specific and is outside the scope of this specification.

Internal Errors may be classified as Corrected Internal Errors or Uncorrectable Internal Errors. A Corrected Internal Error is an error that occurs within a component that has been masked or worked around by hardware without any loss of information or improper operation. An example of a possible Corrected Internal Error is an internal packet buffer memory error corrected by an Error Correcting Code (ECC). An Uncorrectable Internal Error is an error that occurs within a component that results in improper operation of the component. An example of a possible Uncorrectable Internal Error is a memory error that cannot be corrected by an ECC. The only method of recovering from an Uncorrectable Internal Error is reset or hardware replacement.

Reporting of Corrected Internal Errors and Uncorrectable Internal Errors is independently optional. If either is reported, then AER must be implemented.

Header logging is optional for Uncorrectable Internal Errors. When a header is logged, the header is that of the first TLP that was lost or corrupted by the Uncorrectable Internal Error. When header logging is not implemented or a header is not available, a header of all ones is recorded.

Internal Errors that can be associated with a specific PCI Express interface are reported by the Function(s) associated with that Port. Internal Errors detected within Switches that cannot be associated with a specific PCI Express interface are reported by the Upstream Port. Reporting of Internal Errors that cannot be associated with a specific PCI Express interface in all other multi-Port components (e.g., Root Complexes) is outside the scope of this specification.

6.2.10 Downstream Port Containment (DPC)

Downstream Port Containment (DPC) is an optional normative feature of a Downstream Port. DPC halts PCI Express traffic below a Downstream Port after an unmasked uncorrectable error is detected at or below the Port, avoiding the potential spread of any data corruption, and supporting Containment Error Recovery (CER) if implemented by software. A Downstream Port indicates support for DPC by implementing a DPC Extended Capability structure, which contains all DPC control and status bits. See Section 7.9.15 .

DPC is disabled by default, and cannot be triggered unless enabled by software using the DPC Trigger Enable field. When the DPC Trigger Enable field is set to 01b, DPC is enabled and is triggered when the Downstream Port detects an unmasked uncorrectable error or when the Downstream Port receives an ERR_FATAL Message. When the DPC Trigger Enable field is set to 10b, DPC is enabled and is triggered when the Downstream Port detects an unmasked uncorrectable error or when the Downstream Port receives an ERR_NONFATAL or ERR_FATAL Message. In addition to uncorrectable errors of the type managed by the PCI Express Extended Capability and Advanced Error Reporting (AER), RP PIO errors can be handled as uncorrectable errors. See Section 6.2.10.3 . There is also a mechanism described in Section 6.2.10.4 for software or firmware to trigger DPC.

When DPC is triggered due to receipt of an uncorrectable error Message, the Requester ID from the Message is recorded in the DPC Error Source ID Register and that Message is discarded and not forwarded Upstream. When DPC is triggered by an unmasked uncorrectable error, that error will not be signaled with an uncorrectable error Message, even if otherwise enabled. However, when DPC is triggered, DPC can signal an interrupt or send an ERR_COR Message if enabled. See Section 6.2.10.1 and Section 6.2.10.2 .

When DPC is triggered, the Downstream Port immediately Sets the DPC Trigger Status bit and DPC Trigger Reason field to indicate the triggering condition (unmasked uncorrectable error, ERR_NONFATAL, ERR_FATAL, RP_PIO error, or software triggered), and disables its Link by directing the LTSSM to the Disabled state. Once the LTSSM reaches the Disabled state, it remains in that state until the DPC Trigger Status bit is Cleared. To ensure that the LTSSM has time to reach the Disabled state or at least to bring the Link down under a variety of error conditions, software must leave the Downstream Port in DPC until the Data Link Layer Link Active bit in the Link Status Register reads 0b; otherwise, the result is undefined. See Section 7.5.3.8 . See Section 2.9.3 for other important details on Transaction Layer behavior during DPC.

After DPC has been triggered in a Root Port that supports RP Extensions for DPC, the Root Port may require some time to quiesce and clean up its internal activities, such as those associated with DMA read Requests. When the DPC Trigger Status bit is Set and the DPC RP Busy bit is Set, software must leave the Root Port in DPC until the DPC RP Busy bit reads 0b.

After software releases the Downstream Port from DPC, the Port's LTSSM must transition to the Detect state, where the Link will attempt to retrain. Software can use Data Link Layer State Changed interrupts, DL_Active ERR_COR signaling, or both, to signal when the Link reaches the DL_Active state again. See Section 6.7.3.3 and Section 6.2.10.5 .

IMPLEMENTATION NOTE
Data Value of All 1's

Many platforms, including those supporting RP Extensions for DPC, can return a data value of all 1's to software when an error is associated with a PCI Express Configuration, I/O, or Memory Read Request. During DPC, the Downstream Port discards Requests destined for the Link and completes them with an error (i.e., either with an Unsupported Request (UR) or Completer Abort (CA) Completion Status). By ending a series of MMIO or configuration space operations with a read to an address with a known data value not equal to all 1's, software may determine if a Completer has been removed or DPC has been triggered.

Also see the Implementation Note "Use of RP PIO Advisory Error Handling"

IMPLEMENTATION NOTE
Selecting Non-Posted Request Response During DPC

The DPC Completion Control bit determines how a Downstream Port responds to a Non-Posted Request (NPR) received during DPC. The selection needs to take into account how the rest of the platform handles Containment Error Recovery (CER).

While specific CER policy details in a platform are outside the scope of this specification, here are some guidelines based on general considerations.

If the platform or drivers do not support CER policies, it's recommended to select UR Completions, which is the standard behavior when a device is not present.

If the CER strategy relies on software detecting containment by looking for all 1's returned by PIO reads, then a UR Completion may be the more appropriate selection, assuming the RP synthesizes an all 1's return value for PIO reads that return UR Completions. The all 1's synthesis would need to occur for PIO reads that target Configuration Space, Memory Space, and perhaps I/O Space.

If the CER strategy utilizes a mechanism that handles UR and CA Completions differently for PIO reads, then a CA Completion might be the more appropriate selection. CA Completions coming back from a PCIe device normally indicate a device programming model violation, which may need to trigger Port containment and error recovery.

IMPLEMENTATION NOTE
Selecting the DPC Trigger Condition

Non-Fatal Errors are uncorrectable errors that indicate that a particular TLP was unreliable, and in general the associated Function should not continue its normal operation. Fatal errors are uncorrectable errors that indicate that a particular Link and its related hardware are unreliable, and in general the entire hierarchy below that Link should not continue normal operation. This distinction between Non-Fatal and Fatal errors together with the Root Port error containment capabilities can sometimes be used to select the appropriate DPC trigger condition. The following assumes that there is no peer-to-peer traffic between devices.

Some RCs implement a proprietary feature that will be referred to generically as “Function Level Containment” (FLC). This is not an architected feature of PCI Express. A Root Port that implements FLC is capable of containing the traffic associated with a specific Function when a Non-Fatal Error is detected in that traffic. Switch Downstream Ports below a Root Port with FLC should be configured to trigger DPC when the Downstream Port detects an unmasked uncorrectable error itself or when the Downstream Port receives an ERR_FATAL Message. Under this mode, the Switch Downstream Port passes ERR_NONFATAL Messages it receives Upstream without triggering DPC. This enables Root Port FLC to handle Non-Fatal Errors that render a specific Function unreliable and Switch Downstream Port DPC to handle errors that render a sub-tree of the hierarchy domain unreliable. The Downstream Port still needs to trigger DPC for all unmasked uncorrectable errors it detects, since an ERR_NONFATAL it generates will have its own Requester ID, and the FLC hardware in the Root Port would not be able to determine which specific Function below the Switch Downstream Port was responsible for the Non-Fatal Error.

Switch Downstream Ports below a Root Port without FLC should be configured to trigger DPC when the Switch Downstream Port detects an unmasked uncorrectable error or when the Switch Downstream Port receives an ERR_NONFATAL or ERR_FATAL Message. This enables DPC to contain the error to the affected hierarchy below the Link and allow continued normal operation of the unaffected portion of the hierarchy domain.

IMPLEMENTATION NOTE
Software Polling the DPC RP Busy Bit

The DPC RP Busy bit is a means for hardware to indicate to software that the RP needs to remain in DPC containment while the RP does some internal cleanup and quiescing activities. While the details of these activities are implementation specific, the activities will typically complete within a few microseconds or less. However, under worst-case conditions such as those that might occur with certain internal errors in large systems, the busy period might extend substantially, possibly into multiple seconds. If software is unable to tolerate such lengthy delays within the current software context, software may need to rely on using timer interrupts to schedule polling under interrupt.

IMPLEMENTATION NOTE

Determination of DPC Control

DPC may be controlled in some configurations by platform firmware and in other configurations by the operating system. DPC functionality is strongly linked with the functionality in Advanced Error Reporting. To avoid conflicts over whether platform firmware or the operating system have control of DPC, it is recommended that platform firmware and operating systems always link the control of DPC to the control of Advanced Error Reporting.

6.2.10.1 DPC Interrupts

A DPC-capable Downstream Port must support the generation of DPC interrupts. DPC interrupts are enabled by the DPC Interrupt Enable bit in the DPC Control Register. DPC interrupts are indicated by the DPC Interrupt Status bit in the DPC Status Register.

If the Port is enabled for level-triggered interrupt signaling using INTx messages, the virtual INTx wire must be asserted whenever and as long as the following conditions are satisfied:

- The value of the Interrupt Disable bit in the Command register is 0b.
- The value of the DPC Interrupt Enable bit is 1b.
- The value of the DPC Interrupt Status bit is 1b.

Note that all other interrupt sources within the same Function will assert the same virtual INTx wire when requesting service.

If the Port is enabled for edge-triggered interrupt signaling using MSI or MSI-X, an interrupt message must be sent every time the logical AND of the following conditions transitions from FALSE to TRUE:

- The associated vector is unmasked (not applicable if MSI does not support PVM).
- The value of the DPC Interrupt Enable bit is 1b.
- The value of the DPC Interrupt Status bit is 1b.

The Port may optionally send an interrupt message if interrupt generation has been disabled, and the logical AND of the above conditions is TRUE when interrupt generation is subsequently enabled.

The interrupt message will use the vector indicated by the DPC Interrupt Message Number field in the DPC Capability register. This vector may be the same or may be different from the vectors used by other interrupt sources within this Function.

6.2.10.2 DPC ERR_COR Signaling

A DPC-capable Downstream Port must support ERR_COR signaling, independent of whether it supports Advanced Error Reporting (AER) or not. DPC ERR_COR signaling is enabled by the DPC ERR_COR Enable bit in the DPC Control Register. DPC triggering is indicated by the DPC Trigger Status bit in the DPC Status Register. DPC ERR_COR signaling is managed independently of DPC interrupts, and it is permitted to use both mechanisms concurrently.

If the DPC ERR_COR Enable bit is Set, and the Correctable Error Reporting Enable bit in the Device Control Register or the DPC SIG_SFW Enable bit in the DPC Control Register is Set, the Port must send an ERR_COR Message each time the DPC Trigger Status bit transitions from Clear to Set. DPC ERR_COR signaling must not Set the Correctable Error Detected bit in

the Device Status Register, since this event is not handled as an error. If the Downstream Port supports ERR_COR Subclass capability, this DPC ERR_COR signaling event must set the DPC SIG_SFW Status bit in the DPC Status Register and also set the ERR_COR Subclass field in the ERR_COR Message to indicate ECS SIG_SFW.

For a given DPC trigger event, if a Port is going to send both an ERR_COR Message and an MSI/MSI-X transaction, then the Port must send the ERR_COR Message prior to sending the MSI/MSI-X transaction. There is no corresponding requirement if the INTx mechanism is being used to signal DPC interrupts, since INTx Messages won't necessarily remain ordered with respect to ERR_COR Messages when passing through routing elements.

IMPLEMENTATION NOTE

Use of DPC ERR_COR Signaling

It is recommended that operating systems use DPC interrupts for signaling when DPC has been triggered. While DPC ERR_COR signaling indicates the same event, DPC ERR_COR signaling is primarily intended for use by system firmware, when it needs to be notified in order to do its own logging of the event or provide firmware first services.

6.2.10.3 Root Port Programmed I/O (RP PIO) Error Controls

The RP PIO error control registers enable fine-grained control over what happens when Non-Posted Requests that are tracked by the Root Port encounter certain uncorrectable or advisory errors. See Section 2.9.3 for a description of which Non-Posted Requests are tracked. A set of control and status bits exists for receiving Completion with Unsupported Request status (UR Cpl), receiving Completion with Completer Abort status (CA Cpl), and Completion Timeout (CTO) errors. Independent sets of these error bits exist for Configuration Requests, I/O Requests, and Memory Requests. This finer granularity enables more precise error handling for this subset of uncorrectable errors (UR Cpl, CA Cpl, and CTO). As a key example, UR Cpl errors with Memory Read Requests can be configured to trigger DPC for proper containment and error handling, while UR Cpl errors with Configuration Requests can be configured to return all 1's (without trigging DPC) for normal probing and enumeration.

A UR or CA error logged in AER is the result of the Root Port operating in the role of a Completer, and for a received Non-Posted Request, returning a Completion. In contrast, a UR Cpl or CA Cpl error logged as an RP PIO error is the result of the Root Port operating in the role of a Requester, and for an outstanding Non-Posted Request, receiving a Completion. CTO errors logged in both AER and RP PIO are the result of the Root Port operating in the role of a Requester, though the RP PIO error controls support per-space granularity. Depending upon the control register settings, CTO errors can be logged in AER registers, in RP PIO registers, or both. If software unmasks CTO errors in RP PIO, it is recommended that software mask CTO errors in AER in order to avoid unintended interactions.

The RP PIO Header Log Register, RP PIO ImpSpec Log Register, and RP PIO TLP Prefix Log Registers are referred to collectively as the RP PIO log registers. The RP PIO Header Log Register must be implemented; the RP PIO ImpSpec Log Register and RP PIO TLP Prefix Log Register are optional. The RP PIO Log Size field indicates how many DWORDs are allocated for the RP PIO log registers, and from this the allocated size for the RP PIO TLP Prefix Log Register can be calculated. See Section 7.9.15.2 . The RP PIO log registers always record information from a PIO Request, not any associated Completions.

The RP PIO Status, Mask, and Severity registers behave similarly to the Uncorrectable Error Status, Mask, and Severity registers in AER. See Section 7.8.4.2 , Section 7.8.4.3 , and Section 7.8.4.4 >. When an RP PIO error is detected while it is unmasked, the associated bit in the RP PIO Status Register is Set, and the error is recorded in the RP PIO log registers (assuming that RP PIO error logging resources are available). When an RP PIO error is detected while it is masked, the associated bit is still Set in the RP PIO Status Register, but the error does not trigger DPC and the error is not recorded in the RP PIO log registers.

Each unmasked RP PIO error is handled either as uncorrectable or advisory, as determined by the value of the corresponding bit in the RP PIO Severity Register. If the associated Severity bit is Set, the error is handled as uncorrectable, triggering DPC (assuming that DPC is enabled) and signaling this event with a DPC interrupt and/or ERR_COR (if enabled). If the associated Severity bit is Clear, the error is handled as advisory (without triggering DPC) and signaled with ERR_COR (if enabled).

IMPLEMENTATION NOTE
Use of RP PIO Advisory Error Handling

Each RP PIO error can be handled either as uncorrectable or advisory. Uncorrectable error handling usually logs the error, triggers DPC, and signals the event either with a DPC interrupt, an ERR_COR, or both. Advisory error handling usually logs the error and signals the event with ERR_COR.

RP PIO advisory error handling can be used by software in certain cases to handle RP PIO errors robustly without incurring the disruption caused if DPC is triggered in the RP. If an RP PIO Exception is not enabled for a given error, an all 1's value is returned whenever the error occurs. If the error does not trigger DPC, software may be uncertain if the all 1's value returned by a given PIO read is the actual data value returned by the Completion versus indicating that an error occurred with that PIO read. If software enables advisory error handling for that error, instances of that error will be logged, enabling software to distinguish the two cases.

The use of RP PIO advisory error handling is notably beneficial if DPC is triggered in a Switch Downstream Port, and that causes one or more Completion Timeouts in the RP as a side-effect, as described in Section 2.9.3 . If the RP handles Completion Timeout errors as advisory, this avoids DPC being triggered in the RP, permitting continued operation with the other Switch Downstream Ports.

The RP PIO First Error Pointer, RP PIO Header Log, and RP PIO TLP Prefix Log behave similarly to the First Error Pointer, Header Log, and TLP Prefix Log in AER. The RP PIO First Error Pointer is defined to be valid when its value indicates a bit in the RP PIO Status Register that is Set. When the RP PIO First Error Pointer is valid, the RP PIO log registers contain the information associated with the indicated error. The RP PIO ImpSpec Log, if implemented, contains implementation-specific information, e.g., the source of the Request TLP.

In contrast to AER, where the recording of CTO error information in the AER log registers is optional, RP PIO implementations must support recording RP PIO CTO error information in the RP PIO log registers.

If an error is detected with a received Completion TLP associated with an outstanding PIO Request, the set of RP PIO error control bits used to govern the error handling is determined in a similar manner. The DPC Completion Control bit determines whether UR or CA applies, and the Space (Configuration, I/O, or Memory) is that of the associated PIO Request. For example, if the DPC Completion Control bit is configured for CA, and a Root Port receives a poisoned Completion for a PIO Memory Read Request, the Mem CA Cpl bit (bit 17) is used in the RP PIO control and status registers for handling the error.

The RP PIO SysError Register provides a means to generate a System Error when an RP PIO error occurs. If an unmasked RP PIO error is detected while its associated bit in the RP PIO SysError Register is Set, a System Error is generated.

The RP PIO Exception Register provides a means to generate a synchronous processor exception[109] when an error occurs with certain tracked Non-Posted Requests that are generated by a processor instruction. See Section 2.9.3 . This exception must support all such tracked read Requests, and may optionally support Configuration write, I/O write, and AtomicOp Requests. If an error with an exception-supported Non-Posted Request is detected[110] or a Completion for it is synthesized, and its associated bit in the RP PIO Exception Register is Set, the processor instruction that generated the Non-Posted Request must take a synchronous exception. This still applies even if the RP PIO or AER controls specify that the error be handled as masked or advisory.

109. "Exception" is used as a generic term for a variety of mechanisms used by processors, including interrupts, traps, machine checks, instruction aborts, etc.
110. This includes any errors with the Completion TLP itself (e.g., Malformed TLP) or where the Completion Status is other than Successful Completion.

The details of a processor instruction taking a synchronous exception are processor-specific, but at a minimum, the mechanism must be able to interrupt the normal processor instruction flow either before completion of the instruction that generated the Non-Posted Request, or immediately following that instruction. The intent is that exception handling routines in system firmware, the operating system, or both, can examine the cause of the exception and take corrective action if necessary.

If an RP PIO error occurs with a processor-generated read or AtomicOp Request, and the RP PIO Exception Register value does not cause an exception, a value of all 1's must be returned for the instruction that generated the Request.

IMPLEMENTATION NOTE
Synchronous Exception Implementation

The exact mechanism for implementing synchronous exceptions is processor and platform specific. One possible implementation is poisoning the data returned to the processor for a read or AtomicOp Request that encounters an error. While this approach is likely to work with those Requests, it might not work with Configuration and I/O write Requests since they return no data.

Another possible implementation is marking the response transaction for processor-generated Non-Posted Requests with some other type of indication of the Request having failed, e.g., a "hard fail" response. This approach is more likely to work with all processor-generated Non-Posted Requests.

IMPLEMENTATION NOTE
RP PIO Mask Bit Behavior and Rationale

For a given RP PIO error, the associated mask bit in the RP PIO Mask Register affects its associated status bit setting, error logging, and error signaling in a manner that closely parallels the behavior of mask bits in AER.

SysError generation for a given RP PIO error is primarily controlled by the associated bit in the RP PIO SysError Register, but is also contingent upon the associated RP PIO mask bit being Clear. This behavior was chosen for consistency with AER, and also since it is poor practice to generate a SysError without logging the reason.

Exception generation for a given RP PIO error is independent of the associated RP PIO mask bit value. Usage Models are envisioned where an RP PIO error needs to generate an Exception without logging an RP PIO error or triggering DPC.

Root Port error handling for tracked Non-Posted Requests with errors other than receiving UR and CA Completions is governed by a combination of AER and RP PIO error controls. Examples are CTO[111] , Poisoned TLP Received, and Malformed TLP. For a given error managed by AER, the associated AER Mask and Severity bits determine if the error must be handled as an uncorrectable error, handled as an Advisory Non-Fatal Error, or handled as a masked error.

- If the AER-managed error is to be handled as an uncorrectable error (see Section 6.2.2.2), DPC is triggered. The RP PIO SysError and RP PIO Exception bits associated with the Request type and Completion Status apply.
- If the AER-managed error is to be handled as an Advisory Non-Fatal Error (see Section 6.2.3.2.4), DPC is not triggered. The RP PIO SysError and RP PIO Exception bits do apply.
- If the AER-managed error is to be handled as a masked error (see Section 6.2.3.2.2), DPC is not triggered. RP PIO SysError bit does not apply, but the RP PIO Exception bit does apply.

111. CTO errors have status and mask bits in both AER and RP PIO, though RP PIO has independent sets of bits for each of the 3 spaces. Other errors in AER have no equivalent errors in RP PIO.

6.2.10.4 Software Triggering of DPC

If the DPC Software Triggering Supported bit in the DPC Capability register is Set, then software can trigger DPC by writing a 1b to the DPC Software Trigger bit in the DPC Control Register, assuming that DPC is enabled and the Port isn't currently in DPC. This mechanism is envisioned to be useful for software and/or firmware development and testing. It also supports usage models where software or firmware examines RP PIO Exceptions or RP PIO advisory errors, and decides to trigger DPC based upon the situation.

When this mechanism triggers DPC, the DPC Trigger Reason and DPC Trigger Reason Extension fields in the DPC Status Register will indicate this as the reason.

If a Port is already in DPC when a 1b is written to the DPC Software Trigger bit, the Port remains in DPC, and the DPC Trigger Reason and DPC Trigger Reason Extension fields are not modified.

IMPLEMENTATION NOTE

Avoid Disable Link and Hot-Plug Surprise Use With DPC

It is recommended that software not Set the Link Disable bit in the Link Control register while DPC is enabled but not triggered. Setting the Link Disable bit will cause the Link to be directed to DL_Down, invoking some semantics similar to those in DPC, but lacking others. If DPC is enabled, the subsequent arrival of any Posted Requests will likely trigger DPC anyway. If DPC is enabled, the recommended method for software to disable the Link is to write a 1b to the optional DPC Software Trigger bit in the DPC Control Register. If the DPC Software Trigger bit is not implemented, software should disable DPC and use Link Disable instead. If the operating system is performing this action, but DPC is owned by system firmware, the operating system should coordinate disabling DPC with system firmware.

DPC is not recommended for use concurrently with the Hot-Plug Surprise mechanism, indicated by the Hot-Plug Surprise bit in the Slot Capabilities register being Set. Having this bit Set blocks the reporting of Surprise Down errors, preventing DPC from being triggered by this important error, greatly reducing the benefit of DPC. See Section 6.7.4.5 for guidance on slots supporting both mechanisms.

6.2.10.5 DL_Active ERR_COR Signaling

Support for this feature is indicated by the DL_Active ERR_COR Signaling Supported bit in the DPC Capability register. The feature is enabled by the DL_ACTIVE ERR_COR Enable bit in the DPC Control Register. The DL_ACTIVE state is indicated by the Data Link Layer Link Active bit in the Link Status Register. DL_ACTIVE ERR_COR signaling is managed independently of Data Link Layer State Changed interrupts, and it is permitted to use both mechanisms concurrently.

If the DL_ACTIVE ERR_COR Enable bit is Set, and the Correctable Error Reporting Enable bit in the Device Control register or the DPC SIG_SFW Enable bit in the DPC Control Register is Set, the Port must send an ERR_COR Message each time the Link transitions into the DL_ACTIVE state. DL_ACTIVE ERR_COR signaling must not Set the Correctable Error Detected bit in the Device Status register, since this event is not handled as an error. If the Downstream Port supports ERR_COR Subclass capability, this DPC ERR_COR signaling event must set the DPC SIG_SFW Status bit in the DPC Status register and also set the ERR_COR Subclass field in the ERR_COR Message to indicate ECS SIG_SFW. In contrast to Data Link Layer State Changed interrupts, DL_Active ERR_COR signaling only indicates the Link enters the DL_Active state, not when the Link exits the DL_Active state.

For a given DL_ACTIVE event, if a Port is going to send both an ERR_COR Message and an MSI/MSI-X transaction, then the Port must send the ERR_COR Message prior to sending the MSI/MSI-X transaction. There is no corresponding

requirement if the INTx mechanism is being used to signal DL_ACTIVE interrupts, since INTx Messages won't necessarily remain ordered with respect to ERR_COR Messages when passing through routing elements.

IMPLEMENTATION NOTE

Use of DL_ACTIVE ERR_COR Signaling

It is recommended that operating systems use Data Link Layer State Changed interrupts for signaling when DL_ACTIVE changes state. While DL_ACTIVE ERR_COR signaling indicates a subset of the same events, DL_ACTIVE ERR_COR signaling is primarily intended for use by system firmware, when it needs to be notified in order to do Downstream Port configuration or provide firmware first services.

6.3 Virtual Channel Support

6.3.1 Introduction and Scope

The Virtual Channel mechanism provides a foundation for supporting differentiated services within the PCI Express fabric. It enables deployment of independent physical resources that together with traffic labeling are required for optimized handling of differentiated traffic. Traffic labeling is supported using Traffic Class TLP-level labels. The policy for traffic differentiation is determined by the TC/VC mapping and by the VC-based, Port-based, and Function-based arbitration mechanisms. The TC/VC mapping depends on the platform application requirements. These requirements drive the choice of the arbitration algorithms and configurability/programmability of arbiters allows detailed tuning of the traffic servicing policy.

The definition of the Virtual Channel and associated Traffic Class mechanisms is covered in Chapter 2 . The VC configuration/programming model is defined in Section 7.9.1 and Section 7.9.2 .

This section covers VC mechanisms from the system perspective. It addresses the next level of details on:

- Supported TC/VC configurations
- VC-based arbitration - algorithms and rules
- Traffic ordering considerations
- Isochronous support as a specific usage model

6.3.2 TC/VC Mapping and Example Usage

A Virtual Channel is established when one or more TCs are associated with a physical resource designated by a VC ID. Every Traffic Class that is supported on a given path within the fabric must be mapped to one of the enabled Virtual Channels. Every Port must support the default TC0/VC0 pair - this is "hardwired". Any additional TC mapping or additional VC resource enablement is optional and is controlled by system software using the programming model described in Sections 7.9.1 and 7.9.2.

The number of VC resources provisioned within a component or enabled within a given fabric may vary due to implementation and usage model requirements, due to Hot-Plug of disparate components with varying resource capabilities, or due to system software restricting what resources may be enabled on a given path within the fabric.

Some examples to illustrate:

- A set of components (Root Complex, Endpoints, Switches) may only support the mandatory VC0 resource that must have TC0 mapped to VC0. System software may, based on application usage requirements, map one or all non-zero TCs to VC0 as well on any or all paths within the fabric.
- A set of components may support two VC resources, e.g., VC0 and VC1. System software must map TC0/VC0 and in addition, may map one or all non-zero TC labels to either VC0 or VC1. As above, these mappings may be enabled on any or all paths within the fabric. Refer to the examples below for additional information.
- A Switch may be implemented with eight Ports - seven x1 Links with two VC resources and one x16 Link with one VC resource. System software may enable both VC resources on the x1 Links and assign one or more additional TCs to either VC thus allowing the Switch to differentiate traffic flowing between any Ports. The x16 Link must also be configured to map any non-TC0 traffic to VC0 if such traffic is to flow on this Link. Note: multi-Port components (Switches and Root Complex) are required to support independent TC/VC mapping per Port.

In any of the above examples, system software has the ability to map one, all, or a subset of the TCs to a given VC. Should system software wish to restrict the number of traffic classes that may flow through a given Link, it may configure only a subset of the TCs to the enabled VC resources. Any TLP indicating a TC that has not been mapped to an enabled VC resource must be treated as a Malformed TLP. This is referred to as TC Filtering. Flow Control credits for this TLP will be lost, and an uncorrectable error will be generated, so software intervention will usually be required to restore proper operation after a TC Filtering event occurs.

A graphical example of TC filtering is illustrated in Figure 6-4 , where TCs (2:6) are not mapped to the Link that connects Endpoint A and the Switch. This means that the TLPs with TCs (2:6) are not allowed between the Switch and Endpoint A.

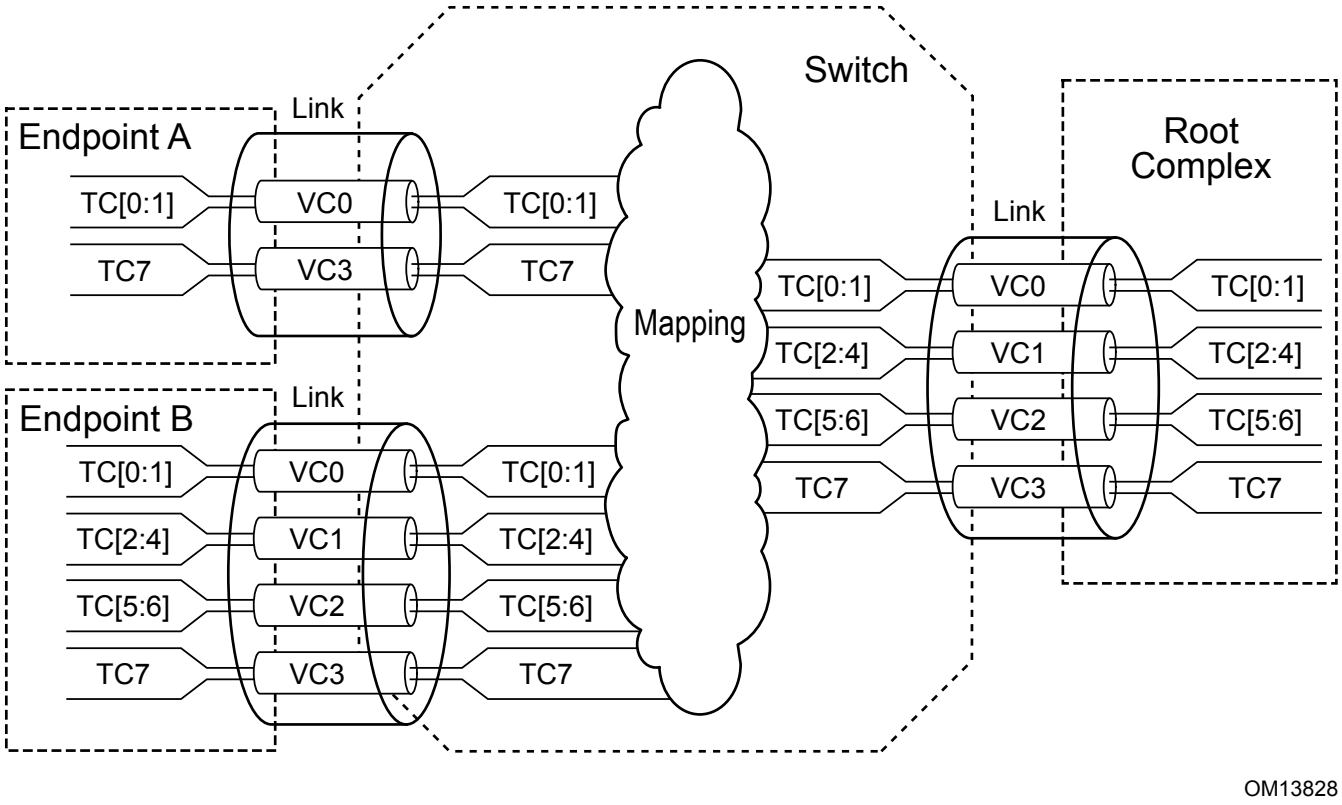

Figure 6-4 TC Filtering Example

Figure 6-5 shows an example of TC to VC mapping. A simple Switch with one Downstream Port and one Upstream Port connects an Endpoint to a Root Complex. At the Upstream Port, two VCs (VC0 and VC1) are enabled with the following mapping: TC(0-6)/VC0, TC7/VC1. At the Downstream Port, only VC0 is enabled and all TCs are mapped to VC0. In this example while TC7 is mapped to VC0 at the Downstream Port, it is re-mapped to VC1 at the Upstream Port. Although the Endpoint only supports VC0, when it labels transactions with different TCs, transactions associated with TC7 from/to the Endpoint can take advantage of the second Virtual Channel enabled between the Switch and the Root Complex.

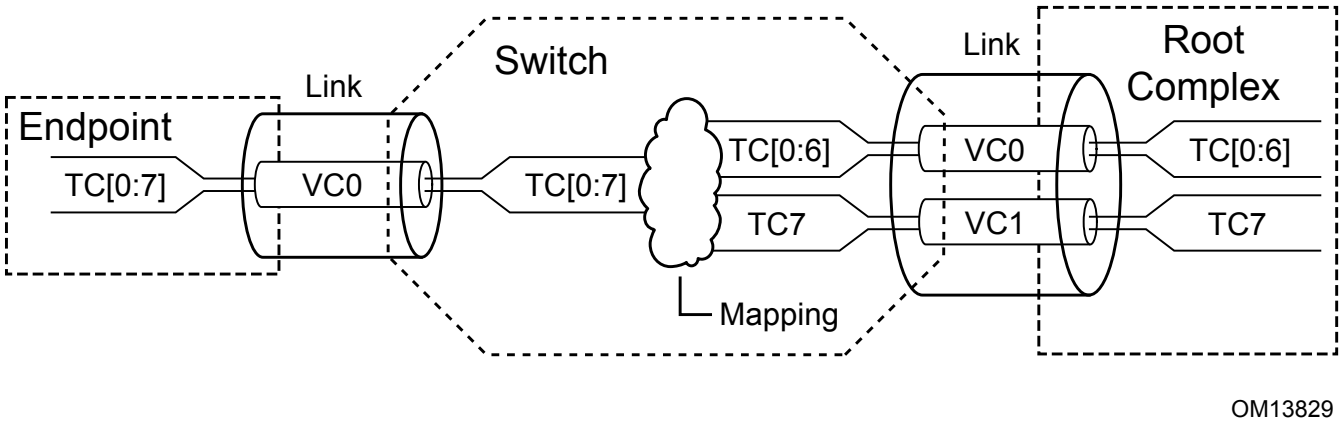

Figure 6-5 TC to VC Mapping Example

IMPLEMENTATION NOTE
Multiple TCs Over a Single VC

A single VC implementation may benefit from using multiple TCs. TCs provide ordering domains that may be used to differentiate traffic within the Endpoint or the Root Complex independent of the number of VCs supported.

In a simple configuration, where only VC0 is supported, traffic differentiation may not be accomplished in an optimum manner since the different TCs cannot be physically segregated. However, the benefits of carrying multiple TCs can still be exploited particularly in the small and “shallow” topologies where Endpoints are connected directly to Root Complex rather than through cascaded Switches. In these topologies traffic that is targeting Root Complex only needs to traverse a single Link, and an optimized scheduling of packets on both sides (Endpoint and Root Complex) based on TCs may accomplish significant improvement over the case when a single TC is used. Still, the inability to route differentiated traffic through separate resources with fully independent flow control and independent ordering exposes all of the traffic to the potential head-of-line blocking conditions. Optimizing Endpoint internal architecture to minimize the exposure to the blocking conditions can reduce those risks.

6.3.3 VC Arbitration

Arbitration is one of the key aspects of the Virtual Channel mechanism and is defined in a manner that fully enables configurability to the specific application. In general, the definition of the VC-based arbitration mechanism is driven by the following objectives:

- To prevent false transaction timeouts and to guarantee data flow forward progress
- To provide differentiated services between data flows within the fabric

- To provide guaranteed bandwidth with deterministic (and reasonably small) end-to-end latency between components

Links are bidirectional, i.e., each Port can be an Ingress or an Egress Port depending on the direction of traffic flow. This is illustrated by the example of a 3-Port Switch in Figure 6-6 , where the paths for traffic flowing between Switch Ports are highlighted with different types of lines. In the following sections, VC Arbitration is defined using a Switch arbitration model since the Switch represents a functional superset from the arbitration perspective.

In addition, one-directional data flow is used in the description.

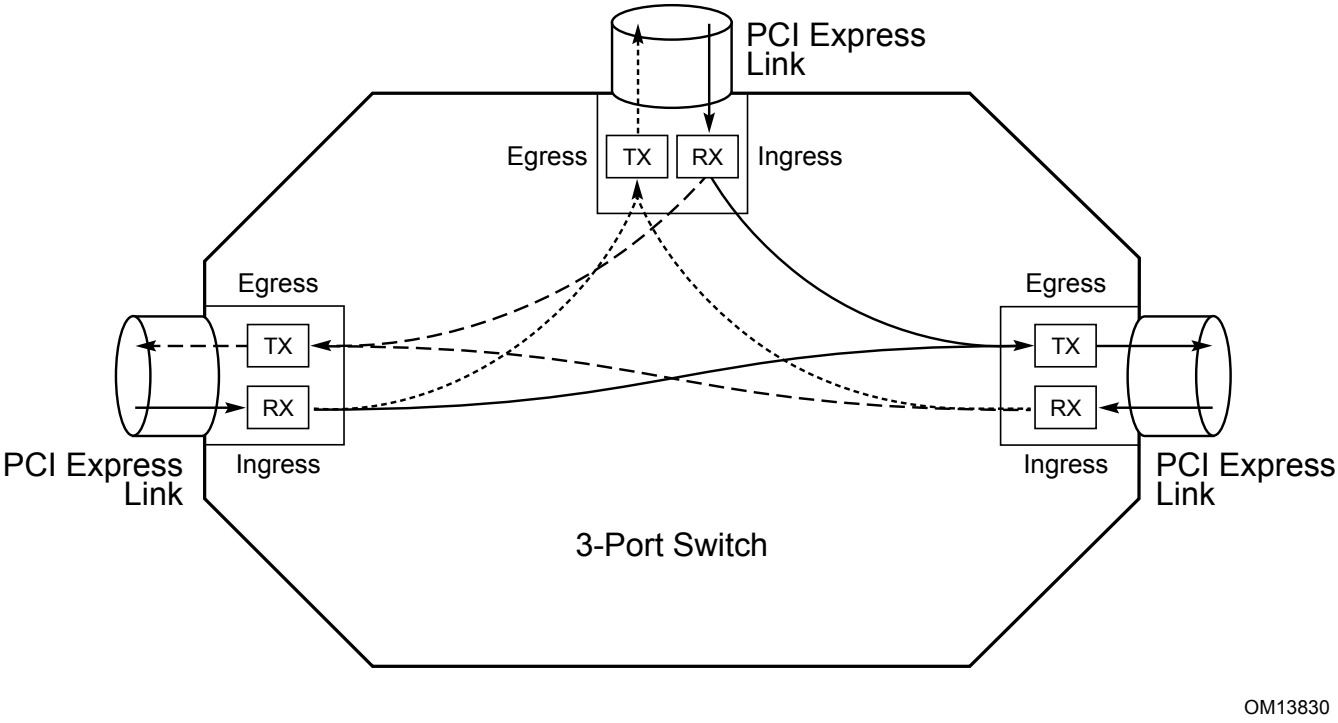

Figure 6-6 An Example of Traffic Flow Illustrating Ingress and Egress

6.3.3.1 Traffic Flow and Switch Arbitration Model

The following set of figures (Figure 6-7 and Figure 6-8) illustrates traffic flow through the Switch and summarizes the key aspects of the arbitration.

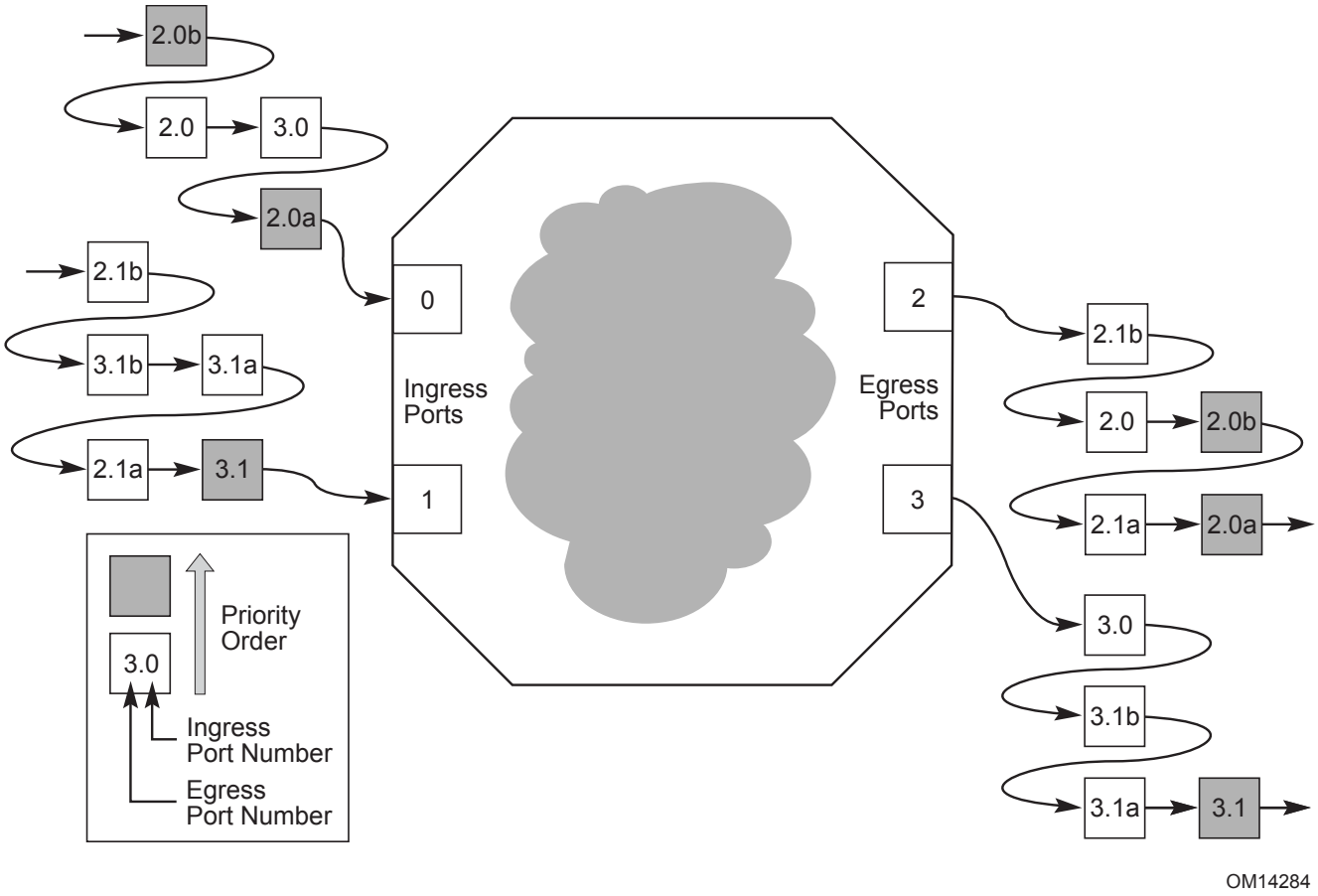

Figure 6-7 An Example of Differentiated Traffic Flow Through a Switch

At each Ingress Port an incoming traffic stream is represented in Figure 6-7 by small boxes. These boxes represent packets that are carried within different VCs that are distinguished using different levels of gray. Each of the boxes that represents a packet belonging to different VC includes designation of Ingress and Egress Ports to indicate where the packet is coming from and where it is going to. For example, designation "3.0" means that this packet is arriving at Port #0 (Ingress) and is destined to Port #3 (Egress). Within the Switch, packets are routed and serviced based on Switch internal arbitration mechanisms.

Switch arbitration model defines a required arbitration infrastructure and functionality within a Switch. This functionality is needed to support a set of arbitration policies that control traffic contention for an Egress Port from multiple Ingress Ports.

Figure 6-8 shows a conceptual model of a Switch highlighting resources and associated functionality in ingress to egress direction. Note that each Port in the Switch can have the role of an Ingress or Egress Port. Therefore, this figure only shows one particular scenario where the 4-Port Switch in this example has ingress traffic on Port #0 and Port #1, that targets Port #2 as an Egress Port. A different example may show different flow of traffic implying different roles for Ports on the Switch. The PCI Express architecture enables peer-to-peer communication through the Switch and, therefore, possible scenarios using the same example may include multiple separate and simultaneous ingress to egress flows (e.g., Port 0 to Port 2 and Port 1 to Port 3).

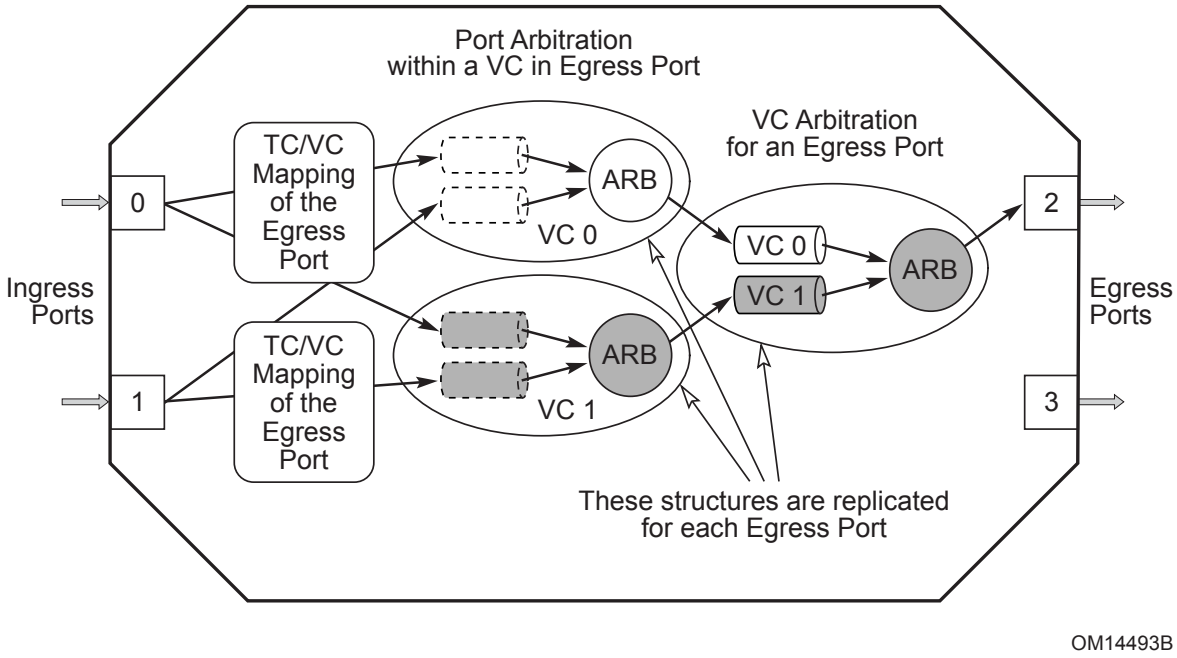

Figure 6-8 Switch Arbitration Structure

The following two steps conceptually describe routing of traffic received by the Switch on Port 0 and Port 1 and destined to Port 2. First, the target Egress Port is determined based on address/routing information in the TLP header. Secondly, the target VC of the Egress Port is determined based on the TC/VC map of the Egress Port. Transactions that target the same VC in the Egress Port but are from different Ingress Ports must be arbitrated before they can be forwarded to the corresponding resource in the Egress Port. This arbitration is referred to as the Port Arbitration.

Once the traffic reaches the destination VC resource in the Egress Port, it is subject to arbitration for the shared Link. From the Egress Port point of view this arbitration can be conceptually defined as a simple form of multiplexing where the multiplexing control is based on arbitration policies that are either fixed or configurable/programmable. This stage of arbitration between different VCs at an Egress Port is called the VC Arbitration of the Egress Port.

Independent of VC arbitration policy, a management/control logic associated with each VC must observe transaction ordering and flow control rules before it can make pending traffic visible to the arbitration mechanism.

IMPLEMENTATION NOTE

VC Control Logic at the Egress Port

VC control logic at every Egress Port includes:

- VC Flow Control logic
- VC Ordering Control logic

Flow control credits are exchanged between two Ports connected to the same Link. Availability of flow control credits is one of the qualifiers that VC control logic must use to decide when a VC is allowed to compete for the shared Link resource (i.e., Data Link Layer transmit/retry buffer). If a candidate packet cannot be submitted due to the lack of an adequate number of flow control credits, VC control logic must mask the presence of pending packet to prevent blockage of traffic from other VCs. Note that since each VC includes buffering resources for Posted Requests, Non-Posted Requests, and Completion packets, the VC control logic must also take into account availability of flow control credits for the particular candidate packet. In addition, VC control logic must observe ordering rules (see Section 2.4 for more details) for Posted/Non-Posted/Completion transactions to prevent deadlocks and violation of producer/consumer ordering model.

6.3.3.2 VC Arbitration - Arbitration Between VCs

This specification defines a default VC prioritization via the VC Identification (VC ID) assignment, i.e., the VC IDs are arranged in ascending order of relative priority in the Virtual Channel Capability structure or Multi-Function Virtual Channel Capability structure. The example in Figure 6-9 illustrates a Port that supports eight VCs with VC0 treated as the lowest priority and VC7 as the highest priority.

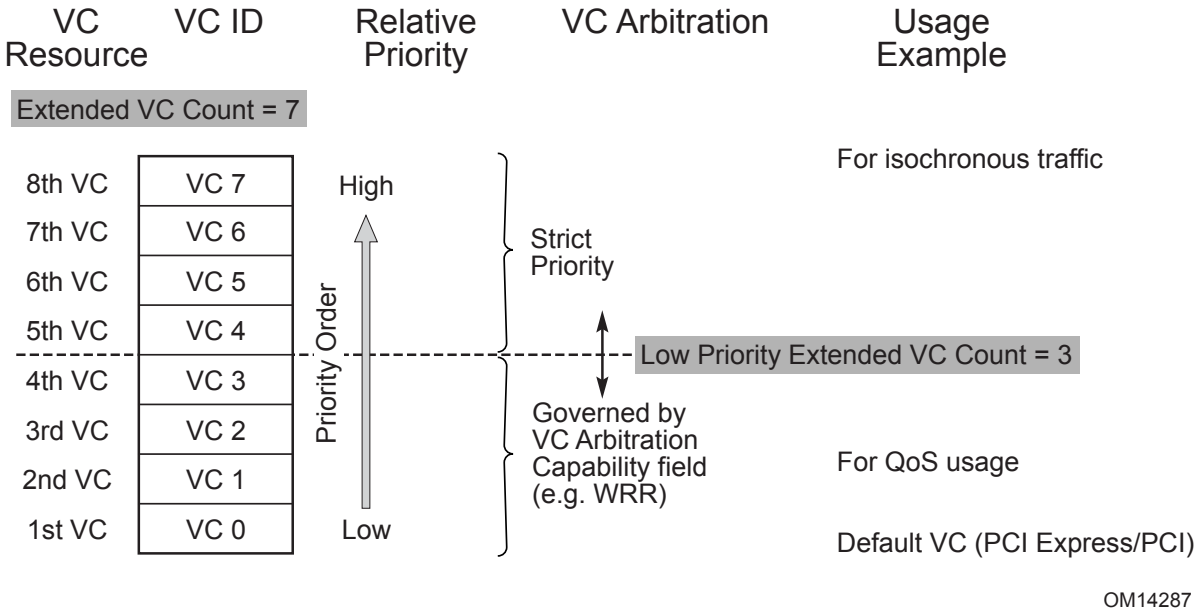

Figure 6-9 VC ID and Priority Order - An Example

The availability of default prioritization does not restrict the type of algorithms that may be implemented to support VC arbitration - either implementation-specific or one of the architecture-defined methods:

- Strict Priority - Based on inherent prioritization, i.e., VC0 = lowest, VC7 = highest
- Round Robin (RR) - Simplest form of arbitration where all VCs have equal priority
- Weighted RR - Programmable weight factor determines the level of service

If strict priority arbitration is supported by the hardware for a subset of the VC resources, software can configure the VCs into two priority groups - a lower and an upper group. The upper group is treated as a strict priority arbitration group while the lower group is arbitrated to only when there are no packets to process in the upper group. Figure 6-9 illustrates an example configuration that supports eight VCs separated into two groups - the lower group consisting of VC0-VC3 and the upper group consisting of VC4-VC7. The arbitration within the lower group can be configured to one of the supported arbitration methods. The Low Priority Extended VC Count field in the Port VC Capability Register 1 indicates the size of this group. The arbitration methods are listed in the VC Arbitration Capability field in the Port VC Capability Register 2. Refer to Section 7.9.1 and Section 7.9.2 for details. When the Low Priority Extended VC Count field is set to zero, all VCs are governed by the strict-priority VC arbitration; when the field is equal to the Extended VC Count, all VCs are governed by the VC arbitration indicated by the VC Arbitration Capability field.

6.3.3.2.1 Strict Priority Arbitration Model

Strict priority arbitration enables minimal latency for high-priority transactions. However, there is potential danger of bandwidth starvation should it not be applied correctly. Using strict priority requires all high-priority traffic to be regulated in terms of maximum peak bandwidth and Link usage duration. Regulation must be applied either at the transaction injection Port/Function or within subsequent Egress Ports where data flows contend for a common Link. System software must configure traffic such that lower priority transactions will be serviced at a sufficient rate to avoid transaction timeouts.

6.3.3.2.2 Round Robin Arbitration Model

Round Robin arbitration is used to provide, at the transaction level, equal[112] opportunities to all traffic. Note that this scheme is used where different unordered streams need to be serviced with the same priority.

In the case where differentiation is required, a Weighted Round Robin scheme can be used. The WRR scheme is commonly used in the case where bandwidth regulation is not enforced by the sources of traffic and therefore it is not possible to use the priority scheme without risking starvation of lower priority traffic. The key is that this scheme provides fairness during traffic contention by allowing at least one arbitration win per arbitration loop. Assigned weights regulate both minimum allowed bandwidth and maximum burstiness for each VC during the contention. This means that it bounds the arbitration latency for traffic from different VCs. Note that latencies are also dependent on the maximum packet sizes allowed for traffic that is mapped onto those VCs.

One of the key usage models of the WRR scheme is support for QoS policy where different QoS levels can be provided using different weights.

Although weights can be fixed (by hardware implementation) for certain applications, to provide more generic support for different applications, components that support the WRR scheme are recommended to implement programmable WRR. Programming of WRR is controlled using the software interface defined in Sections 7.9.1 and 7.9.2.

112. Note that this does not imply equivalence and fairness in the terms of bandwidth usage.

6.3.3.3 Port Arbitration - Arbitration Within VC

For Switches, Port Arbitration refers to the arbitration at an Egress Port between traffic coming from other Ingress Ports that is mapped to the same VC. For Root Ports, Port Arbitration refers to the arbitration at a Root Egress Port between peer-to-peer traffic coming from other Root Ingress Ports that is mapped to the same VC. For RCRBs, Port Arbitration refers to the arbitration at the RCRB (e.g., for host memory) between traffic coming from Root Ports that is mapped to the same VC. An inherent prioritization scheme for arbitration among VCs in this context is not applicable since it would imply strict arbitration priority for different Ports. Traffic from different Ports can be arbitrated using the following supported schemes:

- Hardware-fixed arbitration scheme, e.g., Round Robin
- Programmable WRR arbitration scheme
- Programmable Time-based WRR arbitration scheme

Hardware-fixed RR or RR-like scheme is the simplest to implement since it does not require any programmability. It makes all Ports equal priority, which is acceptable for applications where no software-managed differentiation or per-Port-based bandwidth budgeting is required.

Programmable WRR allows flexibility since it can operate as flat RR or if differentiation is required, different weights can be applied to traffic coming from different Ports in the similar manner as described in Section 6.3.3.2 . This scheme is used where different allocation of bandwidth needs to be provided for different Ports.

A Time-based WRR is used for applications where not only different allocation of bandwidth is required but also a tight control of usage of that bandwidth. This scheme allows control of the amount of traffic that can be injected from different Ports within a certain fixed period of time. This is required for certain applications such as isochronous services, where traffic needs to meet a strict deadline requirement. Section 6.3.4 provides basic rules to support isochronous applications. For more details on time-based arbitration and on the isochronous service as a usage model for this arbitration scheme refer to Appendix A.

6.3.3.4 Multi-Function Devices and Function Arbitration

The multi-Function arbitration model defines an optional arbitration infrastructure and functionality within a Multi-Function Device. This functionality is needed to support a set of arbitration policies that control traffic contention for the device's Upstream Egress Port from its multiple Functions.

Figure 6-10 shows a conceptual model of a Multi-Function Device highlighting resources and associated functionality. Note that each Function optionally contains a VC Capability structure, which if present manages TC/VC mapping, optional Port Arbitration, and optional VC Arbitration, all within the Function. The MFVC Capability structure manages TC/VC mapping, optional Function Arbitration, and optional VC Arbitration for the device's Upstream Egress Port. Together these resources enable enhanced QoS management for Upstream requests. However, unlike a complete Switch with devices on its Downstream Ports, the Multi-Function Device model does not support full QoS management for peer-to-peer requests between Functions or for Downstream requests.

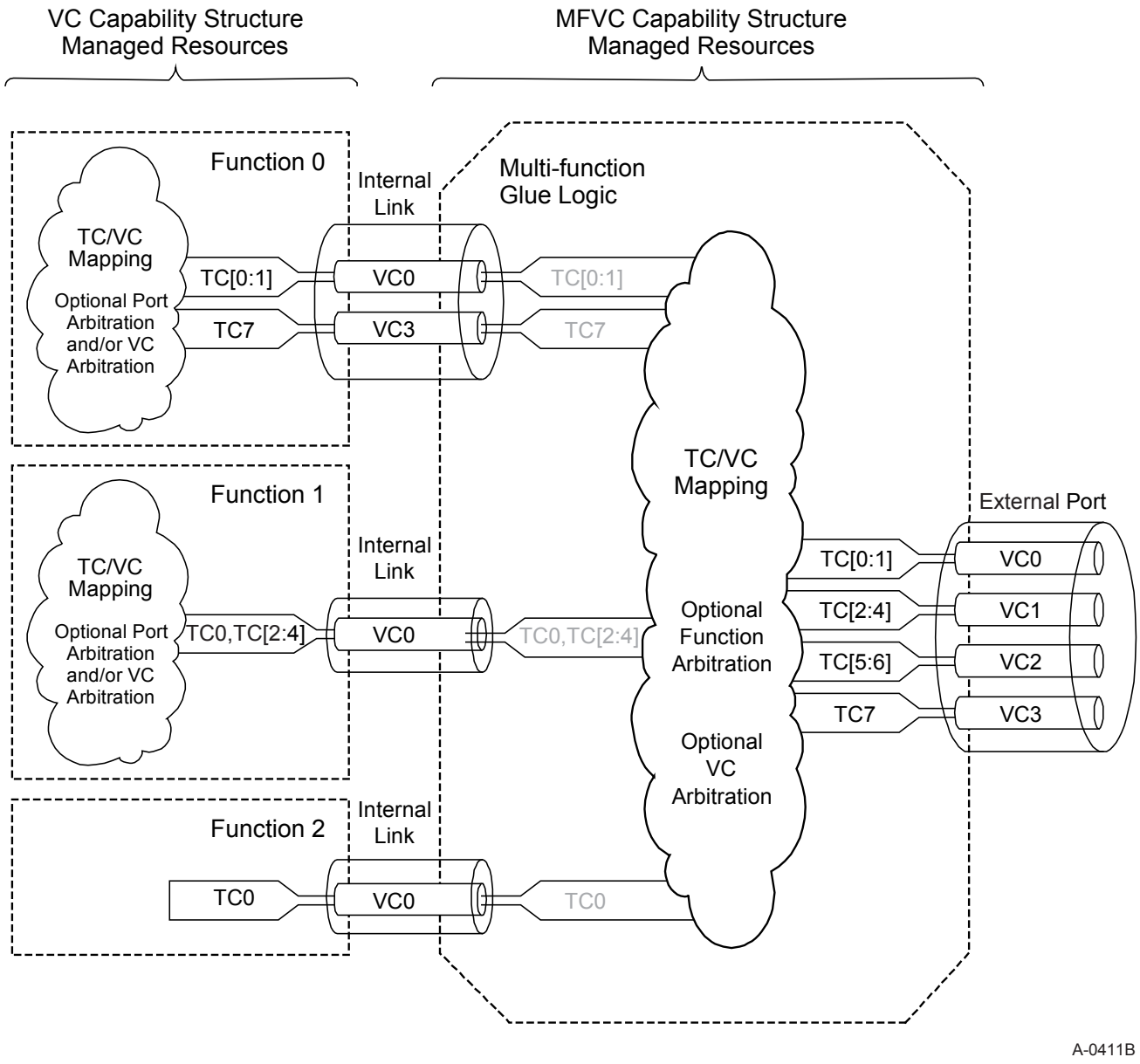

Figure 6-10 Multi-Function Arbitration Model

QoS for an Upstream request originating at a Function is managed as follows. First, a Function-specific mechanism applies a TC to the request. For example, a device driver might configure a Function to tag all its requests with TC7.

Next, if the Function contains a VC Capability structure, it specifies the TC/VC mapping to one of the Function's VC resources (perhaps the Function's single VC resource). In addition, the VC Capability structure supports the enablement and configuration of the Function's VC resources.

If the Function is a Switch and the target VC resource supports Port Arbitration, this mechanism governs how the Switch's multiple Downstream Ingress Ports arbitrate for that VC resource. If the Port Arbitration mechanism supports time-based WRR, this also governs the injection rate of requests from each Downstream Ingress Port.

If the Function supports VC arbitration, this mechanism manages how the Function's multiple VC resources arbitrate for the conceptual internal link to the MFVC resources.

Once a request packet conceptually arrives at MFVC resources, address/routing information in the TLP header determines whether the request goes Upstream or peer-to-peer to another Function. For the case of peer-to-peer, QoS management is left to unarchitected device-specific mechanisms. For the case of Upstream, TC/VC mapping in the MFVC Capability structure determines which VC resource the request will target. The MFVC Capability structure also supports enablement and configuration of the VC resources in the multi-Function glue logic. If the target VC resource supports Function Arbitration, this mechanism governs how the multiple Functions arbitrate for this VC resource. If the Function Arbitration mechanism supports time-based WRR, this governs the injection rate of requests for each Function into this VC resource.

Finally, if the MFVC Capability structure supports VC Arbitration, this mechanism governs how the MFVC's multiple VCs compete for the device's Upstream Egress Port. Independent of VC arbitration policy, management/control logic associated with each VC must observe transaction ordering and flow control rules before it can make pending traffic visible to the arbitration mechanism.

IMPLEMENTATION NOTE

Multi-Function Arbitration Error Behavior

Table 6-6 shows the expected error behavior associated with the example topology shown in Figure 6-10 .

Table 6-6 Multi-Function Arbitration Error Model Example

Source	TC	Destination			
		Function 0	Function 1	Function 2	External Port
Function 0	0	n/a	OK	OK	OK
	1		MF @ F1	MF @ F2	OK
	2 - 6		MF @ F0	MF @ F0	MF @ F0
	7		MF @ F1	MF @ F2	OK
Function 1	0	OK	n/a	OK	OK
	1	MF @ F1		MF @ F1	MF @ F1
	2 - 4	MF @ F0		MF @ F2	OK
	5 - 7	MF @ F1		MF @ F1	MF @ F1
Function 2	0	OK	OK	n/a	OK
	1 - 7	MF @ F2	MF @ F2		MF @ F2
External Port	0	OK	OK	OK	n/a
	1	OK	MF @ F1	MF @ F2	
	2 - 4	MF @ F0	OK		
	5 - 6		MF @ F1		
	7	OK			

Legend:

OK	Success
MF @ F0	Malformed TLP, reported at Function 0
MF @ F1	Malformed TLP, reported at Function 1
MF @ F2	Malformed TLP, reported at Function 2
n/a	Not Applicable (Function/Port sending to itself)

IMPLEMENTATION NOTE

Multi-Function Devices without the MFVC Capability Structure

If a Multi-Function Device lacks an MFVC Capability structure, the arbitration of data flows from different Functions of a Multi-Function Device is beyond the scope of this specification However, if a Multi-Function Device supports TCs other than TC0 and does not implement an MFVC Capability structure, it must implement a single VC Capability structure in Function 0 to provide architected TC/VC mappings for the Link.

6.3.4 Isochronous Support

Servicing isochronous data transfer requires a system to provide not only guaranteed data bandwidth but also deterministic service latency. The isochronous support mechanisms are defined to ensure that isochronous traffic receives its allocated bandwidth over a relevant period of time while also preventing starvation of the other traffic in the system. Isochronous support mechanisms apply to communication between Endpoint and Root Complex as well as to peer-to-peer communication.

Isochronous service is realized through proper use of mechanisms such as TC transaction labeling, VC data-transfer protocol, and TC-to-VC mapping. End-to-end isochronous service requires software to set up proper configuration along the path between the Requester and the Completer. This section describes the rules for software configuration and the rules hardware components must follow to provide end-to-end isochronous services. More information and background material regarding isochronous applications and isochronous service design guidelines can be found in Appendix A.

6.3.4.1 Rules for Software Configuration

System software must obey the following rules to configure PCI Express fabric for isochronous traffic:

- Software must designate one or more TCs for isochronous transactions.
- Software must ensure that the Attribute fields of all isochronous requests targeting the same Completer are fixed and identical.
- Software must configure all VC resources used to support isochronous traffic to be serviced (arbitrated) at the requisite bandwidth and latency to meet the application objectives. This may be accomplished using strict priority, WRR, or hardware-fixed arbitration.
- Software should not intermix isochronous traffic with non-isochronous traffic on a given VC.
- Software must observe the Maximum Time Slots capability reported by the Port or RCRB.
- Software must not assign all Link capacity to isochronous traffic. This is required to ensure the requisite forward progress of other non-isochronous transactions to avoid false transaction timeouts.
- Software must limit the Max_Payload_Size for each path that supports isochronous to meet the isochronous latency. For example, all traffic flowing on a path from an isochronous capable device to the Root Complex should be limited to packets that do not exceed the Max_Payload_Size required to meet the isochronous latency requirements.
- Software must set Max_Read_Request_Size of an isochronous-configured device with a value that does not exceed the Max_Payload_Size set for the device.

6.3.4.2 Rules for Requesters

A Requester requiring isochronous services must obey the following rules:

- The value in the Length field of read requests must never exceed Max_Payload_Size.
- If isochronous traffic targets the Root Complex and the RCRB indicates it cannot meet the isochronous bandwidth and latency requirements without requiring all transactions to set the No Snoop attribute bit, indicated by setting the Reject Snoop Transactions bit, then this bit must be set within the TLP header else the transaction will be rejected.

6.3.4.3 Rules for Completers

A Completer providing isochronous services must obey the following rules:

- A Completer should not apply flow control induced backpressure to uniformly injected isochronous requests under normal operating conditions.
- A Completer must report its isochronous bandwidth capability in the Maximum Time Slots field in the VC Resource Capability register. Note that a Completer must account for partial writes.
- A Completer must observe the maximum isochronous transaction latency.
- A Root Complex as a Completer must implement at least one RCRB and support time-based Port Arbitration for the associated VCs. Note that time-based Port Arbitration only applies to request transactions.

6.3.4.4 Rules for Switches and Root Complexes

A Switch providing isochronous services must obey the following rules. The same rules apply to Root Complexes that support isochronous data flows peer-to-peer between Root Ports, abbreviated in this section as “P2P-RC”.

- An isochronous-configured Switch or P2P-RC Port should not apply flow control induced backpressure to uniformly injected isochronous requests under normal operating conditions.
- An isochronous-configured Switch or P2P-RC Port must observe the maximum isochronous transaction latency.
- A Switch or P2P-RC component must support time-based Port Arbitration for each Port that supports one or more VCs capable of supporting isochronous traffic. Note that time-based Port Arbitration applies to request transactions but not to completion transactions.

6.3.4.5 Rules for Multi-Function Devices

A Multi-Function Device that includes an MFVC Capability structure providing isochronous services must obey the following rules:

- MFVC glue logic configured for isochronous operation should not apply backpressure to uniformly injected isochronous requests from its Functions under normal operating conditions.
- The MFVC Capability structure must support time-based Function Arbitration for each VC capable of supporting isochronous traffic. Note that time-based Function Arbitration applies only to Upstream request

transactions; it does not apply to any Downstream or peer-to-peer request transactions, nor to any completion transactions.

A Multi-Function Device that lacks an MFVC Capability structure has no architected mechanism to provide isochronous services for its multiple Functions concurrently.

6.4 Device Synchronization

System software requires a “stop” mechanism for ensuring that there are no outstanding transactions for a particular device in a system. For example, without such a mechanism renumbering Bus Numbers during system operation may cause the Requester ID (which includes the Bus Number) for a given device to change while Requests or Completions for that device are still in flight, and may thus be rendered invalid due to the change in the Requester ID. It is also desirable to be able to ensure that there are no outstanding transactions during a Hot-Plug orderly removal.

The details of stop mechanism implementation depend on the device hardware, device driver software, and system software. However, the fundamental requirements which must be supported to allow system software management of the fabric include the abilities to:

- Block the device from generating new Requests
- Block the generation of Requests issued to the device
- Determine that all Requests being serviced by the device have been completed
- Determine that all non-posted Requests initiated by the device have completed
- Determine that all posted Requests initiated by the device have reached their destination

The ability of the driver and/or system software to block new Requests from the device is supported by the Bus Master Enable, SERR# Enable, and Interrupt Disable bits in the Command register (Section7.5.1.1.3) of each device Function, and other such control bits.

Requests issued to the device are generally under the direct control of the driver, so system software can block these Requests by directing the driver to stop generating them (the details of this communication are system software specific). Similarly, Requests serviced by the device are normally under the device driver’s control, so determining the completion of such requests is usually trivial.

The Transactions Pending bit provides a consistent way on a per-Function basis for software to determine that all non-posted Requests issued by the device have been completed (see Section 7.5.3.5).

Determining that posted Requests have reached their destination is handled by generating a transaction to “flush” any outstanding Requests. Writes to system memory using TC0 will be flushed by host reads of the device, and so require no explicit flush protocol. Writes using TCs other than TC0 require some type of flush synchronization mechanism. The mechanism itself is implementation specific to the device and its driver software. However, in all cases the device hardware and software implementers should thoroughly understand the ordering rules described in Section 2.4 . This is especially true if the Relaxed Ordering or ID-Based Ordering attributes are set for any Requests initiated by the device.

IMPLEMENTATION NOTE

Flush Mechanisms

In a simple case such as that of an Endpoint communicating only with host memory through TC0, “flush” can be implemented simply by reading from the Endpoint. If the Endpoint issues writes to main memory using TCs other than TC0, “flush” can be implemented with a memory read on the corresponding TCs directed to main memory. The memory read needs to be performed on all TCs that the Endpoint is using.

If a memory read is used to “flush” outstanding transactions, but no actual read is required, it may be desirable to use the zero-length read semantic described in Section 2.2.5 .

Peer-to-peer interaction between devices requires an explicit synchronization protocol between the involved devices, even if all communication is through TC0. For a given system, the model for managing peer-to-peer interaction must be established. System software, and device hardware and software must then conform to this model. The requirements for blocking Request generation and determining completion of Requests match the requirements for non-peer interaction, however the determination that Posted Requests have reached peer destination device(s) requires an explicit synchronization mechanism. The mechanism itself is implementation specific to the device, its driver software, and the model used for the establishment and disestablishment of peer communications.

6.5 Locked Transactions

6.5.1 Introduction

Locked Transaction support is required to prevent deadlock in systems that use legacy software which causes the accesses to I/O devices. Note that some CPUs may generate locked accesses as a result of executing instructions that implicitly trigger lock. Some legacy software misuses these transactions and generates locked sequences even when exclusive access is not required. Since locked accesses to I/O devices introduce potential deadlocks apart from those mentioned above, as well as serious performance degradation, PCI Express Endpoints are prohibited from supporting locked accesses, and new software must not use instructions which will cause locked accesses to I/O devices. Legacy Endpoints support locked accesses only for compatibility with existing software.

Only the Root Complex is allowed to initiate Locked Requests on PCI Express. Locked Requests initiated by Endpoints and Bridges are not supported. This is consistent with limitations for locked transaction use outlined in [PCI] (Appendix F - Exclusive Accesses).

This section specifies the rules associated with supporting locked accesses from the Host CPU to Legacy Endpoints, including the propagation of those transactions through Switches and PCI Express/PCI Bridges.

6.5.2 Initiation and Propagation of Locked Transactions - Rules

Locked transaction sequences are generated by the Host CPU(s) as one or more reads followed by a number of writes to the same location(s). When a lock is established, all other traffic is blocked from using the path between the Root Complex and the locked Legacy Endpoint or Bridge.

- A locked transaction sequence or attempted locked transaction sequence is initiated on PCI Express using the “lock”-type Read Request/Completion (MRdLk/CplDLk) and terminated with the Unlock Message

    - Locked Requests which are completed with a status other than Successful Completion do not establish lock (explained in detail in the following sections)
    - Regardless of the status of any of the Completions associated with a locked sequence, all locked sequences and attempted locked sequences must be terminated by the transmission of an Unlock Message.
    - MRdLk, CplDLk, and Unlock semantics are allowed only for the default Traffic Class (TC0)
    - Only one locked transaction sequence attempt may be in progress at a given time within a single hierarchy domain
- The Unlock Message is sent from the Root Complex down the locked transaction path to the Completer, and may be broadcast from the Root Complex to all Endpoints and Bridges
    - Any device which is not involved in the locked sequence must ignore this Message
- Any violation of the rules for initiation and propagation of locked transactions can result in undefined device and/or system behavior
    - The initiation and propagation of a locked transaction sequence through PCI Express is performed as follows:
- A locked transaction sequence is started with a MRdLk Request
    - Any successive reads for the locked transaction sequence must also use MRdLk Requests
    - The Completions for any MRdLk Request use the CplDLk Completion type for successful Requests, and the CplLk Completion type for unsuccessful Requests
- If any read associated with a locked sequence is completed unsuccessfully, the Requester must assume that the atomicity of the lock is no longer assured, and that the path between the Requester and Completer is no longer locked
- All writes for the locked sequence use MWr Requests
- The Unlock Message is used to indicate the end of a locked sequence
    - A Switch propagates Unlock Messages to the locked Egress Port
- Upon receiving an Unlock Message, a Legacy Endpoint or Bridge must unlock itself if it is in a locked state
    - If not locked, or if the Receiver is a PCI Express Endpoint or Bridge which does not support lock, the Unlock Message is ignored and discarded

6.5.3 Switches and Lock - Rules

Switches must distinguish transactions associated with locked sequences from other transactions to prevent other transactions from interfering with the lock and potentially causing deadlock. The following rules cover how this is done. Note that locked accesses are limited to TC0, which is always mapped to VC0.

- When a Switch propagates a MRdLk Request from the Ingress Port (closest to the Root Complex) to the Egress Port, it must block all Requests which map to the default Virtual Channel (VC0) from being propagated to the Egress Port
    - If a subsequent MRdLk Request is Received at this Ingress Port addressing a different Egress Port, the behavior of the Switch is undefined
      Note: This sort of split-lock access is not supported by PCI Express and software must not cause such a locked access. System deadlock may result from such accesses.
- When the CplDLk for the first MRdLk Request is returned, if the Completion indicates a Successful Completion status, the Switch must block all Requests from all other Ports from being propagated to either of the Ports involved in the locked access, except for Requests which map to non-VC0 on the Egress Port

- The two Ports involved in the locked sequence must remain blocked as described above until the Switch receives the Unlock Message (at the Ingress Port for the initial MRdLk Request)
  - The Unlock Message must be forwarded to the locked Egress Port
  - The Unlock Message may be broadcast to all other Ports
  - The Ingress Port is unblocked once the Unlock Message arrives, and the Egress Port(s) which were blocked are unblocked following the Transmission of the Unlock Message out of the Egress Ports
    - Ports which were not involved in the locked access are unaffected by the Unlock Message

6.5.4 PCI Express/PCI Bridges and Lock - Rules

The requirements for PCI Express/PCI Bridges are similar to those for Switches, except that, because PCI Express/PCI Bridges use only the default Virtual Channel and Traffic Class, all other traffic is blocked during the locked access. The requirements on the PCI bus side of the PCI Express/PCI Bridge match the requirements for a PCI/PCI Bridge (see [PCI-to-PCI-Bridge-1.2] and [PCIe-to-PCI-PCI-X-Bridge-1.0]).

6.5.5 Root Complex and Lock - Rules

A Root Complex is permitted to support locked transactions as a Requester. If locked transactions are supported, a Root Complex must follow the sequence described in Section 6.5.2 to perform a locked access. The mechanisms used by the Root Complex to interface PCI Express to the Host CPU(s) are outside the scope of this document.

6.5.6 Legacy Endpoints

Legacy Endpoints are permitted to support locked accesses, although their use is discouraged. If locked accesses are supported, Legacy Endpoints must handle them as follows:

- The Legacy Endpoint becomes locked when it Transmits the first Completion for the first Read Request of the locked access with a Successful Completion status
  - If the completion status is not Successful Completion, the Legacy Endpoint does not become locked
  - Once locked, the Legacy Endpoint must remain locked until it receives the Unlock Message
- While locked, a Legacy Endpoint must not issue any Requests using TCs which map to the default Virtual Channel (VC0)
  Note that this requirement applies to all possible sources of Requests within the Endpoint, in the case where there is more than one possible source of Requests.
  - Requests may be issued using TCs which map to VCs other than the default Virtual Channel

6.5.7 PCI Express Endpoints

PCI Express Endpoints do not support lock. A PCI Express Endpoint must treat a MRdLk Request as an Unsupported Request (see Chapter 2).

6.6 PCI Express Reset - Rules

This section specifies the PCI Express Reset mechanisms. This section covers the relationship between the architectural mechanisms defined in this document and the reset mechanisms defined in this document. Any relationship between the PCI Express Conventional Reset and component or platform reset is component or platform specific (except as explicitly noted).

6.6.1 Conventional Reset

Conventional Reset includes all reset mechanisms other than Function Level Reset. There are two categories of Conventional Resets: Fundamental Reset and resets that are not Fundamental Reset. This section applies to all types of Conventional Reset.

In all form factors and system hardware configurations, there must, at some level, be a hardware mechanism for setting or returning all Port states to the initial conditions specified in this document - this mechanism is called “Fundamental Reset”. This mechanism can take the form of an auxiliary signal provided by the system to a component or adapter card, in which case the signal must be called PERST#, and must conform to the rules specified in Section 4.2.4.9.1 . When PERST# is provided to a component or adapter, this signal must be used by the component or adapter as Fundamental Reset. When PERST# is not provided to a component or adapter, Fundamental Reset is generated autonomously by the component or adapter, and the details of how this is done are outside the scope of this document. If a Fundamental Reset is generated autonomously by the component or adapter, and if power is supplied by the platform to the component/adapter, the component/adapter must generate a Fundamental Reset to itself if the supplied power goes outside of the limits specified for the form factor or system.

- There are three distinct types of Conventional Reset: cold, warm, and hot:
  - A Fundamental Reset must occur following the application of power to the component. This is called a cold reset.
  - In some cases, it may be possible for the Fundamental Reset mechanism to be triggered by hardware without the removal and re-application of power to the component. This is called a warm reset. This document does not specify a means for generating a warm reset.
  - There is an in-band mechanism for propagating Conventional Reset across a Link. This is called a hot reset and is described in Section 4.2.4.9.2 .

    There is an in-band mechanism for software to force a Link into Electrical Idle, “disabling” the Link. The Disabled LTSSM state is described in Section 4.2.5.9 , the Link Disable control bit is described in Section 7.5.3.7 , and the Downstream Port Containment mechanism is described in Section 6.2.10 . Disabling a Link causes Downstream components to undergo a hot reset.

    Note also that the Data Link Layer reporting DL_Down status is in some ways identical to a hot reset - see Section 2.9 .
- On exit from any type of Conventional Reset (cold, warm, or hot), all Port registers and state machines must be set to their initialization values as specified in this document, except for sticky registers (see Section 7.4).
  - Note that, from a device point of view, any type of Conventional Reset (cold, warm, hot, or DL_Down) has the same effect at the Transaction Layer and above as would RST# assertion and deassertion in conventional PCI.
- On exit from a Fundamental Reset, the Physical Layer will attempt to bring up the Link (see Section 4.2.5). Once both components on a Link have entered the initial Link Training state, they will proceed through Link initialization for the Physical Layer and then through Flow Control initialization for VC0, making the Data Link and Transaction Layers ready to use the Link.

- Following Flow Control initialization for VC0, it is possible for TLPs and DLLPs to be transferred across the Link.

Following a Conventional Reset, some devices may require additional time before they are able to respond to Requests they receive. Particularly for Configuration Requests it is necessary that components and devices behave in a deterministic way, which the following rules address.

The first set of rules addresses requirements for components and devices:

- A component must enter the LTSSM Detect state within 20 ms of the end of Fundamental Reset (Link Training is described in Section 4.2.4).
  - Note: In some systems, it is possible that the two components on a Link may exit Fundamental Reset at different times. Each component must observe the requirement to enter the initial active Link Training state within 20 ms of the end of Fundamental Reset from its own point of view.
- On the completion of Link Training (entering the DL_Active state, see Section 3.2), a component must be able to receive and process TLPs and DLLPs.

The second set of rules addresses requirements placed on the system:

- To allow components to perform internal initialization, system software must wait a specified minimum period following the end of a Conventional Reset of one or more devices before it is permitted to issue Configuration Requests to those devices, unless Readiness Notifications mechanisms are used (see Section 6.23 .).
  - With a Downstream Port that does not support Link speeds greater than 5.0 GT/s, software must wait a minimum of 100 ms before sending a Configuration Request to the device immediately below that Port.
  - With a Downstream Port that supports Link speeds greater than 5.0 GT/s, software must wait a minimum of 100 ms after Link training completes before sending a Configuration Request to the device immediately below that Port. Software can determine when Link training completes by polling the Data Link Layer Link Active bit or by setting up an associated interrupt (see Section 6.7.3.3).
  - A system must guarantee that all components intended to be software visible at boot time are ready to receive Configuration Requests within the applicable minimum period based on the end of Conventional Reset at the Root Complex - how this is done is beyond the scope of this specification.
  - Note: Software should use 100 ms wait periods only if software enables CRS Software Visibility. Otherwise, Completion timeouts, platform timeouts, or lengthy processor instruction stalls may result. See the Configuration Request Retry Status Implementation Note in Section 2.3.1 .
- Following a Conventional Reset of a device, within 1.0 s the device must be able to receive a Configuration Request and return a Successful Completion if the Request is valid. This period is independent of how quickly Link training completes. If Readiness Notifications mechanisms are used (see Section 6.23), this period may be shorter.
- Unless Readiness Notifications mechanisms are used, the Root Complex and/or system software must allow at least 1.0 s after a Conventional Reset of a device, before determining that the device is broken if it fails to return a Successful Completion status for a valid Configuration Request. This period is independent of how quickly Link training completes.
  Note: This delay is analogous to the T_{rhfa} parameter specified for PCI/PCI-X, and is intended to allow an adequate amount of time for devices which require self initialization.
- When attempting a Configuration access to devices on a PCI or PCI-X bus segment behind a PCI Express/PCI(-X) Bridge, the timing parameter T_{rhfa} must be respected.

For this second set of rules, if system software does not have direct visibility into the state of Fundamental Reset (e.g., Hot-Plug; see Section 6.7), software must base these timing parameters on an event known to occur after the end of Fundamental Reset.

When a Link is in normal operation, the following rules apply:

- If, for whatever reason, a normally operating Link goes down, the Transaction and Data Link Layers will enter the DL_Inactive state (see Sections 2.9 and 3.2.1).
- For any Root or Switch Downstream Port, setting the Secondary Bus Reset bit of the Bridge Control register associated with the Port must cause a hot reset to be sent (see Section 4.2.4.9.2).
- For a Switch, the following must cause a hot reset to be sent on all Downstream Ports:
  - Setting the Secondary Bus Reset bit of the Bridge Control register associated with the Upstream Port
  - The Data Link Layer of the Upstream Port reporting DL_Down status. In Switches that support Link speeds greater than 5.0 GT/s, the Upstream Port must direct the LTSSM of each Downstream Port to the Hot Reset state, but not hold the LTSSMs in that state. This permits each Downstream Port to begin Link training immediately after its hot reset completes. This behavior is recommended for all Switches.
  - Receiving a hot reset on the Upstream Port

Certain aspects of Fundamental Reset are specified in this document and others are specific to a platform, form factor and/or implementation. Specific platforms, form factors or application spaces may require the additional specification of the timing and/or sequencing relationships between the components of the system for Fundamental Reset. For example, it might be required that all PCI Express components within a chassis observe the assertion and deassertion of Fundamental Reset at the same time (to within some tolerance). In a multi-chassis environment, it might be necessary to specify that the chassis containing the Root Complex be the last to exit Fundamental Reset.

In all cases where power and PERST# are supplied, the following parameters must be defined:

- T_{pvperl} - PERST# must remain active at least this long after power becomes valid
- T_{perst} - When asserted, PERST# must remain asserted at least this long
- T_{fail} - When power becomes invalid, PERST# must be asserted within this time

Additional parameters may be specified.

In all cases where a reference clock is supplied, the following parameter must be defined:

- $T_{perst\text{-}clk}$ - PERST# must remain active at least this long after any supplied reference clock is stable

Additional parameters may be specified.

6.6.2 Function Level Reset (FLR)

The FLR mechanism enables software to quiesce and reset Endpoint hardware with Function-level granularity. Three example usage models illustrate the benefits of this feature:

- In some systems, it is possible that the software entity that controls a Function will cease to operate normally. To prevent data corruption, it is necessary to stop all PCI Express and external I/O (not PCI Express) operations being performed by the Function. Other defined reset operations do not guarantee that external I/O operations will be stopped.

- In a partitioned environment where hardware is migrated from one partition to another, it is necessary to ensure that no residual "knowledge" of the prior partition be retained by hardware, for example, a user's secret information entrusted to the first partition but not to the second. Further, due to the wide range of Functions, it is necessary that this be done in a Function-independent way.
- When system software is taking down the software stack for a Function and then rebuilding that stack, it is sometimes necessary to return the state to an uninitialized state before rebuilding the Function's software stack.

Implementation of FLR is optional (not required), but is strongly recommended.

FLR applies on a per Function basis. Only the targeted Function is affected by the FLR operation. The Link state must not be affected by an FLR.

FLR modifies the Function state described by this specification as follows:

- Function registers and Function-specific state machines must be set to their initialization values as specified in this document, except for the following:
  - sticky-type registers (ROS, RWS, RW1CS)
  - registers defined as type HwInit
  - these other fields or registers:
    - Captured Slot Power Limit Value in the Device Capabilities Register
    - Captured Slot Power Limit Scale in the Device Capabilities Register
    - Max_Payload_Size in the Device Control Register
    - Active State Power Management (ASPM) Control in the Link Control Register
    - Read Completion Boundary (RCB) in the Link Control Register
    - Common Clock Configuration in the Link Control Register
    - Extended Synch in the Link Control Register
    - Enable Clock Power Management in the Link Control Register
    - Hardware Autonomous Width Disable in Link Control Register
    - Hardware Autonomous Speed Disable in the Link Control 2 Register
    - Link Equalization Request 8.0 GT/s in the Link Status 2 Register
    - Link Equalization Request 16.0 GT/s in the 16.0 GT/s Status Register
    - Enable Lower SKP OS Generation Vector in the Link Control 3 register
    - Lane Equalization Control Register in the Secondary PCI Express Extended Capability structure
    - 16.0 GT/s Lane Equalization Control Register in the Physical Layer 16.0 GT/s Extended Capability structure
    - All registers in the Virtual Channel Extended Capability structure
    - All registers in the Multi-Function Virtual Channel Extended Capability structure
    - All registers in the Data Link Feature Extended Capability structure
    - All registers in the Physical Layer 16.0 GT/s Extended Capability structure
    - All registers in the Physical Layer 32.0 GT/s Extended Capability structure
    - All registers in the Lane Margining at the Receiver Extended Capability structure

- It is strongly recommended that the following registers are also not reset to their initialization values:
  - ARI Control Register in the ARI Extended Capability Structure
  - All registers in the L1 PM Substates Extended Capability structure
  - All registers in the Latency Tolerance Reporting Capability structure
  - All registers in the Precision Time Management Capability structure

  Future revisions of this specification may change this recommendation to a requirement.

Note that the controls that enable the Function to initiate requests on PCI Express are cleared, including Bus Master Enable, MSI Enable, and the like, effectively causing the Function to become quiescent on the Link.

Note that Port state machines associated with Link functionality including those in the Physical and Data Link Layers are not reset by FLR, and VC0 remains initialized following an FLR.

- Any outstanding INTx interrupt asserted by the Function must be deasserted by sending the corresponding Deassert_INTx Message prior to starting the FLR.

Note that when the FLR is initiated to a Function of a Multi-Function Device, if another Function continues to assert a matching INTx, no Deassert_INTx Message will be transmitted.

After an FLR has been initiated by writing a 1b to the Initiate Function Level Reset bit, the Function must complete the FLR within 100 ms. If software initiates an FLR when the Transactions Pending bit is 1b, then software must not initialize the Function until allowing adequate time for any associated Completions to arrive, or to achieve reasonable certainty that any remaining Completions will never arrive. For this purpose, it is recommended that software allow as much time as provided by the pre-FLR value for Completion Timeout on the device. If Completion Timeouts were disabled on the Function when FLR was issued, then the delay is system dependent but must be no less than 100 ms. If Function Readiness Status (FRS - see Section 6.23.2) is implemented, then system software is permitted to issue Configuration Requests to the Function immediately following receipt of an FRS Message indicating Configuration-Ready, however, this does not necessarily indicate that outstanding Requests initiated by the Function have completed.

Note that upon receipt of an FLR, a device Function may either clear all transaction status including Transactions Pending or set the Completion Timeout to its default value so that all pending transactions will time out during FLR execution. Regardless, the Transactions Pending bit must be clear upon completion of the FLR.

Since FLR modifies Function state not described by this specification (in addition to state that is described by this specification), it is necessary to specify the behavior of FLR using a set of criteria that, when applied to the Function, show that the Function has satisfied the requirements of FLR. The following criteria must be applied using Function-specific knowledge to evaluate the Function's behavior in response to an FLR:

- The Function must not give the appearance of an initialized adapter with an active host on any external interfaces controlled by that Function. The steps needed to terminate activity on external interfaces are outside of the scope of this specification.
  - For example, a network adapter must not respond to queries that would require adapter initialization by the host system or interaction with an active host system, but is permitted to perform actions that it is designed to perform without requiring host initialization or interaction. If the network adapter includes multiple Functions that operate on the same external network interface, this rule affects only those aspects associated with the particular Function reset by FLR.
- The Function must not retain within itself software readable state that potentially includes secret information associated with any preceding use of the Function. Main host memory assigned to the Function must not be modified by the Function.
  - For example, a Function with internal memory readable directly or indirectly by host software must clear or randomize that memory.

- The Function must return to a state such that normal configuration of the Function's PCI Express interface will cause it to be useable by drivers normally associated with the Function

When an FLR is initiated, the targeted Function must behave as follows:

- The Function must return the Completion for the configuration write that initiated the FLR operation and then initiate the FLR.
- While an FLR is in progress:
  - If a Request arrives, the Request is permitted to be silently discarded (following update of flow control credits) without logging or signaling it as an error.
  - If a Completion arrives, the Completion is permitted to be handled as an Unexpected Completion or to be silently discarded (following update of flow control credits) without logging or signaling it as an error.
  - While a Function is required to complete the FLR operation within the time limit described above, the subsequent Function-specific initialization sequence may require additional time. If additional time is required, the Function must return a Configuration Request Retry Status (CRS) Completion Status when a Configuration Request is received after the time limit above. After the Function responds to a Configuration Request with a Completion status other than CRS, it is not permitted to return CRS until it is reset again.

IMPLEMENTATION NOTE

Avoiding Data Corruption From Stale Completions

An FLR causes a Function to lose track of any outstanding non-posted Requests. Any corresponding Completions that later arrive are referred to as being "stale". If software issues an FLR while there are outstanding Requests, and then re-enables the Function for operation without waiting for potential stale Completions, any stale Completions that arrive afterwards may cause data corruption by being mistaken by the Function as belonging to Requests issued since the FLR.

Software can avoid data corruption from stale Completions in a variety of ways. Here's a possible algorithm:

1. Software that's performing the FLR synchronizes with other software that might potentially access the Function directly, and ensures such accesses do not occur during this algorithm.
2. Software clears the entire Command register, disabling the Function from issuing any new Requests.
3. Software polls the Transactions Pending bit in the Device Status register either until it is clear or until it has been long enough that software is reasonably certain that Completions associated with any remaining outstanding Transactions will never arrive. On many platforms, the Transactions Pending bit will usually clear within a few milliseconds, so software might choose to poll during this initial period using a tight software loop. On rare cases when the Transactions Pending bit does not clear by this time, software will need to poll for a much longer platform-specific period (potentially seconds), so software might choose to conduct this polling using a timer-based interrupt polling mechanism.
4. Software initiates the FLR.
5. Software waits 100 ms.
6. Software reconfigures the Function and enables it for normal operation.

6.7 PCI Express Native Hot-Plug

The PCI Express architecture is designed to natively support both hot-add and hot-removal (“hot-plug”) of cables, add-in cards, and modules. PCI Express native hot-plug provides a “toolbox” of mechanisms that allow different user/operator models to be supported using a self-consistent infrastructure. These mechanisms may be used to implement orderly addition/removal that relies on coordination with the operating system (e.g., traditional PCI hot-plug), as well as async removal, which proceeds without lock-step synchronization with the operating system. This section defines the set of hot-plug mechanisms and specifies how the elements of hot-plug, such as indicators and push buttons, must behave if implemented in a system.

6.7.1 Elements of Hot-Plug

Table 6-7 lists the physical elements comprehended in this specification for support of hot-plug models. A form factor specification must define how these elements are used in that form factor. For a given form factor specification, it is possible that only some of the available hot-plug elements are required, or even that none of these elements are required. In all cases, the form factor specification must define all assumptions and limitations placed on the system or the user by the choice of elements included. Silicon component implementations that are intended to be used only with selected form factors are permitted to support only those elements that are required by the associated form factor(s).

Table 6-7 Elements of Hot-Plug

Element	Purpose
Indicators	Show the power and attention state of the slot
Manually-operated Retention Latch (MRL)	Holds adapter in place
MRL Sensor	Allows the Port and system software to detect the MRL being opened
Electromechanical Interlock	Prevents removal of adapter from slot
Attention Button	Allows user to request hot-plug operations
Software User Interface	Allows user to request hot-plug operations
Slot Numbering	Provides visual identification of slots
Power Controller	Software-controlled electronic component or components that control power to a slot or adapter and monitor that power for fault conditions
Out-of-band Presence Detect	Method of determining physical presence of an adapter in a slot that does not rely on the Physical Layer

6.7.1.1 Indicators

Two indicators are defined: the Power Indicator and the Attention Indicator. Each indicator is in one of three states: on, off, or blinking. Hot-plug system software has exclusive control of the indicator states by writing the command registers associated with the indicator (with one exception noted below). The indicator requirements must be included in all form factor specifications. For a given form factor, the indicators may be required or optional or not applicable at all.

The hot-plug capable Port controls blink frequency, duty cycle, and phase of the indicators. Blinking indicators must operate at a frequency of between 1 and 2 Hz, with a 50% (+/- 5%) duty cycle. Blinking indicators are not required to be synchronous or in-phase between Ports.

Indicators may be physically located on the chassis or on the adapter (see the associated form factor specification for Indicator location requirements). Regardless of the physical location, logical control of the indicators is by the Downstream Port of the Upstream component on the Link.

The Downstream Port must not change the state of an indicator unless commanded to do so by software, except for platforms capable of detecting stuck-on power faults (relevant only when a power controller is implemented). In the case of a stuck-on power fault, the platform is permitted to override the Downstream Port and force the Power Indicator to be on (as an indication that the adapter should not be removed). The handling by system software of stuck-on faults is optional and not described in this specification. Therefore, the platform vendor must ensure that this feature, if implemented, is addressed via other software, platform documentation, or by other means.

6.7.1.1.1 Attention Indicator

The Attention Indicator, which must be yellow or amber in color, indicates that an operational problem exists or that the hot-plug slot is being identified so that a human operator can locate it easily.

Table 6-8 Attention Indicator States

Indicator Appearance	Meaning
Off	Normal - Normal operation
On	Attention - Operational problem at this slot
Blinking	Locate - Slot is being identified at the user's request

Attention Indicator Off

The Attention Indicator in the Off state indicates that neither the adapter (if one is present) nor the hot-plug slot requires attention.

Attention Indicator On

The Attention Indicator in the On state indicates that an operational problem exists at the adapter or slot.

An operational problem is a condition that prevents continued operation of an adapter. The operating system or other system software determines whether a specific condition prevents continued operation of an adapter and whether lighting the Attention Indicator is appropriate. Examples of operational problems include problems related to external cabling, adapter, software drivers, and power faults. In general, the Attention Indicator in the On state indicates that an operation was attempted and failed or that an unexpected event occurred.

The Attention Indicator is not used to report problems detected while validating the request for a hot-plug operation. Validation is a term applied to any check that system software performs to assure that the requested operation is viable, permitted, and will not cause problems. Examples of validation failures include denial of permission to perform a hot-plug operation, insufficient power budget, and other conditions that may be detected before a hot-plug request is accepted.

Attention Indicator Blinking

A blinking Attention Indicator indicates that system software is identifying this slot for a human operator to find. This behavior is controlled by a user (for example, from a software user interface or management tool).

6.7.1.1.2 Power Indicator

The Power Indicator, which must be green in color, indicates the power state of the slot. Table 6-9 lists the Power Indicator states.

Table 6-9 Power Indicator States

Indicator Appearance	Meaning
Off	Power Off - Insertion or removal of the adapter is permitted.
On	Power On - Insertion or removal of the adapter is not permitted.
Blinking	Power Transition - Hot-plug operation is in progress and insertion or removal of the adapter is not permitted.

Power Indicator Off

The Power Indicator in the Off state indicates that insertion or removal of the adapter is permitted. Main power to the slot is off if required by the form factor. Note that, depending on the form factor, other power/signals may remain on, even when main power is off and the Power Indicator is off. In an example using the [CEM] form factor, if the platform provides Vaux to hot-plug slots and the MRL is closed, any signals switched by the MRL are connected to the slot even when the Power Indicator is off. Signals switched by the MRL are disconnected when the MRL is opened. System software must cause a slot's Power Indicator to be turned off when the slot is not powered and/or it is permissible to insert or remove an adapter. Refer to the appropriate form factor specification for details.

Power Indicator On

The Power Indicator in the On state indicates that the hot-plug operation is complete and that main power to the slot is On and that insertion or removal of the adapter is not permitted.

Power Indicator Blinking

A blinking Power Indicator indicates that the slot is powering up or powering down and that insertion or removal of the adapter is not permitted.

The blinking Power Indicator also provides visual feedback to the operator when the Attention Button is pressed or when hot-plug operation is initiated through the hot-plug software interface.

6.7.1.2 Manually-operated Retention Latch (MRL)

An MRL is a manually-operated retention mechanism that holds an adapter in the slot and prevents the user from removing the device. The MRL rigidly holds the adapter in the slot so that cables may be attached without the risk of creating intermittent contact. MRLs that hold down two or more adapters simultaneously are permitted in platforms that do not provide MRL Sensors.

6.7.1.3 MRL Sensor

The MRL Sensor is a switch, optical device, or other type of sensor that reports the position of a slot's MRL to the Downstream Port. The MRL Sensor reports closed when the MRL is fully closed and open at all other times (that is, if the MRL fully open or in an intermediate position).

If a power controller is implemented for the slot, the slot main power must be automatically removed from the slot when the MRL Sensor indicates that the MRL is open. If signals such as Vaux and SMBus are switched by the MRL, then these signals must be automatically removed from the slot when the MRL Sensor indicates that the MRL is open and must be

restored to the slot when the MRL Sensor indicates that MRL has closed again. Refer to the appropriate form factor specification to identify the signals, if any, switched by the MRL.

Note that the Hot-Plug Controller does not autonomously change the state of either the Power Indicator or the Attention Indicator based on MRL sensor changes.

IMPLEMENTATION NOTE

MRL Sensor Handling

In the absence of an MRL sensor, for some form factors, out-of-band presence detect may be used to handle the switched signals. In this case, when out-of-band presence detect indicates the absence of an adapter in a slot, the switched signals will be automatically removed from the slot.

If an MRL Sensor is implemented without a corresponding MRL Sensor input on the Hot-Plug Controller, it is recommended that the MRL Sensor be routed to power fault input of the Hot-Plug Controller. This allows an active adapter to be powered off when the MRL is opened.

6.7.1.4 Electromechanical Interlock

An electromechanical interlock is a mechanism for physically locking the adapter or MRL in place until system software releases it. The state of the electromechanical interlock is set by software and must not change except in response to a subsequent software command. In particular, the state of the electromechanical interlock must be maintained even when power to the hot-plug slot is removed.

The current state of the electromechanical interlock must be reflected at all times in the Electromechanical Interlock Status bit in the Slot Status register, which must be updated within 200 ms of any commanded change. Software must wait at least 1 second after issuing a command to toggle the state of the Electromechanical Interlock before another command to toggle the state can be issued. Systems may optionally expand control of interlocks to provide physical security of the adapter.

6.7.1.5 Attention Button

The Attention Button is a momentary-contact push button switch, located adjacent to each hot-plug slot or on the adapter that is pressed by the user to initiate a hot-plug operation at that slot. Regardless of the physical location of the button, the signal is processed and indicated to software by hot-plug hardware associated with the Downstream Port corresponding to the slot.

The Attention Button must allow the user to initiate both hot add and hot remove operations regardless of the physical location of the button.

If present, the Power Indicator provides visual feedback to the human operator (if the system software accepts the request initiated by the Attention Button) by blinking. Once the Power Indicator begins blinking, a 5-second abort interval exists during which a second depression of the Attention Button cancels the operation.

If an operation initiated by an Attention Button fails for any reason, it is recommended that system software present an error message explaining the failure via a software user interface or add the error message to a system log.

6.7.1.6 Software User Interface

System software provides a user interface that allows hot insertions and hot removals to be initiated and that allows occupied slots to be monitored. A detailed discussion of hot-plug user interfaces is operating system specific and is therefore beyond the scope of this document.

On systems with multiple hot-plug slots, the system software must allow the user to initiate operations at each slot independent of the states of all other slots. Therefore, the user is permitted to initiate a hot-plug operation on one slot using either the software user interface or the Attention Button while a hot-plug operation on another slot is in process, regardless of which interface was used to start the first operation.

6.7.1.7 Slot Numbering

A Physical Slot Identifier (as defined in [PCI-Hot-Plug-1.1], Section 1.5) consists of an optional chassis number and the physical slot number of the slot. The physical slot number is a chassis unique identifier for a slot. System software determines the physical slot number from registers in the Port. Chassis number 0 is reserved for the main chassis. The chassis number for other chassis must be a non-zero value obtained from a PCI-to-PCI Bridge's Chassis Number register (see [PCI-to-PCI-Bridge-1.2], Section 13.4).

Regardless of the form factor associated with each slot, each physical slot number must be unique within a chassis.

6.7.1.8 Power Controller

The power controller is an element composed of one or more discrete components that acts under control of software to set the power state of the hot-plug slot as appropriate for the specific form factor. The power controller must also monitor the slot for power fault conditions (as defined in the associated form factor specification) that occur on the slot's main power rails and, if supported, auxiliary power rail.

If a power controller is not present, the power state of the hot-plug slot must be set automatically by the hot-plug controller in response to changes in the presence of an adapter in the slot.

The power controller monitors main and auxiliary power faults independently. If a power controller detects a main power fault on the hot-plug slot, it must automatically set its internal main power fault latch and remove main power from the hot-plug slot (without affecting auxiliary power). Similarly, if a power controller detects an auxiliary power fault on the hot-plug slot, it must automatically set its internal auxiliary power fault latch and remove auxiliary power from the hot-plug slot (without affecting main power). Power must remain off to the slot as long as the power fault condition remains latched, regardless of any writes by software to turn on power to the hot-plug slot. The main power fault latch is cleared when software turns off power to the hot-plug slot. The mechanism by which the auxiliary power fault latch is cleared is form factor specific but generally requires auxiliary power to be removed from the hot-plug slot. For example, one form factor may remove auxiliary power when the MRL for the slot is opened while another may require the adapter to be physically removed from the slot. Refer to the associated form factor specifications for specific requirements.

Since the Power Controller Control bit in the Slot Control register reflects the last value written and not the actual state of the power controller, this means there may be an inconsistency between the value of the Power Controller Control bit and the state of the power to the slot in a power fault condition. To determine whether slot is off due to a power fault, software must use the power fault software notification to detect power faults. To determine that a requested power-up operation has otherwise failed, software must use the hot-plug slot power-up time out mechanism described in Section 6.7.3.3 .

Software must not assume that writing to the Slot Control register to change the power state of a hot-plug slot causes an immediate power state transition. After turning power on, software must wait for a Data Link Layer State Changed event, as described in Section 6.7.3.3 . After turning power off, software must wait for at least 1 second before taking any action

that relies on power having been removed from the hot-plug slot. For example, software is not permitted to turn off the power indicator (if present) or attempt to turn on the power controller before completing the 1 second wait period.

6.7.2 Registers Grouped by Hot-Plug Element Association

The registers described in this section are grouped by hot-plug element to convey all registers associated with implementing each element. Register fields associated with each Downstream Port implementing a hot-plug capable slot are located in the Device Capabilities, Slot Capabilities, Slot Control, Slot Status, and Slot Capabilities 2 registers in the PCI Express Capability structure (see Section 7.5.3). Registers reporting the presence of hot-plug elements associated with the device Function on an adapter are located in the Device Capabilities register (also in the PCI Express Capability structure).

6.7.2.1 Attention Button Registers

Attention Button Present (Slot Capabilities Register and Device Capabilities Register) - This bit indicates if an Attention Button is electrically controlled by the chassis (Slot Capabilities Register) or by the adapter (Device Capabilities Register).

Attention Button Pressed (Slot Status Register - This bit is set when an Attention Button electrically controlled by the chassis is pressed.

Attention Button Pressed Enable (Slot Control Register - When Set, this bit enables software notification on an Attention Button Pressed event (see Section 6.7.3.4).

6.7.2.2 Attention Indicator Registers

Attention Indicator Present (Slot Capabilities Register and Device Capabilities Register) - This bit indicates if an Attention Indicator is electrically controlled by the chassis (Slot Capabilities Register) or by the adapter (Device Capabilities REgister).

Attention Indicator Control (Slot Control Register) - When written, sets an Attention Indicator electrically controlled by the chassis to the written state.

6.7.2.3 Power Indicator Registers

Power Indicator Present (Slot Capabilities Register and Device Capabilities Register) - This bit indicates if a Power Indicator is electrically controlled by the chassis (Slot Capabilities Register) or by the adapter (Device Capabilities Register).

Power Indicator Control (Slot Control Register) - When written, sets a Power Indicator electrically controlled by the chassis to the written state.

6.7.2.4 Power Controller Registers

Power Controller Present (Slot Capabilities Register) - This bit indicates if a Power Controller is implemented.

Power Controller Control (Slot Control Register) - Turns the Power Controller on or off according to the value written.

Power Fault Detected (Slot Status Register) - This bit is set when a power fault is detected at the slot or the adapter.

Power Fault Detected Enable (Slot Control Register) - When Set, this bit enables software notification on a power fault event (see Section 6.7.3.4).

6.7.2.5 Presence Detect Registers

In-Band PD Disable Supported (Slot Capabilities 2 Register) - This bit indicates if the slot supports the disabling of in-band presence detect, which allows the out-of-band presence detect state to be reported independently of the in-band presence detect state.

In-Band PD Disable (Slot Control Register) - When Set, this bit disables the in-band presence detect mechanism from affecting the Presence Detect State bit, allowing that bit to be dedicated to reporting out-of-band presence detect.

Presence Detect State (Slot Status Register) - This bit indicates the presence of an adapter in the slot.

Presence Detect Changed (Slot Status Register) - This bit is set when a presence detect state change is detected.

Presence Detect Changed Enable (Slot Control Register) - When Set, this bit enables software notification on a presence detect changed event (see Section 6.7.3.4).

6.7.2.6 MRL Sensor Registers

MRL Sensor Present (Slot Capabilities Register) - This bit indicates if an MRL Sensor is implemented.

MRL Sensor Changed (Slot Status Register) - This bit is set when the value of the MRL Sensor state changes.

MRL Sensor Changed Enable (Slot Control Register) - When Set, this bit enables software notification on a MRL Sensor changed event (see Section 6.7.3.4).

MRL Sensor State (Slot Status Register) - This register reports the status of the MRL Sensor if one is implemented.

6.7.2.7 Electromechanical Interlock Registers

Electromechanical Interlock Present (Slot Capabilities Register) - This bit indicates if an Electromechanical Interlock is implemented.

Electromechanical Interlock Status (Slot Status Register) - This bit reflects the current state of the Electromechanical Interlock.

Electromechanical Interlock Control (Slot Control Register) - This bit when set to 1b toggles the state of the Electromechanical Interlock.

6.7.2.8 Command Completed Registers

No Command Completed Support (Slot Capabilities Register) - This bit when set to 1b indicates that this slot does not generate software notification when an issued command is completed by the Hot-Plug Controller.

Command Completed (Slot Status Register) - This bit is set when the Hot-Plug Controller completes an issued command and is ready to accept the next command.

Command Completed Interrupt Enable (Slot Control Register) - When Set, this bit enables software notification (see Section 6.7.3.4) when a command is completed by the hot-plug control logic.

6.7.2.9 Port Capabilities and Slot Information Registers

Slot Implemented (PCI Express Capabilities Register) - When Set, this bit indicates that the Link associated with this Downstream Port is connected to a slot.

Physical Slot Number (Slot Capabilities Register) - This hardware initialized field indicates the physical slot number attached to the Port.

Hot-Plug Capable (Slot Capabilities Register) - When Set, this bit indicates this slot is capable of supporting hot-plug.

Hot-Plug Surprise (Slot Capabilities Register) - When Set, this bit indicates that the Hot-Plug Surprise mechanism for handling async removal is enabled for this slot. See Section 6.7.6 .

6.7.2.10 Hot-Plug Interrupt Control Register

Hot-Plug Interrupt Enable (Slot Control Register) - When Set, this bit enables generation of the hot-plug interrupt on enabled hot-plug events.

6.7.3 PCI Express Hot-Plug Events

A Downstream Port with hot-plug capabilities supports the following hot-plug events:

- Slot Events:
  - Attention Button Pressed
  - Power Fault Detected
  - MRL Sensor Changed
  - Presence Detect Changed
- Command Completed Events
- Data Link Layer State Changed Events

Each of these events has a status field, which indicates that an event has occurred but has not yet been processed by software, and an enable field, which indicates whether the event is enabled for software notification. Some events also have a capability field, which indicates whether the event type is supported on the Port. The grouping of these fields by event type is listed in Section 6.7.2 , and each individual field is described in Section 7.5.3 .

6.7.3.1 Slot Events

A Downstream Port with hot-plug capabilities monitors the slot it controls for the slot events listed above. When one of these slot events is detected, the Port indicates that the event has occurred by setting the status field associated with the event. At that point, the event is pending until software clears the status field.

Once a slot event is pending on a particular slot, all subsequent events of that type are ignored on that slot until the event is cleared. The Port must continue to monitor the slot for all other slot event types and report them as they occur.

If enabled through the associated enable field, slot events must generate a software notification. If the event is not supported on the Port as indicated by the associated capability field, software must not enable software notification for the event. The mechanism by which this notification is reported to software is described in Section 6.7.3.4 .

6.7.3.2 Command Completed Events

Since changing the state of some hot-plug elements may not happen instantaneously, PCI Express supports hot-plug commands and command completed events. All hot-plug capable Ports are required to support hot-plug commands and, if the capability is reported, command completed events.

Software issues a command to a hot-plug capable Downstream Port by issuing a write transaction that targets any portion of the Port's Slot Control register. A single write to the Slot Control register is considered to be a single command, even if the write affects more than one field in the Slot Control register. In response to this transaction, the Port must carry out the requested actions and then set the associated status field for the command completed event. The Port must process the command normally even if the status field is already set when the command is issued. If a single command results in more than one action being initiated, the order in which the actions are executed is unspecified. All actions associated with a single command execution must not take longer than 1 second.

If command completed events are not supported as indicated by a value of 1b in the No Command Completed Support field of the Slot Capabilities register, a hot-plug capable Port must process a write transaction that targets any portion of the Port's Slot Control register without any dependency on previous Slot Control writes. Software is permitted to issue multiple Slot Control writes in sequence without any delay between the writes.

If command completed events are supported, then software must wait for a command to complete before issuing the next command. However, if the status field is not set after the 1 second limit on command execution, software is permitted to repeat the command or to issue the next command. If software issues a write before the Port has completed processing of the previous command and before the 1 second time limit has expired, the Port is permitted to either accept or discard the write. Such a write is considered a programming error, and could result in a discrepancy between the Slot Control register and the hot plug element state. To recover from such a programming error and return the controller to a consistent state, software must issue a write to the Slot Control register which conforms to the command completion rules.

If enabled through the associated enable field, the completion of a commands must generate a software notification. The exception to this rule is a command that occurs as a result of a write to the Slot Control register that disables software notification of command completed events. Such a command must be processed as described above, but must not generate a software notification.

6.7.3.3 Data Link Layer State Changed Events

The Data Link Layer State Changed event provides an indication that the state of the Data Link Layer Link Active bit in the Link Status Register has changed. Support for Data Link Layer State Changed events and software notification of these events are required for hot-plug capable Downstream Ports. If this event is supported, the Port sets the status field associated with the event when the value in the Data Link Layer Link Active bit changes.

This event allows software to indirectly determine when power has been applied to a newly hot-plugged adapter. Software must wait for 100 ms after the Data Link Layer Link Active bit reads 1b before initiating a configuration access to the hot added device (see Section 6.6). Software must allow 1 second after the Data Link Layer Link Active bit reads 1b before it is permitted to determine that a hot plugged device which fails to return a Successful Completion for a Valid Configuration Request is a broken device (see Section 6.6).

The Data Link Layer State Changed event must occur within 1 second of the event that initiates the hot-insertion. If a power controller is supported, the time out interval is measured from when software initiated a write to the Slot Control register to turn on the power. If a power controller is not supported, the time out interval is measured from presence detect slot event. Software is allowed to time out on a hot add operation if the Data Link Layer State Changed event does not occur within 1 second. The action taken by software after such a timeout is implementation specific.

6.7.3.4 Software Notification of Hot-Plug Events

A hot-plug capable Downstream Port must support generation of an interrupt on a hot-plug event. As described in Sections 6.7.3.1 and 6.7.3.2, each hot-plug event has both an enable bit for interrupt generation and a status bit that indicates when an event has occurred but has not yet been processed by software. There is also a Hot-Plug Interrupt Enable bit in the Slot Control register that serves as a master enable/disable bit for all hot-plug events.

If the Port is enabled for level-triggered interrupt signaling using the INTx messages, the virtual INTx wire must be asserted whenever and as long as the following conditions are satisfied:

- The Interrupt Disable bit in the Command register is set to 0b.
- The Hot-Plug Interrupt Enable bit in the Slot Control register is set to 1b.
- At least one hot-plug event status bit in the Slot Status register and its associated enable bit in the Slot Control register are both set to 1b.

Note that all other interrupt sources within the same Function will assert the same virtual INTx wire when requesting service.

If the Port is enabled for edge-triggered interrupt signaling using MSI or MSI-X, an interrupt message must be sent every time the logical AND of the following conditions transitions from FALSE to TRUE:

- The associated vector is unmasked (not applicable if MSI does not support PVM).
- The Hot-Plug Interrupt Enable bit in the Slot Control register is set to 1b.
- At least one hot-plug event status bit in the Slot Status register and its associated enable bit in the Slot Control register are both set to 1b.

Note that PME and Hot-Plug Event interrupts (when both are implemented) always share the same MSI or MSI-X vector, as indicated by the Interrupt Message Number field in the PCI Express Capabilities register.

The Port may optionally send an MSI when there are hot-plug events that occur while interrupt generation is disabled, and interrupt generation is subsequently enabled.

If wake generation is required by the associated form factor specification, a hot-plug capable Downstream Port must support generation of a wakeup event (using the PME mechanism) on hot-plug events that occur when the system is in a sleep state or the Port is in device state D1, D2, or $D3_{Hot}$.

Software enables a hot-plug event to generate a wakeup event by enabling software notification of the event as described in Section 6.7.3.1 . Note that in order for software to disable interrupt generation while keeping wakeup generation enabled, the Hot-Plug Interrupt Enable bit must be cleared. For form factors that support wake generation, a wakeup event must be generated if all three of the following conditions occur:

- The status register for an enabled event transitions from Clearto Set
- The Port is in device state D1, D2, or $D3_{Hot}$, and
- The PME_En bit in the Port's Power Management Control/Status register is Set

  Note that the Hot-Plug Controller generates the wakeup on behalf of the hot-plugged device, and it is not necessary for that device to have auxiliary (or main) power.

6.7.4 *System Firmware Intermediary (SFI)* Support

The System Firmware Intermediary (SFI) Capability is an optional normative feature of a Downstream Port. Some SFI functionality is focused on hot-pluggable slots, as indicated by the Hot-Plug Capable bit in the Slot Capabilities register being Set, while some SFI functionality is useful outside that context. If a Downstream Port supports an SFI Capability structure, the following bits must be Set:

- Data Link Layer Link Active Reporting Capable bit in the Link Capabilities register
- DRS Supported bit in the Link Capabilities 2 register
- ERR_COR Subclass Capable bit in the Device Capabilities register

6.7.4.1 SFI ERR_COR Event Signaling

The SFI Capability has no support for generating INTx or MSI/MSI-X interrupts, since the capability is intended for use by system firmware.

A Downstream Port with SFI must support ERR_COR signaling, regardless of whether it supports Advanced Error Reporting (AER) or not. SFI ERR_COR event signaling is enabled independently by the SFI OOB PD Changed Enable, SFI DLL State Changed Enable, and SFI DRS Signaling Enable bits in the SFI Control Register. These events are indicated by the SFI OOB PD Changed, SFI DLL State Changed, and SFI DRS Received bits in the SFI Status Register.

If the Correctable Error Reporting Enable bit in the Device Control Register is Set, the Port must send an ERR_COR Message each time one of the enabled conditions becomes satisfied. SFI ERR_COR event signaling must not Set the Correctable Error Detected bit in the Device Status Register, since this event is not handled as an error.

IMPLEMENTATION NOTE

ERR_COR Signaling for DPC DL_Active vs. SFI DLL State Changed

DPC implements ERR_COR signaling for DL_Active, whereas SFI implements ERR_COR signaling for SFI DLL State Changed, which are related but non-identical conditions. The DL_Active condition occurs when the Data Link Layer Link Active bit in the Link Status register changes from 0b to 1b, and this bit can be masked by the SFI DLL State Mask bit in the SFI Control register. The SFI DLL State Changed condition occurs when the SFI DLL State bit in the SFI Status Register changes its value either by becoming Set or becoming Clear, and this condition is always based on the actual Data Link Layer state.

6.7.4.2 SFI Downstream Port Filtering (DPF)

Downstream Port Filtering (DPF) is a mechanism where a Downstream Port can handle specified Request TLPs that target Components below it as if the Link is in DL_Down. See Section 2.9.1 .

DPF has two modes of filtering Request TLPs that target Components below the Downstream Port. The first mode filters all such Request TLPs; the second mode filters only Configuration Request TLPs. Other TLPs must not be filtered or blocked by DPF.

One key use case for DPF is guaranteeing that asynchronous system software activities like bus scans do not unintentionally send Configuration Requests to devices that are not yet ready following a Conventional Reset, since such accesses result in undefined hardware behavior. See Section 6.6.1 .

Another key use case for DPF is supporting firmware first functionality, enabling system firmware, when notified of an async hot add, to configure the newly added device before making the device visible to the operating system. For this use case, the SFI CAM mechanism enables the Downstream Port itself to generate Configuration Request TLPs targeting Downstream Components, and those TLPs are not filtered or blocked by the DPF mechanism. See Section 6.7.4.3 , Section 7.9.21.5 , and Section 7.9.21.6 .

6.7.4.3 SFI CAM

The SFI Configuration Access Method (CAM) provides a means for SFI-aware system firmware to have the Downstream Port proxy (pass through) Configuration Requests targeting Components below the Downstream Port when DPF is enabled. The SFI CAM is always enabled.

To use the SFI CAM, software first writes to the SFI CAM Address Register, specifying the target Configuration address. Software then reads or writes the SFI CAM Data Register to cause a proxied Configuration Request to be generated and transmitted to the Downstream Component.

The following rules apply:

- All TLP fields used for the proxied Configuration Request are identical to those in the Configuration Request that targeted the SFI CAM Data Register, with the following exceptions:
  - The target Bus Number, Device Number, and Function Number come from the SFI CAM Address Register.
  - The Extended Register Number and Register Number come from the SFI CAM Address Register.
  - The LCRC is regenerated.
  - If present, the ECRC is regenerated.
- The SFI CAM must not apply the Completion Timeout mechanism to the Request.
- System firmware must ensure that between the time it writes to the SFI CAM Address Register and its subsequent read or write of the SFI CAM Data Register completes, no other threads modify the SFI CAM Address Register; otherwise, the result is undefined.
- If there is a detected error associated with the proxied Configuration Request, this is a reported error associated with the Downstream Port implementing the SFI CAM (see Section 6.2).
- Completions flowing Upstream must be passed through the Downstream Port unmodified.

IMPLEMENTATION NOTE
Serialized Use of the SFI CAM Address and Data Registers

As described above, system firmware must ensure that between the time it writes to the SFI CAM Address Register and its subsequent read or write of the SFI CAM Data Register completes, no other threads modify the SFI CAM Address Register. For example, a semaphore or other synchronization mechanism can be used to ensure this serialization.

For platforms where a processor store instruction to Configuration Space is effectively posted, software must still ensure that the resulting Configuration Write completes before another software thread modifies the SFI CAM Data Register. On such platforms, the mechanism for determining when a Configuration Write completes is platform specific.

Given appropriate serialization, the SFI CAM works correctly with Configuration Requests that result in CRS Completions, even when the Root Complex automatically re-issues the Configuration Request as a new Request. The re-issued Configuration Request will again be sent to the SFI CAM Data Register, and the associated Downstream Port will again generate a Configuration Request targeting the Downstream Component. As long as the SFI CAM Address Register isn't modified by other software until the Configuration Request completes, the sequence can repeat indefinitely until a non-CRS Completion is returned or a Completion Timeout occurs.

When CRS Software Visibility is enabled, the SFI CAM still works correctly with Configuration Requests that result in CRS Completions. Any Completions with a CRS Completion Status flow back to the original Requester, which handles them as required by CRS Software Visibility semantics. See Section 2.3.2 .

IMPLEMENTATION NOTE
Use of Assigned Bus Numbers with the SFI CAM

When a Downstream Port has DPF enabled, the SFI CAM can be used by SFI-aware system firmware to configure and access the sub-hierarchy below the Port without other software being able to do so. While the Bus Number configuration below the Port is generally not visible to other software, Bus Numbers configured for use below the Port should be limited to those already assigned to the Port since TLPs coming Upstream through the Port may contain IDs with the configured Bus Numbers. If any errors are detected and logged with those TLPs, the Bus Numbers can become visible to other software, creating confusion if they overlap with Bus Numbers used elsewhere in the system.

6.7.4.4 SFI Interactions with Readiness Notifications

The SFI Capability is able to mask the reporting of received Device Readiness Status (DRS) Messages as well as emulate them being received. This functionality is useful when SFI's Downstream Port Filtering (DPF) mechanism is being used to block operating system visibility of a device or sub-hierarchy below the Downstream Port.

Rules:

- When the SFI DRS Mask bit is Set, the DRS Message Received bit in the Link Status 2 Register value must be 0b.
- The SFI DRS Received bit must always indicate the actual state of the DRS Message Received condition.
- When the SFI DRS Mask bit is Clear and a 1b is written to the SFI DRS Trigger bit, the Downstream Port must behave as if a DRS Message was received.

IMPLEMENTATION NOTE
SFI Transparent Optimizations for Device Readiness

Certain devices may need more time to become Configuration-Ready following a hot-add operation than permitted. See Section 6.6.1 .

If system firmware is aware of such devices, it can use the SFI DPF mechanism to block operating system visibility of a newly added device, wait the necessary amount of time for the device to become Configuration-Ready, and then expose the device to the operating system.

To avoid the operating system from unnecessarily waiting additional time for the newly exposed device to become Configuration-Ready, system firmware can use the SFI DRS Trigger bit to have the Downstream Port emulate the reception of a DRS Message. An operating system that supports DRS can then immediately discover and configure the newly exposed device.

The newly exposed device doesn't necessarily need to be DRS capable itself. Since an Upstream Port is expressly permitted to send DRS Messages even when its DRS Supported bit is Clear, the Downstream Port above it can legitimately emulate receiving a DRS Message from it even if it is incapable of sending DRS Messages.

It should also be noted that in cases where system firmware is aware of a device becoming Configuration-Ready early, system firmware can expose this to the operating system using the SFI DRS Trigger mechanism.

Although SFI is not intended to be used by operating system software, it is recommended that operating systems used in platforms supporting SFI implement support for DRS, so that the system as a whole can have the benefits of this optimized Device Readiness timing.

IMPLEMENTATION NOTE
SFI DPF and Function Readiness Status (FRS) Messages

Downstream Port Filtering (DPF) does not affect the generation or propagation of FRS Messages. No FRS Messages are generated by a device when it becomes ready as part of an async hot-add operation. However, if system firmware performs operations on a device that result in FRS events, the resulting FRS Messages may be visible to the operating system. See Section 2.2.8.6.4 and Section 6.23.2 .

6.7.4.5 SFI Suppression of Hot-Plug Surprise Functionality

If a slot supports Hot-Plug Surprise (HPS) functionality as indicated by the Hot-Plug Surprise bit in the Slot Capabilities Register being Set, the SFI HPS Suppress bit in the SFI Control Register can be used to force the Hot-Plug Surprise bit to be Clear, and disable the associated Hot-Plug Surprise functionality.

HPS suppression is useful when a Downstream Port / slot combination supports both HPS and Downstream Port Containment (DPC). DPC is not recommended for concurrent use with HPS, so if a slot has HPS capability enabled, DPC should not be enabled. If software wishes to use DPC, software should first Set the SFI HPS Suppress bit in order to disable HPS functionality, allowing DPC to function properly.

IMPLEMENTATION NOTE

Software Negotiation of Hot-Plug Surprise Functionality

Assuming that system firmware owns the SFI Capability structure, it is recommended that for backward compatibility with older operating systems, Hot-Plug Surprise functionality be enabled by default on slots supporting async removal. Then, if the slot also supports DPC and the operating system wishes to use it instead, the operating system will request that HPS be suppressed by system firmware, and system firmware will determine whether to Set or Clear the SFI HPS Suppress bit.

6.7.5 Firmware Support for Hot-Plug

Some systems that include hot-plug capable Root Ports and Switches that are released before ACPI-compliant operating systems with native hot-plug support are available, can use ACPI firmware for propagating hot-plug events. Firmware control of the hot-plug registers must be disabled if an operating system with native support is used. Platforms that provide ACPI firmware to propagate hot-plug events must also provide a mechanism to transfer control to the operating system. The details of this method are described in the *PCI Firmware Specification*.

6.7.6 Async Removal

Async removal refers to the removal of an adapter or disabling of a Downstream Port Link due to error containment without prior warning to the operating system. This is in contrast to orderly removal, where removal operations are performed in a lock-step manner with the operating system through a well defined sequence of user actions and system management facilities. For example, the user presses the Attention Button to request permission from the operating system to remove the adapter, but the user doesn't actually remove the adapter from the slot until the operating system has quiesced activity to the adapter and granted permission for removal.

Since async removal proceeds before the rest of the PCI Express hierarchy or operating system necessarily becomes aware of the event, special consideration is required beyond that needed for standard PCI hot-plug. This section outlines PCI Express events that may occur as a side effect of async removal and mechanisms for handling async removal.

Since async removal may be unexpected to both the Physical and Data Link Layers of the Downstream Port associated with the slot, Correctable Errors may be reported as a side effect of the event (i.e. Receiver Error, Bad TLP, and Bad DLLP). If these errors are reported, software should handle them as an expected part of this event.

Requesters may experience Completion Timeouts associated with Requests that were accepted, but will never be completed by removed Completers. Any resulting Completion Timeout errors in this context should be handled as an expected part of this event.

Async removal may result in a transition from DL_Active to DL_Down in the Downstream port. This transition may result in a Surprise Down error. In addition, Requesters in the PCI Express hierarchy domain may not become immediately aware of this transition and continue to issue Requests to removed Completers that must be handled by the Downstream Port associated with the slot.

Either Downstream Port Containment (DPC) or the Hot-Plug Surprise (HPS) mechanism may be used to support async removal as part of an overall async hot-plug architecture. See Appendix I for the associated reference model.

IMPLEMENTATION NOTE

Hot-Plug Surprise Mechanism Deprecated for Async Hot-Plug

The Hot-Plug Surprise (HPS) mechanism, as indicated by the Hot-Plug Surprise bit in the Slot Capabilities Register being Set, is deprecated for use with async hot-plug. DPC is the recommended mechanism for supporting async hot-plug. See Section 6.7.4.4 for guidance on slots supporting both mechanisms.

With async removal, using HPS has serious downsides. Uncorrectable errors other than those that inherently bring down the Link need to be configured either to crash the system, be handled asynchronously by software, or be ignored. These include uncorrectable errors associated with Posted Memory Writes, TLPs with poisoned data, and Completion Timeouts. Uncorrectable errors ignored or handled asynchronously by software may make it impossible for the driver to determine which high-level operations complete successfully versus those that do not.

DPC provides a robust mechanism for supporting async removal. The TLP stream cleanly stops upon an uncorrectable error that triggers DPC. Operating System / driver stacks that support Containment Error Recovery (CER) can fully and transparently recover from many transient PCIe uncorrectable errors. DPC can support async removal and CER concurrently

6.8 Power Budgeting Capability

With the addition of a hot-plug capability for adapters, the need arises for the system to be capable of properly allocating power to any new devices added to the system. This capability is a separate and distinct function from power management and a basic level of support is required to ensure proper operation of the system. The power budgeting concept puts in place the building blocks that allow devices to interact with systems to achieve these goals. There are many ways in which the system can implement the actual power budgeting capabilities, and as such, they are beyond the scope of this specification.

Implementation of the Power Budgeting Capability is optional for devices that are implemented either in a form factor which does not require hot-plug support, or that are integrated on the system board. Form factor specifications may require support for power budgeting. The devices and/or adapters are required to remain under the configuration power limit specified in the corresponding electromechanical specification until they have been configured and enabled by the system. The system should guarantee that power has been properly budgeted prior to enabling an adapter.

6.8.1 System Power Budgeting Process Recommendations

It is recommended that system firmware provide the power budget management agent the following information:

- Total system power budget (power supply information).
- Total power allocated by system firmware (system board devices).
- Total number of slots and the types of slots.

System firmware is responsible for allocating power for all devices on the system board that do not have power budgeting capabilities. The firmware may or may not include devices that are connected to the standard power rails. When the firmware allocates the power for a device that implements the Power Budgeting Capability it must set the System Allocated bit to 1b in the Power Budget Capability register to indicate that it has been properly allocated. The

power budget manager is responsible for allocating all PCI Express devices including system board devices that have the Power Budgeting Capability and have the System Allocated bit Clear. The power budget manager is responsible for determining if hot-plugged devices can be budgeted and enabled in the system.

There are alternate methods which may provide the same functionality, and it is not required that the power budgeting process be implemented in this manner.

6.9 Slot Power Limit Control

PCI Express provides a mechanism for software controlled limiting of the maximum power per slot that an adapter (associated with that slot) can consume. If supported, the Emergency Power Reduction State, over-rides the mechanisms listed here (see Section 6.25). The key elements of this mechanism are:

- Slot Power Limit Value and Scale fields of the Slot Capabilities register implemented in the Downstream Ports of a Root Complex or a Switch
- Captured Slot Power Limit Value and Scale fields of the Device Capabilities register implemented in Endpoint, Switch, or PCI Express-PCI Bridge Functions present in an Upstream Port
- Set_Slot_Power_Limit Message that conveys the content of the Slot Power Limit Value and Scale fields of the Slot Capabilities register of the Downstream Port (of a Root Complex or a Switch) to the corresponding Captured Slot Power Limit Value and Scale fields of the Device Capabilities register in the Upstream Port of the component connected to the same Link

Power limits on the platform are typically controlled by the software (for example, platform firmware) that comprehends the specifics of the platform such as:

- Partitioning of the platform, including slots for I/O expansion using adapters
- Power delivery capabilities
- Thermal capabilities

This software is responsible for correctly programming the Slot Power Limit Value and Scale fields of the Slot Capabilities registers of the Downstream Ports connected to slots. After the value has been written into the register within the Downstream Port, it is conveyed to the adapter using the Set_Slot_Power_Limit Message (see Section 2.2.8.5). The recipient of the Message must use the value in the Message data payload to limit usage of the power for the entire adapter, unless the adapter will never exceed the lowest value specified in the corresponding form factor specification. It is required that device driver software associated with the adapter be able (by reading the values of the Captured Slot Power Limit Value and Scale fields of the Device Capabilities register) to configure hardware of the adapter to guarantee that the adapter will not exceed the imposed limit. In the case where the platform imposes a limit that is below the minimum needed for adequate operation, the device driver will be able to communicate this discrepancy to higher level configuration software. Configuration software is required to set the Slot Power Limit to one of the maximum values specified for the corresponding form factor based on the capability of the platform.

The following rules cover the Slot Power Limit control mechanism:

For Adapters:

- Until and unless a Set_Slot_Power_Limit Message is received indicating a Slot Power Limit value greater than the lowest value specified in the form factor specification for the adapter's form factor, the adapter must not consume more than the lowest value specified.
- An adapter must never consume more power than what was specified in the most recently received Set_Slot_Power_Limit Message or the minimum value specified in the corresponding form factor specification, whichever is higher.

- Components with Endpoint, Switch, or PCI Express-PCI Bridge Functions that are targeted for integration on an adapter where total consumed power is below the lowest limit defined for the targeted form factor are permitted to ignore Set_Slot_Power_Limit Messages, and to return a value of 0 in the Captured Slot Power Limit Value and Scale fields of the Device Capabilities register
  - Such components still must be able to receive the Set_Slot_Power_Limit Message without error but simply discard the Message value

For Root Complex and Switches which source slots:

- Configuration software must not program a Set_Slot_Power_Limit value that indicates a limit that is lower than the lowest value specified in the form factor specification for the slot's form factor.

IMPLEMENTATION NOTE

Example Adapter Behavior Based on the Slot Power Limit Control Capability

The following power limit scenarios are examples of how an adapter must behave based on the Slot Power Limit control capability. The form factor limits are representations, and should not be taken as actual requirements.

Note: Form factor #1 has a maximum power requirement of 40 W and 25 W; form factor #2 has a maximum power requirement of 15 W.

Scenario 1: An Adapter Consuming 12 W

- If the adapter is plugged into a form factor #1 40 W slot, the Slot Power Limit control mechanism is followed, and the adapter operates normally.
- If the adapter is plugged into a form factor #1 25 W slot, the Slot Power Limit control mechanism is followed, and the adapter operates normally.
- If the adapter is plugged into a form factor #2 15 W slot, the Slot Power Limit control mechanism is followed, and the adapter operates normally.

In all cases, since the adapter operates normally within all the form factors, it can ignore any of the slot power limit Messages.

Scenario 2: An Adapter Consuming 18 W

- If the adapter is plugged into a form factor #1 40 W slot, the Slot Power Limit control mechanism is followed, and the adapter operates normally.
- If the adapter is plugged into a form factor #1 25 W slot, the Slot Power Limit control mechanism is followed, and the adapter operates normally.
- If the adapter is plugged into a form factor #2 15 W slot, the Slot Power Limit control mechanism is followed, and the adapter must scale down to 15 W or disable operation. An adapter that does not scale within any of the power limits for a given form factor will always be disabled in that form factor and should not be used.

In this case, if the adapter is only to be used in form factor #1, it can ignore any of the slot power limit Messages. To be useful in form factor #2, the adapter should be capable of scaling to the power limit of form factor #2.

Scenario 3: An Adapter Consuming 30 W

- If the adapter is plugged into a form factor #1 40 W slot, the Slot Power Limit control mechanism is followed, and the device operates normally.
- If the adapter is plugged into a form factor #1 25 W slot, the Slot Power Limit control mechanism is followed, and the device must scale down to 25 W or disable operation.
- If the adapter is plugged into a form factor #2 15 W slot, the Slot Power Limit control mechanism is followed, and the adapter must scale down to 15 W or disable operation. An adapter that does not scale within any of the power limits for a given form factor will always be disabled in that form factor and should not be used.

In this case, since the adapter consumes power above the lowest power limit for a slot, the adapter must be capable of scaling or disabling to prevent system failures. Operation of adapters at power levels that exceed the capabilities of the slots in which they are plugged must be avoided.

IMPLEMENTATION NOTE
Slot Power Limit Control Registers

Typically Slot Power Limit register fields within Downstream Ports of a Root Complex or a Switch will be programmed by platform-specific software. Some implementations may use a hardware method for initializing the values of these registers and, therefore, do not require software support.

Components with Endpoint, Switch, or PCI Express-PCI Bridge Functions that are targeted for integration on the adapter where total consumed power is below the lowest limit defined for that form factor are allowed to ignore Set_Slot_Power_Limit Messages. Note that components that take this implementation approach may not be compatible with potential future defined form factors. Such form factors may impose lower power limits that are below the minimum required by a new adapter based on the existing component.

IMPLEMENTATION NOTE
Auto Slot Power Limit Disable

In some environments host software may wish to directly manage the transmission of a Set_Slot_Power_Limit message by performing a Configuration Write to the Slot Capabilities register rather than have the transmission automatically occur when the Link transitions from a non-DL_Up to a DL_Up status. This allows host software to limit power supply surge current by staggering the transition of Endpoints to a higher power state following a Link Down or when multiple Endpoints are simultaneously hot-added due to cable or adapter insertion.

6.10 Root Complex Topology Discovery

A Root Complex may present one of the following topologies to configuration software:

- A single opaque Root Complex such that software has no visibility with respect to internal operation of the Root Complex. All Root Ports are independent of each other from a software perspective; no mechanism exists to manage any arbitration among the various Root Ports for any differentiated services.
- A single Root Complex Component such that software has visibility and control with respect to internal operation of the Root Complex Component. As shown in Figure 6-11 , software views the Root Ports as Ingress Ports for the component. The Root Complex internal Port for traffic aggregation to a system Egress Port or an internal sink unit (such as memory) is represented by an RCRB structure. Controls for differentiated services are provided through a Virtual Channel Capability structure located in the RCRB.

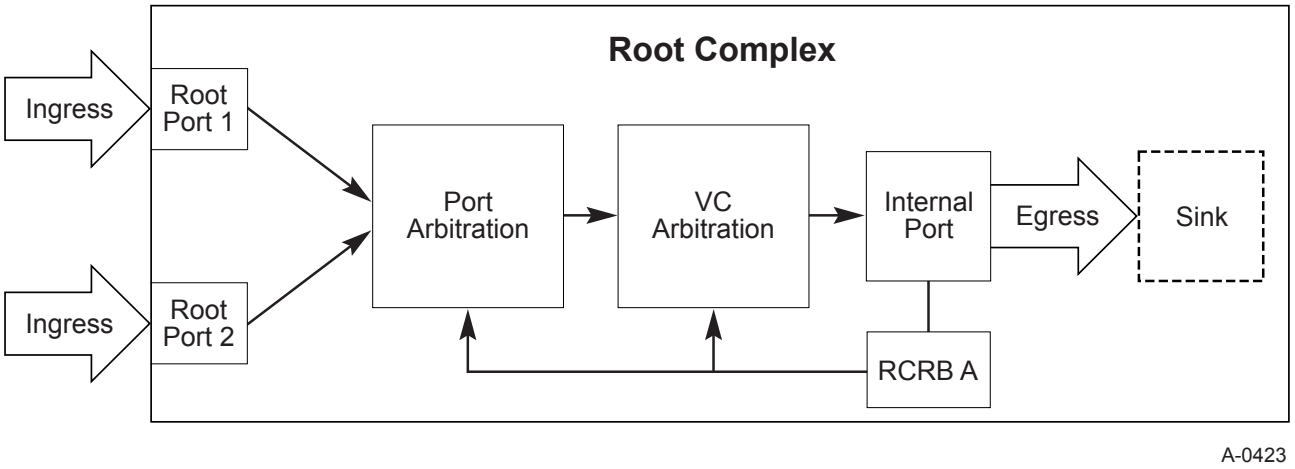

Figure 6-11 Root Complex Represented as a Single Component

- Multiple Root Complex Components such that software not only has visibility and control with respect to internal operation of a given Root Complex Component but also has the ability to discover and control arbitration between different Root Complex Components. As shown in Figure 6-12 , software views the Root Ports as Ingress Ports for a given component. An RCRB structure controls egress from the component to other Root Complex Components (RCRB C) or to an internal sink unit such as memory (RCRB A). In addition, an RCRB structure (RCRB B) may also be present in a given component to control traffic from other Root Complex Components. Controls for differentiated services are provided through Virtual Channel Capability structures located appropriately in the RCRBs respectively.

More complex topologies are possible as well.

A Root Complex topology can be represented as a collection of logical Root Complex Components such that each logical component has:

- One or more Ingress Ports.
- An Egress Port.
- Optional associated Virtual Channel capabilities located either in the Configuration Space (for Root Ports) or in an RCRB (for internal Ingress/Egress Ports) if the Root Complex supports Virtual Channels.
- Optional devices/Functions integrated in the Root Complex.

In order for software to correctly program arbitration and other control parameters for PCI Express differentiated services, software must be able to discover a Root Complex's internal topology. Root Complex topology discovery is accomplished by means of the Root Complex Link Declaration Capability as described in Section 7.9.8 .

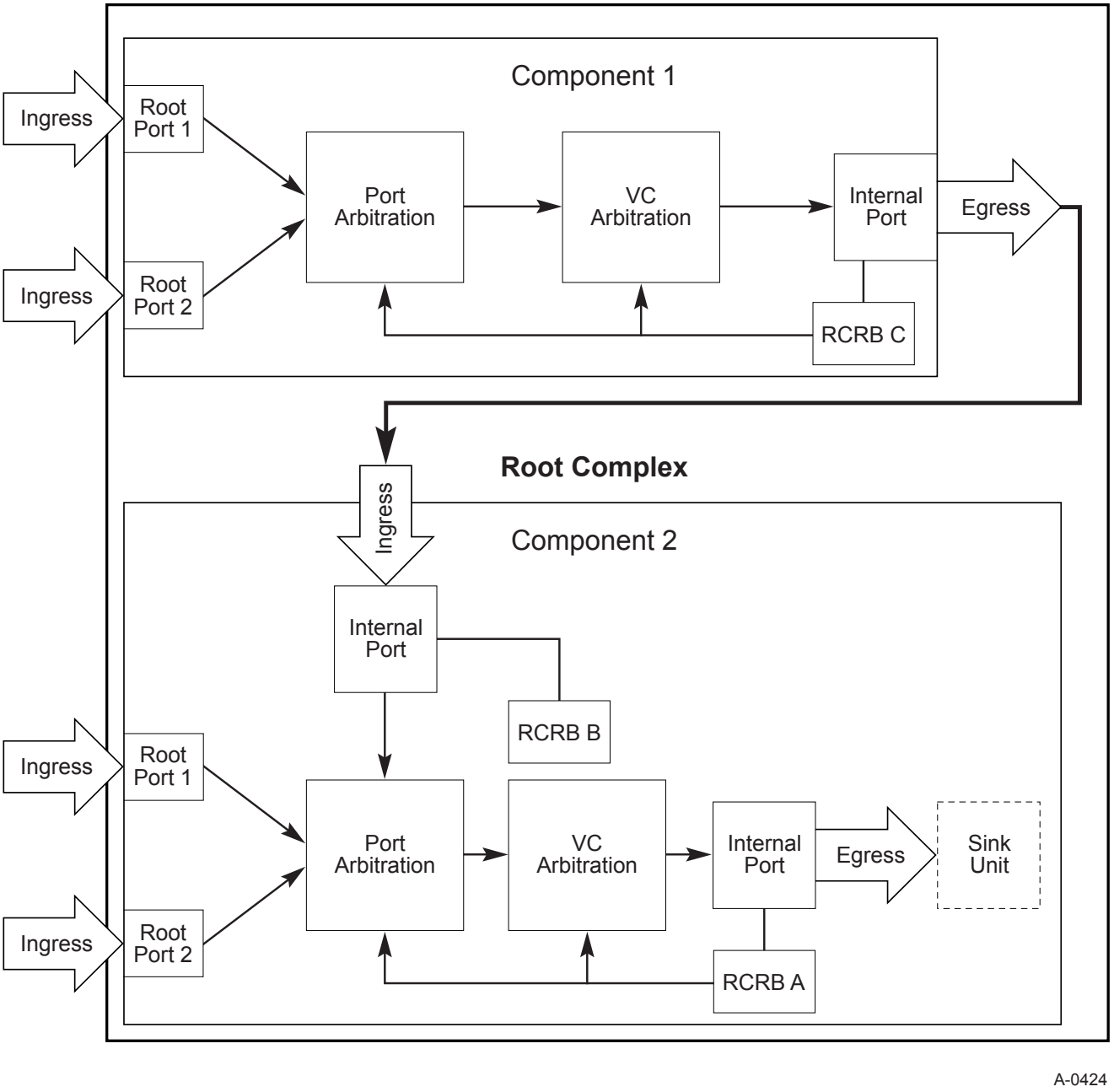

Figure 6-12 Root Complex Represented as Multiple Components

6.11 Link Speed Management

This section describes how Link speed management is coordinated between the LTSSM (Section 4.2.6) and the software Link observation and control mechanisms (see Section 7.5.3.6 , Section 7.5.3.7 , Section 7.5.3.8 , Section 7.5.3.18 , Section 7.5.3.19 , and Section 7.5.3.20).

The Target Link Speed field in the Link Control 2 register in the Downstream Port sets the upper bound for the Link speed. Except as described below, the Upstream component must attempt to maintain the Link at the Target Link Speed,

or at the highest speed supported by both components on the Link (as reported by the values in the training sets - see Section 4.2.4.1), whichever is lower.

Any Upstream Port or Downstream Port with the Hardware Autonomous Speed Disable bit in the Link Control 2 register clear is permitted to autonomously change the Link speed using implementation specific criteria.

If the reliability of the Link is unacceptably low, then either component is permitted to lower the Link speed by removing the unreliable Link speed from the list of supported speeds advertised in the training sets the component transmits. The criteria for determination of acceptable Link reliability are implementation specific, and are not dependent on the setting of the Hardware Autonomous Speed Disable bit.

During any given speed negotiation it is possible that one or both components will advertise a subset of all speeds supported, as a means to cap the post-negotiation Link speed. It is permitted for a component to change its set of advertised supported speeds without requesting a Link speed change by driving the Link through Recovery without setting the speed change bit.

When a component's attempt to negotiate to a particular Link speed fails, that component is not permitted to attempt negotiation to that Link speed, or to any higher Link speed, until 200 ms has passed from the return to L0 following the failed attempt, or until the other component on the Link advertises support for the higher Link speed through its transmitted training sets (with or without a request to change the Link speed), whichever comes first.

Software is permitted to restrict the maximum speed of Link operation and set the preferred Link speed by setting the value in the Target Link Speed field in the Upstream component. After modifying the value in the Target Link Speed field, software must trigger Link retraining by writing 1b to the Retrain Link bit. Software is notified of any Link speed changes (as well as any Link width changes) through the Link Bandwidth Notification Mechanism.

Software is permitted to cause a Link to transition to the Polling.Compliance LTSSM state at a particular speed by writing the Link Control 2 register in both components with the same value in the Target Link Speed field and Setting the Enter Compliance bit, and then initiating a Hot Reset on the Link (through the Downstream Port).

Note that this will take the Link to a DL_Down state and therefore cannot be done transparently to other software that is using the Link. The Downstream Port will return to Polling.Active when the Enter Compliance bit is cleared.

6.12 Access Control Services (ACS)

ACS defines a set of control points within a PCI Express topology to determine whether a TLP is to be routed normally, blocked, or redirected. ACS is applicable to RCs, Switches, and Multi-Function Devices.[113] For ACS requirements, single-Function devices that are SR-IOV capable must be handled as if they were Multi-Function Devices, since they essentially behave as Multi-Function Devices after their Virtual Functions (VFs) are enabled.

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.

ACS provides the following types of access control:

- ACS Source Validation
- ACS Translation Blocking
- ACS P2P Request Redirect
- ACS P2P Completion Redirect

113. Applicable Functions within Multi-Function Devices specifically include PCI Express Endpoints, Switch Upstream Ports, Legacy PCI Express Endpoints, and Root Complex Integrated Endpoints.

- ACS Upstream Forwarding
- ACS P2P Egress Control
- ACS Direct Translated P2P
- ACS I/O Request Blocking
- ACS DSP Memory Target Access
- ACS USP Memory Target Access
- ACS Unclaimed Request Redirect

The specific requirements for each of these are discussed in the following section.

ACS hardware functionality is disabled by default, and is enabled only by ACS-aware software. With the exception of ACS Source Validation, ACS access controls are not applicable to Multicast TLPs (see Section 6.14), and have no effect on them.

6.12.1 ACS Component Capability Requirements

ACS functionality is reported and managed via ACS Extended Capability structures. PCI Express components are permitted to implement ACS Extended Capability structures in some, none, or all of their applicable Functions. The extent of what is implemented is communicated through capability bits in each ACS Extended Capability structure. A given Function with an ACS Extended Capability structure may be required or forbidden to implement certain capabilities, depending upon the specific type of the Function and whether it is part of a Multi-Function Device.

ACS is never applicable to a PCI Express to PCI Bridge Function or a Root Complex Event Collector Function, and such Functions must never implement an ACS Extended Capability structure.

6.12.1.1 ACS Downstream Ports

This section applies to Root Ports and Switch Downstream Ports that implement an ACS Extended Capability structure. This section applies to Downstream Port Functions both for single-Function devices and Multi-Function Devices.

- ACS Source Validation: must be implemented.

  When enabled, the Downstream Port tests the Bus Number from the Requester ID of each Upstream Request received by the Port to determine if it is associated with the Secondary side of the virtual bridge associated with the Downstream Port, by either or both of:

  - Determining that the Requester ID falls within the Bus Number “aperture” of the Port - the inclusive range specified by the Secondary Bus Number register and the Subordinate Bus Number register.
  - If FPB is implemented and enabled, determining that the Requester ID is associated with the bridge’s Secondary Side by the application of the FPB Routing ID mechanism.
    If the Bus Number from the Requester ID of the Request is not within this aperture, this is a reported error (ACS Violation) associated with the Receiving Port (see Section 6.12.5 .)

    Completions are never affected by ACS Source Validation.

IMPLEMENTATION NOTE
Upstream Messages and ACS Source Validation

Functions are permitted to transmit Upstream Messages before they have been assigned a Bus Number. Such messages will have a Requester ID with a Bus Number of 00h. If the Downstream Port has ACS Source Validation enabled, these Messages (see Table F-1 and Section 6.23.1) will likely be detected as an ACS Violation error.

- ACS Translation Blocking: must be implemented.
  When enabled, the Downstream Port checks the Address Type (AT) field of each Upstream Memory Request received by the Port. If the AT field is not the default value, this is a reported error (ACS Violation) associated with the Receiving Port (see Section 6.12.5). This error must take precedence over ACS Upstream Forwarding and any applicable ACS P2P control mechanisms.

  Completions are never affected by ACS Translation Blocking.

- ACS P2P Request Redirect: must be implemented by Root Ports that support peer-to-peer traffic with other Root Ports;[114] must be implemented by Switch Downstream Ports.

  ACS P2P Request Redirect is subject to interaction with the ACS P2P Egress Control and ACS Direct Translated P2P mechanisms (if implemented). Refer to Section 6.12.3 for more information.

  When ACS P2P Request Redirect is enabled in a Switch Downstream Port, peer-to-peer Requests must be redirected Upstream towards the RC.

  When ACS P2P Request Redirect is enabled in a Root Port, peer-to-peer Requests must be sent to Redirected Request Validation logic within the RC that determines whether the Request is “reflected” back Downstream towards its original target, or blocked as an ACS Violation error. The algorithms and specific controls for making this determination are not architected by this specification.

  Downstream Ports never redirect Requests that are traveling Downstream.

  Completions are never affected by ACS P2P Request Redirect.

- ACS P2P Completion Redirect: must be implemented by Root Ports that implement ACS P2P Request Redirect; must be implemented by Switch Downstream Ports.
  The intent of ACS P2P Completion Redirect is to avoid ordering rule violations between Completions and Requests when Requests are redirected. Refer to Section 6.12.6 for more information.

  ACS P2P Completion Redirect does not interact with ACS controls that govern Requests.

  When ACS P2P Completion Redirect is enabled in a Switch Downstream Port, peer-to-peer Completions[115] that do not have the Relaxed Ordering Attribute bit set (1b) must be redirected Upstream towards the RC. Otherwise, peer-to-peer Completions must be routed normally.

  When ACS P2P Completion Redirect is enabled in a Root Port, peer-to-peer Completions that do not have the Relaxed Ordering bit set must be handled such that they do not pass Requests that are sent to Redirected Request Validation logic within the RC. Such Completions must eventually be sent Downstream towards their original peer-to-peer targets, without incurring additional ACS access control checks.

  Downstream Ports never redirect Completions that are traveling Downstream.

  Requests are never affected by ACS P2P Completion Redirect.

114. Root Port indication of ACS P2P Request Redirect or ACS P2P Completion Redirect support does not imply any particular level of peer-to-peer support by the Root Complex, or that peer-to-peer traffic is supported at all
115. This includes Read Completions, AtomicOp Completions, and other Completions with or without Data.

- ACS Upstream Forwarding: must be implemented by Root Ports if the RC supports Redirected Request Validation; must be implemented by Switch Downstream Ports.
  When ACS Upstream Forwarding is enabled in a Switch Downstream Port, and its Ingress Port receives an Upstream Request or Completion TLP targeting the Port's own Egress Port, the Port must instead forward the TLP Upstream towards the RC.

  When ACS Upstream Forwarding is enabled in a Root Port, and its Ingress Port receives an Upstream Request or Completion TLP that targets the Port's own Egress Port, the Port must handle the TLP as follows. For a Request, the Root Port must handle it the same as a Request that the Port "redirects" with the ACS P2P Request Redirect mechanism. For a Completion, the Root Port must handle it the same as a Completion that the Port "redirects" with the ACS P2P Completion Redirect mechanism.

  When ACS Upstream Forwarding is not enabled on a Downstream Port, and its Ingress Port receives an Upstream Request or Completion TLP that targets the Port's own Egress Port, the handling of the TLP is undefined.

- ACS P2P Egress Control: implementation is optional.
  ACS P2P Egress Control is subject to interaction with the ACS P2P Request Redirect and ACS Direct Translated P2P mechanisms (if implemented). Refer to Section 6.12.3 for more information.

  A Switch that supports ACS P2P Egress Control can be selectively configured to block peer-to-peer Requests between its Downstream Ports. Software can configure the Switch to allow none or only a subset of its Downstream Ports to send peer-to-peer Requests to other Downstream Ports. This is configured on a per Downstream Port basis.

  An RC that supports ACS P2P Egress Control can be selectively configured to block peer-to-peer Requests between its Root Ports. Software can configure the RC to allow none or only a subset of the Hierarchy Domains to send peer-to-peer Requests to other Hierarchy Domains. This is configured on a per Root Port basis.

  With ACS P2P Egress Control in Downstream Ports, controls in the Ingress Port ("sending" Port) determine if the peer-to-peer Request is blocked, and if so, the Ingress Port handles the ACS Violation error per Section 6.12.5 .

  Completions are never affected by ACS P2P Egress Control.

- ACS Direct Translated P2P: must be implemented by Root Ports that support Address Translation Services (ATS) and also support peer-to-peer traffic with other Root Ports;[116] must be implemented by Switch Downstream Ports.

  When ACS Direct Translated P2P is enabled in a Downstream Port, peer-to-peer Memory Requests whose Address Type (AT) field indicates a Translated address must be routed normally ("directly") to the peer Egress Port, regardless of ACS P2P Request Redirect and ACS P2P Egress Control settings. All other peer-to-peer Requests must still be subject to ACS P2P Request Redirect and ACS P2P Egress Control settings.

  Completions are never affected by ACS Direct Translated P2P.

- ACS I/O Request Blocking: must be implemented by Root Ports and Switch Downstream Ports that support ACS Enhanced Capability.

  When enabled, the Port must handle an Upstream I/O Request received by the Port's Ingress as an ACS Violation.

- ACS DSP Memory Target Access: must be implemented by Root Ports and Switch Downstream Ports that support ACS Enhanced Capability and that have applicable Memory BAR Space to protect.

116. Root Port indication of ACS Direct Translated P2P support does not imply any particular level of peer-to-peer support by the Root Complex, or that peer-to-peer traffic is supported at all.

ACS DSP Memory Target Access determines how an Upstream Request received by the Downstream Port's Ingress and targeting any Memory BAR Space[117] associated with an applicable Downstream Port is handled. The Request can be blocked, redirected, or allowed to proceed directly to its target. In a Switch, all Downstream Ports are applicable, including the one on which the Request was received. In a Root Complex, the set of applicable Root Ports is implementation specific, but always includes the one on which the Request was received.

- ACS USP Memory Target Access: must be implemented by Switch Downstream Ports that support ACS Enhanced Capability and that have applicable Memory BAR Space in the Switch Upstream Port to protect; is not applicable to Root Ports.

  ACS USP Memory Target Access determines how an Upstream Request received by the Switch Downstream Port's Ingress and targeting any Memory BAR Space[118] associated with the Switch's Upstream Port is handled. The Request can be blocked, redirected, or allowed to proceed directly to its target.

  If any Functions other than the Switch Upstream Port are associated with the Upstream Port, this field has no effect on accesses to their Memory BAR Space[119] . Such access is controlled by the ACS Extended Capability (if present) in the Switch Upstream Port.

- ACS Unclaimed Request Redirect: must be implemented by Switch Downstream Ports that support ACS Enhanced Capability; is not applicable to Root Ports.

  When enabled, incoming Requests received by the Switch Downstream Port's Ingress and targeting Memory Space within the memory window of a Switch Upstream Port that is not within a memory window or Memory BAR Target of any Downstream Port within the Switch are redirected Upstream out of the Switch.

  When not enabled, such Requests are handled by the Switch Downstream Port as an Unsupported Request (UR).

6.12.1.2 ACS Functions in SR-IOV Capable and Multi-Function Devices

This section applies to Multi-Function Device ACS Functions, with the exception of Downstream Port Functions, which are covered in the preceding section. For ACS requirements, single-Function devices that are SR-IOV capable must be handled as if they were Multi-Function Devices.

- ACS Source Validation: must not be implemented.
- ACS Translation Blocking: must not be implemented.
- ACS P2P Request Redirect: must be implemented by Functions that support peer-to-peer traffic with other Functions. This includes SR-IOV Virtual Functions (VFs).

  ACS P2P Request Redirect is subject to interaction with the ACS P2P Egress Control and ACS Direct Translated P2P mechanisms (if implemented). Refer to Section 6.12.3 for more information.

  When ACS P2P Request Redirect is enabled in a Multi-Function Device that is not an RCiEP, peer-to-peer Requests (between Functions of the device) must be redirected Upstream towards the RC.

  It is permitted but not required to implement ACS P2P Request Redirect in an RCiEP. When ACS P2P Request Redirect is enabled in an RCiEP, peer-to-peer Requests, defined as all Requests that do not target system memory, must be sent to implementation-specific logic within the Root Complex that determines whether the

117. This also includes any Memory Space allocated by an Expansion ROM Base Address register (BAR). This also includes any Memory Space allocated by EA entries with a BEI value of 0, 1, 7, or 8. See Section 7.8.5.3 .
118. This also includes any Memory Space allocated by an Expansion ROM Base Address register (BAR). This also includes any Memory Space allocated by EA entries with a BEI value of 0, 1, 7, or 8. See Section 7.8.5.3 .
119. This also includes any Memory Space allocated by an Expansion ROM Base Address register (BAR). This also includes any Memory Space allocated by EA entries with a BEI value of 0, 1, 7, or 8. See Section 7.8.5.3 .

Request is directed towards its original target, or blocked as an ACS Violation error. The algorithms and specific controls for making this determination are not architected by this specification.

Completions are never affected by ACS P2P Request Redirect.

- ACS P2P Completion Redirect: must be implemented by Functions that implement ACS P2P Request Redirect. The intent of ACS P2P Completion Redirect is to avoid ordering rule violations between Completions and Requests when Requests are redirected. Refer to Section 6.12.6 for more information.

  ACS P2P Completion Redirect does not interact with ACS controls that govern Requests.

  When ACS P2P Completion Redirect is enabled in a Multi-Function Device that is not an RCiEP, peer-to-peer Completions that do not have the Relaxed Ordering bit set must be redirected Upstream towards the RC. Otherwise, peer-to-peer Completions must be routed normally.

  Requests are never affected by ACS P2P Completion Redirect.

- ACS Upstream Forwarding: must not be implemented.
- ACS P2P Egress Control: implementation is optional; is based on Function Numbers or Function Group Numbers; controls peer-to-peer Requests between the different Functions within the multi-Function or SR-IOV capable device.

  ACS P2P Egress Control is subject to interaction with the ACS P2P Request Redirect and ACS Direct Translated P2P mechanisms (if implemented). Refer to Section 6.12.3 for more information.

  Each Function within a Multi-Function Device that supports ACS P2P Egress Control can be selectively enabled to block peer-to-peer communication with other Functions or Function Groups[120] within the device. This is configured on a per Function basis.

  With ACS P2P Egress Control in multi-Function or SR-IOV capable devices, controls in the "sending" Function determine if the Request is blocked, and if so, the "sending" Function handles the ACS Violation error per Section 6.12.5 .

  When ACS Function Groups are enabled in an ARI Device (ACS Function Groups Enable is Set), ACS P2P Egress Controls are enforced on a per Function Group basis instead of a per Function basis. See Section 6.13 .

  Completions are never affected by ACS P2P Egress Control.

- ACS Direct Translated P2P: must be implemented if the Multi-Function Device Function supports Address Translation Services (ATS) and also peer-to-peer traffic with other Functions.

  When ACS Direct Translated P2P is enabled in a Multi-Function Device, peer-to-peer Memory Requests whose Address Type (AT) field indicates a Translated address must be routed normally (“directly”) to the peer Function, regardless of ACS P2P Request Redirect and ACS P2P Egress Control settings. All other peer-to-peer Requests must still be subject to ACS P2P Request Redirect and ACS P2P Egress Control settings.

  Completions are never affected by ACS Direct Translated P2P.

6.12.1.3 Functions in Single-Function Devices

This section applies to single-Function device Functions, with the exception of Downstream Port Functions and SR-IOV capable Functions, which are covered in a preceding section. For ACS requirements, single-Function devices that are SR-IOV capable must be handled as if they were Multi-Function Devices.

No ACS capabilities are applicable, and the Function must not implement an ACS Extended Capability structure.

120. ACS Function Groups capability is optional for ARI Devices that implement ACS P2P Egress Controls.

6.12.2 Interoperability

The following rules govern interoperability between ACS and non-ACS components:

- When ACS P2P Request Redirect and ACS P2P Completion Redirect are not being used, ACS and non-ACS components may be intermixed within a topology and will interoperate fully. ACS can be enabled in a subset of the ACS components without impacting interoperability.
- When ACS P2P Request Redirect, ACS P2P Completion Redirect, or both are being used, certain components in the PCI Express hierarchy must support ACS Upstream Forwarding (of Upstream redirected Requests). Specifically:
  The associated Root Port[121] must support ACS Upstream Forwarding. Otherwise, how the Root Port handles Upstream redirected Request or Completion TLPs is undefined. The RC must also implement Redirected Request Validation.

  Between each ACS component where P2P TLP redirection is enabled and its associated Root Port, any intermediate Switches must support ACS Upstream Forwarding. Otherwise, how such Switches handle Upstream redirected TLPs is undefined.

6.12.3 ACS Peer-to-Peer Control Interactions

With each peer-to-peer Request, multiple ACS control mechanisms may interact to determine whether the Request is routed directly towards its peer-to-peer target, blocked immediately as an ACS Violation, or redirected Upstream towards the RC for access validation. Peer-to-peer Completion redirection is determined exclusively by the ACS P2P Completion Redirect mechanism.

If ACS Direct Translated P2P is enabled in a Port/Function, peer-to-peer Memory Requests whose Address Type (AT) field indicates a Translated address must be routed normally (“directly”) to the peer Port/Function, regardless of ACS P2P Request Redirect and ACS P2P Egress Control settings. Otherwise such Requests, and unconditionally all other peer-to-peer Requests, must be subject to ACS P2P Request Redirect and ACS P2P Egress Control settings. Specifically, the applicable Egress Control Vector bit, along with the ACS P2P Egress Control Enable bit (E) and the ACS P2P Request Redirect Enable bit (R), determine how the Request is handled. It must be noted that atomicity of accesses cannot be guaranteed if ACS peer-to-peer Request Redirect targets a legacy device location that can be the target of a locked access. Refer to Section 7.7.8 for descriptions of these control bits. Table 6-10 specifies the interactions.

Table 6-10 ACS P2P Request Redirect and ACS P2P Egress Control Interactions

Control Bit E (b)	Control Bit R (b)	Egress Control Vector Bit for the Associated Egress Switch Port, Root Port, Function, or Function Group	Required Handling for Peer-to-Peer Requests
0	0	X - Don’t care	Route directly to peer-to-peer target
0	1	X - Don’t Care	Redirect Upstream
1	0	1	Handle as an ACS Violation
1	0	0	Route directly to peer-to-peer target
1	1	1	Redirect Upstream

121. Not applicable for ACS Redirect between Functions of a multi-Function Root Complex Integrated Endpoint.

Control Bit E (b)	Control Bit R (b)	Egress Control Vector Bit for the Associated Egress Switch Port, Root Port, Function, or Function Group	Required Handling for Peer-to-Peer Requests
1	1	0	Route directly to peer-to-peer target

6.12.4 ACS Enhanced Capability

ACS Enhanced Capability is an additional set of ACS control mechanisms to improve the level of isolation and protection provided by ACS. ACS Enhanced Capability defines the following additional access control mechanisms:

- ACS I/O Request Blocking
- ACS DSP Memory Target Access
- ACS USP Memory Target Access
- ACS Unclaimed Request Redirect

Through these mechanisms, ACS Enhanced Capability provides protection and consistent handling of Requests directed toward regions not covered by the original ACS mechanisms.

IMPLEMENTATION NOTE

ACS Redirect and Guest Physical Addresses (GPAs)

ACS redirect mechanisms were originally architected to enable fine-grained access control for P2P Memory Requests, by redirecting selected Requests Upstream to the RC, where validation logic determines whether to allow or deny access. However, ACS redirect mechanisms can also ensure that Functions under the direct control of VMs have their DMA Requests routed correctly to the Translation Agent in the host, which then translates their guest physical addresses (GPAs) into host physical addresses (HPAs).

GPA ranges used for Memory Space vs. DMA are not guaranteed to coincide with HPA ranges, which the PCIe fabric uses for Memory Request routing and access control. If any GPAs used for DMA fall with within the HPA ranges used for Memory Space, legitimate or malicious packet misrouting can result.

ACS redirect mechanisms can ensure that Upstream Memory Requests with GPAs intended for DMA never get routed to HPA Memory ranges. ACS P2P Request Redirect handles this for (1) peer accesses between Functions within a Multi-Function Device and (2) peer accesses between Downstream Ports within a Switch or RC. ACS P2P Egress Control with redirect handles this in a more fine-grained manner for the same two cases.

Redirect mechanisms introduced with ACS Enhanced Capability handle this for additional cases. ACS DSP Memory Target Access with redirect handles this for Downstream Port Memory Resource ranges. ACS USP Memory Target Access with redirect handles this for Switch Upstream Port Memory Resource ranges. In Switches, ACS Unclaimed Request Redirect handles this for any areas within Upstream Port Memory apertures that are not handled by the other ACS redirect mechanisms. Together these ACS redirect mechanisms can ensure that Upstream Memory Requests with GPAs intended for DMA are always routed or redirected to the Translation Agent in the host, and those with GPAs intended for P2P are still routed as originally architected.

6.12.5 ACS Violation Error Handling

ACS Violations may occur due to either hardware or software defects/failures. To assist in fault isolation and root cause analysis, it is recommended that AER be implemented in ACS components. AER prefix/header logging and the Prefix Log/ Header Log registers may be used to determine the prefix/header of the offending Request. The ACS Violation Status, Mask, and Severity bits provide positive identification of the error and increased control over error logging and signaling.

When an ACS Violation is detected, the ACS component that operates as the Completer[122] must do the following:

- For Non-Posted Requests, the Completer must generate a Completion with a Completer Abort (CA) Completion Status.
- The Completer must log and signal the ACS Violation as indicated in Figure 6-2 . Note the following:
  - Even though the Completer uses a CA Completion Status when it sends a Completion, the Completer must log an ACS Violation error instead of a Completer Abort error.
  - If the severity of the ACS Violation is non-fatal and the Completer sends a Completion with CA Completion Status, this case must be handled as an Advisory Non-Fatal Error as described in Section 6.2.3.2.4.1 .
- The Completer[123] must set the Signaled Target Abort bit in either its Status register or Secondary Status register as appropriate.

6.12.6 ACS Redirection Impacts on Ordering Rules

When ACS P2P Request Redirect is enabled, some or all peer-to-peer Requests are redirected, which can cause ordering rule violations in some cases. This section explores those cases, plus a similar case that occurs with RCs that implement “Request Retargeting” as an alternative mechanism for enforcing peer-to-peer access control.

6.12.6.1 Completions Passing Posted Requests

When a peer-to-peer Posted Request is redirected, a subsequent peer-to-peer non-RO[124] Completion that is routed directly can effectively pass the redirected Posted Request, violating the ordering rule that non-RO Completions must not pass Posted Requests. Refer to Section 2.4.1 for more information.

ACS P2P Completion Redirect can be used to avoid violating this ordering rule. When ACS P2P Completion Redirect is enabled, all peer-to-peer non-RO Completions will be redirected, thus taking the same path as redirected peer-to-peer Posted Requests. Enabling ACS P2P Completion Redirect when some or all peer-to-peer Requests are routed directly will not cause any ordering rule violations, since it is permitted for a given Completion to be passed by any TLP other than another Completion with the same Transaction ID.

As an alternative mechanism to ACS P2P Request Redirect for enforcing peer-to-peer access control, some RCs implement “Request Retargeting”, where the RC supports special address ranges for “peer-to-peer” traffic, and the RC will retarget validated Upstream Requests to peer devices. Upon receiving an Upstream Request targeting a special address range, the RC validates the Request, translates the address to target the appropriate peer device, and sends the Request back Downstream. With retargeted Requests that are Non-posted, if the RC does not modify the Requester ID,

122. In all cases but one, the ACS component that detects the ACS Violation also operates as the Completer. The exception case is when Root Complex Redirected Request Validation logic disallows a redirected Request. If the redirected Request came through a Root Port, that Root Port must operate as the Completer. If the redirected Request came from a Root Complex Integrated Endpoint, the associated Root Complex Event Collector must operate as the Completer.
123. Similarly, if the Request was Non-Posted, when the Requester receives the resulting Completion with CA Completion Status, the Requester must set the Received Target Abort bit in either its Status register or Secondary Status register as appropriate. Note that for the case of a Multi-Function Device incurring an ACS Violation error with a peer-to-peer Request between its Functions, the same Function might serve both as Requester and Completer.
124. In this section, “non-RO” is an abbreviation characterizing TLPs whose Relaxed Ordering Attribute field is not set.

the resulting Completions will travel “directly” peer-to-peer back to the original Requester, creating the possibility of non-RO Completions effectively passing retargeted Posted Requests, violating the same ordering rule as when ACS P2P Request Redirect is being used. ACS P2P Completion Redirect can be used to avoid violating this ordering rule here as well.

If ACS P2P Request Redirect and RC P2P Request Retargeting are not being used, there is no envisioned benefit to enabling ACS P2P Completion Redirect, and it is recommended not to do so because of potential performance impacts.

IMPLEMENTATION NOTE

Performance Impacts with ACS P2P Completion Redirect

While the use of ACS P2P Completion Redirect can avoid ordering violations with Completions passing Posted Requests, it also may impact performance. Specifically, all redirected Completions will have to travel up to the RC from the point of redirection and back, introducing extra latency and possibly increasing Link and RC congestion.

Since peer-to-peer Completions with the Relaxed Ordering bit set are never redirected (thus avoiding performance impacts), it is strongly recommended that Requesters be implemented to maximize the proper use of Relaxed Ordering, and that software enable Requesters to utilize Relaxed Ordering by setting the Enable Relaxed Ordering bit in the Device Control Register.

If software enables ACS P2P Request Redirect, RC P2P Request Retargeting, or both, and software is certain that proper operation is not compromised by peer-to-peer non-RO Completions passing peer-to-peer[125] Posted Requests, it is recommended that software leave ACS P2P Completion Redirect disabled as a way to avoid its performance impacts.

6.12.6.2 Requests Passing Posted Requests

When some peer-to-peer Requests are redirected but other peer-to-peer Requests are routed directly, the possibility exists of violating the ordering rules where Non-posted Requests or non-RO Posted Requests must not pass Posted Requests. Refer to Section 2.4.1 for more information.

These ordering rule violation possibilities exist only when ACS P2P Request Redirect and ACS Direct Translated P2P are both enabled. Software should not enable both these mechanisms unless it is certain either that such ordering rule violations cannot occur, or that proper operation will not be compromised if such ordering rule violations do occur.

125. These include true peer-to-peer Requests that are redirected by the ACS P2P Request Redirect mechanism, as well as “logically peer-to-peer” Requests routed to the Root Complex that the Root Complex then retargets to the peer device.

IMPLEMENTATION NOTE
Ensuring Proper Operation with ACS Direct Translated P2P

The intent of ACS Direct Translated P2P is to optimize performance in environments where Address Translation Services (ATS) are being used with peer-to-peer communication whose access control is enforced by the RC. Permitting peer-to-peer Requests with Translated addresses to be routed directly avoids possible performance impacts associated with redirection, which introduces extra latency and may increase Link and RC congestion.

For the usage model where peer-to-peer Requests with Translated addresses are permitted, but those with Untranslated addresses are to be blocked as ACS Violations, it is recommended that software enable ACS Direct Translated P2P and ACS P2P Request Redirect, and configure the Redirected Request Validation logic in the RC to block the redirected Requests with Untranslated addresses. This configuration has no ordering rule violations associated with Requests passing Posted Requests.

For the usage model where some Requesters use Translated addresses exclusively with peer-to-peer Requests and some Requesters use Untranslated addresses exclusively with peer-to-peer Requests, and the two classes of Requesters do not communicate peer-to-peer with each other, proper operation is unlikely to be compromised by redirected peer-to-peer Requests (with Untranslated addresses) being passed by direct peer-to-peer Requests (with Translated addresses). It is recommended that software not enable ACS Direct Translated P2P unless software is certain that proper operation is not compromised by the resulting ordering rule violations.

For the usage model where a single Requester uses both Translated and Untranslated addresses with peer-to-peer Requests, again it is recommended that software not enable ACS Direct Translated P2P unless software is certain that proper operation is not compromised by the resulting ordering rule violations. This requires a detailed analysis of the peer-to-peer communications models being used, and is beyond the scope of this specification.

6.13 Alternative Routing-ID Interpretation (ARI)

Routing IDs, Requester IDs, and Completer IDs are 16-bit identifiers traditionally composed of three fields: an 8-bit Bus Number, a 5-bit Device Number, and a 3-bit Function Number. With ARI, the 16-bit field is interpreted as two fields instead of three: an 8-bit Bus Number and an 8-bit Function Number - the Device Number field is eliminated. This new interpretation enables an ARI Device to support up to 256 Functions [0..255] instead of 8 Functions [0..7].

ARI is controlled by a new set of optional capability and control register bits. These provide:

- Software the ability to detect whether a component supports ARI.
- Software the ability to configure an ARI Downstream Port so the logic that determines when to turn a Type 1 Configuration Request into a Type 0 Configuration Request no longer enforces a restriction on the traditional Device Number field being 0.
- Software the ability to configure an ARI Device to assign each Function to a Function Group. Controls based on Function Groups may be preferable when finer granularity controls based on individual Functions are not required.
  - If Multi-Function VC arbitration is supported and enabled, arbitration can optionally be based on Function Groups instead of individual Functions.
  - If ACS P2P Egress Controls are supported and enabled, access control can optionally be based on Function Groups instead of individual Functions.

The following illustrates an example flow for enabling these capabilities and provides additional details on their usage:

1. Software enumerates the PCI Express hierarchy and determines whether the ARI Extended Capability is supported.
   a. For an ARI Downstream Port, the capability is communicated through the Device Capabilities 2 register.
   b. For an ARI Device, the capability is communicated through the ARI Extended Capability structure.
   c. ARI has no impact on the base enumeration algorithms used in platforms today.
2. Software enables ARI functionality in each component.
   a. In an ARI Downstream Port immediately above an ARI Device, software sets the ARI Forwarding Enable bit in the Device Control 2 register. Setting this bit ensures the logic that determines when to turn a Type 1 Configuration Request into a Type 0 Configuration Request no longer enforces a restriction on the traditional Device Number field being 0.
   b. In an ARI Device, Extended Functions must respond if addressed with a Type 0 Configuration Request. It is necessary for ARI-aware software to enable ARI Forwarding in the Downstream Port immediately above the ARI Device, in order for ARI-aware software to discover and configure the Extended Functions.
   c. If an ARI Device implements a Multi-Function VC Capability structure with Function arbitration, and also implements MFVC Function Groups, ARI-aware software categorizes Functions into Function Groups.
      i. Each Function is assigned to a Function Group represented by a ***Function Group Number***.
      ii. A maximum of 8 Function Groups can be configured.
      iii. Within the Multi-Function VC Arbitration Table, a Function Group Number is used in place of a Function Number in each arbitration slot.
         1. Arbitration occurs on a Function Group basis instead of an individual Function basis.
         2. All other aspects of Multi-Function VC arbitration remain unchanged. See Section 7.9.2.10 for additional details.
      iv. Function arbitration within each Function Group is implementation-specific.
   d. If an ARI Device supports ACS P2P Egress Control, access control can be optionally implemented on a Function Group basis.
   e. To improve the enumeration performance and create a more deterministic solution, software can enumerate Functions through a linked list of Function Numbers. The next linked list element is communicated through each Function's ARI Capability Register.
      i. Function 0 acts as the head of a linked list of Function Numbers. Software detects a non-zero Next Function Number field within the ARI Capability Register as the next Function within the linked list. Software issues a configuration probe using the Bus Number captured by the Device and the Function Number derived from the ARI Capability Register to locate the next associated Function's configuration space.
      ii. Function Numbers may be sparse and non-sequential in their consumption by an ARI Device.

With an ARI Device, the Phantom Functions Supported field within each Function's Device Capabilities register (see Section 7.5.3.3 , Table 7-19) must be set to 00b to indicate that Phantom Functions are not supported. The Extended Tag Field Enable bit and the 10-Bit Tag Requester Enable bit can still be used to enable each Function to support higher numbers of outstanding Requests. See Section 2.2.6.2 .

Figure 6-13 shows an example system topology with two ARI Devices, one below a Root Port and one below a Switch. For access to Extended Functions in ARI Device X, Root Port A must support ARI Forwarding and have it enabled by

software. For access to Extended Functions in ARI Device Y, Switch Downstream Port D must support ARI Forwarding and have it enabled by software. With this configuration, it is recommended that software not enable ARI Forwarding in Root Port B or Switch Downstream Port C.

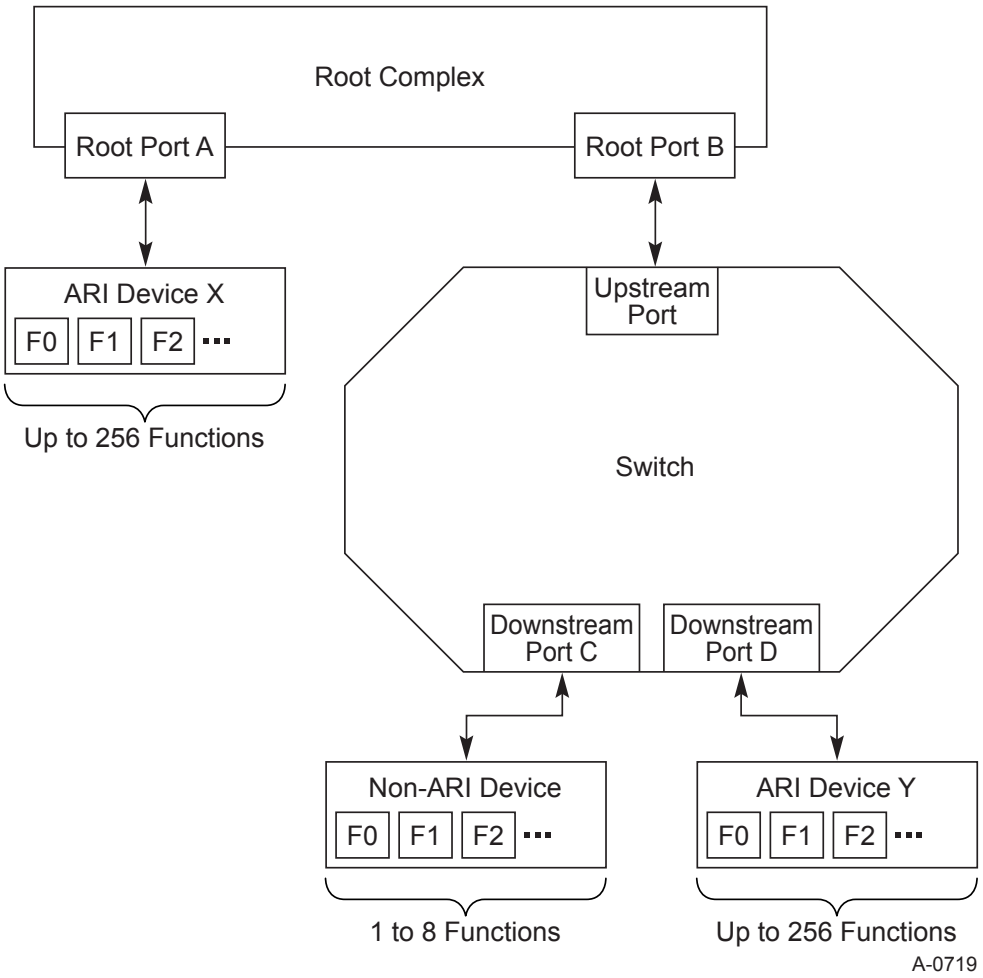

Figure 6-13 Example System Topology with ARI Devices

IMPLEMENTATION NOTE

ARI Forwarding Enable Being Set Inappropriately

It is strongly recommended that software in general Set the ARI Forwarding Enable bit in a Downstream Port only if software is certain that the device immediately below the Downstream Port is an ARI Device. If the bit is Set when a non-ARI Device is present, the non-ARI Device can respond to Configuration Space accesses under what it interprets as being different Device Numbers, and its Functions can be aliased under multiple Device Numbers, generally leading to undesired behavior.

Following a hot-plug event below a Downstream Port, it is strongly recommended that software Clear the ARI Forwarding Enable bit in the Downstream Port until software determines that a newly added component is in fact an ARI Device.

IMPLEMENTATION NOTE

ARI Forwarding Enable Setting at Firmware/Operating System Control Handoff

It is strongly recommended that firmware not have the ARI Forwarding Enable bit Set in a Downstream Port upon control handoff to an operating system unless firmware knows that the operating system is ARI-aware. With this bit Set, a non-ARI-aware operating system might be able to discover and enumerate Extended Functions in an ARI Device below the Downstream Port, but such an operating system would generally not be able to manage Extended Functions successfully, since it would interpret there being multiple Devices below the Downstream Port instead of a single ARI Device. As one example of many envisioned problems, the interrupt binding for INTx virtual wires would not be consistent with what the non-ARI-aware operating system would expect.

6.14 Multicast Operations

The Multicast Capability structure defines a Multicast address range, the segmentation of that range into a number, N, of equal sized Multicast Windows, and the association of each Multicast Window with a Multicast Group, MCG. Each Function that supports Multicast within a component implements a Multicast Capability structure that provides routing directions and permission checking for each MCG for TLPs passing through or to the Function. The Multicast Group is a field of up to 6 bits in width embedded in the address beginning at the MC_Index_Position, as defined in Section 7.9.11.4 .

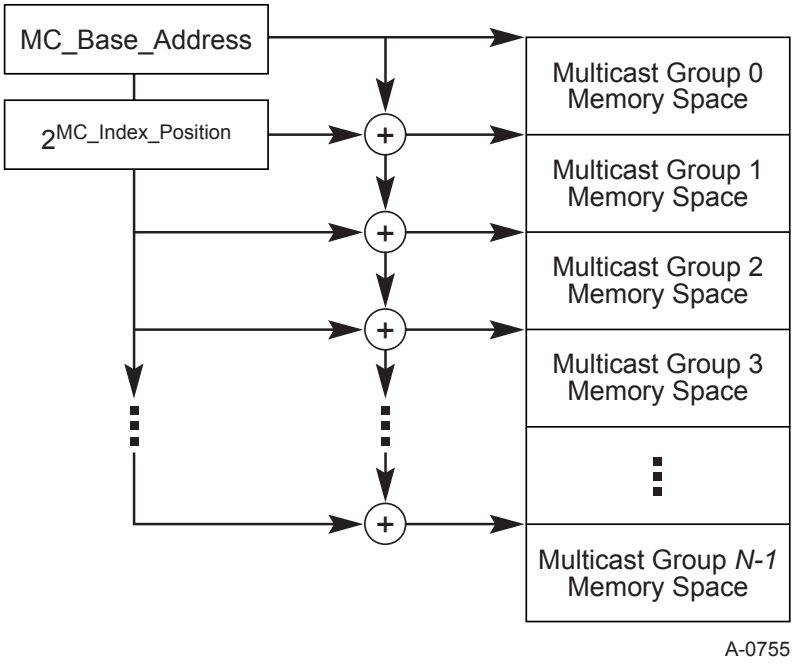

Figure 6-14 Segmentation of the Multicast Address Range

6.14.1 Multicast TLP Processing

A Multicast Hit occurs if all of the following are true:

- MC_Enable is Set
- TLP is a Memory Write or an Address Routed Message, both of which are Posted Requests
- $Address_{TLP}$ >= MC_Base_Address
- $Address_{TLP}$ < (MC_Base_Address + ($2^{MC\_Index\_Position}$ * (MC_Num_Group + 1)))

In this step, each Switch Ingress Port and other components use values of MC_Enable, MC_Base_Address, MC_Index_Position, and MC_Num_Group from any one of their Functions. Software is required to configure all Functions of a Switch and all Functions of a Multi-Function Upstream Port to have the same values in each of these fields and results are indeterminate if this is not the case.

If the address in a Non-Posted Memory Request hits in a Multicast Window, no Multicast Hit occurs and the TLP is processed normally per the base specification - i.e., as a unicast.

If a Multicast Hit occurs, the only ACS access control that can still apply is ACS Source Validation. In particular, neither ACS redirection nor the ACS Egress Control vector affects operations during a Multicast Hit.

If a Multicast Hit occurs, normal address routing rules do not apply. Instead, the TLP is processed as follows:

The Multicast Group is extracted from the address in the TLP using any Function's values for MC_Base_Address and MC_Index_Position. Specifically:

MCG = (($Address_{TLP}$ - MC_Base_Address) >> MC_Index_Position) & 3Fh

In this process, the component may use any Function's values for MC_Base_Address and MC_Index_Position. Which Function's values are used is device-specific.

Components next check the MC_Block_All and the MC_Block_Untranslated bits corresponding to the extracted MCG. Switches and Root Ports check Multicast TLPs in their Ingress Ports using the MC_Block_All and MC_Block_Untranslated registers associated with the Ingress Port. Endpoint Functions check Multicast TLPs they are preparing to send, using their MC_Block_All and MC_Block_Untranslated registers. If the MC_Block_All bit corresponding to the extracted MCG is set, the TLP is handled as an MC Blocked TLP. If the MC_Block_Untranslated bit corresponding to the extracted MCG is set and the TLP contains an Untranslated Address, the TLP, is also handled as an MC Blocked TLP.

IMPLEMENTATION NOTE
MC_Block_Untranslated and PIO Writes

Programmed I/O (PIO) Writes to Memory Space generally have Untranslated addresses since there is no architected mechanism for software to control the Address Type (AT) field for PIO Requests. Thus, if it's necessary for a given Switch to Multicast any PIO Writes, software should ensure that the appropriate MC_Block_Untranslated bits in the Upstream Port of that Switch are Clear. Otherwise, the Switch Upstream Port may block PIO Writes that legitimately target Multicast Windows. Since it may be necessary for software to clear MC_Block_Untranslated bits in a Switch Upstream Port for the sake of PIO Writes, the following are strongly recommended for a Root Complex capable of Address translation:

- All Integrated Endpoints each implement a Multicast Capability structure to provide access control for sending Untranslated Multicast TLPs.
- All peer-to-peer capable Root Ports each implement a Multicast Capability structure to provide access control for Untranslated Multicast TLPs that are forwarded peer-to-peer.

For similar reasons, with Multicast-capable Switch components where the Upstream Port is a Function in a Multi-Function Device, it is strongly recommended that any Endpoints in that Multi-Function Device each implement a Multicast Capability structure.

IMPLEMENTATION NOTE
Multicast Window Size

Each ultimate Receiver of a Multicast TLP may have a different Multicast Window size requirement. At one extreme, a Multicast Window may be required to cover a range of memory implemented within the device. At the other, it may only need to cover a particular offset at which a FIFO register is located. The MC_Window_Size_Requested field within the Multicast Capability register is used by an Endpoint to advertise the size of Multicast Window that it requires.

Unless available address space is limited, resource allocation software may be able to treat each request as a minimum and set the Multicast Window size via MC_Index_Position to accommodate the largest request. In some cases, a request for a larger window size can be satisfied by configuring a smaller window size and assigning the same membership to multiple contiguous MCGs.

IMPLEMENTATION NOTE
Multicast, ATS, and Redirection

The ACS P2P Request Redirection and ACS Direct Translated P2P mechanisms provide a means where P2P Requests with Untranslated Addresses can be redirected to the Root Complex (RC) for access control checking, whereas P2P Requests with Translated Addresses can be routed "directly" to their P2P targets for improved performance. No corresponding redirection mechanism exists for Multicast TLPs.

To achieve similar functionality, an RC might be configured to provide one or more target Memory Space ranges that are not in the Multicast address range, but the RC maps to "protected" Multicast Windows. Multicast TLP senders either with or without ATS capability then target these RC Memory Space ranges in order to access the protected Multicast Windows indirectly. When either type of sender targets these ranges with Memory Writes, each TLP that satisfies the access control checks will be reflected back down by the RC with a Translated Address targeting a protected Multicast Window.[126] ATS-capable senders can request and cache Translated Addresses using the RC Memory Space range, and then later use those Translated Addresses for Memory Writes that target protected Multicast Windows directly and can be Multicast without a taking a trip through the RC.

For hardware enforcement that only Translated Addresses can be used to target the protected Multicast Windows directly, software Sets appropriate MCG bits in the MC_Block_Untranslated register in all applicable Functions throughout the platform. Each MCG whose bit is set will cause its associated Multicast Window to be protected from direct access using Untranslated Addresses.

If the TLP is not blocked in a Switch or Root Complex it is forwarded out all of the Ports, except its Ingress Port, whose MC_Receive bit corresponding to the extracted MCG is set. In an Endpoint, it is consumed by all Functions whose MC_Receive bit corresponding to the extracted MCG is set. If no Ports forward the TLP or no Functions consume it, the TLP is silently dropped.

To prevent loops, it is prohibited for a Root Port or a Switch Port to forward a TLP back out its Ingress Port, even if so specified by the MC_Receive register associated with the Port. An exception is the case described in the preceding Implementation Note, where an RC reflects a unicast TLP that came in on an Ingress Root Port to a Multicast Window. In that case, when specified by the MC_Receive register associated with that Ingress Root Port, the RC is required to send the reflected TLP out the same Root Port that it originally came in.

A Multicast Hit suspends normal address routing, including default Upstream routing in Switches. When a Multicast Hit occurs, the TLP will be forwarded out only those Egress Ports whose MC_Receive bit associated with the MCG extracted from the address in the TLP is set. If the address in the TLP does not decode to any Downstream Port using normal address decode, the TLP will be copied to the Upstream Port only if so specified by the Upstream Port's MC_Receive register.

6.14.2 Multicast Ordering

No new ordering rules are defined for processing Multicast TLPs. All Multicast TLPs are Posted Requests and follow Posted Request ordering rules. Multicast TLPs are ordered per normal ordering rules relative to other TLPs in a component's ingress stream through the point of replication. Once copied into an egress stream, a Multicast TLP follows the same ordering as other Posted Requests in the stream.

126. If the original sender belongs to the MCG associated with this Window, the original sender will also receive a copy of the reflected TLP.

6.14.3 Multicast Capability Structure Field Updates

Some fields of the Multicast Capability structure may be changed at any time. Others cannot be changed with predictable results unless the MC_Enable bit is Clear in every Function of the component. The latter group includes MC_Base_Address and MC_Index_Position.

Fields which software may change at any time include MC_Enable, MC_Num_Group, MC_Receive, MC_Block_All, and MC_Block_Untranslated. Updates to these fields must themselves be ordered. Consider, for example, TLPs A and B arriving in that order at the same Ingress Port and in the same TC. If A uses value X for one of these fields, then B must use the same value or a newer value.

For Multi-Function Upstream Switch Ports Multicast TLPs received by one Switch or transmitted by one Endpoint Function are presented to the other parallel Endpoint Functions and the Downstream Switch Ports of the other parallel Switches (Functions are considered to be parallel if they are in the same Device. A single Multicast TLP is forwarded Upstream when any of the Upstream Switch Functions has the appropriate MC_Receive bit Set.

6.14.4 *MC Blocked TLP* Processing

When a TLP is blocked by the MC_Block_All or the MC_Block_Untranslated mechanisms, the TLP is dropped. The Function blocking the TLP serves as the Completer. The Completer must log and signal this MC Blocked TLP error as indicated in Figure 6-2 . In addition, the Completer must set the Signaled Target Abort bit in either its Status register or Secondary Status register as appropriate. To assist in fault isolation and root cause analysis, it is highly recommended that AER be implemented in Functions with Multicast capability.

In Root Complexes and Switches, if the error occurs with a TLP received by an Ingress Port, the error is reported by that Ingress Port. If the error occurs in an Endpoint Function preparing to send the TLP, the error is reported by that Endpoint Function.

6.14.5 *MC_Overlay* Mechanism

The MC_Overlay mechanism is provided to allow a single BAR in an Endpoint that doesn't contain a Multicast Capability structure to be used for both Multicast and unicast TLP reception. Software can configure the MC_Overlay mechanism to affect this by setting the MC_Overlay_BAR in a Downstream Port so that the Multicast address range, or a portion of it, is remapped (overlaid) onto the Memory Space range accepted by the Endpoint's BAR. At the Upstream Port of a Switch, the mechanism can be used to overlay a portion of the Multicast address range onto a Memory Space range associated with host memory.

A Downstream Port's MC_Overlay mechanism applies to TLPs exiting that Port. An Upstream Port's MC_Overlay mechanism applies to TLPs exiting the Switch heading Upstream. A Port's MC_Overlay mechanism does not apply to TLPs received by the Port, to TLPs targeting memory space within the Port, or to TLPs routed Peer-to-Peer between Functions in a Multi-Function Upstream Port.

When enabled, the overlay operation specifies that bits in the address in the Multicast TLP, whose bit numbers are equal to or higher than the MC_Overlay_Size field, be replaced by the corresponding bits in the MC_Overlay_BAR. In other words:

```
If (MC_Overlay_Size < 6)
    Then Egress_TLP_Addr = Ingress_TLP_Addr;
    Else Egress_TLP_Addr = { MC_Overlay_BAR[63:MC_Overlay_Size],
                             Ingress_TLP_Addr[MC_Overlay_Size-1:0] };
```

Equation 6-1 MC_Overlay Transform rules

If the TLP with modified address contains the optional ECRC, the unmodified ECRC will almost certainly indicate an error. The action to be taken if a TLP containing an ECRC is Multicast copied to an Egress Port that has MC_Overlay enabled depends upon whether or not optional support for ECRC regeneration is implemented. All of the contingent actions are outlined in Table 6-11 . If MC_Overlay is not enabled, the TLP is forwarded unmodified. If MC_Overlay is enabled and the TLP has no ECRC, the modified TLP, with its address replaced as specified in the previous paragraph is forwarded. If the TLP has an ECRC but ECRC regeneration is not supported, then the modified TLP is forwarded with its ECRC dropped and the TD bit in the header cleared to indicate no ECRC attached. If the TLP has an ECRC and ECRC regeneration is supported, then an ECRC check is performed before the TLP is forwarded. If the ECRC check passes, the TLP is forwarded with regenerated ECRC. If the ECRC check fails, the TLP is forwarded with inverted regenerated ECRC.

Table 6-11 ECRC Rules for MC_Overlay

MC_Overlay Enabled	TLP has ECRC	ECRC Regeneration Supported	Action if ECRC Check Passes	Action if ECRC Check Fails
No	x	x	Forward TLP unmodified	
Yes	No	x	Forward modified TLP	
Yes	Yes	No	Forward modified TLP with ECRC dropped and TD bit clear	
Yes	Yes	Yes	Forward modified TLP with regenerated ECRC	Forward modified TLP with inverted regenerated ECRC

IMPLEMENTATION NOTE
MC_Overlay and ECRC Regeneration

Switch and Root Complex Ports have the option to support ECRC regeneration. If ECRC regeneration is supported, then it is highly advised to do so robustly by minimizing the time between checking the ECRC of the original TLP and replacing it with an ECRC computed on the modified TLP. The TLP is unprotected during this time, leaving a data integrity hole if the pre-check and regeneration aren't accomplished in the same pipeline stage.

Stripping the ECRC from Multicast TLPs passing through a Port that has MC_Overlay enabled but doesn't support ECRC regeneration allows the receiving Endpoint to enable ECRC checking. In such a case, the Endpoint will enjoy the benefits of ECRC on non-Multicast TLPs without detecting ECRC on Multicast TLPs modified by the MC_Overlay mechanism.

When Multicast ECRC regeneration is supported, and an ECRC error is detected prior to TLP modification, then inverting the regenerated ECRC ensures that the ECRC error isn't masked by the regeneration process.

IMPLEMENTATION NOTE
Multicast to Endpoints That Don't Have Multicast Capability

An Endpoint Function that doesn't contain a Multicast Capability structure cannot distinguish Multicast TLPs from unicast TLPs. It is possible for a system designer to take advantage of this fact to employ such Endpoints as Multicast targets. The primary requirement for doing so is that the base and limit registers of the virtual PCI to PCI Bridge in the Switch Port above the device be configured to overlap at least part of the Multicast address range or that the MC_Overlay mechanism be employed. Extending this reasoning, it is even possible that a single Multicast target Function could be located on the PCI/PCI-X side of a PCI Express to PCI/PCI-X Bridge.

If an Endpoint without a Multicast Capability structure is being used as a Multicast target and the MC_Overlay mechanism isn't used, then it may be necessary to read from the Endpoint's Memory Space using the same addresses used for Multicast TLPs. Therefore, Memory Reads that hit in a Multicast Window aren't necessarily errors. Memory Reads that hit in a Multicast Window and that don't also hit in the aperture of an RCiEP or the Downstream Port of a Switch will be routed Upstream, per standard address routing rules, and be handled as a UR there.

IMPLEMENTATION NOTE
Multicast in a Root Complex

A Root Complex with multiple Root Ports that supports Multicast may implement as many Multicast Capability structures as its implementation requires. If it implements more than one, software should ensure that certain fields, as specified in Section 6.14.3 , are configured identically. To support Multicast to RCiEPs, the implementation needs to expose all TLPs identified as Multicast via the MC_Base_Address register to all potential Multicast target Endpoints integrated within it. Each such Integrated Endpoint then uses the MC_Receive register in its Multicast Capability structure to determine if it should receive the TLP.

IMPLEMENTATION NOTE
Multicast and Multi-Function Devices

All Port Functions and Endpoint Functions that are potential Multicast targets need to implement a Multicast Capability structure so that each has its own MC_Receive vector. Within a single component, software should configure the MC_Enable, MC_Base_Address, MC_Index_Position, and MC_Num_Group fields of these Capability structures identically. That being the case, it is sufficient to implement address decoding logic on only one instance of the Multicast BAR in the component.

IMPLEMENTATION NOTE

Congestion Avoidance

The use of Multicast increases the output link utilization of Switches to a degree proportional to both the size of the Multicast groups used and the fraction of Multicast traffic to total traffic. This results in an increased risk of congestion and congestion spreading when Multicast is used.

To mitigate this risk, components that are intended to serve as Multicast targets should be designed to consume Multicast TLPs at wire speed. Components that are intended to serve as Multicast sources should consider adding a rate limiting mechanism.

In many applications, the application's Multicast data flow will have an inherent rate limit and can be accommodated without causing congestion. Others will require an explicit mechanism to limit the injection rate, selection of a Switch with buffers adequate to hold the requisite bursts of Multicast traffic without asserting flow control, or selection of Multicast target components capable of sinking the Multicast traffic at the required rate. It is the responsibility of the system designer to choose the appropriate mechanisms and components to serve the application.

IMPLEMENTATION NOTE

The Host as a Multicast Recipient

For general-purpose systems, it is anticipated that the Multicast address range will usually not be configured to overlap with Memory Space that's directly mapped to host memory. If host memory is to be included as a Multicast recipient, the Root Complex may need to have some sort of I/O Memory Management Unit (IOMMU) that is capable of remapping portions of Multicast Windows to host memory, perhaps with page-level granularity. Alternatively, the MC_Overlay mechanism in the Upstream Port of a Switch can be used to overlay a portion of the Multicast address range onto host memory.

For embedded systems that lack an IOMMU, it may be feasible to configure Multicast Windows overlapping with Memory Space that's directly mapped to host memory, thus avoiding the need for an IOMMU. Specific details of this approach are beyond the scope of this specification.

6.15 Atomic Operations (AtomicOps)

An Atomic Operation (AtomicOp) is a single PCI Express transaction that targets a location in Memory Space, reads the location's value, potentially writes a new value back to the location, and returns the original value. This "read-modify-write" sequence to the location is performed atomically. AtomicOps include the following:

- FetchAdd (Fetch and Add): Request contains a single operand, the "add" value
  - Read the value of the target location.
  - Add the "add" value to it using two's complement arithmetic ignoring any carry or overflow.
  - Write the sum back to the target location.
  - Return the original value of the target location.
- Swap (Unconditional Swap): Request contains a single operand, the "swap" value